SYLLABUS
COMPUTER NETWORKS LABORATORY
V SEMESTER

Subject Code: 15CSL57	**I.A. Marks: 20**
Hours/Week: 03	**Exam Hours: 03**
Total Hours: 40	**Exam Marks: 80**

PART A

Implement the following in NS2/NS3:

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

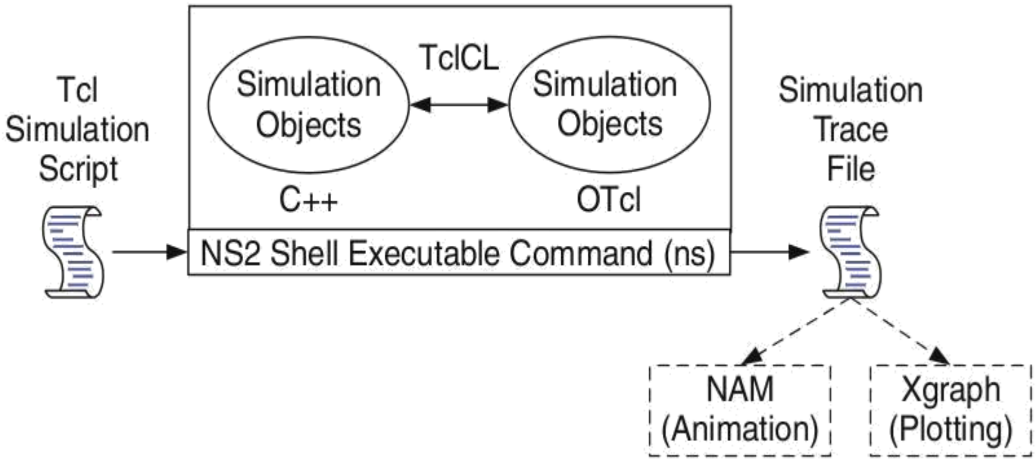
Implement the following in Java:

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.

3.INTRODUCTION TO NS-2

- Widely known as NS2, is simply an event driven simulation tool.
- Useful in studying the dynamic nature of communication networks.
- Simulation of wired as well as wireless network functions and protocols (e.g., routing algorithms, TCP, UDP) can be done using NS2.
- In general, NS2 provides users with a way of specifying such network protocols and simulating their corresponding behaviours.

Basic Architecture of NS2



Tcl scripting

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

Basics of TCL

Syntax: command arg1 arg2 arg3

- **Hello World!**

  puts stdout{Hello, World!}

  Hello, World!

- **Variables** Command Substitution

  set a 5 set len [string length foobar]

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

- **Simple Arithmetic**
  expr 7.2 / 4

- **Procedures**

```
proc Diag {a b} {

set c [expr sqrt($a * $a + $b * $b)]

return $c }

puts "Diagonal of a 3, 4 right triangle is [Diag 3 4]"
```

Output: Diagonal of a 3, 4 right triangle is 5.0

- **Loops**

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

. . .                           . . .

    }                           }
```

Wired TCL Script Components

- Create the event scheduler
- Open new files & turn on the tracing
- Create the node
- 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 NS2

An ns simulation starts with the command

```
set ns [new Simulator]
```

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

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

#Open the Trace file

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

#Open the NAM trace file

```
set namfile [open out.nam w]
$ns namtrace-all $namfile
```

The above creates a data trace file called “out.tr” and a nam visualization trace file called “out.nam”. Within the tcl script, these files are not called explicitly by their names, but instead by pointers that are declared above and called “tracefile1” and “namfile” respectively.Remark that they begins with a # symbol. The second line open the file “out.tr” to be used forwriting, declared with the letter “w”. The third line uses a simulator method called trace-all that have as parameter the name of the file where the traces will go.

The last line tells the simulator to record all simulation traces in NAM input format. It also gives the file name that the trace will be written to later by the command $ns flush-trace. In our case, this will be the file pointed at by the pointer “$namfile”,i.e the file “out.tr”.

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

#Define a ‘finish’ procedure

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

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

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

```
$ns at 125.0 “finish”
```

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

The simulation can then begin using the command

