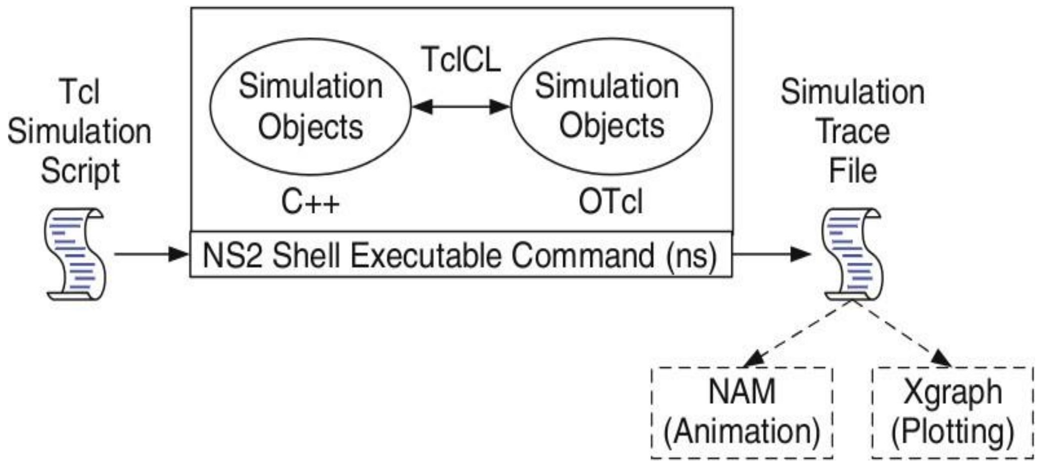
Introduction to NS-2

- 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 behaviours.

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

- **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.

#Define a ‘finish’ procedure

```
Proc finish { } {
global ns tracefile1 namfile
$ns flush-trace
Close $tracefile1
Close$namfile
Execnamout.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”
```

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 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 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 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"
```

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

```
Awk -F"|" -f empawk.awk empn.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 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 **"unset USERNAME"**
- **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

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]    /* Letter S is capital */
set nf [open lab1.nam w] /* open a nam trace file in write mode */
$ns namtrace-all $nf       /* nf – nam file */

set tf [open lab1.tr w] /* tf- trace file */
$ns trace-all $tf

proc finish { } {     /* provide space b/w proc and finish and all are in small case */
global ns nf tf
$ns flush-trace      /* clears trace file contents */
close $nf
close $tf
exec nam lab1.nam &
exit 0
}

set n0 [$ns node] /* creates 4 nodes */
set n1 [$ns node]
set n2 [$ns node]
set n3 [$ns node]

$ns duplex-link $n0 $n2 200Mb 10ms DropTail /*Letter M is capital Mb*/
$ns duplex-link $n1 $n2 100Mb 5ms DropTail /*D and T are capital*/
$ns duplex-link $n2 $n3 1Mb 1000ms DropTail

$ns queue-limit $n0 $n2 10
$ns queue-limit $n1 $n2 10

set udp0 [new Agent/UDP] /* Letters A,U,D and P are capital */
$ns attach-agent $n0 $udp0

set cbr0 [new Application/Traffic/CBR] /* A,T,C,B and R are capital*/
$cbr0 set packetSize_ 500 /*S is capital, space after underscore*/
$cbr0 set interval_ 0.005
$cbr0 attach-agent $udp0

set udp1 [new Agent/UDP]
$ns attach-agent $n1 $udp1

set cbr1 [new Application/Traffic/CBR]
$cbr1 attach-agent $udp1


set udp2 [new Agent/UDP]
$ns attach-agent $n2 $udp2
```

```
set cbr2 [new Application/Traffic/CBR]
$cbr2 attach-agent $udp2


set null0 [new Agent/Null] /* A and N are capital */
$ns attach-agent $n3 $null0

$ns connect $udp0 $null0
$ns connect $udp1 $null0

$ns at 0.1 "$cbr0 start"
$ns at 0.2 "$cbr1 start"
$ns at 1.0 "finish"

$ns run
```

<u>AWK file</u> (Open a new editor using "vi 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);
  }
 }
```

/*immediately after END should open braces '{'

```
END{
   printf("The number of packets dropped =%d\n",c);
}
```

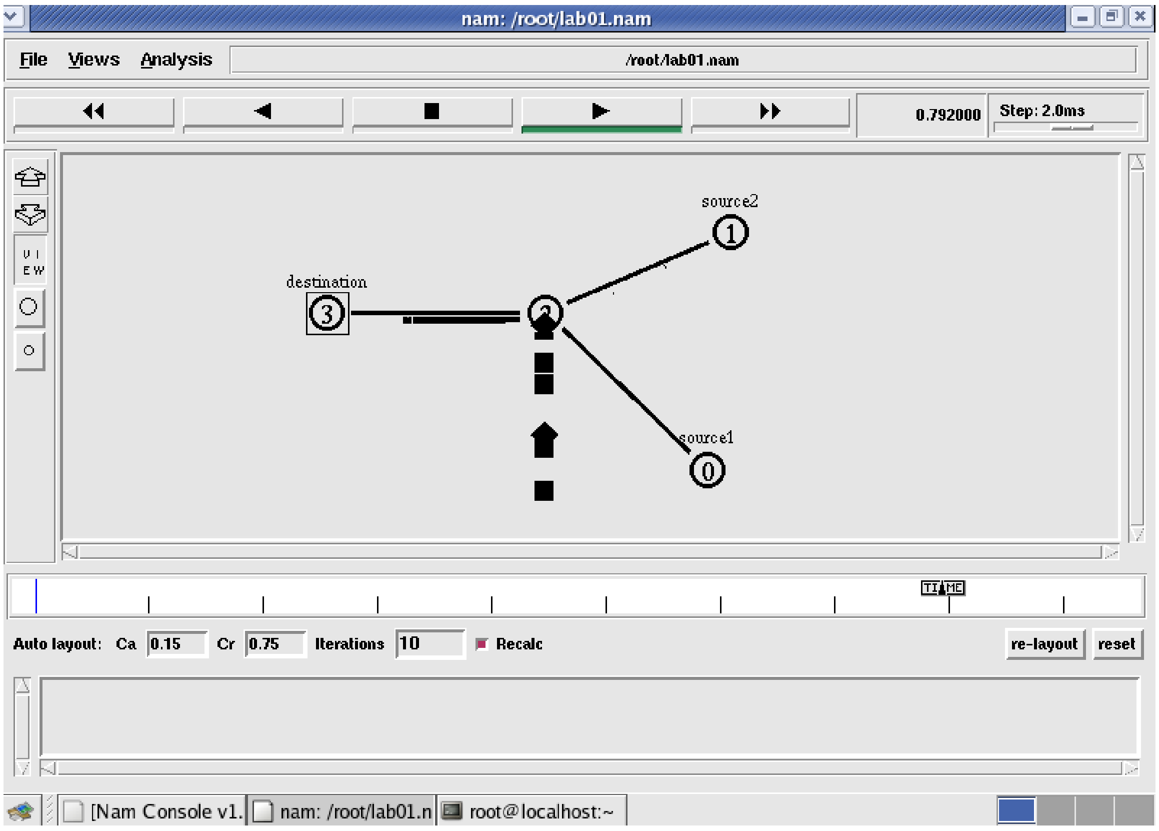
<u>Steps for execution</u>

1) Open vi editor and type program. Program name should have the extension " **.tcl** "
   [root@localhost ~]# vi lab1.tcl
2) Save the program by pressing **"ESC key"** first, followed by **"Shift and :"** keys simultaneously and type **"wq"** and press **Enter key**.
3) Open vi editor and type **awk** program. Program name should have the extension "**.awk** "
   [root@localhost ~]# vi lab1.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
   [root@localhost~]# ns lab1.tcl
   i) Here **"ns"** indicates network simulator. We get the topology shown in the snapshot.
   ii) Now press the play button in the simulation window and the simulation will begins.

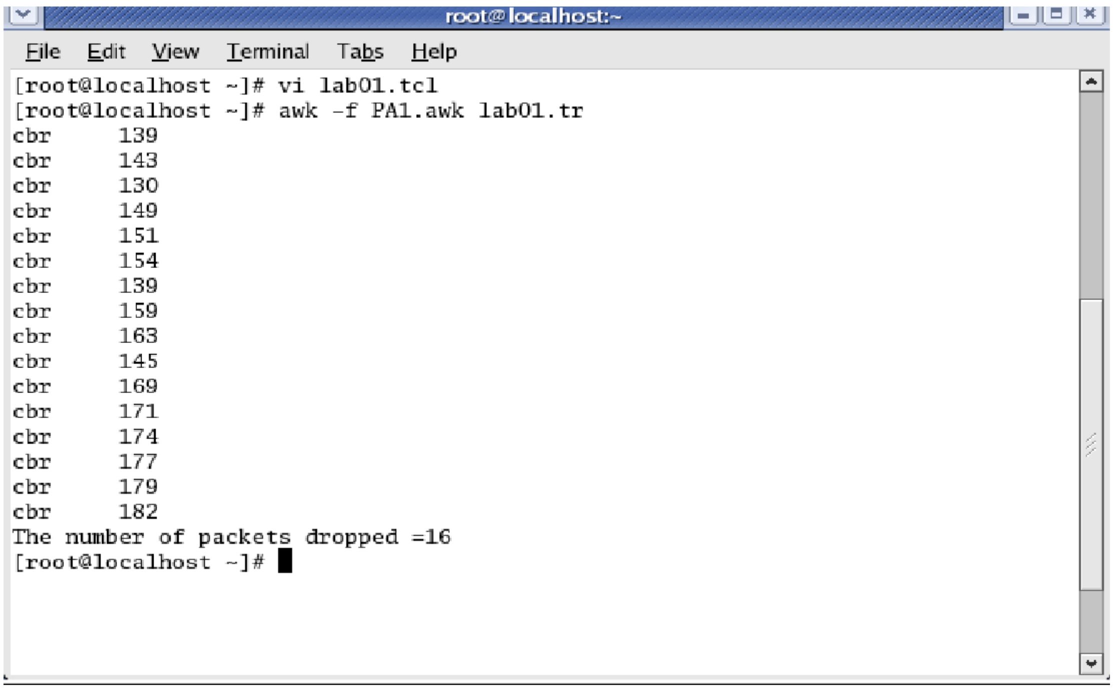
6) After simulation is completed run **awk file** to see the output ,
   [root@localhost~]# awk –f lab1.awk lab1.tr
7) To see the trace file contents open the file as ,
   [root@localhost~]# vi lab1.tr

<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>Topology</u>



Output



```
[root@localhost ~]# vi lab01.tcl
[root@localhost ~]# awk -f PA1.awk lab01.tr
cbr     139
cbr     143
cbr     130
cbr     149
cbr     151
cbr     154
cbr     139
cbr     159
cbr     163
cbr     145
cbr     169
cbr     171
cbr     174
cbr     177
cbr     179
cbr     182
The number of packets dropped =16
[root@localhost ~]#
```

Note:

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

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]

$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] /* letters A and P should be capital */
$ns attach-agent $n0 $p1
$p1 set packetSize_ 50000
$p1 set interval_ 0.0001

set p2 [new Agent/Ping] /* letters A and P should be capital */
$ns attach-agent $n1 $p2

set p3 [new Agent/Ping] /* letters A and P should be capital */
$ns attach-agent $n2 $p3
$p3 set packetSize_ 30000
$p3 set interval_ 0.00001

set p4 [new Agent/Ping] /* letters A and P should be capital */
$ns attach-agent $n3 $p4

set p5 [new Agent/Ping] /* letters A and P should be capital */
$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 "vi command" and write awk file and save with ".awk" extension)

