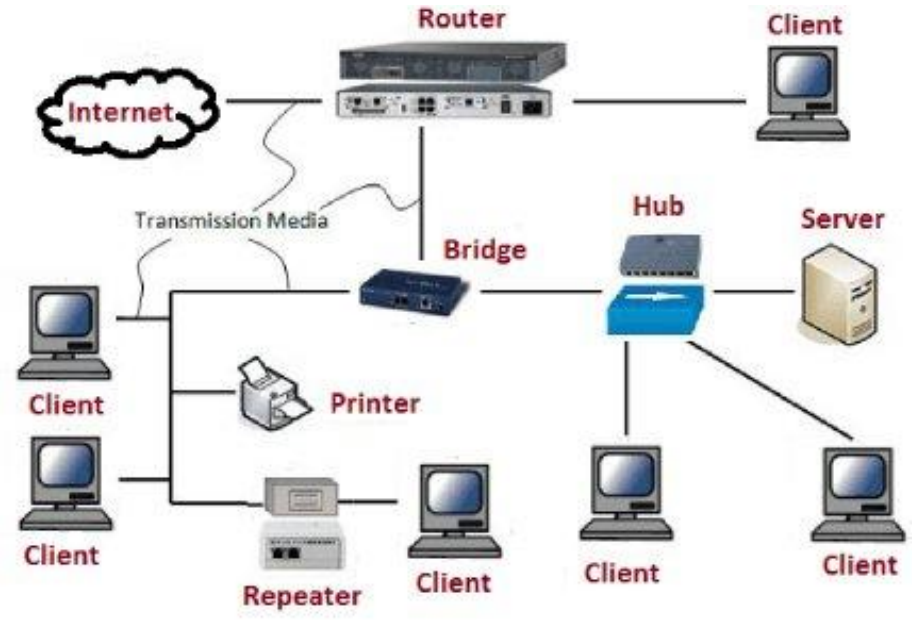
CAMBRIDGE INSTITUTE OF TECHNOLOGY

K.R. PURAM, BENGALURU-560036

DEPARTMENT OF INFORMATION SCIENCE AND ENGINEERING

COMPUTER NETWORK LABORATORY (18CSL57)



LAB MANUAL

(2021-2022)

Semester: V

Prepared by: Prof. Teena KB & Prof. Triveni N

COMPUTER NETWORK LABORATORY
(Effective from the academic year 2018 -2019)
SEMESTER – V

Course Code	**18CSL57**	**CIE Marks**	40
Number of Contact Hours/Week	0:2:2	**SEE Marks**	60
Total Number of Lab Contact Hours	36	**Exam Hours**	03

Credits – 2

Course Learning Objectives: This course (18CSL57) will enable students to:

- Demonstrate operation of network and its management commands
- Simulate and demonstrate the performance of GSM and CDMA
- Implement data link layer and transport layer protocols.

Descriptions (if any):

- For the experiments below modify the topology and parameters set for the experiment and take multiple rounds of reading and analyze the results available in log files. Plot necessary graphs and conclude. Use NS2/NS3.
- **Installation procedure of the required software must be demonstrated, carried out in groups and documented in the journal.**

Programs List:

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

Laboratory Outcomes: The student should be able to:

- Analyze and Compare various networking protocols.
- Demonstrate the working of different concepts of networking.
- Implement, analyze and evaluate networking protocols in NS2 / NS3 and JAVA programming language

Conduct of Practical Examination:

- Experiment distribution
    - For laboratories having only one part: Students are allowed to pick one experiment from the lot with equal opportunity.

- For laboratories having PART A and PART B: Students are allowed to pick one experiment from PART A and one experiment from PART B, with equal opportunity.

- Change of experiment is allowed only once and marks allotted for procedure to be made zero of the changed part only.
- Marks Distribution *(Courseed to change in accoradance with university regulations)*
  i) For laboratories having only one part – Procedure + Execution + Viva-Voce: 15+70+15 = 100 Marks
  j) For laboratories having PART A and PART B
    i. Part A – Procedure + Execution + Viva = 6 + 28 + 6 = 40 Marks
    ii. Part B – Procedure + Execution + Viva = 9 + 42 + 9 = 60 Marks

Course Learning Objectives

This course (18CSL57) will enable students to:

- **Demonstrate operation of network and its management commands**
- **Simulate and demonstrate the performance of GSM and CDMA**
- **Implement data link layer and transport layer protocols**

Course Outcomes

The student should be able to:

CO1: Analyze and Compare various networking protocols.

CO2: Demonstrate the working of different concepts of networking.

CO3: Implement, analyze and evaluate networking protocols in NS2 / NS3 and JAVA programming language

Index

Introduction to NS-2

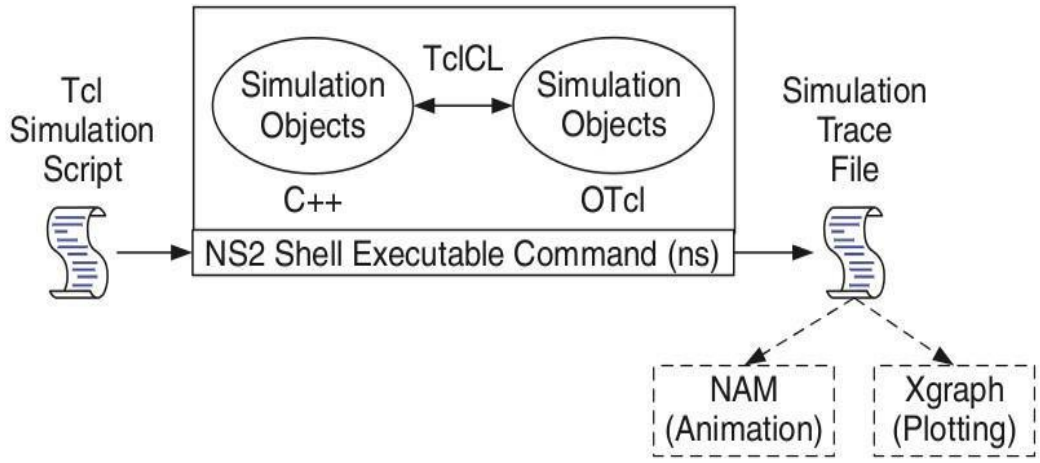
• Widely known as NS2, is simply an event driven simulation tool.

• 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 (Tool command language) scripting

- TCL[Tool command language] is also an scripting programming language
- NS2 also uses OTCL[Object oriented extension of TCL]
- OTCL is also written in C++ [OTCL=OOPS concepts+ TCL]
- OTCL-configures the system while C++ is also used to implement the code that is frequently executed
- Tcl is a general purpose scripting language.
- 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

```
set b $a
```

•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]

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.

The 1st line open the file "out.tr" to be used for writing, declared with the letter "w".

The 2nd line uses a simulator method called trace-all that have as parameter the name of the file where the traces will go.

The 3rd line open the file "out.nam" to be used for writing, declared with the letter "w".

The 4th 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 "**$namfile**" and "**$tracefile1**".

The termination of the program is done using a "finish" procedure.

#Define a 'finish' procedure

```
Proc finish { } {

global ns tracefile1 namfile

$ns flush trace

Close $tracefile1
Close $namfile

Exec nam out.nam &
Exit 0
```

The word proc declares a procedure in this case called finish and without arguments.
The word global is used to tell that we are using variables declared outside the procedure.
The simulator method "flush-trace" will dump the traces on the respective files.
The tcl command "close" closes the trace files defined before and exec executes the nam program for visualization.
The command exit will ends the application and return the number 0 as status to the system.
Zero is the default for a clean exit.
Other values can be used to say that is a exit because something fails.

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

$ns at 125.0 "finish"

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 single 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:

The command

set tcp [new Agent/TCP]

$ns attach-agent $n0 $tcp

Defines the source node of the tcp

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 tcl 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:

Options are listed here

/-bd <color> (Border)

Xgraph [options] file-name

This specifies the border color of the xgraph window.

/-bg <color> (Background)

This specifies the background color of the xgraph window.

/-fg<color> (Foreground)

This specifies the foreground color of the xgraph window.

/-lf <fontname> (LabelFont)

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

/-t<string> (Title Text)

This string is centered at the top of the graph.

/-x <unit name> (XunitText)

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

/-y <unit name> (YunitText)

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

Awk- An Advanced

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

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

Syntax: awk option ‘selection_criteria {action’ file(s)

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

Example: $ awk ‘/manager/ {print}’ emp.lst

Variables

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

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.

NS-2 installation steps in Linux

➢ 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

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.

- **TCL file**

```
set ns [new Simulator]
set nf [open lab1.nam w]
$ns namtrace-all $nf
set tf [open lab1.tr w]
$ns trace-all $tf

proc finish { } {
global ns nf tf
$ns flush-trace
close $nf
close $tf
exec nam lab1.nam &
exit 0
}

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

$ns duplex-link $n0 $n2 200Mb 10ms DropTail
$ns duplex-link $n1 $n2 100Mb 5ms DropTail
```

```
$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]

$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 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]

$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
```

- **Awk file**

```
BEGIN{
c=0;
}
{
if($1=="d")
{
c++;
printf(" %s\t %s\t%s\t%s\t%s\n",$2,$3,$4,$5,$11);
}
}
END{
printf("the number of packets dropped=%d\n",c);
}
```

Steps for execution

1) Open vi editor and type program. Program name should have the extension ― .tcl

[root@localhost ~]# vi lab1.tcl

or

[use gedit:

[root@localhost ~]# gedit lab1.tcl

Type the code and save]

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

Here ―ns indicates network simulator. We will get the required topology in the simulator