```
$ns run
```

Definition of a network of links and nodes

The way to define a node is

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

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

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

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

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

To define a directional link instead of a bi-directional one, we should replace “duplex-link” by “simplex-link”.

In NS, an output queue of a node is implemented as a part of each link whose input is that node. The definition of the link then includes the way to handle overflow at that queue.

In our case, if the buffer capacity of the output queue is exceeded then the last packet to arrive is dropped. Many alternative options exist, such as the RED (Random Early Discard) mechanism, the FQ (Fair Queuing), the DRR (Deficit Round Robin), the stochastic Fair Queuing (SFQ) and the CBQ (which including a priority and a round-robin scheduler).

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

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

Agents and Applications

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

FTP over TCP

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

There are number variants of the TCP protocol, such as Tahoe, Reno, NewReno, Vegas.

The type of agent appears in the first line:

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

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

The command

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

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

#Setup a UDP connection

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

#setup a CBR over UDP connection

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

Above shows the definition of a CBR application using a UDP agent

The command **$ns attach-agent $n4 $sink** defines the destination node. The command **$ns connect $tcp $sink** finally makes the TCP connection between the source and destination nodes.

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

When we have several flows, we may wish to distinguish them so that we can identify them with different colors in the visualization part. This is done by the command **$tcp set fid_1** that assigns to the TCP connection a flow identification of "1".We shall later give the flow identification of "2" to the UDP connection.

CBR over UDP

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

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

```
$cbr set interval_ 0.005
```

The packet size can be set to some value using

```
$cbr set packetSize_ <packet size>
```

Scheduling Events

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

$ns at <time><event>

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

$ns at 0.1 “$cbr start”

$ns at 1.0 “ $ftp start”

$ns at 124.0 “$ftp stop”

Structure of Trace Files

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

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

1. The first field is the event type. It is given by one of four possible symbols r, +, -, d which correspond respectively to receive (at the output of the link), enqueued, dequeued and dropped.
2. The second field gives the time at which the event occurs.
3. Gives the input node of the link at which the event occurs.
4. Gives the output node of the link at which the event occurs.
5. Gives the packet type (eg CBR or TCP)
6. Gives the packet size
7. Some flags
8. This is the flow id (fid) of IPv6 that a user can set for each flow at the input OTcl script one can further use this field for analysis purposes; it is also used when specifying stream color for the NAM display.
9. This is the source address given in the form of “node.port”.

10. This is the destination address, given in the same form.

11. This is the network layer protocol's packet sequence number. Even though UDP implementations in a real network do not use sequence number, ns keeps track of UDP packet sequence number for analysis purposes

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

XGRAPH

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

Syntax:

Xgraph [options] file-name

Options are listed here

/-bd <color> (Border)

This specifies the border color of the xgraph window.

/-bg <color> (Background)

This specifies the background color of the xgraph window.

/-fg<color> (Foreground)

This specifies the foreground color of the xgraph window.

/-lf <fontname> (LabelFont)

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

/-t<string> (Title Text)

This string is centered at the top of the graph.

/-x <unit name> (XunitText)

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

/-y <unit name> (YunitText)

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

Awk- An Advanced

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

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

Syntax:

awk option ‘selection_criteria {action}’ file(s)

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

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

Variables

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

$ awk –F”|” ‘$3 == “director” && $6 > 6700 {
count =count+1

printf “ %3f %20s %-12s %d\n”, count,$2,$3,$6 }’empn.lst

THE –f OPTION: STORING awk PROGRAMS INA FILE

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

$ cat empawk.awk

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

Awk –F”|” –f empawk.awk empn.lst

THE BEGIN AND END SECTIONS

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

BEGIN {action}

END {action}

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

BUILT-IN VARIABLES

Awk has several built-in variables. They are all assigned automatically, though it is also possible for a user to reassign some of them. You have already used NR, which signifies the record number of the current line. We'll now have a brief look at some of the other variable. ***The FS Variable:*** as stated elsewhere, awk uses a contiguous string of spaces as the default field delimiter. FS redefines this field separator, which in the sample database happens to be the |. When used at all, it must occur in the BEGIN section so that the body of the program knows its value before it starts processing:

BEGIN {FS="|"}

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

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

BEGIN { OFS="~" }

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

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

$awk 'BEGIN {FS = "|"}

NF! =6 {

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

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

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

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

PART -A

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

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

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

$ns color 1 "red"
$ns color 2 "blue"

$n0 label "Source/udp0"
$n1 label "Source/udp1"
$n2 label "Router"
$n3 label "Destination/Null"

$ns duplex-link $n0 $n2 10Mb 300ms DropTail
$ns duplex-link $n1 $n2 10Mb 300ms DropTail
$ns duplex-link $n2 $n3 1Mb 300ms DropTail


$ns set queue-limit $n0 $n2 10
$ns set queue-limit $n1 $n2 10
$ns set queue-limit $n2 $n3 5


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

set null3 [new Agent/Null]
$ns attach-agent $n3 $null3

set udp1 [new Agent/UDP]
$ns attach-agent $n1 $udp1
set cbr1 [new Application/Traffic/CBR]
```