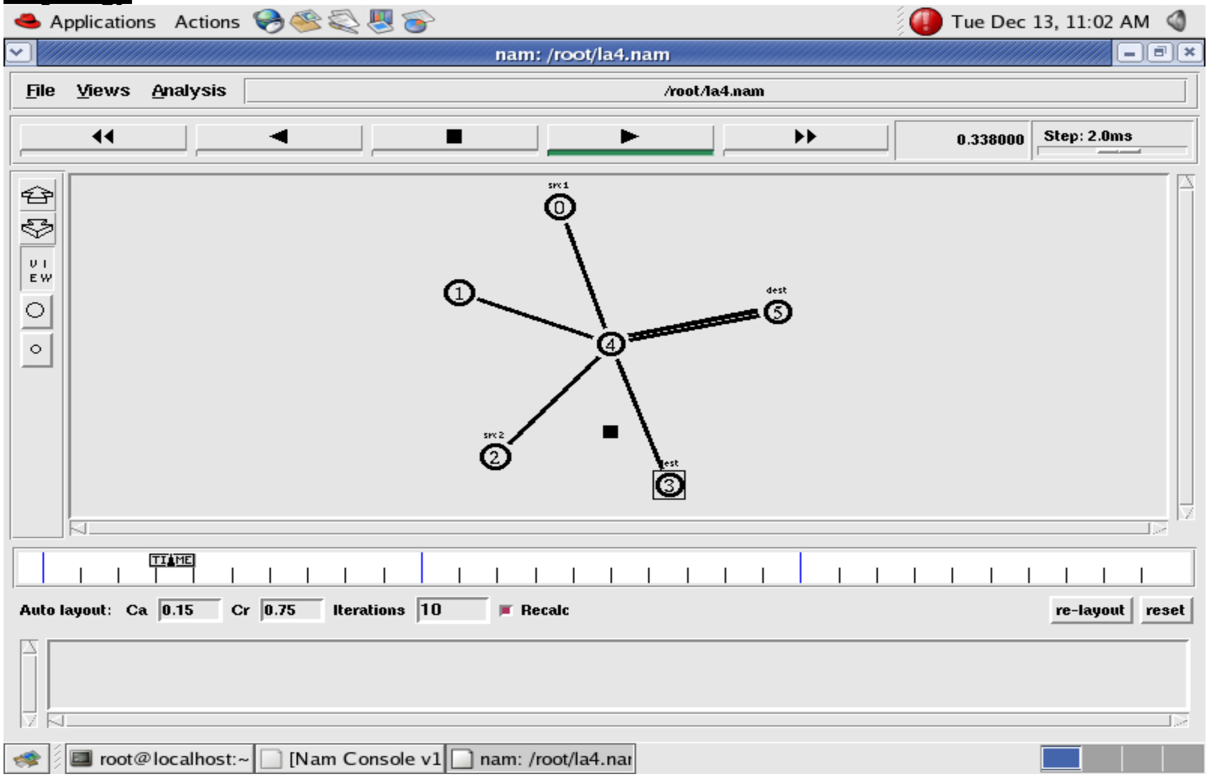
```
BEGIN{
drop=0;
}
{
 if($1= ="d" )
  {
   drop++;
   }
}
END{
printf("Total number of %s packets dropped due to congestion =%d\n",$5,drop);
}
```

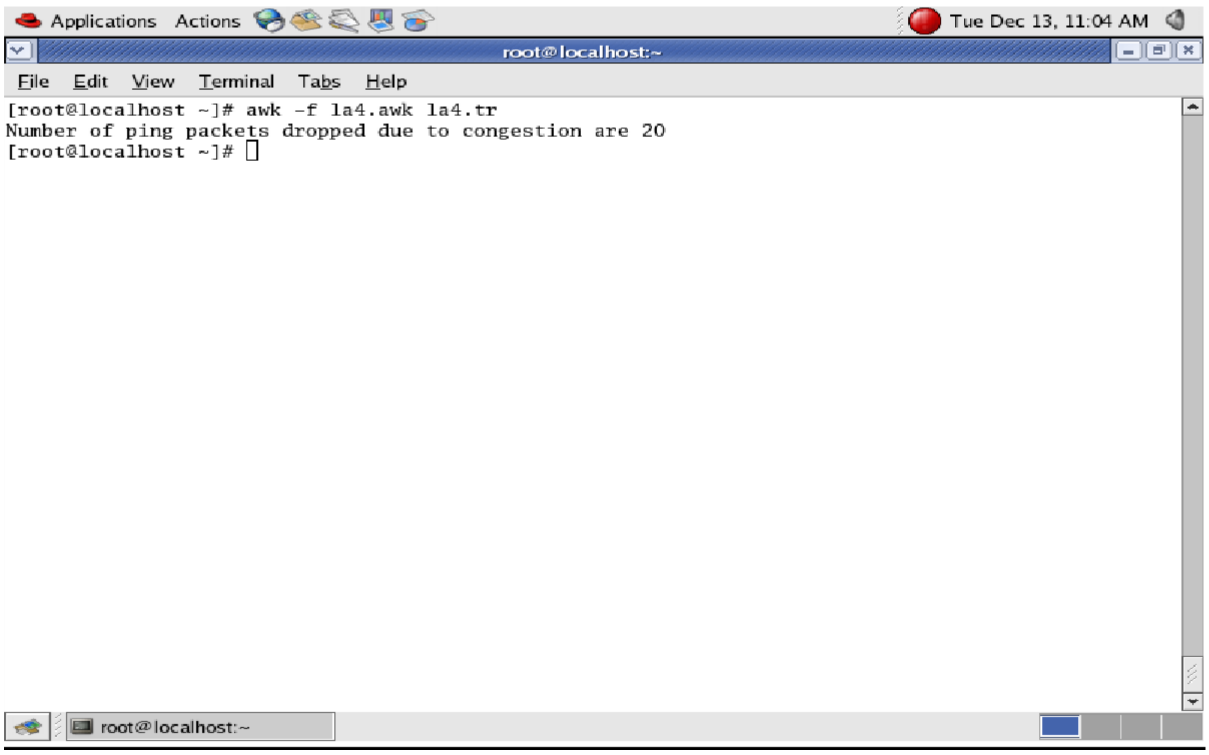
Steps for execution

1) Open vi editor and type program. Program name should have the extension " **.tcl** "
   [root@localhost ~]# vi lab2.tcl
2) Save the program by pressing **"ESC key"** first, followed by **"Shift and :"** keys simultaneously and type **"wq"** and press **Enter key**.
3) Open vi editor and type **awk** program. Program name should have the extension "**.awk** "
   [root@localhost ~]# vi 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
   [root@localhost~]# ns lab2.tcl
   i) Here **"ns"** indicates network simulator. We get the topology shown in the snapshot.
   ii) 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 ,
   [root@localhost~]# awk –f lab2.awk lab2.tr

7) To see the trace file contents open the file as ,
   [root@localhost~]# vi lab2.tr

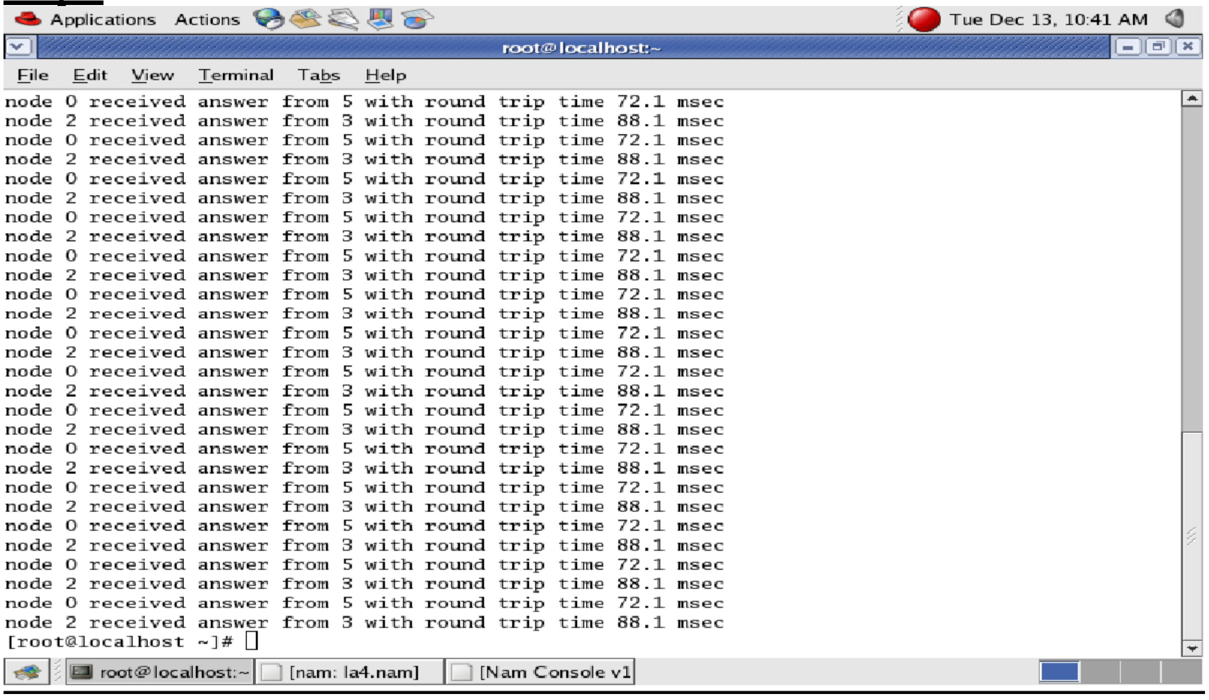
Topology





```
[root@localhost ~]# awk -f la4.awk la4.tr
Number of ping packets dropped due to congestion are 20
[root@localhost ~]#
```

Output



```
node 0 received answer from 5 with round trip time 72.1 msec
node 2 received answer from 3 with round trip time 88.1 msec
node 0 received answer from 5 with round trip time 72.1 msec
node 2 received answer from 3 with round trip time 88.1 msec
node 0 received answer from 5 with round trip time 72.1 msec
node 2 received answer from 3 with round trip time 88.1 msec
node 0 received answer from 5 with round trip time 72.1 msec
node 2 received answer from 3 with round trip time 88.1 msec
node 0 received answer from 5 with round trip time 72.1 msec
node 2 received answer from 3 with round trip time 88.1 msec
node 0 received answer from 5 with round trip time 72.1 msec
node 2 received answer from 3 with round trip time 88.1 msec
node 0 received answer from 5 with round trip time 72.1 msec
node 2 received answer from 3 with round trip time 88.1 msec
node 0 received answer from 5 with round trip time 72.1 msec
node 2 received answer from 3 with round trip time 88.1 msec
node 0 received answer from 5 with round trip time 72.1 msec
node 2 received answer from 3 with round trip time 88.1 msec
node 0 received answer from 5 with round trip time 72.1 msec
node 2 received answer from 3 with round trip time 88.1 msec
node 0 received answer from 5 with round trip time 72.1 msec
node 2 received answer from 3 with round trip time 88.1 msec
node 0 received answer from 5 with round trip time 72.1 msec
node 2 received answer from 3 with round trip time 88.1 msec
node 0 received answer from 5 with round trip time 72.1 msec
node 2 received answer from 3 with round trip time 88.1 msec
node 0 received answer from 5 with round trip time 72.1 msec
node 2 received answer from 3 with round trip time 88.1 msec
[root@localhost ~]#
```

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.