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 ,

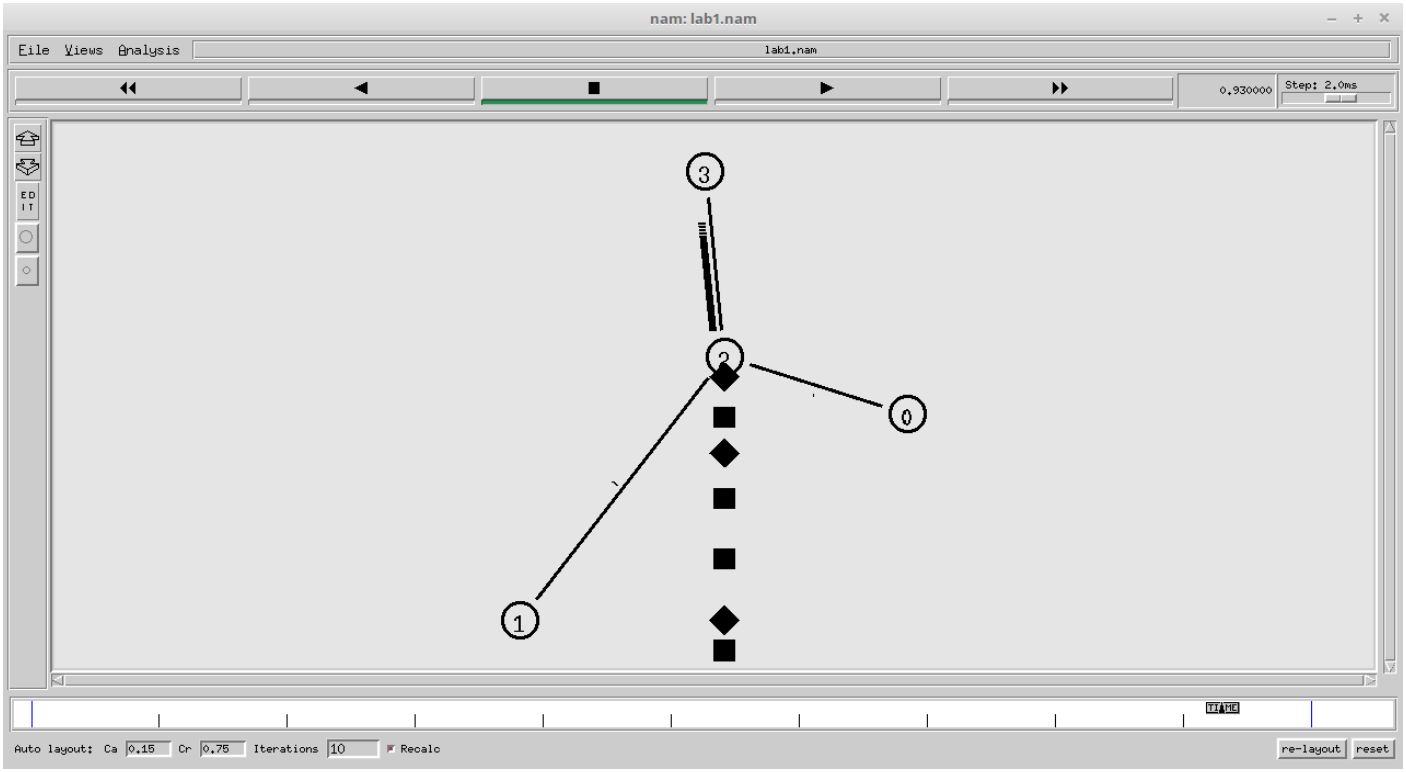
[root@localhost~]# awk –f lab1.awk lab1.tr

7) To see the trace file contents open the file as ,

[root@localhost~]# vi lab1.tr

Trace file contains 12 columns:-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

OUTPUT :




```
cseise@cseise ~
File  Edit  View  Search  Terminal  Help

cseise@cseise ~ $ awk -f lab1.awk lab1.tr
 0.726267         2      3       cbr     139
 0.74002          2      3       cbr     126
 0.74502          2      3       cbr     127
 0.76002          2      3       cbr     130
 0.771267         2      3       cbr     151
 0.782517         2      3       cbr     154
 0.79002          2      3       cbr     136
 0.801267         2      3       cbr     159
 0.81502          2      3       cbr     141
 0.82002          2      3       cbr     142
 0.83502          2      3       cbr     145
 0.846267         2      3       cbr     171
 0.857517         2      3       cbr     174
 0.86502          2      3       cbr     151
 0.88002          2      3       cbr     154
 0.89502          2      3       cbr     157
 0.906267         2      3       cbr     187
 0.91502          2      3       cbr     161
 0.92502          2      3       cbr     163
 0.936267         2      3       cbr     195
 0.958767         2      3       cbr     201
 0.97502          2      3       cbr     173
 0.98502          2      3       cbr     175
 0.988767         2      3       cbr     209
the number of packets dropped=24
```

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.

- **TCL file**

```
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 p0 [new Agent/Ping]
$ns attach-agent $n0 $p0
$p0 set packetSize_  50000
$p0 set interval_  0.0001

set p1 [new Agent/Ping]
$ns attach-agent $n1 $p1

set p2 [new Agent/Ping]
$ns attach-agent $n2 $p2
$p2 set packetSize_  30000
$p2 set interval_  0.00001

set p3 [new Agent/Ping]
$ns attach-agent $n3 $p3

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"
}

$ns connect $p0 $p5
$ns connect $p2 $p3

proc finish { } {
global ns nf tf
$ns flush-trace
close $nf
close $tf
exec nam lab2.nam &
exit 0
}

$ns at 0.1 "$p0 send"
$ns at 0.2 "$p0 send"
$ns at 0.3 "$p0 send"
$ns at 0.4 "$p0 send"
$ns at 0.5 "$p0 send"
$ns at 0.6 "$p0 send"
$ns at 0.7 "$p0 send"
$ns at 0.8 "$p0 send"
$ns at 0.9 "$p0 send"
$ns at 1.0 "$p0 send"
$ns at 1.1 "$p0 send"
$ns at 1.2 "$p0 send"
$ns at 1.3 "$p0 send"
$ns at 1.4 "$p0 send"
$ns at 1.5 "$p0 send"
$ns at 1.6 "$p0 send"
$ns at 1.7 "$p0 send"
$ns at 1.8 "$p0 send"
$ns at 1.9 "$p0 send"
$ns at 2.0 "$p0 send"
$ns at 2.1 "$p0 send"
$ns at 2.2 "$p0 send"
$ns at 2.3 "$p0 send"
$ns at 2.4 "$p0 send"
$ns at 2.5 "$p0 send"
$ns at 2.6 "$p0 send"
$ns at 2.7 "$p0 send"
$ns at 2.8 "$p0 send"
$ns at 2.9 "$p0 send"

$ns at 0.1 "$p2 send"
$ns at 0.2 "$p2 send"
```

```
$ns at 0.3 "$p2 send"
$ns at 0.4 "$p2 send"
$ns at 0.5 "$p2 send"
$ns at 0.6 "$p2 send"
$ns at 0.7 "$p2 send"
$ns at 0.8 "$p2 send"
$ns at 0.9 "$p2 send"
$ns at 1.0 "$p2 send"
$ns at 1.1 "$p2 send"
$ns at 1.2 "$p2 send"
$ns at 1.3 "$p2 send"
$ns at 1.4 "$p2 send"
$ns at 1.5 "$p2 send"
$ns at 1.6 "$p2 send"
$ns at 1.7 "$p2 send"
$ns at 1.8 "$p2 send"
$ns at 1.9 "$p2 send"
$ns at 2.0 "$p2 send"
$ns at 2.1 "$p2 send"
$ns at 2.2 "$p2 send"
$ns at 2.3 "$p2 send"
$ns at 2.4 "$p2 send"
$ns at 2.5 "$p2 send"
$ns at 2.6 "$p2 send"
$ns at 2.7 "$p2 send"
$ns at 2.8 "$p2 send"
$ns at 2.9 "$p2 send"

$ns at 3.0 "finish"
$ns run
```

- **Awk file**

```
BEGIN{

drop=0;

}

{

if($1=="d")

{

drop++;

}

}
```

```
END{
printf("total no of %s type packets dropped=%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

or

[use gedit:

[root@localhost ~]# gedit lab2.tcl

Type the code and save]

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

Here ―ns indicates network simulator. We will get the required topology in the simulator

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 ,

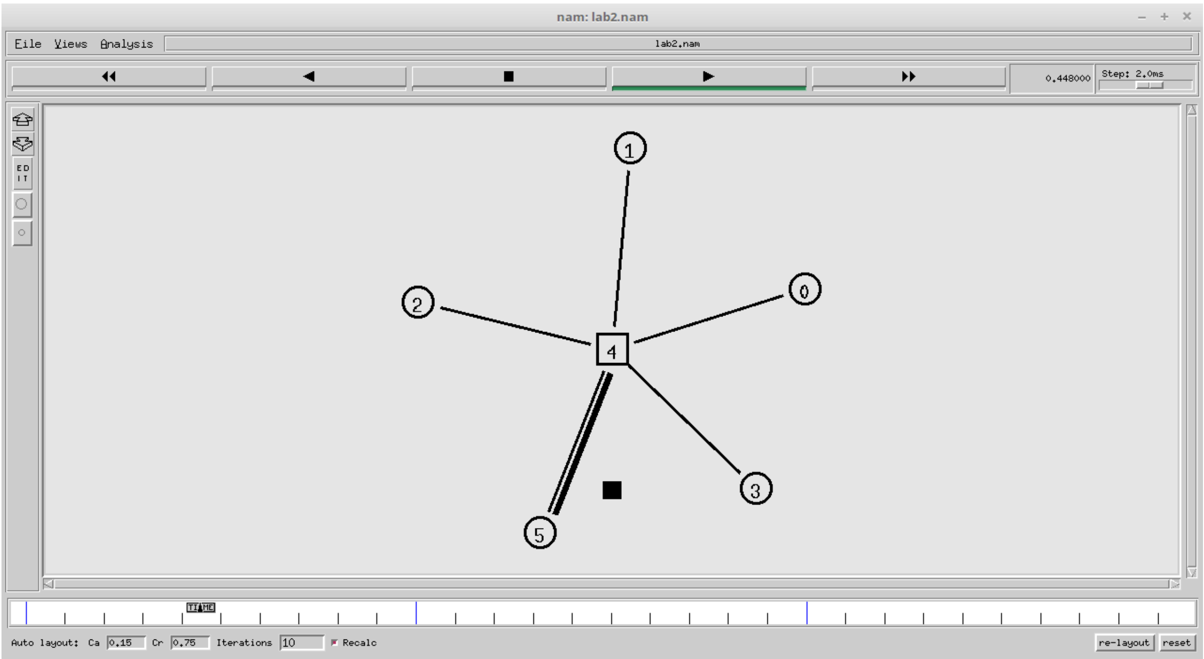
[root@localhost~]# awk –f lab2.awk lab2.tr

7) To see the trace file contents open the file as ,

[root@localhost~]# vi lab2.tr

Trace file contains 12 columns:-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

OUTPUT:




```
cseise@cseise ~ $ 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
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
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
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
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
cseise@cseise ~ $ awk -f lab2.awk lab2.tr
total no of ping type packets dropped=20
cseise@cseise ~ $
```

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

Note:
$ns make-lan nodelist bw delay LL ifq MAC channel phy

Creates a lan from a set of nodes given by <nodelist>. Bandwidth, delay characteristics along with the link-layer, Interface queue, Mac layer and channel type for the lan also needs to be defined.

Default values used are as follows:
<LL> .. LL
<ifq>.. Queue/DropTail
<MAC>.. Mac
<channel>.. Channel and
<phy>.. Phy/WiredPhy

- **TCL file**

```
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"
$ns at 14 "$ftp0 stop"

$ns at 0.2 "$ftp2 start"
$ns at 8 "$ftp2 stop"
$ns at 10 "$ftp2 start"
$ns at 15 "$ftp2 stop"
$ns at 16 "finish"
$ns run
```

- **Awk script**

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

END{

}

Steps for execution

1) Open vi editor and type program. Program name should have the extension — .tcl

[root@localhost ~]# vi lab3.tcl

or

[use gedit:

[root@localhost ~]# gedit lab3.tcl

Type the code and save]

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

Here —ns indicates network simulator. We will get the required topology in the simulator

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 lab3.awk file1.tr>a1

[root@localhost~]# awk –f lab3.awk file2.tr>a2

[root@localhost~]# xgraph a1 a2

7) To see the trace file contents open the file as ,

[root@localhost~]# vi lab3.tr

Trace file contains 12 columns:-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

8. To see the tcp trace of cwnd (congestion window)

[root@localhost~]# vi file1.tr

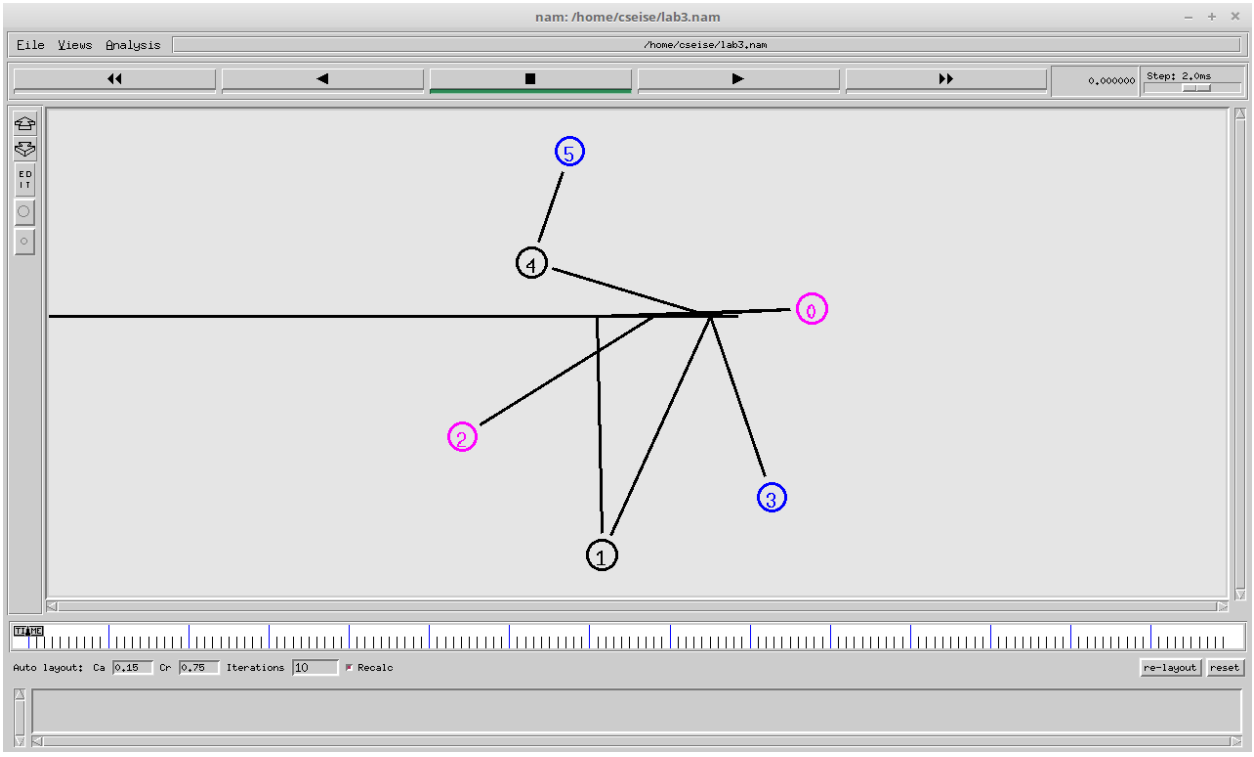
Or

[root@localhost~]# vi file2.tr

The format of these lines is

1. time
2. source node of the flow
3. source port (as above, an abstract connection endpoint, not a simulated TCP port)
4. destination node of the flow
5. destination port
6. name of the traced variable
7. value of the traced variable

OUTPUT




xgraph
Close
Hdcpy
About
Anim
Replot
Deriv
X Graph
a1
a2
32.0000
30.0000
28.0000
26.0000
24.0000
22.0000
20.0000
18.0000
16.0000
14.0000
12.0000
10.0000
8.0000
6.0000
4.0000
2.0000
0.0000
0.0000
1.0000
2.0000
3.0000
4.0000
5.0000
6.0000
7.0000
8.0000
9.0000
10.0000
11.0000
12.0000
13.0000
14.0000
15.0000
X

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.

An extended service set (ESS) is one or more interconnected basic service sets (BSSs) and their associated LANs. Each BSS consists of a single access point (AP) together with all wireless client devices (stations, also called STAs) creating a local or enterprise 802.11 wireless LAN (WLAN). 802.11 and *802.11x* refers to a family of specifications developed by the IEEE for **w***ireless* **LAN** (WLAN) technology. 802.11 specifies an over-the-air interface between a wireless client and a base station or between two wireless clients.

The topography for mobilenodes needs to be defined. It should be defined before creating mobilenodes. Normally flat topology is created by specifying the length and width of the topography using the following primitive:

set topo [new Topography]

$topo load_flatgrid $opt(x) $opt(y) optional:res

where opt(x) and opt(y) are the boundaries used in simulation.

This initializes the grid for the topography object. <X> and <Y> are the x-y co-ordinates for the topology and are used for sizing the grid. The grid resolution may be passed as <res>. A default value of 1 is normally used.

$ns_ node-config -addressingType \<usually flat or hierarchical used for wireless topologies\>

-adhocRouting \<adhoc rotuing protocol like PUMA, DSR, TORA, AODV, DSDV etc\>

-llType \<LinkLayer\>

-macType \<MAC type like Mac/802_11\>

-propType \<Propagation model like Propagation/TwoRayGround\>

-ifqType \<interface queue type like Queue/DropTail/PriQueue\>

-ifqLen \<interface queue length like 50\>

-phyType \<network inteface type like Phy/WirelessPhy\>

-antType \<antenna type like Antenna/OmniAntenna\>

-channelType \<Channel type like Channel/WirelessChannel\>

-topoInstance \<the topography instance\>

-wiredRouting \<turning wired routing ON or OFF\>

-mobileIP \<setting the flag for mobileIP ON or OFF\>

-energyModel \<EnergyModel type\>

-initialEnergy \<specified in Joules\>

-rxPower \<specified in W\>

-txPower \<specified in W\>

-agentTrace \<tracing at agent level turned ON or OFF\>

-routerTrace \<tracing at router level turned ON or OFF\>

-macTrace \<tracing at mac level turned ON or OFF\>

-movementTrace \<mobilenode movement logging turned ON or OFF\>

This command is used typically to configure for a mobilenode.

create-god num_nodes

This command is used to create a God(general operation director) instance. The number of mobile nodes is passed as argument which is used by God to create a matrix to store connectivity information of the topology.

Creating Node movements

The mobilenode is designed to move in a three dimensional topology. However the third dimension (Z) is not used. That is the mobilenode is assumed to move always on a flat terrain with Z always equal to 0. Thus the mobilenode has X, Y, Z(=0) co-ordinates that is continually adjusted as the node moves. There are two mechanisms to induce movement in mobilenodes. In the first method, starting position of the node and its future destinations may be set explicitly. These directives are normally included in a separate movement scenario file.

The start-position and future destinations for a mobilenode may be set by using the following APIs:

$node set X_ \<x1\>

$node set Y_ \<y1\>

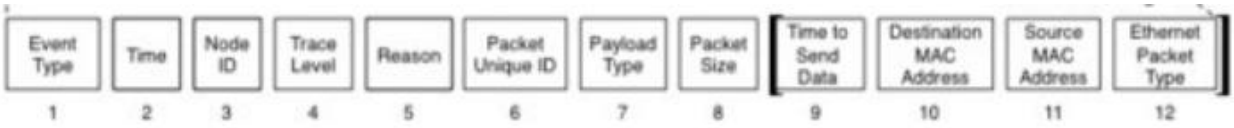
$node set Z_ \<z1\>

$ns at $time $node setdest \<x2\> \<y2\> \<speed\>

At $time sec, the node would start moving from its initial position of (x1,y1) towards a destination (x2,y2) at the defined speed.

In this method the node-movement-updates are triggered whenever the position of the node at a given time is required to be known.

NS2 Wireless Trace Format



- **TCL file**

```
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 lab4.nam &
close $tf
exit 0
}
```

```
$ns at 250 "finish"
$ns run
```

- **Awk script**

```
BEGIN{
count1=0
count2=0
pack1=0
pack2=0
time1=0
time2=0
}

{
if($1=="r" && $3=="_1_"&& $4=="AGT")
{
count1++
pack1=pack1+$8 t
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

or

[use gedit:

[root@localhost ~]# gedit lab4.tcl

Type the code and save]

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

Here ―ns indicates network simulator. We will get the required topology in the simulator

ii) Now press the play button in the simulation window and the simulation will

begins.