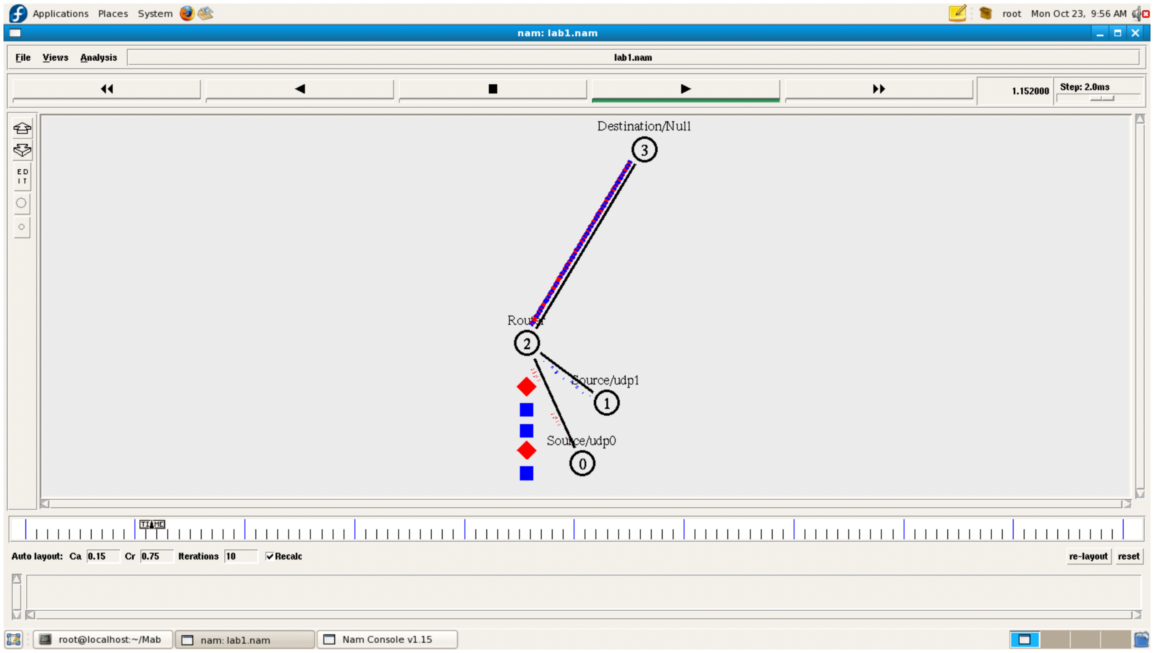
```
$cbr1 attach-agent $udp1


$udp0 set class_ 1
$udp1 set class_ 2


$ns connect $udp0 $null3
$ns connect $udp1 $null3

$cbr1 set packetSize_ 500Mb
$cbr1 set interval_ 0.005

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

$ns at 0.1 "$cbr0 start"
$ns at 0.1 "$cbr1 start"
$ns at 10.0 "finish"
$ns run
```