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 l
abel "dest1"

$ns make-lan "$n0 $n1 $n2 $n3 $n4" 100Mb 100ms LL Queue/DropTail Mac/802_3
/* should come in single line */
$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_ /* must put underscore ( _ ) after cwnd and no space between
them*/
$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 "vi command" and write awk file and save with ".awk" extension) cwnd:- means congestion window

```
BEGIN {
}
{
if($6= ="cwnd_") /* don't leave space after writing cwnd_ */
printf("%f\t%f\t\n",$1,$7); /* you must put \n in printf */
}
END {
}
```

Steps for execution

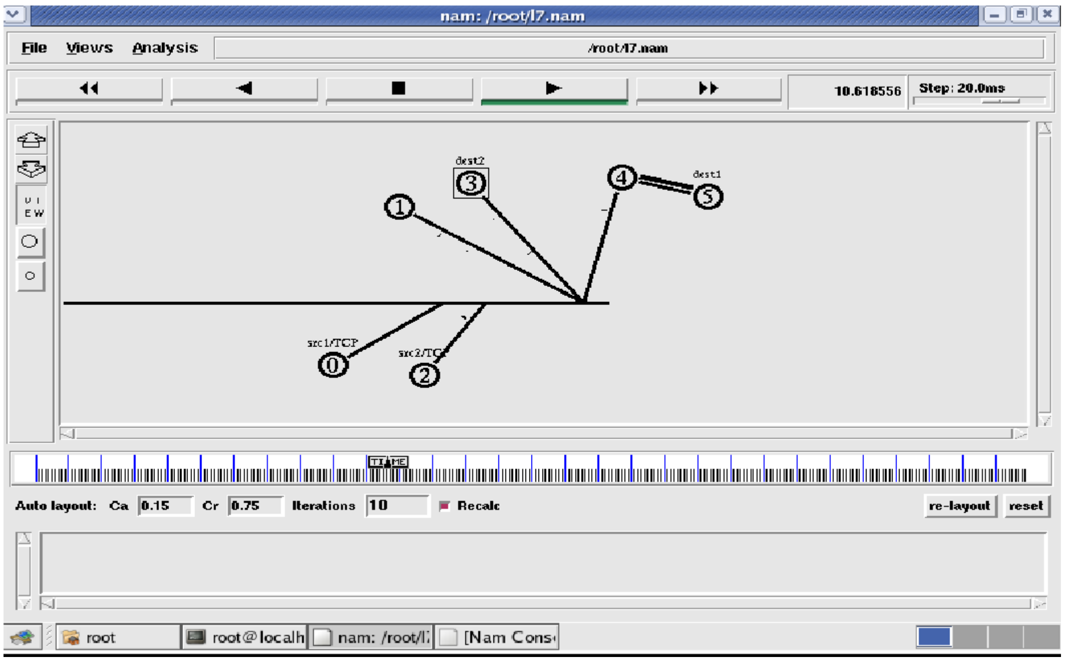
1) Open vi editor and type program. Program name should have the extension " **.tcl** "
   [root@localhost ~]# vi lab3.tcl
2) Save the program by pressing **"ESC key"** first, followed by **"Shift and :"** keys simultaneously and type **"wq"** and press **Enter key**.
3) Open vi editor and type **awk** program. Program name should have the extension "**.awk** "
   [root@localhost ~]# vi lab3.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
   [root@localhost~]# ns lab3.tcl
6) After simulation is completed run **awk file** to see the output ,
   i. **[root@localhost~]# awk –f lab3.awk file1.tr > a1**
   ii. **[root@localhost~]# awk –f lab3.awk file2.tr > a2**

iii. **[root@localhost~]# xgraph a1 a2**

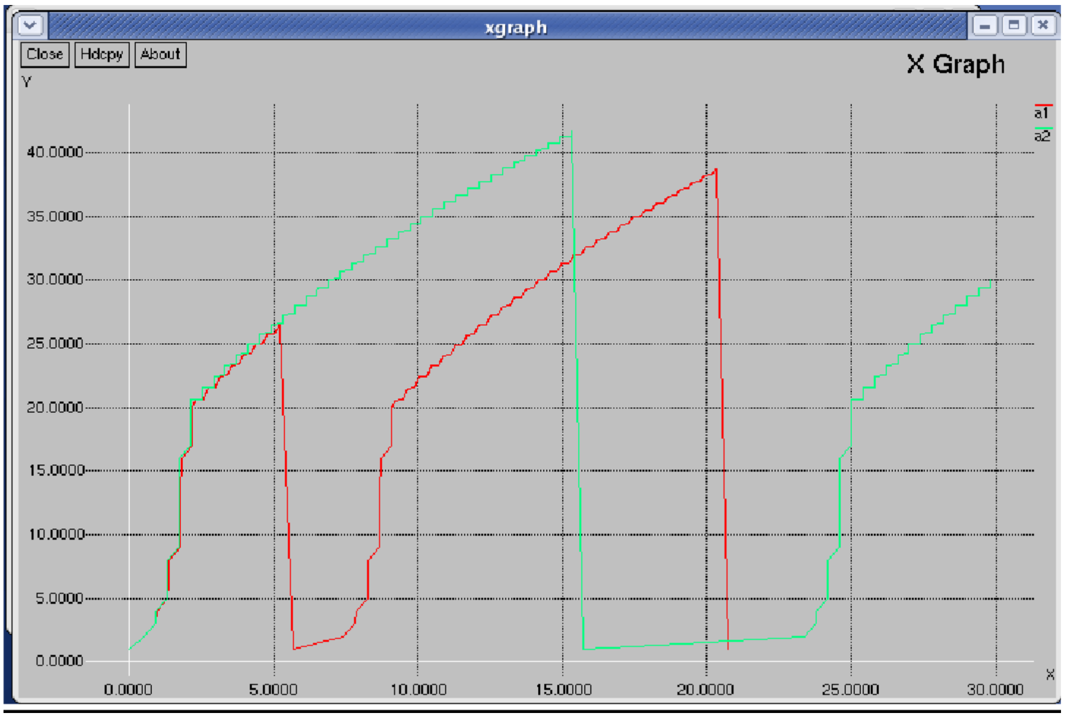
7) Here we are using the congestion window trace files i.e. **file1.tr** and **file2.tr** and we are redirecting the contents of those files to new files say **a1** and **a2** using **output redirection operator (>)**.
8) To see the trace file contents open the file as ,

   [root@localhost~]# vi lab3.tr

Topology



Output



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]
set tf [open lab4.tr w]
$ns trace-all $tf
set topo [new Topography]
$topo load_flatgrid 1000 1000
set nf [open lab4.nam w]
$ns namtrace-all-wireless $nf 1000 1000

$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 $topo \
                -agentTrace ON \
                -routerTrace ON
create-god 3
set n0 [$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 lab8.nam &
        close $tf
        exit 0
}
$ns at 250 "finish"
$ns run
```

<u>AWK file</u> (Open a new editor using “vi 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", ((count2*pack2*8)/(time2*1000000)));
```

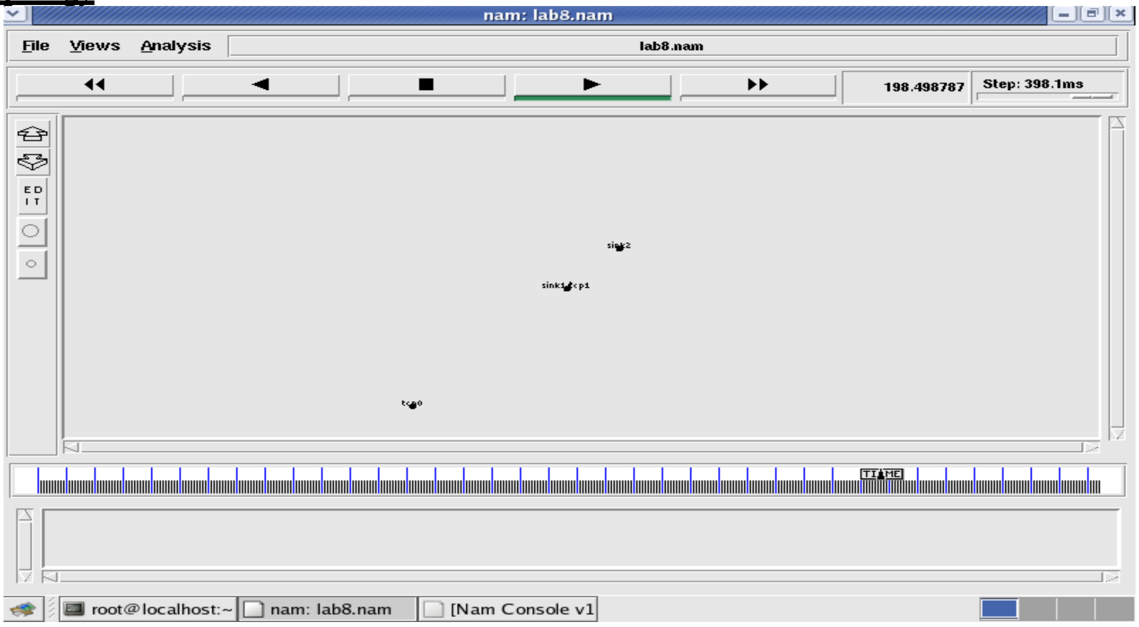
}

Steps for execution

1) Open vi editor and type program. Program name should have the extension " **.tcl** "
   [root@localhost ~]# vi lab4.tcl
2) Save the program by pressing **"ESC key"** first, followed by **"Shift and :"** keys simultaneously and type **"wq"** and press **Enter key**.
3) Open vi editor and type **awk** program. Program name should have the extension "**.awk** "
   [root@localhost ~]# vi lab4.awk
4) Save the program by pressing **"ESC key"** first, followed by **"Shift and :"** keys simultaneously and type **"wq"** and press **Enter key**.

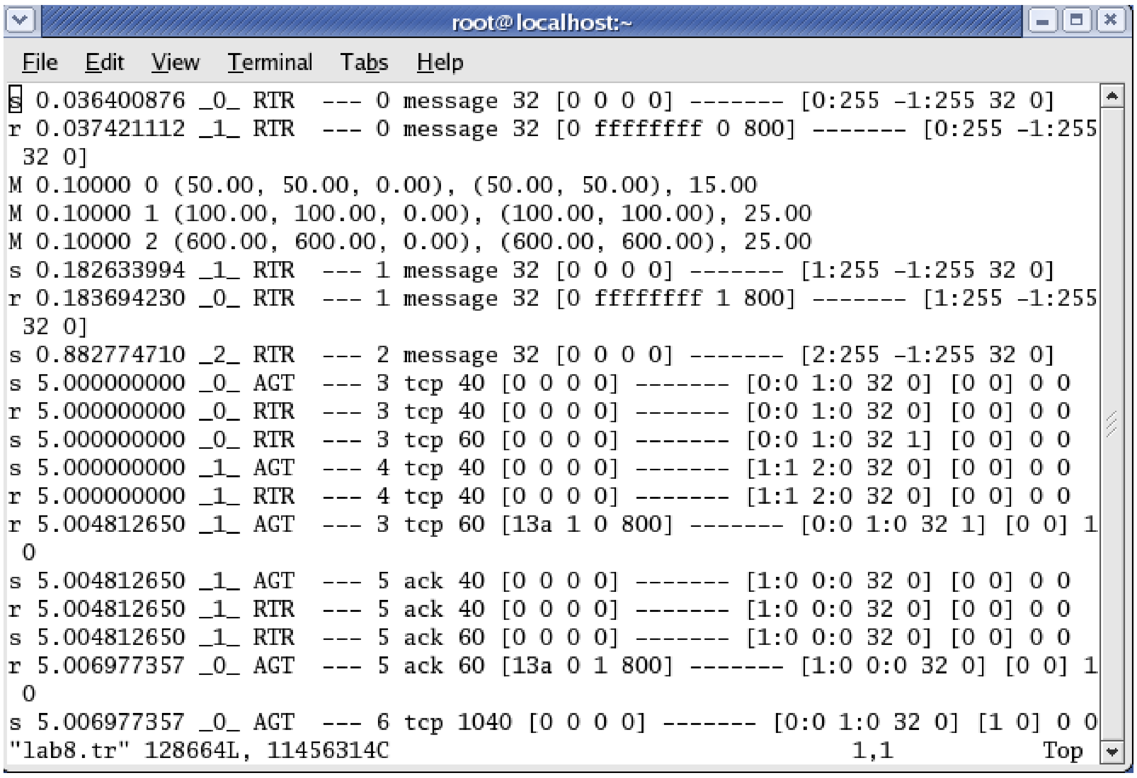
5) Run the simulation program
   [root@localhost~]# ns lab4.tcl
   i) Here **"ns"** indicates network simulator. We get the topology shown in the snapshot.
   ii) Now press the play button in the simulation window and the simulation will begins.
6) After simulation is completed run **awk file** to see the output ,
   [root@localhost~]# awk –f lab4.awk lab4.tr
7) To see the trace file contents open the file as ,
   [root@localhost~]# vi lab4.tr