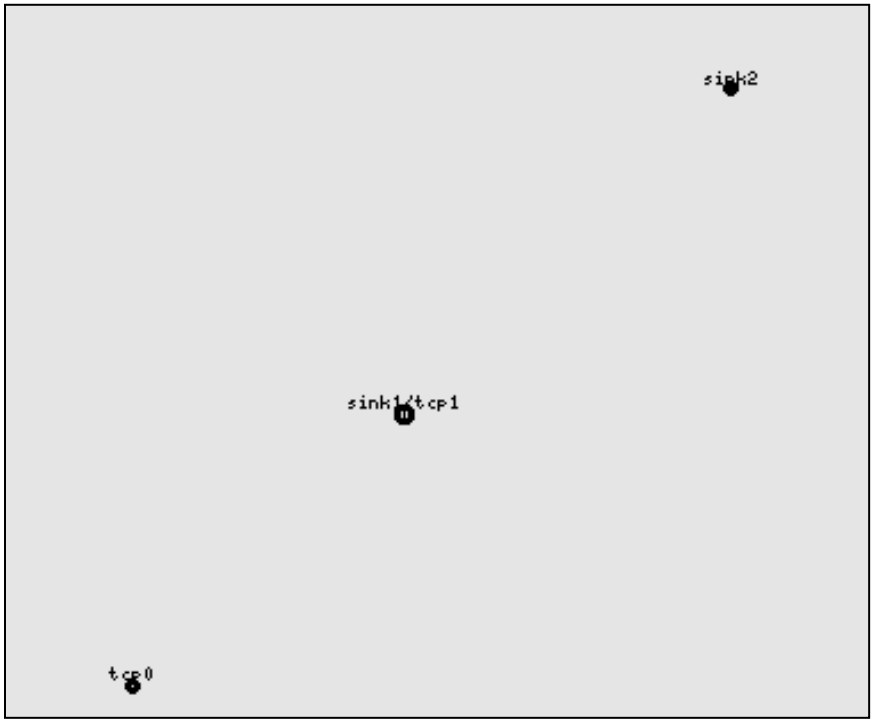
6) After simulation is completed run awk file to see the output ,

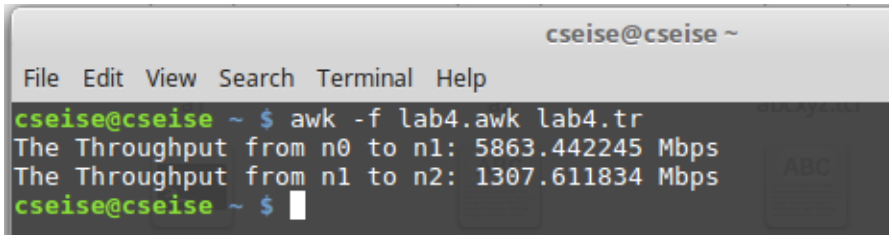
[root@localhost~]# awk –f lab4.awk lab4.tr

7) To see the trace file contents open the file as ,

[root@localhost~]# vi lab4.tr

OUTPUT

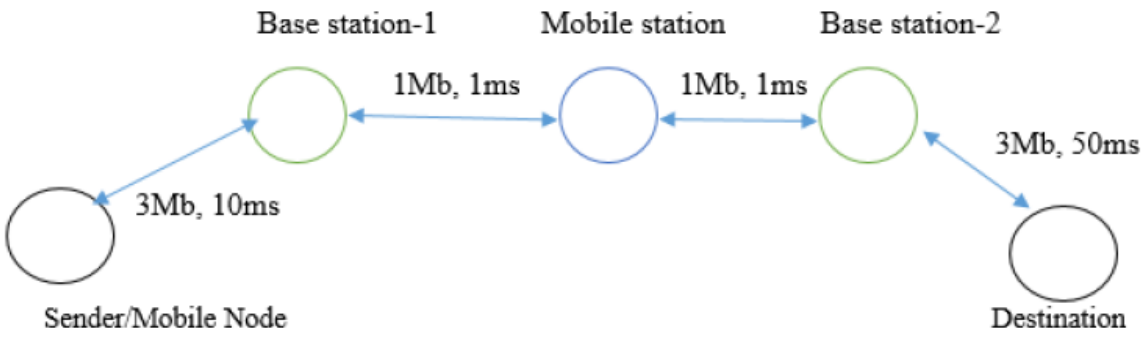

sink2
sink1/tcp1
tcp0


cseise@cseise ~
File Edit View Search Terminal Help
cseise@cseise ~ $ awk -f lab4.awk lab4.tr
The Throughput from n0 to n1: 5863.442245 Mbps
The Throughput from n1 to n2: 1307.611834 Mbps
cseise@cseise ~ $

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)

Design:



Important note:

Change the directory (as shown in below figure) before typing the code :

- **TCL file**

```
set opt(title) zero;
set opt(stop) 100;
# Stop time
set opt(ecn) 0 ;
# Topology
set opt(type) gsm ; #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(gsmbuf) 10 ; # buffer size for gsm

#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
```

```
#default buffer size in packets
set buf(gsm) 10
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 bandwidth $nodes(bs2) $nodes(ms) $bwDL($t) simplex
$ns bandwidth $nodes(ms) $nodes(bs2) $bwUL($t) simplex
$ns delay $nodes(bs1) $nodes(ms) $propDL($t) simplex
$ns delay $nodes(ms) $nodes(bs1) $propDL($t) simplex
$ns delay $nodes(bs2) $nodes(ms) $propDL($t) simplex
$ns delay $nodes(ms) $nodes(bs2) $propDL($t) simplex
$ns queue-limit $nodes(bs1) $nodes(ms) $buf($t)
$ns queue-limit $nodes(ms) $nodes(bs1) $buf($t)
$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
```

Steps for execution

1) Open vi editor and change directory:

[root@localhost ~]#cd /home/cseise/Downloads/ns-allinone-2.35/ns-2.35/tcl/ex/wireless-scripts

2) Open vi editor and type program. Program name should have the extension — .tcl

[root@localhost ~]# vi lab5.tcl

or

[use gedit:

[root@localhost ~]# gedit lab5.tcl

Type the code and save]

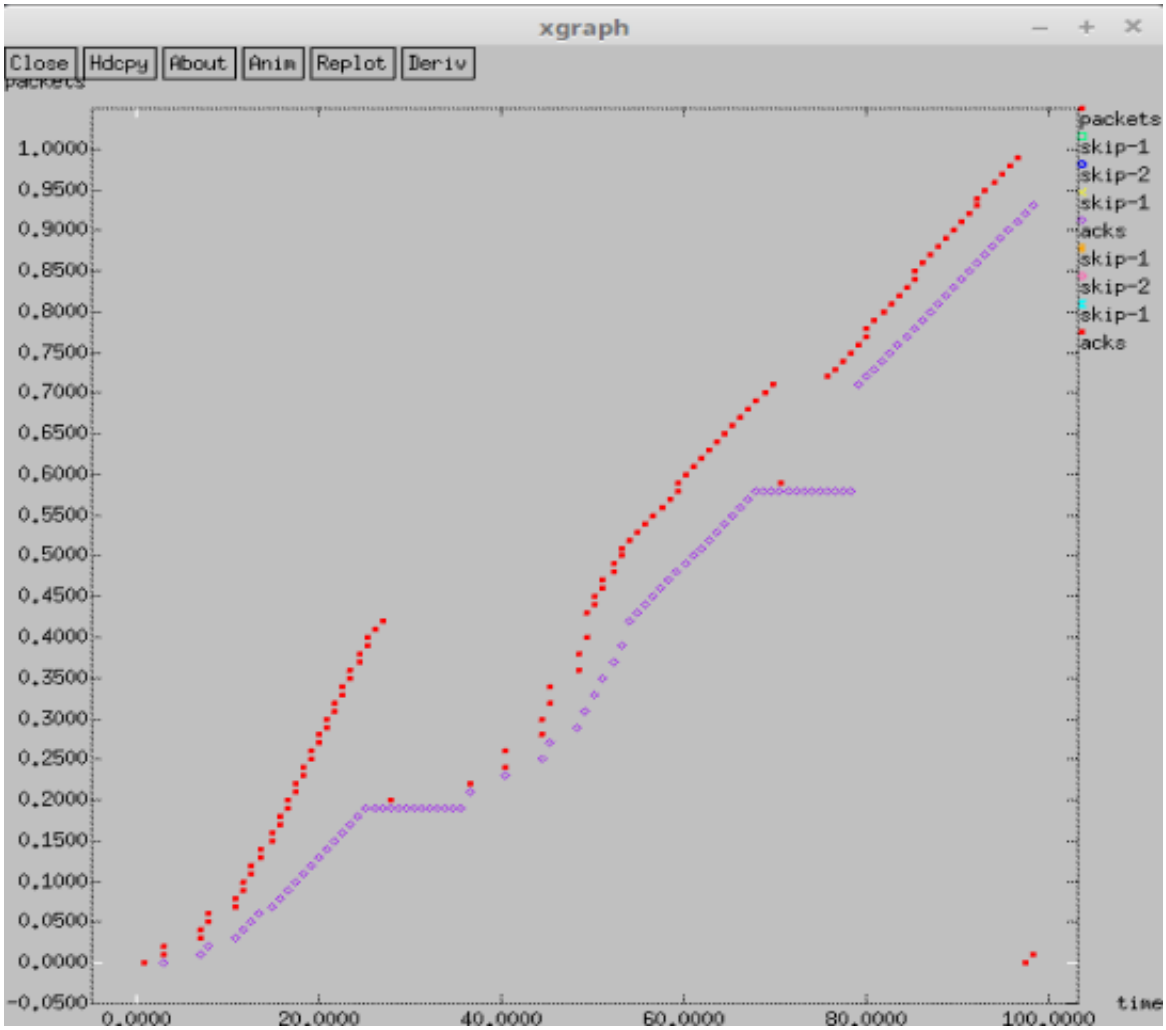
3) Save the program by pressing —ESC key first, followed by —Shift and keys

simultaneously and type —wq and press Enter key.