Topology



Node 1 and 2 are communicating

Trace file

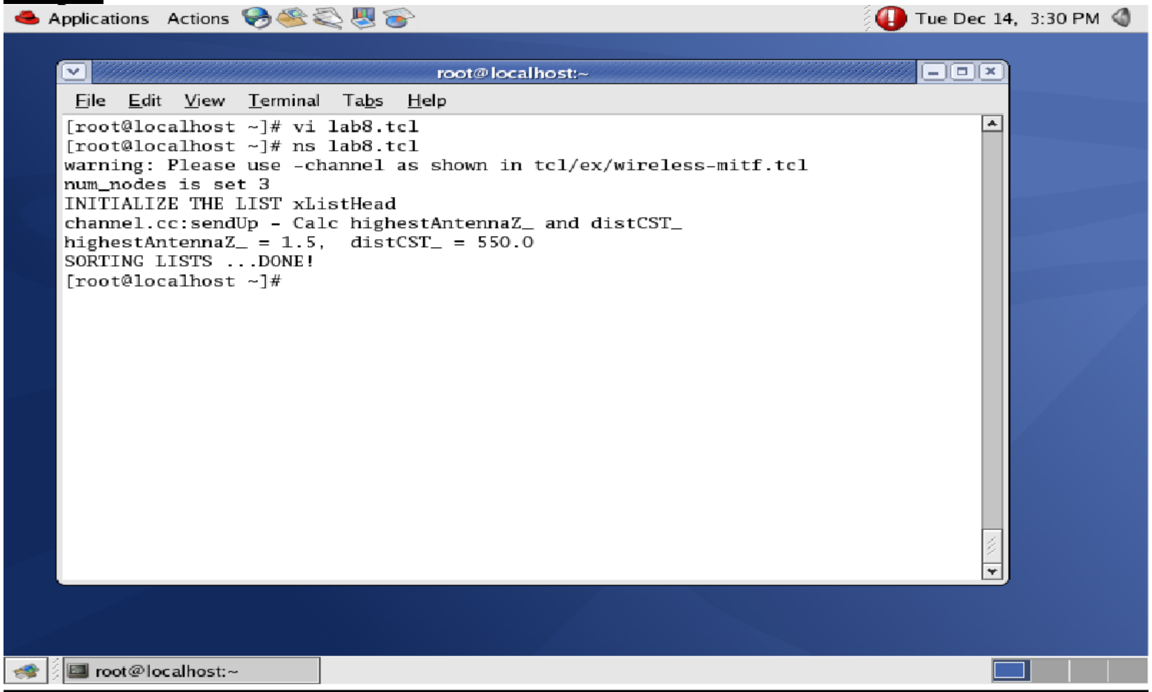


File Edit View Terminal Tabs Help

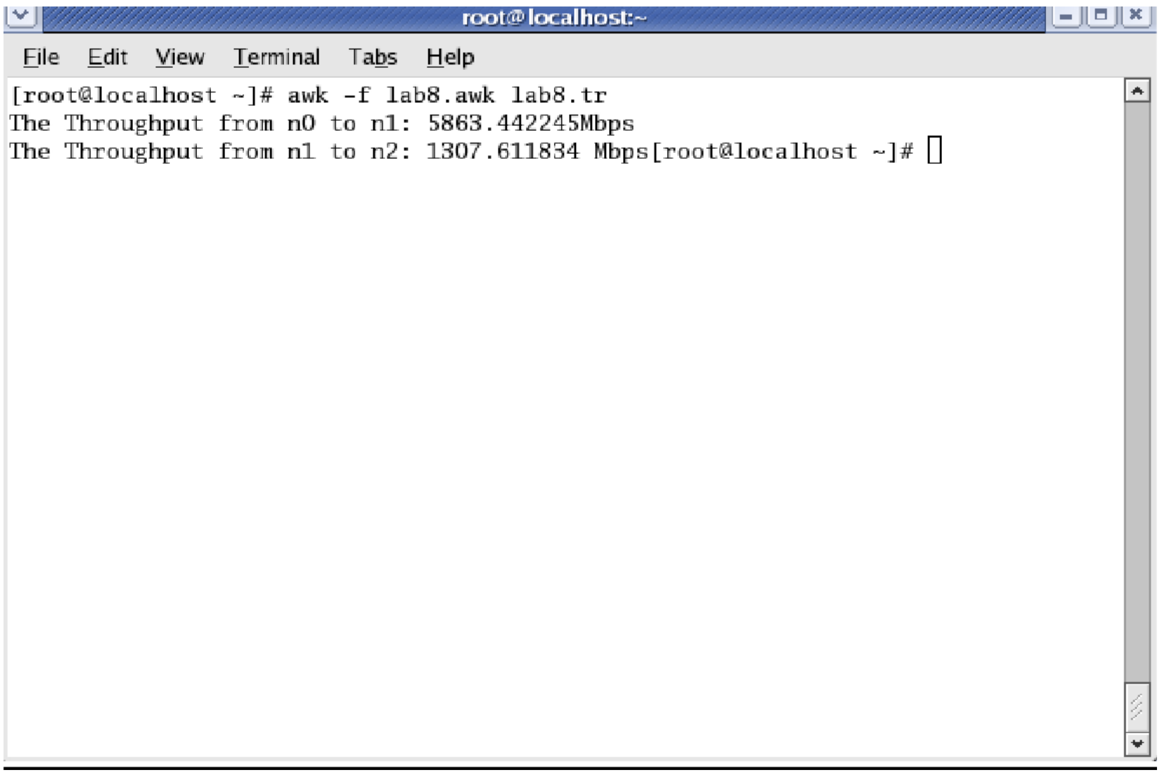
```
s 0.036400876 _0_ RTR  --- 0 message 32 [0 0 0 0] ------- [0:255 -1:255 32 0]
r 0.037421112 _1_ RTR  --- 0 message 32 [0 ffffffff 0 800] ------- [0:255 -1:255
 32 0]
M 0.10000 0 (50.00, 50.00, 0.00), (50.00, 50.00), 15.00
M 0.10000 1 (100.00, 100.00, 0.00), (100.00, 100.00), 25.00
M 0.10000 2 (600.00, 600.00, 0.00), (600.00, 600.00), 25.00
s 0.182633994 _1_ RTR  --- 1 message 32 [0 0 0 0] ------- [1:255 -1:255 32 0]
r 0.183694230 _0_ RTR  --- 1 message 32 [0 ffffffff 1 800] ------- [1:255 -1:255
 32 0]
s 0.882774710 _2_ RTR  --- 2 message 32 [0 0 0 0] ------- [2:255 -1:255 32 0]
s 5.000000000 _0_ AGT  --- 3 tcp 40 [0 0 0 0] ------- [0:0 1:0 32 0] [0 0] 0 0
r 5.000000000 _0_ RTR  --- 3 tcp 40 [0 0 0 0] ------- [0:0 1:0 32 0] [0 0] 0 0
s 5.000000000 _0_ RTR  --- 3 tcp 60 [0 0 0 0] ------- [0:0 1:0 32 1] [0 0] 0 0
s 5.000000000 _1_ AGT  --- 4 tcp 40 [0 0 0 0] ------- [1:1 2:0 32 0] [0 0] 0 0
r 5.000000000 _1_ RTR  --- 4 tcp 40 [0 0 0 0] ------- [1:1 2:0 32 0] [0 0] 0 0
r 5.004812650 _1_ AGT  --- 3 tcp 60 [13a 1 0 800] ------- [0:0 1:0 32 1] [0 0] 1
 0
s 5.004812650 _1_ AGT  --- 5 ack 40 [0 0 0 0] ------- [1:0 0:0 32 0] [0 0] 0 0
r 5.004812650 _1_ RTR  --- 5 ack 40 [0 0 0 0] ------- [1:0 0:0 32 0] [0 0] 0 0
s 5.004812650 _1_ RTR  --- 5 ack 60 [0 0 0 0] ------- [1:0 0:0 32 0] [0 0] 0 0
r 5.006977357 _0_ AGT  --- 5 ack 60 [13a 0 1 800] ------- [1:0 0:0 32 0] [0 0] 1
 0
s 5.006977357 _0_ AGT  --- 6 tcp 1040 [0 0 0 0] ------- [0:0 1:0 32 0] [1 0] 0 0
"lab8.tr" 128664L, 11456314C                                  1,1           Top
```

Here **"M"** indicates mobile nodes, **"AGT"** indicates Agent Trace, **"RTR"** indicates Router Trace

Output



```
[root@localhost ~]# vi lab8.tcl
[root@localhost ~]# ns lab8.tcl
warning: Please use -channel as shown in tcl/ex/wireless-mitf.tcl
num_nodes is set 3
INITIALIZE THE LIST xListHead
channel.cc:sendUp - Calc highestAntennaZ_ and distCST_
highestAntennaZ_ = 1.5,  distCST_ = 550.0
SORTING LISTS ...DONE!
[root@localhost ~]#
```



```
[root@localhost ~]# awk -f lab8.awk lab8.tr
The Throughput from n0 to n1: 5863.442245Mbps
The Throughput from n1 to n2: 1307.611834 Mbps[root@localhost ~]#
```

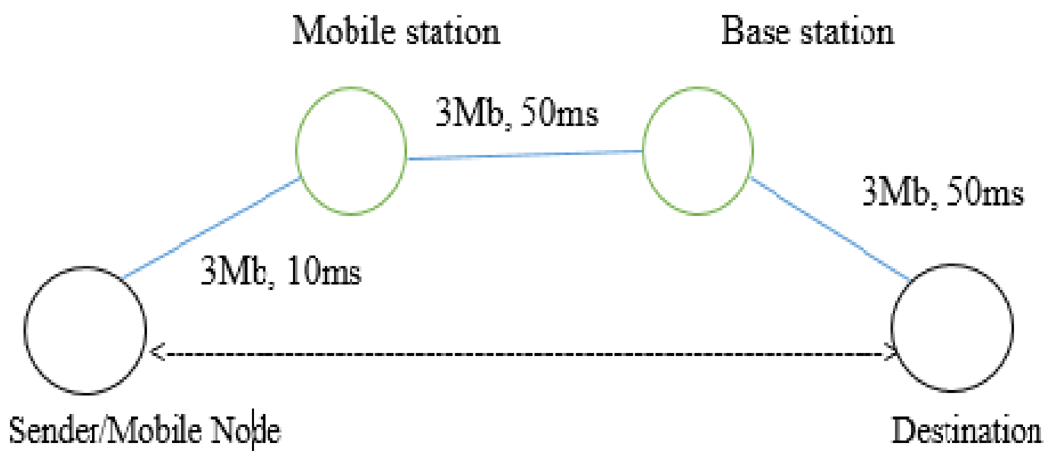
5. Implement and study the performance of GSM on NS2/NS3 (Using MAC layer) or equivalent environment.

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)

Design:



Source Code:

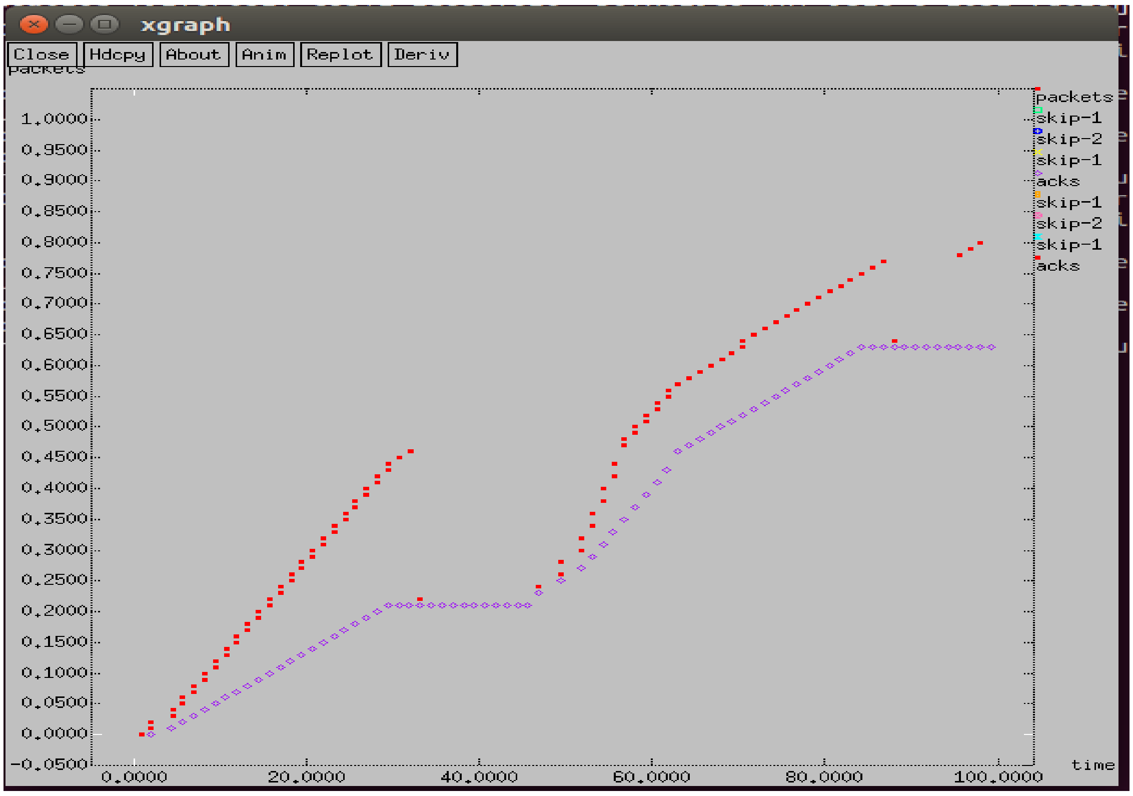
```
# General Parameters
set opt(title) zero ;
set opt(stop) 100 ;# Stop time.
set opt(ecn) 0 ;
# Topology
set opt(type) umts ;#type of link:
set opt(secondDelay) 55 ;# average delay of access links in ms
# AQM parameters
set opt(minth) 30
; set opt(maxth) 0
;
set opt(adaptive) 1 ;# 1 for Adaptive RED, 0 for plain RED #
Traffic generation.
```

```
set opt(flows) 0 ;# number of long-lived TCP flows
set opt(window) 30 ;# window for long-lived traffic
set opt(web) 2 ;# number of web sessions
# Plotting statistics.
set opt(quiet) 0 ;# popup anything
set opt(wrap) 100 ;# wrap plots
set opt(srcTrace) is ;# where to plot traffic set
opt(dstTrace) bs2 ;# where to plot traffic set
opt(umtsbuf) 10 ; # buffer size for umts
#default downlink bandwidth in bps
set bwDL(umts) 384000
#default uplink bandwidth in bps
set bwUL(umts) 64000
#default downlink propagation delay in seconds
set propDL(umts) .150
#default uplink propagation delay in seconds
set propUL(umts) .150
#default buffer size in packets
set buf(umts) 20
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 10ms 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 50ms DropTail
puts "Cell Topology"
}
proc set_link_params {t} {
global ns nodes bwUL bwDL propUL propDL buf
$ns bandwidth $nodes(bs1) $nodes(ms) $bwDL($t) simplex
$ns bandwidth $nodes(ms) $nodes(bs1) $bwUL($t) simplex
$ns delay $nodes(bs1) $nodes(ms) $propDL($t) simplex
$ns delay $nodes(ms) $nodes(bs1) $propDL($t) simplex
$ns queue-limit $nodes(bs1) $nodes(ms) $buf($t)
$ns queue-limit $nodes(ms) $nodes(bs1) $buf($t)
$ns bandwidth $nodes(bs2) $nodes(ms) $bwDL($t) simplex
$ns bandwidth $nodes(ms) $nodes(bs2) $bwUL($t) simplex
$ns delay $nodes(bs2) $nodes(ms) $propDL($t) simplex
$ns delay $nodes(ms) $nodes(bs2) $propDL($t) simplex
$ns queue-limit $nodes(bs2) $nodes(ms) $buf($t)
$ns queue-limit $nodes(ms) $nodes(bs2) $buf($t)
```

```
}
# RED and TCP parameters
Queue/RED set summarystats_ true
Queue/DropTail set summarystats_ true
Queue/RED set adaptive_ $opt(adaptive)
Queue/RED set q_weight_ 0.0
Queue/RED set thresh_ $opt(minth)
Queue/RED set maxthresh_ $opt(maxth)
Queue/DropTail set shrink_drops_ true
Agent/TCP set ecn_ $opt(ecn)
Agent/TCP set window_ $opt(window)
DelayLink set avoidReordering_ true

#source web.tcl
#Create topology
switch $opt(type)
{
gsm-
gprs-
umts {cell_topo}
}
set_link_params $opt(type)
$ns insert-delayer $nodes(ms) $nodes(bs1) [new Delayer]
$ns insert-delayer $nodes(bs1) $nodes(ms) [new Delayer]
$ns insert-delayer $nodes(ms) $nodes(bs2) [new Delayer]
$ns insert-delayer $nodes(bs2) $nodes(ms) [new Delayer]
# Set up forward TCP connection
if {$opt(flows) == 0} {
set tcp1 [$ns create-connection TCP/Sack1 $nodes(is) TCPSink/Sack1 $nodes(lp) 0]
set ftp1 [[set tcp1] attach-app FTP]
$ns at 0.8 "[set ftp1] start"
}
if {$opt(flows) > 0} {
set tcp1 [$ns create-connection TCP/Sack1 $nodes(is) TCPSink/Sack1 $nodes(lp) 0]
set ftp1 [[set tcp1] attach-app FTP]
$tcp1 set window_ 100
$ns at 0.0 "[set ftp1] start"
$ns at 3.5 "[set ftp1] stop"
set tcp2 [$ns create-connection TCP/Sack1 $nodes(is) TCPSink/Sack1 $nodes(lp) 0]
set ftp2 [[set tcp2] attach-app FTP]
$tcp2 set window_ 3
$ns at 1.0 "[set ftp2] start"
$ns at 8.0 "[set ftp2] stop"
}
proc stop {} {
global nodes opt
nf
set wrap $opt(wrap)
set sid [$nodes($opt(srcTrace)) id]
set did [$nodes($opt(dstTrace)) id]
```

```
if {$opt(srcTrace) == "is"} {
set a "-a out.tr"
} else {
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 $GETRC -s $sid -d $did -f 1 out.tr | \
$RAW2XG -s 0.01 -m $wrap -r >> plot.xgr
exec $GETRC -s $did -d $sid -f 1 out.tr | \
$RAW2XG -s 0.01 -m $wrap -a >> plot.xgr
exec ./xg2gp.awk plot.xgr
if {!$opt(quiet)} {
exec xgraph -bb -tk -nl -m -x time -y packets plot.xgr &
}

exit 0
}
$ns at $opt(stop) "stop"
$ns run
```

Output:

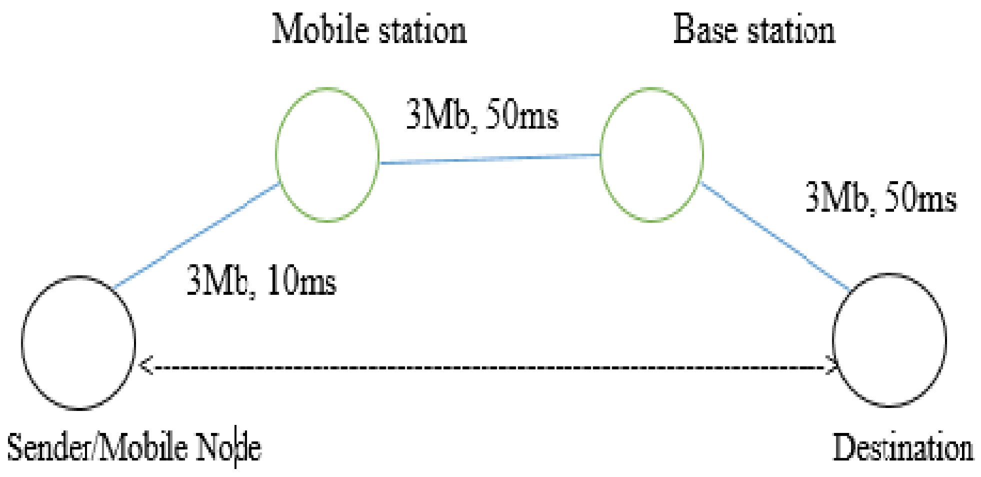


6. Implement and study the performance of CDMA on NS2/NS3 (Using stack called Call net) or equivalent environment.

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)

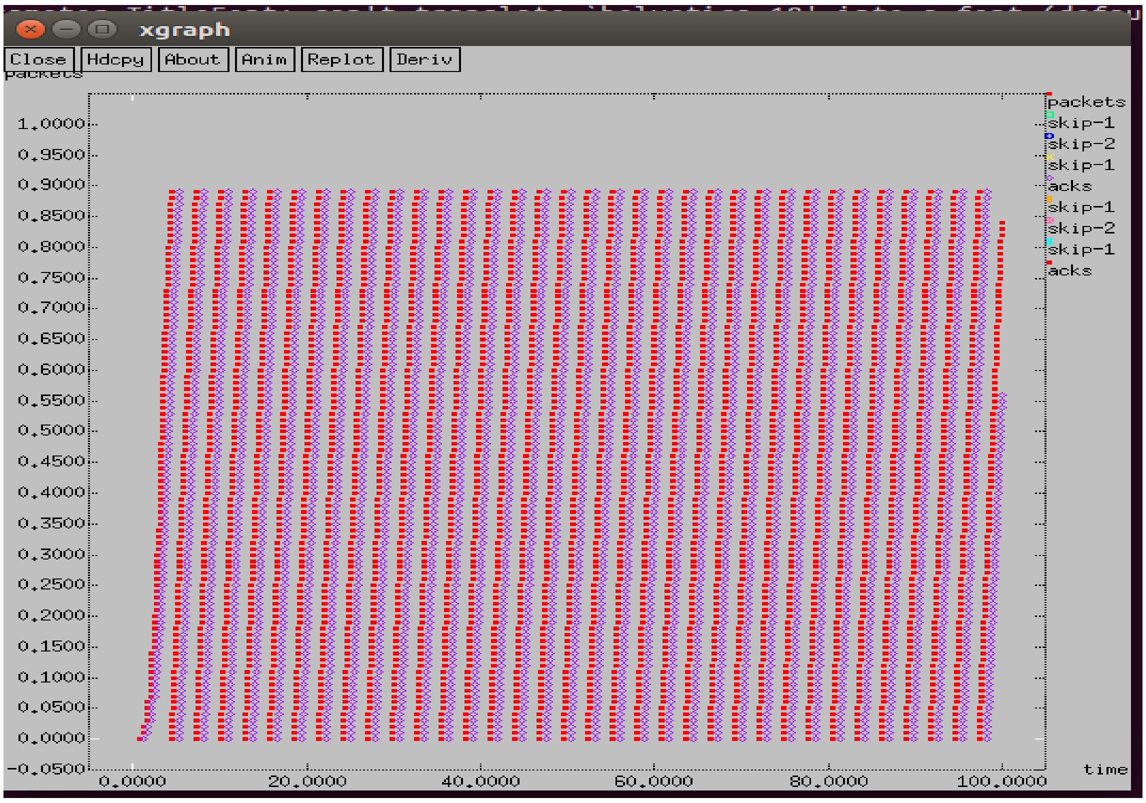


```
# General Parameters set
opt(title) zero;
set opt(stop) 100;# Stop time. set
opt(ecn) 0;# Topology
set opt(type) umts;#type of link:
set opt(secondDelay) 55;# average delay of  access links in ms # AQM
parameters
set opt(minth) 30;
set opt(maxth) 0;
set opt(adaptive) 1; # 1 for Adaptive RED, 0 for plain RED # Traffic
generation.
set opt(flows) 0;# number of long-lived TCP flows set
```