4) Run the simulation program

[root@localhost~]# ns lab5.tcl

OUTPUT

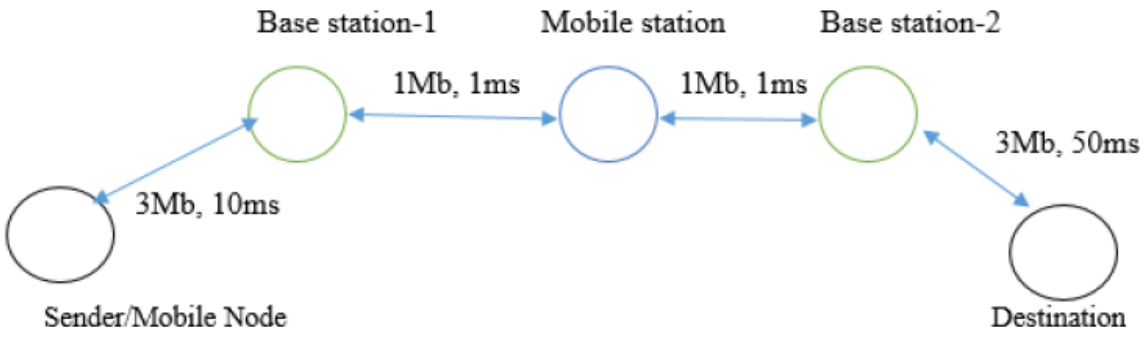

xgraph
Close
Hdcpy
About
Anim
Replot
Deriv
packets
1.0000
0.9500
0.9000
0.8500
0.8000
0.7500
0.7000
0.6500
0.6000
0.5500
0.5000
0.4500
0.4000
0.3500
0.3000
0.2500
0.2000
0.1500
0.1000
0.0500
0.0000
-0.0500
0.0000
20.0000
40.0000
60.0000
80.0000
100.0000
time
packets
skip-1
skip-2
skip-1
acks
skip-1
skip-2
skip-1
acks

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)

Design:



Important note:

Change the directory (as shown in below figure) before typing the code :

- **TCL file**

```
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 gsm


#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 bandwidth $nodes(bs2) $nodes(ms) $bwDL($t) simplex
$ns bandwidth $nodes(ms) $nodes(bs2) $bwUL($t) simplex
$ns delay $nodes(bs1) $nodes(ms) $propDL($t) simplex
$ns delay $nodes(ms) $nodes(bs1) $propDL($t) simplex
$ns delay $nodes(bs2) $nodes(ms) $propDL($t) simplex
$ns delay $nodes(ms) $nodes(bs2) $propDL($t) simplex
$ns queue-limit $nodes(bs1) $nodes(ms) $buf($t)
$ns queue-limit $nodes(ms) $nodes(bs1) $buf($t)
$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
```

Steps for execution

1) Open vi editor and change directory:

[root@localhost ~]#cd /home/cseise/Downloads/ns-allinone-2.35/ns-2.35/tcl/ex/wireless-scripts

2) Open vi editor and type program. Program name should have the extension — .tcl

[root@localhost ~]# vi lab6.tcl

or

[use gedit:

[root@localhost ~]# gedit lab6.tcl

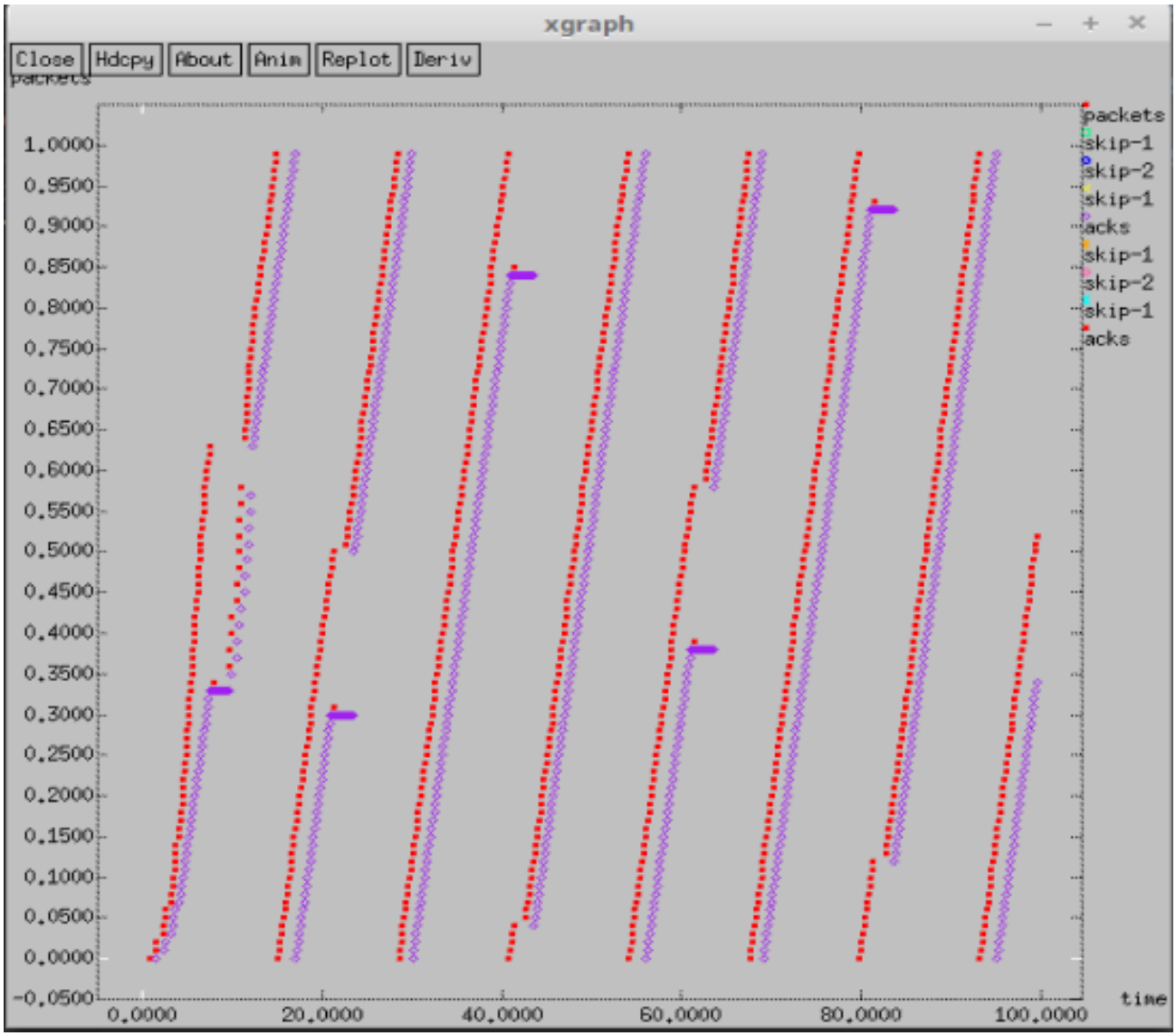
Type the code and save]

3) Save the program by pressing —ESC key first, followed by —Shift and keys simultaneously and type —wq and press Enter key.

4) Run the simulation program

[root@localhost~]# ns lab6.tcl

OUTPUT:


xgraph
Close
Hdcopy
About
Anim
Replot
Deriv
packets
1.0000
0.9500
0.9000
0.8500
0.8000
0.7500
0.7000
0.6500
0.6000
0.5500
0.5000
0.4500
0.4000
0.3500
0.3000
0.2500
0.2000
0.1500
0.1000
0.0500
0.0000
-0.0500
0.0000
20.0000
40.0000
60.0000
80.0000
100.0000
time
packets
skip-1
skip-2
skip-1
acks
skip-1
skip-2
skip-1
acks

PART-B

Java is a general-purpose computer programming language that is simple, concurrent, class-based, object-oriented language. The compiled Java code can run on all platforms that support Java without the need for recompilation hence Java is called as "write once, run anywhere" (WORA).The Java compiled intermediate output called “byte-code” that can run on any Java virtual machine (JVM) regardless of computer architecture. The language derives much of its syntax from C and C++, but it has fewer low-level facilities than either of them.

In Linux operating system Java libraries are preinstalled. It’s very easy and convenient to compile and run Java programs in Linux environment. To compile and run Java Program is a two-step process:

1. Compile Java Program from Command Prompt

[root@host ~]# javac Filename.java

The Java compiler (Javac) compiles java program and generates a byte-code with the same file name and .class extension.

2. Run Java program from Command Prompt

[root@host ~]# java Filename

The java interpreter (Java) runs the byte-code and gives the respective output. It is important to note that in above command we have omitted the .class suffix of the byte- code (Filename.class).

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 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.

	CRC-CCITT	CRC-16	CRC-32
Checksum Width	16 bits	16 bits	32 bits
Generator Polynomial	10001000000100001	11000000000000101	10000010011000001000111011011011

International Standard CRC Polynomials

Algorithm

- Read datastream into a variable : "data"
- Read generator polymanial into a variable: "GP"
- Create an integer array of size [data.length+(GP.length-1)] : "dividend"

  (dividend will be initialised with zeroes, thus no need to append any zeroes)
- Convert every character in "data" into integer and store in the array "dividend"

```
For (i=0 to i<data.length)
        dividend[i]= integer ( data[i] )
```

- Create an integer array of size GP.length : "divisor"
- Convert every character in "GP" into integer and store in the array "divisor"

```
For (i=0 to i<GP.length)
```

divisor[i]= integer (GP[i])

- to calculate CRC

```
For( i=0 to i<data.length)
    If( dividend[i] is equal to 1)
        Then
            For( j=0 to j<GP. Length)
                dividend[i+j]= dividend[i+j] xor divisor[j]
```

- In the dividend array only last (GP.length-1) elements will be CRC and initial 0 to data.length will be modified hence reload the array with data for displaying purpose.

```
For (i=0 to i<data.length)

    dividend[i]= integer( data[i] )
```

- To display CRC: for(i=0 to i<dividend. Length)
  print dividend[i]

//to check CRC

- Read datastream along with CRC into a variable : "data"
- Create an integer array of size [data.length] : "dividend"
- Convert every character in "data" into integer and store in the array "dividend"

```
For (i=0 to i<data.length)
    dividend[i]= integer ( data[i] )
```

- Divide dividend by divisor

```
For( i=0 to i<data.length)
    If( dividend[i] is equal to 1)
        Then
            For( j=0 to j<GP. Length)
                dividend[i+j]= dividend[i+j] xor divisor[j]
```

- Declare a Boolean variable to check if remainder is zero(initialize it as True) : Valid = TRUE
- Check if all elements of dividend array are equal to zero, if any element is zero make valid=False,

```
For(i=0 to i<dividend.length)
    If( dividend[i] is equal to 1)
        Then valid= false
```

- If valid= true

```
    Then print data is valid
Else print data is invalid
```

SOURCE CODE:

```
import java.util.Scanner;
class CRC{
public static void main(String args[]){
Scanner sc = new Scanner(System.in);
//Input Data Stream
System.out.print("Enter data stream: ");
String datastream = sc.nextLine();
System.out.print("Enter generator: ");
String generator = sc.nextLine();
int data[] = new int[datastream.length() + generator.length()-1];
int divisor[] = new int[generator.length()];
for(int i=0;i<datastream.length();i++)
data[i] = Integer.parseInt(datastream.charAt(i)+"");
for(int i=0;i<generator.length();i++)
divisor[i] = Integer.parseInt(generator.charAt(i)+"");
//Calculation of CRC
for(int i=0;i<datastream.length();i++){
if(data[i]==1)
for(int j=0;j<divisor.length;j++)
data[i+j] ^= divisor[j];
}
//Display CRC
System.out.print("The CRC code is: ");
for(int i=0;i<datastream.length();i++)
data[i] = Integer.parseInt(datastream.charAt(i)+"");
for(int i=0;i<data.length;i++)
System.out.print(data[i]);
System.out.println();

//Check for input CRC code
System.out.print("Enter CRC code: ");
datastream = sc.nextLine();
```

```
data = new int[datastream.length() + generator.length()-1];
for(int i=0;i<datastream.length();i++)
data[i] = Integer.parseInt(datastream.charAt(i)+"");

//Calculation of remainder
for(int i=0;i<datastream.length();i++){
if(data[i]==1)
for(int j=0;j<divisor.length;j++)
data[i+j] ^= divisor[j];
}
//Display validity of data
boolean valid = true;
for(int i=0;i<data.length;i++)
if(data[i]==1){
valid = false;
break;
}
if(valid==true) System.out.println("Data stream is valid");
else System.out.println("Data stream is invalid. CRC error occured.");
}
}
```

OUTPUT:

```
Enter data stream: 1001
Enter generator: 1011
The CRC code is: 1001110
Enter CRC code: 1001111
Data stream is invalid. CRC error occured.

Enter data stream: 1001
Enter generator: 1011
The CRC code is: 1001110
Enter CRC code: 1001110
Data stream is valid
```

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.