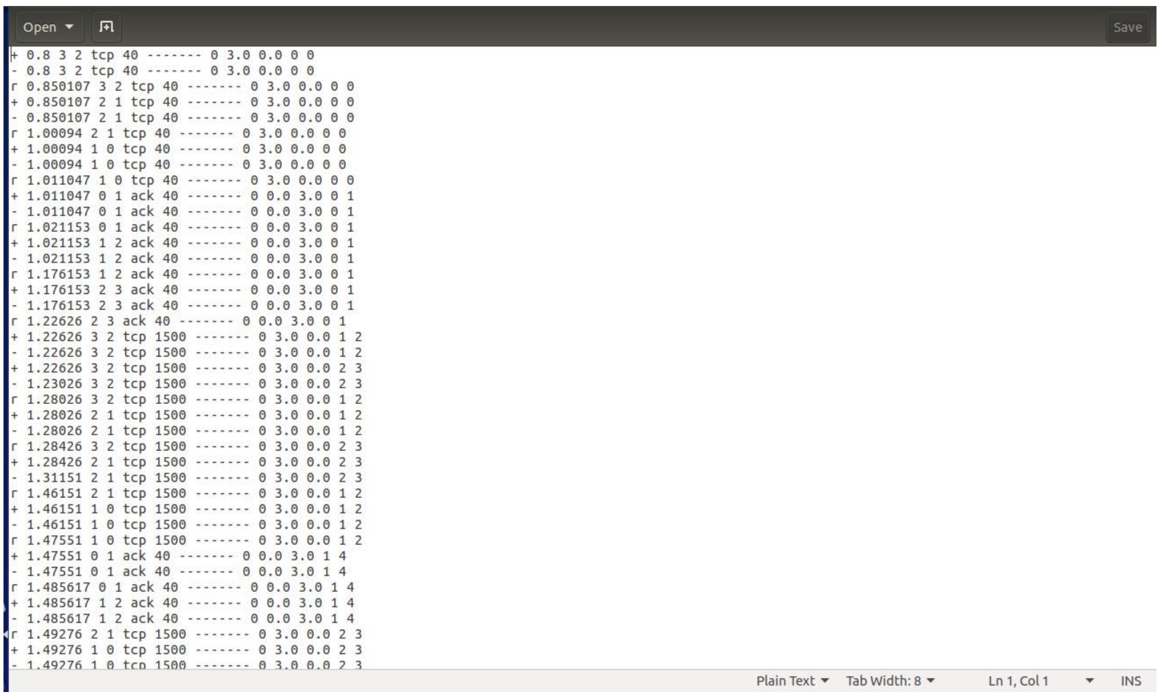
```
opt(window) 30 ;# window for long-lived traffic set opt(web)
2;# number of web sessions
# Plotting statistics.
set opt(quiet) 0; # popup anythng set
opt(wrap) 100 ;# wrap plots
set opt(srcTrace) is ;# where to plot traffic set
opt(dstTrace) bs2 ;# where to plot traffic set
opt(umtsbuf) 10; # buffer size for umts #default
downlink bandwidth in bps
set bwDL(umts) 384000
#default uplink bandwidth in bps set
bwUL(umts) 64000
#default downlink propagation delay in seconds set
propDL(umts) .150
#default uplink propagation delay in seconds set
propUL(umts) .150
#default buffer size in packets set
buf(umts) 20
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 10ms 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 50ms DropTail
puts "Cell Topology"
}

proc set_link_params {t} {
global ns nodes bwUL bwDL propUL propDL buf
$ns bandwidth $nodes(bs1) $nodes(ms) $bwDL($t) simplex
$ns bandwidth $nodes(ms) $nodes(bs1) $bwUL($t) simplex
$ns delay $nodes(bs1) $nodes(ms) $propDL($t) simplex
$ns delay $nodes(ms) $nodes(bs1) $propDL($t) simplex
$ns queue-limit $nodes(bs1) $nodes(ms) $buf($t)
$ns queue-limit $nodes(ms) $nodes(bs1) $buf($t)
$ns bandwidth $nodes(bs2) $nodes(ms) $bwDL($t) simplex
$ns bandwidth $nodes(ms) $nodes(bs2) $bwUL($t) simplex
$ns delay $nodes(bs2) $nodes(ms) $propDL($t) simplex
$ns delay $nodes(ms) $nodes(bs2) $propDL($t) simplex
$ns queue-limit $nodes(bs2) $nodes(ms) $buf($t)
$ns queue-limit $nodes(ms) $nodes(bs2) $buf($t)
}
```

```
# RED and TCP parameters Queue/RED set
summarystats_ true Queue/DropTail set
summarystats_ true
Queue/RED set adaptive_ $opt(adaptive)
Queue/RED set q_weight_ 0.0 Queue/RED set
thresh_ $opt(minth) Queue/RED set maxthresh_
$opt(maxth) Queue/DropTail set shrink_drops_
true Agent/TCP set ecn_ $opt(ecn) Agent/TCP set
window_ $opt(window) DelayLink set
avoidReordering_ true source web.tcl
#Create topology switch
$opt(type) {
 umts {cell_topo}
}
set_link_params $opt(type)
$ns insert-delayer $nodes(ms) $nodes(bs1) [new Delayer]
$ns insert-delayer $nodes(bs1) $nodes(ms) [new Delayer]
$ns insert-delayer $nodes(ms) $nodes(bs2) [new Delayer]
$ns insert-delayer $nodes(bs2) $nodes(ms) [new Delayer] # Set up
forward TCP connection
if {$opt(flows) == 0} {
set tcp1 [$ns create-connection TCP/Sack1 $nodes(is) TCPSink/Sack1 $nodes(lp) 0]
set ftp1 [[set tcp1] attach-app FTP]
$ns at 0.8 "[set ftp1] start"
}
if {$opt(flows) > 0} {
set tcp1 [$ns create-connection TCP/Sack1 $nodes(is) TCPSink/Sack1 $nodes(lp) 0]
set ftp1 [[set tcp1] attach-app FTP]
$tcp1 set window_ 100
$ns at 0.0 "[set ftp1] start"
$ns at 3.5 "[set ftp1] stop"

set tcp2 [$ns create-connection TCP/Sack1 $nodes(is) TCPSink/Sack1 $nodes(lp) 0]
set ftp2 [[set tcp2] attach-app FTP]
$tcp2 set window_ 3
$ns at 1.0 "[set ftp2] start"
$ns at 8.0 "[set ftp2] stop"
}
proc stop {} { global nodes
opt nf
set wrap $opt(wrap)
set sid [$nodes($opt(srcTrace)) id]
set did [$nodes($opt(dstTrace)) id]
if {$opt(srcTrace) == "is"} {
set a "-a out.tr"
} else {
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
$GETRC -s $sid -d $did -f 1 out.tr | \
$RAW2XG -s 0.01 -m $wrap -r >> plot.xgr exec
$GETRC -s $did -d $sid -f 1 out.tr | \
$RAW2XG -s 0.01 -m $wrap -a >> plot.xgr exec
./xg2gp.awk plot.xgr
if {!$opt(quiet)} {
exec xgraph -bb -tk -nl -m -x time -y packets plot.xgr &
}
exit 0
}
$ns at $opt(stop) "stop"
$ns run
```

Output:



CDMA Trace File



```
+ 0.8 3 2 tcp 40 ------- 0 3.0 0.0 0 0
- 0.8 3 2 tcp 40 ------- 0 3.0 0.0 0 0
r 0.850107 3 2 tcp 40 ------- 0 3.0 0.0 0 0
+ 0.850107 2 1 tcp 40 ------- 0 3.0 0.0 0 0
- 0.850107 2 1 tcp 40 ------- 0 3.0 0.0 0 0
r 1.00094 2 1 tcp 40 ------- 0 3.0 0.0 0 0
+ 1.00094 1 0 tcp 40 ------- 0 3.0 0.0 0 0
- 1.00094 1 0 tcp 40 ------- 0 3.0 0.0 0 0
r 1.011047 1 0 tcp 40 ------- 0 3.0 0.0 0 0
+ 1.011047 0 1 ack 40 ------- 0 0.0 3.0 0 1
- 1.011047 0 1 ack 40 ------- 0 0.0 3.0 0 1
r 1.021153 0 1 ack 40 ------- 0 0.0 3.0 0 1
+ 1.021153 1 2 ack 40 ------- 0 0.0 3.0 0 1
- 1.021153 1 2 ack 40 ------- 0 0.0 3.0 0 1
r 1.176153 1 2 ack 40 ------- 0 0.0 3.0 0 1
+ 1.176153 2 3 ack 40 ------- 0 0.0 3.0 0 1
- 1.176153 2 3 ack 40 ------- 0 0.0 3.0 0 1
r 1.22626 2 3 ack 40 ------- 0 0.0 3.0 0 1
+ 1.22626 3 2 tcp 1500 ------- 0 3.0 0.0 1 2
- 1.22626 3 2 tcp 1500 ------- 0 3.0 0.0 1 2
+ 1.22626 3 2 tcp 1500 ------- 0 3.0 0.0 2 3
- 1.23026 3 2 tcp 1500 ------- 0 3.0 0.0 2 3
r 1.28026 3 2 tcp 1500 ------- 0 3.0 0.0 1 2
+ 1.28026 2 1 tcp 1500 ------- 0 3.0 0.0 1 2
- 1.28026 2 1 tcp 1500 ------- 0 3.0 0.0 1 2
r 1.28426 3 2 tcp 1500 ------- 0 3.0 0.0 2 3
+ 1.28426 2 1 tcp 1500 ------- 0 3.0 0.0 2 3
- 1.31151 2 1 tcp 1500 ------- 0 3.0 0.0 2 3
r 1.46151 2 1 tcp 1500 ------- 0 3.0 0.0 1 2
+ 1.46151 1 0 tcp 1500 ------- 0 3.0 0.0 1 2
- 1.46151 1 0 tcp 1500 ------- 0 3.0 0.0 1 2
r 1.47551 1 0 tcp 1500 ------- 0 3.0 0.0 1 2
+ 1.47551 0 1 ack 40 ------- 0 0.0 3.0 1 4
- 1.47551 0 1 ack 40 ------- 0 0.0 3.0 1 4
r 1.485617 0 1 ack 40 ------- 0 0.0 3.0 1 4
+ 1.485617 1 2 ack 40 ------- 0 0.0 3.0 1 4
- 1.485617 1 2 ack 40 ------- 0 0.0 3.0 1 4
r 1.49276 2 1 tcp 1500 ------- 0 3.0 0.0 2 3
+ 1.49276 1 0 tcp 1500 ------- 0 3.0 0.0 2 3
- 1.49276 1 0 tcp 1500 ------- 0 3.0 0.0 2 3
```

PART-B

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

7.

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.

```
                      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
```

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.

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:

- 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

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.

Table 1: International Standard CRC Polynomials

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

Error detection with CRC

Consider a message represented by the polynomial M(x)

Consider a *generating polynomial* G(x)

This is used to generate a CRC = C(x) to be appended to M(x).

Note this G(x) is prime.

Steps:

1. Multiply M(x) by highest power in G(x). i.e. Add So much zeros to M(x).
2. Divide the result by G(x). The remainder = C(x).

   Special case: This won't work if bitstring =all zeros. We don't allow such an M(x).But M(x) bitstring = 1 will work, for example. Can divide 1101 into 1000.
3. If: x div y gives remainder c

   that means: x = n y + c

   Hence (x-c) = n y

   (x-c) div y gives remainder 0

   Here (x-c) = (x+c)

   Hence (x+c) div y gives remainder 0
4. Transmit: T(x) = M(x) + C(x)
5. Receiver end: Receive T(x). Divide by G(x), should have remainder 0.

Note if G(x) has order n - highest power is x^n, then G(x) will cover (n+1) bitsand the *remainder* will cover n bits. Add n bits (Zeros) to message.