Algorithm

Create a Class named BellmanFord

Start of class:

Declare attribute to hold number of vertices: num_ver (private int)

Declare attribute to hold Distance table: D (private int array)

Declare a attribute to hold value to indicate “direct link doesnot exists” : MAX

(Use value 999 and Type as public static final int)

Create a constructor with N_V as parameter

Start :

Assign N_V value to num_ver attribute

Allocate size to Distance table

D= size of(N_V+1)

End of constructor

Create a function named Bellmanfordevaluation with parameters : source , adjacency matrix

Start of function

Initialize all the elements of D to MAX

For (i=1 to i<=num_ver)

D[i]=MAX

Make distance to source to zero

D[source]=0

//To calculate distance

//For source node we have distance as zero so we need to iter (num_ver -1) time to calculate the distance. 1st iter calculates a shortest distance at 1 link far, 2nd at 2 links far and on

In each iter for every node to every other node say nodes s and d check

if value in adjacency matrix A[s,d] is not equal MAX, then check

if distance to node d from source is **greater** than distance to node d to source through node s, then assign the new shortest distance else no need to change

Thus , for(i=1 to i<=(num_ver-1))

For(s=1 to s<=num_ver)

For(d=1 to d<= num_ver)

If D[d]> D[s]+A[s][d]

Then D[d] = D[s]+A[s][d]

//now to find out negative edge cycle

Every node to every other node say nodes s and d check

if value in adjacency matrix A[s,d] is not equal MAX, then check

if distance to node d from source is **greater** than distance to node d to source through node s, then print “negative edge cycle detected”

Thus For(s=1 to s<=num_ver)

For(d=1 to d<= num_ver)

If D[d]> D[s]+A[s][d]

Then print “negative edge cycle detected”

//now printing distance table D

For(i=1 to i<=num_ver)

Print : “Distance from” Source “ to” i “is” D[i]

End of function

Main program

Start of main program:

Declare a variable to hold number of vertices: N_V

Declare a variable to hold source vertex: Source

Read number of vertices into N_V (use nextInt( ))

Create a two dimensional integer array to hold the adjacency matrix of size (N_V+1) : A[ ][ ]

Read the adjacency matrix

For (s=1 to s<=N_V)

{

For (d=1 to d<=N_V)

{

Read value for A[s][d] (use nextInt( ))

If s equal to d

Then its diagonal element so make it zero

A[s][d]=0

Else if A[s][d] is equal to zero

Then it indicates there is no direct link between s and d. so assign a large value.

A[s][d] = MAX

}

}

Read the source vertex into variable: Source (use nextInt( ))

Create a object of the class BellmanFord passing N_V as argument for constructor :

B= new BellmanFord (N_V)

Call the function Bellmanfordevaluation with source and A[ ][ ] as arguments:

B. Bellmanfordevaluation (source, A)

End of the main

End of class

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)
```

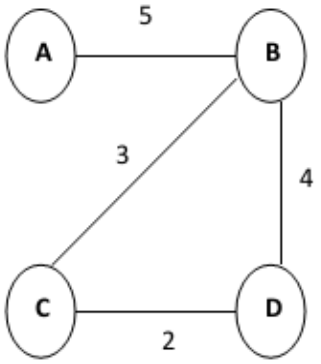
```
{
if (D[dn] > D[sn]+ A[sn][dn])
D[dn] = D[sn] + A[sn][dn];
}
}
}
}
for (int sn = 1; sn <= num_ver; sn++)
{
for (int dn = 1; dn <= num_ver; dn++)
{
if (A[sn][dn] != MAX_VALUE)
{
if (D[dn] > D[sn]+ A[sn][dn])
System.out.println("The Graph contains negative egde cycle");
}
}
}
for (int vertex = 1; vertex <= 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 7
0 3 0 2
0 7 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 5
```

Input graph:



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.

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

TCP − TCP stands for Transmission Control Protocol, which allows for reliable communication between two applications. TCP is typically used over the Internet Protocol, which is referred to as TCP/IP.

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

Socket Programming

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.

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

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

-The server instantiates a ServerSocket object, denoting which port number communication is to occur on.

-The server invokes the accept() method of the ServerSocket class. This method waits until a client connects to the server on the given port.

-After the server is waiting, a client instantiates a Socket object, specifying the server name and the port number to connect to.

-The constructor of the Socket class attempts to connect the client to the specified server and the port number. If communication is established, the client now has a Socket object capable of communicating with the server.

-On the server side, the accept() method returns a reference to a new socket on the server that is connected to the client's socket.

-After the connections are established, communication can occur using I/O streams. Each socket has both an OutputStream and an InputStream. The client's OutputStream is connected to the server's InputStream, and the client's InputStream is connected to the server's OutputStream.

TCP is a two-way communication protocol, hence data can be sent across both streams at the same time. Following are the useful classes providing complete set of methods to implement sockets.

ServerSocket

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

public ServerSocket(int port, int backlog) throws IOException

Similar to the previous constructor, the backlog parameter specifies how many incoming clients to store in a wait queue.

If the ServerSocket constructor does not throw an exception, it means that your application has successfully bound to the specified port and is ready for client requests.

public Socket accept() throws IOException

Waits for an incoming client. This method blocks until either a client connects to the server on the specified port or the socket times out, assuming that the time-out value has been set using the setSoTimeout() method. Otherwise, this method blocks indefinitely.

public InputStream getInputStream() throws IOException

Returns the input stream of the socket. The input stream is connected to the output stream of the remote socket.

public OutputStream getOutputStream() throws IOException

Returns the output stream of the socket. The output stream is connected to the input stream of the remote socket.

The DataOutputStream stream lets you write the primitives to an output source.

The DataInputStream is used in the context of DataOutputStream and can be used to read primitives.

The **java.io.DataInputStream.readUTF()** method reads in a string that has been encoded using a modified UTF-8 format. The string of character is decoded from the UTF and returned as String.

public static Path get(String first,String... more)

Converts a path string, or a sequence of strings that when joined form a path string, to a Path.

public final class Files

This class consists exclusively of static methods that operate on files, directories, or other types of files.

public static byte[ ] readAllBytes(Path path) throws IOException

Reads all the bytes from a file. The method ensures that the file is closed when all bytes have been read or an I/O error, or other runtime exception, is thrown.

Note that this method is intended for simple cases where it is convenient to read all bytes into a byte array. It is not intended for reading in large files.

Parameters:

path - the path to the file

Returns:

a byte array containing the bytes read from the file

String (byte[ ] bytes)

Constructs a new String by decoding the specified array of bytes using the platform's default charset.

Algorithm

Server :

Import the java packages: net and nio.file

Create a class named TCPServer

Start of the class:

Main method (which throws IOException):

Start of main method:

Create an object of the class ServerSocket to represent the welcoming socket of the server : **server**

Create an object of class DataOutputStream to point to the Output stream of the connection and initialize it as null: **out**

Create an object of class DataInputStream to point to the input stream of the connection: **in**

Start a try block

For the welcoming server socket "server" assign the port number and number of connection it can establish in parallel through the constructor of class ServerSocket:

server = new ServerSocket(5000, 1)

Print a statement on the server terminal “Server Waiting for client”

When client request comes, the welcoming server socket “server” accept the request and creates a new connection socket using the class Socket and function accept: socket

Socket socket = server.accept( );

Print a statement “Client connected”

Now assign the object “in” to the input stream of the socket using function socket.getInputStream:

in = new DataInputStream(socket.getInputStream())

Now assign the object “out” to the output stream of the socket using function socket.getOutputStream:

out = new DataOutputStream(socket.getOutputStream())

Create a string variable to hold the file name requested by the client from the input stream of the socket using the function readUTF: filename

fileName = in.readUTF()

Display the filename requested on the server terminal: “File Requested is : filename”

Get the path of the requested file using the get method of class Path and read the bytes from the file and store in a bytes array: filedata

byte[] filedata = Files.readAllBytes(Paths.get(fileName))

Convert the filedata into a string “fileContent” using String constructor

String fileContent = new String(filedata)

Display the fileContent

Write the filecontent into the output stream “out” using the function writeUTF

out.writeUTF(fileContent.toString())

Display a statement "FILE SENT SUCCESSFULLY"

End of try block

Start of catch block:

Print the standard error message “e” using getMessage function

Also send a message to the client "FILE DOESN'T EXISTS” at the output end of the socket using “out” object and the function writeUTF

End of catch block

End of main method

End of class

Client:

Import the java packages: net and nio.file

Create class named TCPClient

Start of Class:

Main method (which throws IOException, InterruptedException):

Start Main method

Create an object of class DataInputStream to point to the input stream of the connection: in

Create an object of class DataOutputStream to point to the Output stream of the connection : out

Create a client socket with IP address of server and port number using Socket function:

Socket socket = new Socket("127.0.0.1", 5000)

Print a statement: "Client Connected to Server"

Print a statement: "Enter the filename to request"

Read the filename into a string object : filename

Now assign the object "in" to the input stream of the socket using function socket.getInputStream:

in = new DataInputStream(socket.getInputStream())

Now assign the object "out" to the output stream of the socket using function socket.getOutputStream:

out = new DataOutputStream(socket.getOutputStream())

Write the filename into the output stream "out" using the function writeUTF

out.writeUTF(filename)

Create a string variable to hold the file content requested by the client from the input stream "in" of the socket using the function readUTF: fileContent

fileContent = in.readUTF()

Check:

if file length is greater than 0

Then

Print the file contents : “fileContent”

Else

Print a statement :"FILE IS EMPTY")

End of Main

End of class

SOURCE CODE:

//TCP SERVER

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

public class TCPServer
{

    public static void main(String[] args) throws IOException
    {
        ServerSocket server;
        DataOutputStream out = null;
        DataInputStream in;

        try
        {
            server = new ServerSocket(5000, 1); //port number and num of connections

            System.out.println("Server Waiting for client");
            Socket socket = server.accept();

            System.out.println("Client connected ");

            in = new DataInputStream(socket.getInputStream());
            out = new DataOutputStream(socket.getOutputStream());
```

```
                String fileName = in.readUTF();
                System.out.println("File Requested is : " + fileName);
                byte[] filedata = Files.readAllBytes(Paths.get(fileName));
                String fileContent = new String(filedata);

                out.writeUTF(fileContent.toString());
                System.out.println("FILE SENT SUCCESSFULLY");
            }
            catch (Exception e)
            {
                System.out.println(e.getMessage());
                out.writeUTF("FILE DOESN'T EXISTS");
            }
        }
}
```

//TCP CLIENT

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

public class TCPClient
{
    public static void main(String[] args) throws IOException,InterruptedException
    {
        DataOutputStream out;
        DataInputStream in;
        Scanner scanner = new Scanner(System.in);

        Socket socket = new Socket("127.0.0.1", 5000); //server IP and Port num
        System.out.println("Client Connected to Server");
        System.out.print("\nEnter the filename to request\n");
```

```
            String filename = scanner.nextLine();

            in = new DataInputStream(socket.getInputStream());
            out = new DataOutputStream(socket.getOutputStream());

            out.writeUTF(filename);

            String fileContent = in.readUTF();

            if (fileContent.length() > 0)
                System.out.println(fileContent);
            else
                System.out.println("FILE IS EMPTY");
        }
}
```

OUTPUT:

```
D:\cn-lab>javac TCPServer.java

D:\cn-lab>java TCPServer
Server Waiting for client
Client connected
File Requested is : test.txt
FILE SENT SUCCESSFULLY

D:\cn-lab>
```

```
D:\cn-lab>javac TCPClient.java

D:\cn-lab>java TCPClient
Client Connected to Server

Enter the filename to request
test.txt
file at server side

hello...

end of file

D:\cn-lab>_
```

Note: Create two different files Client.java and Server.java. Follow the steps given:

1. Open a terminal run the server program and provide the filename to send
2. 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.
3. Send any start bit to start sending file.
4. Refer https://www.tutorialspoint.com/java/java_networking.htm for all the parameters, methods description in socket communication.

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.

public class DatagramSocket

This class represents a socket for sending and receiving datagram packets.

A datagram socket is the 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.

DatagramSocket()

Constructs a datagram socket and binds it to any available port on the local host machine.

DatagramSocket(int port)

Constructs a datagram socket and binds it to the specified port on the local host machine.

public void receive(DatagramPacket p) throws IOException

Receives a datagram packet from this socket. When this method returns, the DatagramPacket's buffer is filled with the data received. The datagram packet also contains the sender's IP address, and the port number on the sender's machine.

This method blocks until a datagram is received. The length field of the datagram packet object contains the length of the received message. If the message is longer than the packet's length, the message is truncated.

Parameters:

p - The DatagramPacket into which to place the incoming data.

public final class DatagramPacket

This class represents a datagram packet.

Datagram packets are used to implement a connectionless packet delivery service. Each message is routed from one machine to another based solely on information contained within that packet. Multiple packets sent from one machine to another might be routed differently, and might arrive in any order. Packet delivery is not guaranteed.

DatagramPacket(byte[ ] buf, int length)

Constructs a DatagramPacket for receiving packets of length length.

DatagramPacket(byte[] buf, int length, InetAddress address, int port)

Constructs a datagram packet for sending packets of length length to the specified port number on the specified host.

public byte[ ] getData( )

Returns the data buffer. The data received or the data to be sent starts from the offset in the buffer, and runs for length long.

Returns: the buffer used to receive or send data

String (byte[ ] bytes, int offset, int length)

Constructs a new String by decoding the specified subarray of bytes using the platform's default charset.

public class InetAddress

This class represents an Internet Protocol (IP) address.

An IP address is either a 32-bit or 128-bit unsigned number used by IP, a lower-level protocol on which protocols like UDP and TCP are built.

public static InetAddress getByName(String host) throws UnknownHostException

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

ALGORITHM:

//SENDER

Import the java packages: scanner and net

Create a class named DSender

Start of the class:

Main method (which throws IOException):

Start of main method:

Print on the sender terminal “Sender”

Create an object of the class DatagramSocket to represent the datagram socket of the sender: **sendersoc**

Print on the sender terminal :“Enter the Message :”

Loop till end of message:

- Read the message to be sent in to a string variable using scanner object: **sendermsg**
- Get the IP address of the receiver using the method getByName( ) of class InetAddress: **ip**

  InetAddress ip = InetAddress.getByName("127.0.0.1")
- Prepare a datagram packet using the class DatagramPacket and sendermsg, sendermsg length and ip address of the receiver and port number: **senderdp**

  DatagramPacket senderdp = new DatagramPacket (sendermsg.getBytes()
  ,sendermsg.length(), ip, 3000)
- Send the prepared packet senderdp through the datagram socket sendersoc

  sendersoc.send(senderdp)

End of Loop

End of Main method

End of Class

//RECEIVER

Import the java packages: scanner and net

Create a class named DReceiver

Start of the class:

Main method (which throws IOException):

Create a buffer of type byte of size 1024 to receive the message: **buf**

Print on the receiver terminal "Receiver"

Create an object of the class DatagramSocket to represent the datagram socket of the Receiver with port number 3000: **recvsoc**

Loop till the message is recived:

- Prepare a datagram packet using the class DatagramPacket with the buffer created : **recvdp**

  DatagramPacket recvdp = new DatagramPacket(buf, 1024)
- receive the message into the packet recvdp through the receiver socket recvsoc

  recvsoc.receive(recvdp)
- extract the data in the packet into a string using methods getData and getLength:**recvmsg**

  String recvmsg = new String(recvdp.getData(), 0,recvdp.getLength())
- print the message on the recever terminal :recvmsg

End of Loop

End of Main method

End of Class

Source Code:

//RECEIVER

```
import java.net.*;
public class DReciever
{
	public static void main(String[] args) throws Exception
	{
		byte[] buf = new byte[1024];
		System.out.println("Receiver");
		DatagramSocket ds = new DatagramSocket(3000);

		while(true)
		{
			DatagramPacket dp = new DatagramPacket(buf, 1024);
			ds.receive(dp);
			String msg = new String(dp.getData(), 0, dp.getLength());
			System.out.println(msg);

		}
	}
}
```

//SENDER

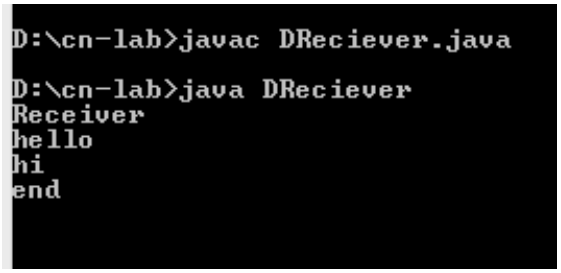
```
import java.net.*;
import java.util.Scanner;
public class DSender
{
	public static void main(String[] args) throws Exception
	{
```

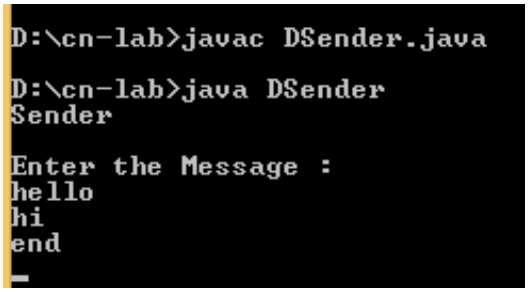
```
            System.out.println("Sender");
            DatagramSocket ds = new DatagramSocket();
            Scanner scanner = new Scanner(System.in);
            System.out.println("\nEnter the Message : ");
            while(true)
            {
                  String msg = scanner.nextLine();
                  InetAddress ip = InetAddress.getByName("127.0.0.1");
                  DatagramPacket dp = new DatagramPacket(msg.getBytes(), msg.length(), ip, 3000);
                  ds.send(dp);

            }
      }
}
```

OUTPUT:

Note: Create two different files DSender.java and DReciever.java. Follow the following steps:

1. Open a terminal run the server program.
2. Open one more terminal run the client program, the sent message will be received.





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

RSA is an example of public key cryptography. It was developed by Rivest, Shamir and Adelman. The RSA algorithm can be used for both public key encryption and digital signatures. Its security is based on the difficulty of factoring large integers.

The RSA algorithm's efficiency requires a fast method for performing the modular exponentiation operation. A less efficient, conventional method includes raising a number (the input) to a power (the secret or public key of the algorithm, denoted e and d, respectively) and taking the remainder of the division with N. A straight-forward implementation performs these two steps of the operation sequentially: first, raise it to the power and second, apply modulo. The RSA algorithm comprises of three steps, which are depicted below:

Key Generation Algorithm

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

Encryption

Sender A does the following:-

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

Decryption

Recipient B does the following:-

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

Algorithm:

Create class named RSA

Start of class

Create a method named keygen

Start of method

Create two random numbers base on time

rand1(with current time),

rand2(with current time*10)

read a initial value for public key: pubkey

create objects p and q of class BigInteger to store the prime value of 32 bit length based on rand1 and rand2 (use probablePrime)

create an object n of class BigInteger to store p*q (use p.multiply(q))

create an object p_1 of class BigInteger to store p-1 (use p.subtract(new BigInteger("1")))

create an object q_1 of class BigInteger to store q-1 (use q.subtract(new BigInteger("1")))

create an object fi of class BigInteger to store (p-1)(q-1) (use p_1.multiply(q_1))

//to find public key

loop till public key is generated

{

Find GCD between fi and publickey,

GCD=fi.gcd(pubkey))

If GCD is equal to one

then break

else increment pulickey value

}

Once public key is finalized,

create an object new_pubkey of class BigInteger to store the finalized public key

(use BigInteger(pubkey + ""))

//To find private key

create an object new_prvkey of class BigInteger to store the the value of inverse of new_public modulo fi

(use new_pubkey.modInverse(fi))

//Print the keys

Print values of new_pubkey and n

Print values of new_prvkey and n

End of method keygen

Create a method called encdec

Start of encdec

Read the public key value into a string variable : pubkey1

create an object pubkey of class BigInteger to store the the value pubkey1

Read the private key value into a string variable : prvkey1

create an object prvkey of class BigInteger to store the the value prvkey1

Read the value of n into a string variable : n1

create an object n of class BigInteger to store the the value n1

Read the message value into a string variable : msg1

create an object msg of class BigInteger to store the the value msg1

//encryption

Create an object cipherVal to store the value of $(msg)^{(pubkey)}$ mod n

(Use msg.modPow(pubkey,n))

Print the cipher text value : cipherVal

//Decription

Create an object plainVal to store the value of $(cipherVal)^{(prvkey)}$ mod n

(Use cipherVal.modPow(prvkey,n))

Print the original msg value : plainVal

End of method encdec

Main function

Start of main

Create an object of class RSA : R

Call the method keygen to generate public and private keys

Call the method encdec to perform encryption and decryption

End of main

End of class

Source Code:

```
import java.math.BigInteger;
import java.util.Random;
import java.util.Scanner;
public class RSA {
        public void keygen()
        {

        Random rand1=new Random(System.currentTimeMillis());
        Random rand2=new Random(System.currentTimeMillis()*10);
        Scanner scanner = new Scanner(System.in);
        System.out.println("Enter a initial value for public key");
        int pubkey=scanner.nextInt();
        BigInteger p=BigInteger.probablePrime(32, rand1);
        BigInteger q=BigInteger.probablePrime(32, rand2);
        BigInteger n=p.multiply(q);
        BigInteger p_1=p.subtract(new BigInteger("1"));
        BigInteger q_1=q.subtract(new BigInteger("1"));
        BigInteger fi=p_1.multiply(q_1);
        while(true)
        {
        BigInteger GCD=fi.gcd(new BigInteger(pubkey+""));
        if(GCD.equals(BigInteger.ONE))
                {
                        break;
                }
        pubkey++;
        }
        BigInteger new_pubkey=new BigInteger(pubkey+"");
        BigInteger new_prvkey=new_pubkey.modInverse(fi);
        System.out.println("public key : "+new_pubkey+","+n);
        System.out.println("private key : "+new_prvkey+","+n);
        }
```

```
    public void encdec()
    {
        Scanner scanner = new Scanner(System.in);
        System.out.println("Enter the public key");
        String pubkey1=scanner.nextLine();
        BigInteger pubkey = new BigInteger(pubkey1);
        System.out.println("Enter the Private key");
        String prvkey1=scanner.nextLine();
        BigInteger prvkey = new BigInteger(prvkey1);
        System.out.println("Enter the value of n");
        String n1=scanner.nextLine();
        BigInteger n = new BigInteger(n1);
        System.out.println("Enter the message key");
        String msg1=scanner.nextLine();
        BigInteger msg=new BigInteger(msg1);
        BigInteger cipherVal=msg.modPow(pubkey,n);
        System.out.println("Cipher text: " + cipherVal);
        BigInteger plainVal=cipherVal.modPow(prvkey,n);
        System.out.println("Plain text:" + plainVal);
    }

    public static void main(String[] args) {
        RSA R= new RSA();
        System.out.println("The Key generation phase");
        R.keygen();
        System.out.println("The Encryption/Decrytion phase");
        R.encdec();
    }

}
```

OUTPUT:

The Key generation phase

Enter a initial value for public key

10

public key : 13,8735161591355742359

private key : 40316130393785714 77,8735161591355742359

The Encryption/Decrytion phase

Enter the public key: 13

Enter the Private key: 4031613039378571477

Enter the value of n: 8735161591355742359

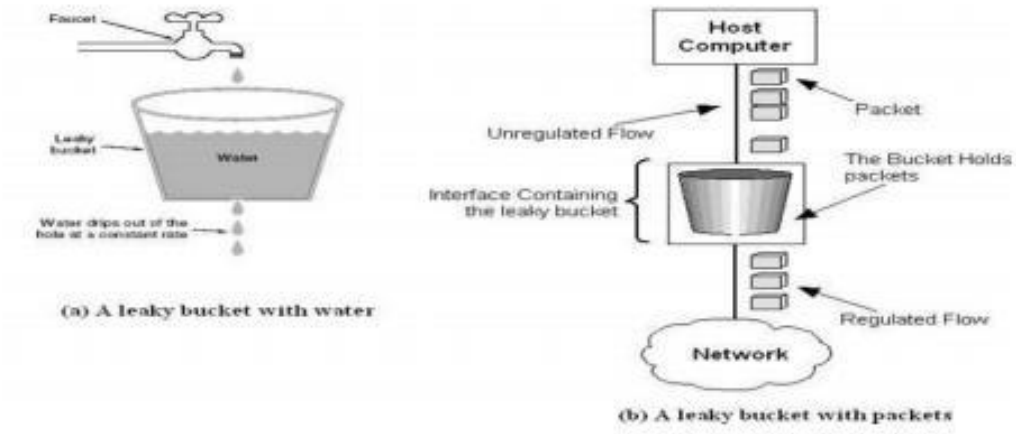
Enter the message key : 787

Cipher text: 6309385004489097632

Plain text: 787

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 (a) 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.

Algorithm:

Create a class named Leakybucket
Start of the class
Main method:
Start of main method
Read the bucket size: n
Read the number of incoming packets: np

Create an array of size np and integer type to hold size of incoming packet : a[ ]

Read the packet size for the incoming packets:

for(i=0 to i< np)

{

Read packet size for i^{th} packet and store it in a[i]

}

Read the output rate: out

//transmitting packets

For each packet i.e for(i=0 to i<np)

{

Print a statement "Transmitting packet of size: value of a[i]"

Check if i^{th} packet size is greater than bucket size i.e., if (a[i]>n)

Then print "packet cannot be transmitted Bucket overflow: value of a[i]"

Else packet can be transmitted, then

Check if packet size is equal to output rate or less than than output rate i.e., **if** (a[i] equal to out or a[i] less than out)

Then print "transmitted: value of a[i] bits"

else if (a[i] is greater than out)

then

while(a[i] not equal to 0 and a[i] is greater than out)

{

Print "transmitted: **out** bits"

Reduce a[i] by out since out no. of bits are transmitted i.e.,

a[i]=a[i]-out;

}end of while

Now transmitting remaining bits which is less than out
print "transmitted: **a[i]** bits;

End of else if (a[i] is greater than out)

End of else packet can be transmitted

End of for each packet

End of main method

End of class LeakyBucket

SOURCE CODE:

```
import java.util.Scanner;
public class Leakybucket {
public static void main(String[] args) {
        Scanner sc= new Scanner(System.in);
        System.out.println("Enter the bucket size:");
        int n=sc.nextInt();
        System.out.println("Enter the number of incomming packets:");
        int np=sc.nextInt();
        int a[ ]=new int[np];
        System.out.println("Enter the incomming packet size:");
        for(int i=0;i<np;i++)
        {
                a[i]=sc.nextInt();
        }
        System.out.println("Enter output rate=");
        int out=sc.nextInt();
        for(int i=0;i<np;i++)
        {
                System.out.println("Transmitting packet of size:"+a[i]);
                if(a[i]>n)
                {
                System.out.println("packet cannot be transmitted, Bucket overflow: "+a[i]+"bits");
                }
                else
                {
                        if (a[i]<=out)
                                System.out.println("    transmitted: "+a[i]+"bits");
                        else if(a[i]>out)
                        {
```

```
                    while(a[i]>out)
                    {
                        System.out.println("    transmitted: "+out+"bits");
                        a[i]=a[i]-out;
                     }
                    System.out.println("    transmitted: "+a[i]+"bits");

                    }
                 }
                }
        }
}
```

OUTPUT:

```
Enter the bucket size: 30
Enter the number of incomming packets: 4
Enter the incomming packet size:
15
35
3
30
Enter output rate=10
Transmitting packet of size:15
        transmitted: 10bits
        transmitted: 5bits
Transmitting packet of size:35
packet cannot be transmitted, Bucket overflow: 35bits
Transmitting packet of size:3
        transmitted: 3bits
Transmitting packet of size:30
        transmitted: 10bits
        transmitted: 10bits
        transmitted: 10bits
```

VIVA QUESTIONS

1. What are functions of different layers?
2. Differentiate between TCP/IP Layers and OSI Layers
3. Differentiate between TCP and UDP.
4. Differentiate between Connectionless and connection oriented connection.
5. What is meant by subnet?
6. What is meant by Gateway?
7. What is an IP address?
8. What is MAC address?
9. What is a Router. 28. What is routing.
10. What is the role of DNS.
11. What type of transport protocol is used for DNS.
12. What protocols does ns support?
13. What Is Simulation?
14. Define Network
15. What is meant by Protocol?
16. What are the constituent parts of NS2?
17. What are the Different topologies available in networks?
18. Which topology requires multipoint conection?
19. Data communication system within a campus is called as?
20. What is meant by WAN?
21. Explain the working of Ring topology?
22. What is ARQ?
23. What is stop and wait protocol?
24. What is stop and wait ARQ?
25. What is usage of sequence number in reliable transmission?
26. What is sliding window?
27. Compare connection oriented and connection less protocols.
28. What is MTU?
29. Explain the working of Distance vector routing.
30. Differentiate Proactive and Reactive routing Protocols.
31. What are the different attributes for calculating the cost of a path?
32. What is Routing?

33. What is Dynamic rouitng?
34. What are the two steps in link state routing?
35. Compare link state and Distance Vector routing
36. What are all the route metric used in Link state routing?
37. What is meant by Encryption?
38. What is Decryption?
39. Encrypt the word “HELLO” using Caesar cipher.
40. What are the different algorithms available for encryption?
41. What type of information can be secured with Cryptography?
42. What are the types of errors?
43. What is Error Detection? What are its methods?
44. What is Redundancy?
45. What is CRC?
46. What is Checksum?
47. Define Socket
48. What is socket programming?
49. What is the function of command bind?
50. What is the syntax for connecting Client and Server
51. What is the command to assign port number to client and server?
52. What does a Socket consists of?
53. What is the concept of Echo?
54. Explain Ping?
55. Mention some advantages of Java Sockets.
56. How do you open a Socket? . What is TCP?
57. Explain the three way Handshake process?
58. Which layer is closer to a user?
59. Explain how TCP avoids a network meltdown?
60. What is the difference between flow control and Error control?