Some CRC polynomials that are actually used

Some CRC polynomials

- CRC-8:

  x^8+x^2+x+1
  - Used in: 802.16 (along with error *correction*).
- CRC-

  CCITT:

  $x^{16}+x^{12}+x^5+1$
  - Used in: HDLC, SDLC, PPP default
- IBM-CRC-16

(ANSI): $x^{16}+x^{15}+x^{2}+1$

- 802.3:

  $x^{32}+x^{26}+x^{23}+x^{22}+x^{16}+x^{12}+x^{11}+x^{10}+x^{8}+x^{7}+x^{5}+x^{4}+x^{2}+x+1$

  - Used in: Ethernet, PPP rootion

```
package crc;
import java.io.*;
import java.util.*;
public class CRC {
	public static void main(String[] args)throws IOException {
		Scanner input = 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=input.nextInt();
		data=new int[data_bits];
		System.out.println("Enter data bits : ");
		for(int i=0; i<data_bits; i++)
		data[i]=input.nextInt();
		System.out.println("Enter number of bits in divisor : ");
		divisor_bits=input.nextInt();
		divisor=new int[divisor_bits];
		System.out.println("Enter Divisor bits : ");
		for(int i=0; i<divisor_bits; i++)
		divisor[i]=input.nextInt();
		tot_length=data_bits+divisor_bits-1;
		div=new int[tot_length];
		rem=new int[tot_length];
		crc=new int[tot_length];
		/*------------------ CRC GENERATION-----------------------*/
		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++) //append dividend and ramainder
		{
		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]);
		/*-------------------ERROR DETECTION---------------------*/
		System.out.println();
		System.out.println("Enter CRC code of "+tot_length+" bits : ");
		for(int i=0; i<crc.length; i++)
		crc[i]=input.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
CASE-1

```
Enter number of data bits :
5
Enter data bits :
1
0
1
1
```

```
1
Enter number of bits in divisor :
3
Enter Divisor bits :
1
0
1
Dividend (after appending 0's) are : 1011100
CRC code :
1011111
Enter CRC code of 7 bits :
1
0
1
1
1
0
0
Error
THANK YOU.... :)
```

CASE-2

```
Enter number of data bits :
5
Enter data bits :
1
1
0
1
1
Enter number of bits in divisor :
3
Enter Divisor bits :
1
1
1
Dividend (after appending 0's) are : 1101100
CRC code :
1101100
Enter CRC code of 7 bits :
1
1
0
1
1
0
0
No Error
THANK YOU.... :)
```

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 upd 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

Implementation Algorithm:

1. send my routing table to all my neighbors whenever my link table changes
2. when I get a routing table from a neighbor on port P with link metric M:
   a. add L to each of the neighbor's metrics
   b. for each entry (D, P', M') in the updated neighbor's table:
      i. if I do not have an entry for D, add (D, P, M') to my routing table
      ii. if I have an entry for D with metric M", add (D, P, M') to my routing table if M' < M"

3. if my routing table has changed, send all the new entries to all my neighbors.

```
package bellman;
import java.util.Scanner;
public class Bellman {
	private int d[];
	private int n;
	public static final int MAX= 9999;
	public static void main(String[] args) {
		int n=0,s;
		Scanner scanner = new Scanner(System.in);
		System.out.println("Enter the number of vertices");
		n = scanner.nextInt();
		int a[][] = new int[n][n];
		System.out.println("Enter the adjacency matrix");
		for (int i = 0; i < n; i++)
		{
		for (int j = 0; j < n; j++)
		{
		a[i][j] = scanner.nextInt();
		if (a[i][j] == 0)
		{
		a[i][j] = MAX;
		}
		}
		}
		System.out.println("Enter the source vertex");
		s = scanner.nextInt();
		Bellman b= new Bellman(n);
		b.Bellmanford(s,a);
		scanner.close();
	}
	public Bellman(int n)
	{
	this.n = n;
	d = new int[n + 1];
	}
	public void Bellmanford(int s, int a[][])
	{
		for (int i = 0; i < n; i++)
		{
		d[i] = MAX;
		}
		d[s] = 0;
		for (int k = 0; k < n - 1; k++)
		{ for (int i = 0; i < n; i++)
```

```
                {
                for (int j = 0; j < n; j++)
                { if (a[i][j] != MAX)
                { if (d[j] > d[i]+ a[i][j])
                { d[j] = d[i]+ a[i][j];
                }
                }
                }
                }
                }
                for (int i = 0; i <n; i++)
                {
                System.out.println("distance of source " + s + " to "+ i + " is " + d[i]);
                }
                }
}
```

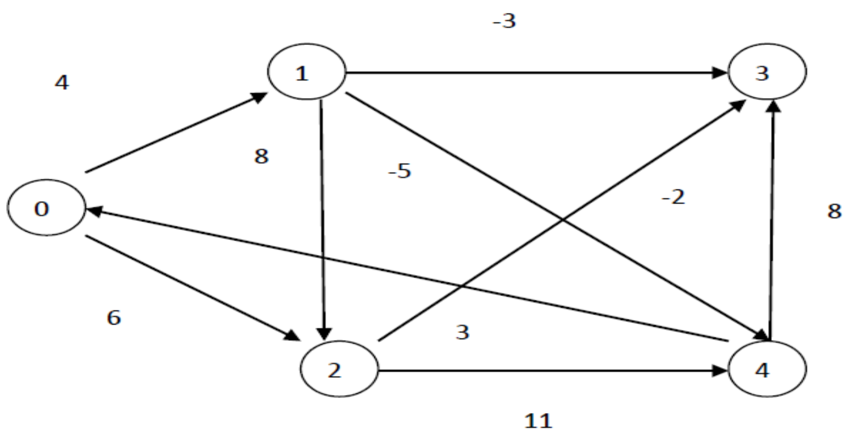
<u>OUTPUT</u>
<u>CASE-1</u>

```
Enter the number of vertices
5
Enter the adjacency matrix
0         4         6       9999        9999
9999      0         8       -3          -5
9999      9999      0       -2          11
9999      9999      9999     0          9999
3         9999      9999    9999        9999

Enter the source vertex
0
distance of source 0 to 0 is 0
distance of source 0 to 1 is 4
distance of source 0 to 2 is 6
distance of source 0 to 3 is 1
distance of source 0 to 4 is -1
```

<u>CASE-2</u>

```
Enter the number of vertices
5
Enter the adjacency matrix
0         3        9999      6        3
9999      9999      9        4        9999
9999      9999     9999     9999      6
2         9999     9999      3        1
9999      9999      0        2        9999

Enter the source vertex
3
```

```
distance of source 3 to 0 is 2
distance of source 3 to 1 is 5
distance of source 3 to 2 is 14
distance of source 3 to 3 is 0
distance of source 3 to 4 is 1
```



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.

// SERVER SIDE PROGRAM (TYPE IN A DIFFERENT WORKSPACE) //

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.

```
package tcpserver;
import java.net.*;
import java.io.*;
public class Server1 {

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

                // establishing the connection with the server
           ServerSocket sersock = new ServerSocket(4000);
           System.out.println("Server ready for connection");
           Socket sock = sersock.accept();          // binding with port: 4000
           System.out.println("Connection is successful and wating");

                              // reading the file name from client
           InputStream istream = sock.getInputStream( );
           BufferedReader fileRead =new BufferedReader(new InputStreamReader(istream));
           String fname = fileRead.readLine( );
                             // reading file contents
           BufferedReader contentRead = new BufferedReader(new FileReader(fname) );

                           // keeping output stream ready to send the contents
           OutputStream ostream = sock.getOutputStream( );
           PrintWriter pwrite = new PrintWriter(ostream, true);

           String str;
           while((str = contentRead.readLine()) !=  null) // reading line-by-line from file
           {
                pwrite.println(str);        // sending each line to client
           }

           sock.close();  sersock.close();      // closing network sockets
           pwrite.close();  fileRead.close(); contentRead.close();
         }
```

```
            }
```

// CLIENT SIDE PROGRAM (TYPE IN A DIFFERENT WORKSPACE) //

```
package tcpclient;
import java.net.*;
import java.io.*;
public class Client {

            public static void main(String[] args)throws Exception
            {
                Socket sock = new Socket( "127.0.0.1", 4000);

        // reading the file name from keyboard. Uses input stream
System.out.print("Enter the file name");
BufferedReader keyRead = new BufferedReader(new InputStreamReader(System.in));
String fname = keyRead.readLine();

  // sending the file name to server. Uses PrintWriter
OutputStream  ostream = sock.getOutputStream( );
PrintWriter pwrite = new PrintWriter(ostream, true);
pwrite.println(fname);
                    // receiving the contents from server.  Uses input stream
InputStream istream = sock.getInputStream();
BufferedReader socketRead = new BufferedReader(new InputStreamReader(istream));

String str;
while((str = socketRead.readLine())  !=  null) // reading line-by-line
{
System.out.println(str);
}
pwrite.close();
socketRead.close();
keyRead.close();
}
}
```

OUTPUT:
Note: As we are using eclipse save and run the server and client program in different workspace

Server side: (First run the sever)

```
Server ready for connection
Connection is successful and waiting for chatting
```

Client side: (After the sever is ready, run the client program...... note file need to saved at the sever side)

Enter the file name BGSIT.txt

BGSIT Providing Quality of Technical Education To Build Tomorrows Engineers.
Dept. of CSE & ISE
Computer Network Laboratory
15CSL57
(contents of the file name BGSIT.txt will be displayed as shown above)

<u>To create the file at server side follow below steps</u>
My computer →C drive→User→Dell→Workspace (server workspace)→New text file

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.

// SERVER SIDE PROGRAM (TYPE IN A DIFFERENT WORKSPACE) //

```
package datagramsocketserver;
import java.io.*;
import java.net.*;
public class Server {

        public static void main(String[] args) throws Exception {
                BufferedReader buff=new BufferedReader(new InputStreamReader(System.in));
                DatagramSocket dsock = new DatagramSocket();
                InetAddress address = InetAddress.getLocalHost( );

                System.out.println("Server is ready");

                    while(true)
                             {
                               Thread.sleep(1000);
                              System.out.println("Enter message to be send to client from server ");
                               String s1 =buff.readLine();
                              byte arr[] = s1.getBytes( );
                     DatagramPacket dpack = new DatagramPacket(arr, arr.length, address, 2000);
                                  dsock.send(dpack);
                             }

                }
                }
```

// CLIENT SIDE PROGRAM (TYPE IN A DIFFERENT WORKSPACE) //

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

public class Client {

        public static void main(String[] args) throws SocketException, IOException {
     DatagramSocket dsock = new DatagramSocket(2000);
     DatagramPacket dpack;
     while(true)
        {
            byte arr1[] = new byte[100];
            dpack = new DatagramPacket(arr1, arr1.length);

            dsock.receive(dpack);

            String str = new String(arr1);

            System.out.println("Message recieved from server");
            System.out.println(str);

        }
    }
}
```

OUTPUT:

Note: As we are using eclipse save and run the server and client program in different workspace

Server side: (First run the sever)

```
Server is ready
Enter message to be send to client from server
```

Client side: (After the sever is ready, run the client program)

(Then come to Server program)

```
Server is ready
Enter message to be send to client from server
BGSIT Providing Quality of Technical Education To Build Tomorrows Engineers.
```

(Above text is the entered message at server side)

(Then come to client Side)

Client Side: (it displays the below message)

```
Message received from server

BGSIT Providing Quality of Technical Education To Build Tomorrows Engineers.
```

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

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.

A very simple example of RSA encryption

This is an extremely simple example using numbers you can work out on a pocket calculator (those of you over the age of 35 can probably even do it by hand on paper).

1. Select primes p = 11, q =

3. 2. n = pq = 11.3 = 33

   phi = (p-1)(q-1) = 10.2 = 20

3. 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, phi) = gcd(e, (p-1)(q-1)) = gcd(3, 20) = 1

4. Compute d such that ed ≡ 1 (mod phi)

   i.e. compute d = $e^{\wedge-1}$ mod phi = $3^{\wedge-1}$ mod 20

   i.e. find a value for d such that phi divides (ed-1)

   i.e. find d such that 20 divides 3d-1.

   Simple testing (d = 1, 2, ...) gives d = 7

   Check: ed-1 = 3.7 - 1 = 20, which is divisible by phi.

5. Public key = (n, e) = (33, 3)

   Private key = (n, d) = (33, 7).

This is actually the smallest possible value for the modulus n for which the RSA algorithm works.

Now say we want to encrypt the message m =
7, c = $m^{\wedge e}$ mod n = $7^{\wedge 3}$ mod 33 = 343 mod 33 =

13. Hence the ciphertext c = 13.

To check decryption we compute
m' = $c^{\wedge d}$ mod n = $13^{\wedge 7}$ mod 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.

One way of calculating m' is as follows:-
m' = $13^{\wedge 7}$ mod 33 = $13^{\wedge(3+3+1)}$ mod 33 = $13^{\wedge 3}$.$13^{\wedge 3}$.13 mod 33
= ($13^{\wedge 3}$ mod 33).($13^{\wedge 3}$ mod 33).(13 mod 33) mod 33
= (2197 mod 33).(2197 mod 33).(13 mod 33) mod 33
= 19.19.13 mod 33 = 4693 mod 33
= 7.

Now if we calculate the cipher text c for all the possible values of m (0 to 32), we get

m	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
c	0	1	8	27	31	26	18	13	17	3	10	11	12	19	5	9	4

m	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32
c	29	24	28	14	21	22	23	30	16	20	15	7	2	6	25	32

Note that all 33 values of m (0 to 32) map to a unique code c in the same range in a sort of random manner. In this case we have nine values of m that map to the same value of c - these are known as *unconcealed messages*. m = 0 and 1 will always do this for any N, no matter how large. But in practice, higher values shouldn't be a problem when we use large values for N.

If we wanted to use this system to keep secrets, we could let A=2, B=3, ..., Z=27. (We specifically avoid 0 and 1 here for the reason given above). Thus the plaintext message "HELLOWORLD" would be represented by the set of integers m1, m2, ...
{9,6,13,13,16,24,16,19,13,5}

Using our table above, we obtain ciphertext integers c1, c2, ...
{3,18,19,19,4,30,4,28,19,26}

Note that this example is no more secure than using a simple Caesar substitution cipher, but it serves to illustrate a simple example of the mechanics of RSA encryption.

Remember that calculating m^e mod n is easy, but calculating the inverse c^-e mod n is very difficult, well, for large n's anyway. However, if we can factor n into its prime factors p and q, the solution becomes easy again, even for large n's. Obviously, if we can get hold of the secret exponent d, the solution is easy, too.

Key Generation Algorithm

1. Generate two large random primes, p and q, of approximately equal size such that their product n = pq is of the required bit length, e.g. 1024 bits. [See note 1].
2. Compute n = pq and (φ) phi = (p-1)(q-1).
3. Choose an integer e, 1 < e < phi, such that gcd(e, phi) = 1. [See note 2].
4. Compute the secret exponent d, 1 < d < phi, such that ed ≡ 1 (mod phi). [See note 3].
5. The public key is (n, e) and the private key is (n, d). The values of p, q, and phi should also be kept secret.

- n is known as the *modulus*.
- e is known as the *public exponent* or *encryption exponent*.
- d is known as the *secret exponent* or *decryption exponent*.

Encryption

Sender A does the following:-

1. Obtains the recipient B's public key (n, e).
2. Represents the plaintext message as a positive integer m [see note 4].
3. Computes the ciphertext c = $m^{\wedge e}$ mod n.
4. Sends the ciphertext c to B.

Decryption

Recipient B does the following:-

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

```
package rsa;
import java.math.BigInteger;
import java.util.Random;
import java.util.Scanner;
public class Rsa {

        public static void main(String[] args) {
                BigInteger p,q,n,z,e,d;
                byte [] encrypted=new byte[1000];
                byte [] decrypted=new byte[1000];
                int range=128;
                  Random random=new Random();
                p=BigInteger.probablePrime(range,random);
                q=BigInteger.probablePrime(range,random);
                e=BigInteger.probablePrime(range, random);
                  n=p.multiply(q);
                  z=p.subtract(BigInteger.ONE).multiply(q.subtract(BigInteger.ONE));
                  while(z.gcd(e).compareTo(BigInteger.ONE)>0&& e.compareTo(z)<0)
                  {
                          e.add(BigInteger.ONE);

                  }
                  d=e.modInverse(z);
                  System.out.println("d:"+d+"\nn"+n);
                  Scanner in=new Scanner(System.in);
                  String text;
                  System.out.println("enter the text");
                  text=in.nextLine();
                  System.out.println("ASCII:"+BytestoString(text.getBytes ()));
                  encrypted=encrypt_decrypt(text.getBytes(), e,n,true);
                  decrypted=encrypt_decrypt(encrypted,d,n,false);
                  System.out.println("decrypted String:"+new String(decrypted));
        }
        public static String BytestoString(byte[] encrypted)
        {
                String test="  ";
                for(int i=0;i<encrypted.length;i++)
                {
                        test=test+encrypted[i]+"";
                }
                return test;

        }

public static byte[] encrypt_decrypt(byte[] message,BigInteger e,BigInteger n,boolean t)
                {
                        BigInteger c=new BigInteger(message).modPow(e,n);
                        if(t)
                        System.out.println("cipher text;"+c);
                        return c.toByteArray ();
                }
        }
```

OUTPUT:

```
enter the text
process
ASCII:  11211411199101115115 32
Ciphertext;12124208235791742960009344698453493753313078467788071314246843583843
4 6587727
decrypted String:process
```

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 (a) and with packets (b)).

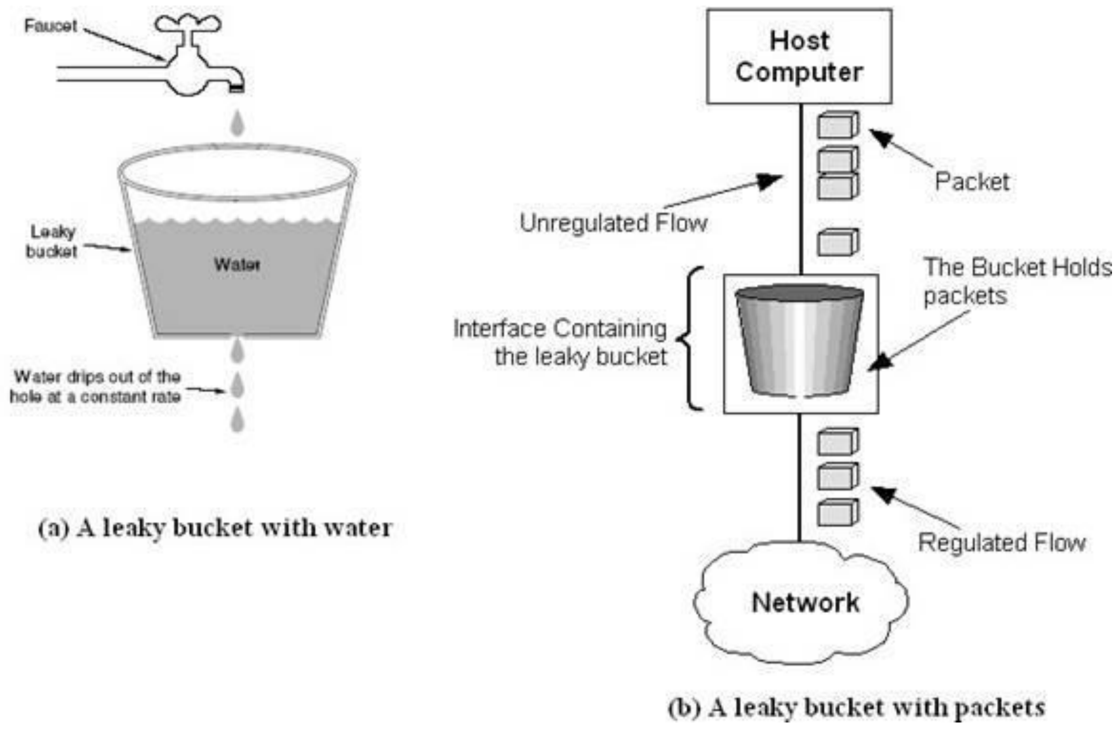

Faucet
Leaky bucket
Water
Water drips out of the hole at a constant rate
(a) A leaky bucket with water
Host Computer
Packet
Unregulated Flow
The Bucket Holds packets
Interface Containing the leaky bucket
Regulated Flow
Network
(b) A leaky bucket with packets

Figure 2.4 - The leaky bucket traffic shaping 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.

Implementation Algorithm:

```
Steps:
1. Read The Data For Packets
2. Read The Queue Size
3. Divide the Data into Packets
4. Assign the random Propagation delays for each packets to input
   into the bucket (input_packet).
5. wlile((Clock++<5*total_
       packets)and
       (out_packets<
       total_paclets))
     a. if (clock == input_packet)
          i. insert into Queue
     b. if (clock % 5 == 0 )
          i. Remove paclet from Queue
6. End
```

```
package leakybucket;
import java.util.Scanner;
public class Leakybuckett {

        public static void main(String[] args) {
Scanner input = new Scanner(System.in);

                System.out.println("enter the output rate");
                int oprate=input.nextInt();

                System.out.println("enter the bucket size");
                int bktsize=input.nextInt();



      for(int i=1;i<=5;i++)

                {
                System.out.println(" the packet number is   "+i)
```

```
                        System.out.println("enter the packet size of  "+i);
                        System.out.println();
                        int pktsize=input.nextInt();
                        System.out.println();

                if(pktsize>bktsize)
                                System.out.println(" bucket overflow");
                                 else
                    {
                            while(pktsize>oprate)
                              {
                               System.out.println(oprate+ "bytes outputted ");

                                   try {
                                         Thread.sleep(1000);
                                      } catch (InterruptedException ie)

                                         {
                                           //Handle exception
                                         }
                            System.out.println();
                            pktsize=pktsize-oprate;

                     }

                if(pktsize>0)
                     {
 System.out.println("last " +pktsize+ "  bytes outputted");
 System.out.println("bucket output sucessfull");
   System.out.println();
                      }

        }


}
}

}
```

OUTPUT:
CASE-1

```
enter the output rate
50
enter the bucket size
500
```

```
 the packet number is   1
enter the packet size of  1

250

50bytes outputted

50bytes outputted

50bytes outputted

50bytes outputted

last 50  bytes outputted
bucket output sucessfull

 the packet number is   2
enter the packet size of  2

510

 bucket overflow
 the packet number is   3
enter the packet size of  3

175

50bytes outputted

50bytes outputted

50bytes outputted

last 25  bytes outputted
bucket output sucessfull

 the packet number is   4
enter the packet size of  4


30

last 30  bytes outputted
bucket output sucessfull

 the packet number is   5
```

```
enter the packet size of  5


55

50bytes outputted

last 5  bytes outputted
bucket output successful
```

CASE- 2

```
enter the output rate
30
enter the bucket size
100
 the packet number is   1
enter the packet size of  1

50

30bytes outputted

last 20  bytes outputted
bucket output sucessfull

 the packet number is   2
enter the packet size of  2

30

last 30  bytes outputted
bucket output sucessfull

 the packet number is   3
enter the packet size of  3

50

30bytes outputted

last 20  bytes outputted
bucket output sucessfull

 the packet number is   4
enter the packet size of  4
```

```
60

30bytes outputted

last 30  bytes outputted
bucket output sucessfull

 the packet number is   5
enter the packet size of  5

60

30bytes outputted

last 30  bytes outputted
bucket output sucessfull
```

CASE- 3

```
enter the output rate
50
enter the bucket size
300
 the packet number is   1
enter the packet size of  1

100

50bytes outputted

last 50  bytes outputted
bucket output sucessfull

 the packet number is   2
enter the packet size of  2

150

50bytes outputted

50bytes outputted

last 50  bytes outputted
bucket output sucessfull
```

```
 the packet number is   3
enter the packet size of  3

350

 bucket overflow
 the packet number is   4
enter the packet size of  4

400

 bucket overflow
 the packet number is   5
enter the packet size of  5

180

50bytes outputted

50bytes outputted

50bytes outputted

last 30  bytes outputted
bucket output sucessfull
```