AWK SCRIPT

```
BEGIN {
#include <stdio.h>
count=0;
}
{
if ($1=="d")
count++;
}
END{
printf("dropped packets %d\n\n", count);
}
```

To Run:

ns lab1.tcl



awk –f lab1.awk lab1.tr

Number of packets dropped in 750

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

```
set ns [ new Simulator ]
set tf [open lab2.tr w ]
$ns trace-all $tf
set nf [ open lab2.nam w ]
$ns namtrace-all $nf
set n0 [ $ns node ]
set n1 [ $ns node ]
set n2 [ $ns node ]
set n3 [ $ns node ]
set n4 [ $ns node ]
set n5 [ $ns node ]
set n6 [ $ns node ]

$n0 label "ping0"
$n4 label "ping4"
$n5 label "ping5"
$n6 label "ping6"
$n2 label "Router"
```

```
$ns color 1 "red"
$ns color 2 "green"

$ns duplex-link $n0 $n2 100Mb 300ms DropTail
$ns duplex-link $n6 $n2 1Mb 300ms DropTail
$ns duplex-link $n5 $n2 100Mb 300ms DropTail
$ns duplex-link $n2 $n4 1Mb 300ms DropTail
$ns duplex-link $n3 $n2 1Mb 300ms DropTail
$ns duplex-link $n1 $n2 1Mb 300ms DropTail

$ns queue-limit $n0 $n2 5
$ns queue-limit $n2 $n6 2
$ns queue-limit $n2 $n4 3
$ns queue-limit $n5 $n2 5

set ping0 [new Agent/Ping]
$ns attach-agent $n0 $ping0

set ping4 [new Agent/Ping]
$ns attach-agent $n4 $ping4

set ping5 [new Agent/Ping]
$ns attach-agent $n5 $ping5

set ping6 [new Agent/Ping]
$ns attach-agent $n6 $ping6

$ping0 set packetSize_ 50000
$ping0 set interval_ 0.0001
$ping5 set packetSize_ 60000
$ping5 set interval_ 0.00001

$ping0 set class_ 1
$ping5 set class_ 2

$ns connect $ping0 $ping4
$ns connect $ping5 $ping6

Agent/Ping instproc recv {from rtt} {
$self instvar node_
puts "The node [$node_ id] received an reply from $from with rtt of $rtt"  }

proc finish { } {
global ns nf tf
```

```
exec nam lab2.nam &
$ns flush-trace
close $tf
close $nf
exit 0
}

$ns rtmodel-at 0.9 down $n2 $n6
$ns rtmodel-at 1.5 up $n2 $n6

$ns at 0.1 "$ping0 send"
$ns at 0.2 "$ping0 send"
$ns at 0.3 "$ping0 send"
$ns at 0.4 "$ping0 send"
$ns at 0.5 "$ping0 send"
$ns at 0.6 "$ping0 send"
$ns at 0.7 "$ping0 send"
$ns at 0.8 "$ping0 send"
$ns at 0.9 "$ping0 send"
$ns at 1.0 "$ping0 send"
$ns at 1.1 "$ping0 send"
$ns at 1.2 "$ping0 send"
$ns at 1.3 "$ping0 send"
$ns at 1.4 "$ping0 send"
$ns at 1.5 "$ping0 send"
$ns at 1.6 "$ping0 send"
$ns at 1.7 "$ping0 send"
$ns at 1.8 "$ping0 send"

$ns at 0.1 "$ping5 send"
$ns at 0.2 "$ping5 send"
$ns at 0.3 "$ping5 send"
$ns at 0.4 "$ping5 send"
$ns at 0.5 "$ping5 send"
$ns at 0.6 "$ping5 send"
$ns at 0.7 "$ping5 send"
$ns at 0.8 "$ping5 send"
$ns at 0.9 "$ping5 send"
$ns at 1.0 "$ping5 send"
$ns at 1.1 "$ping5 send"
$ns at 1.2 "$ping5 send"
$ns at 1.3 "$ping5 send"
$ns at 1.4 "$ping5 send"
$ns at 1.5 "$ping5 send"
$ns at 1.6 "$ping5 send"
```

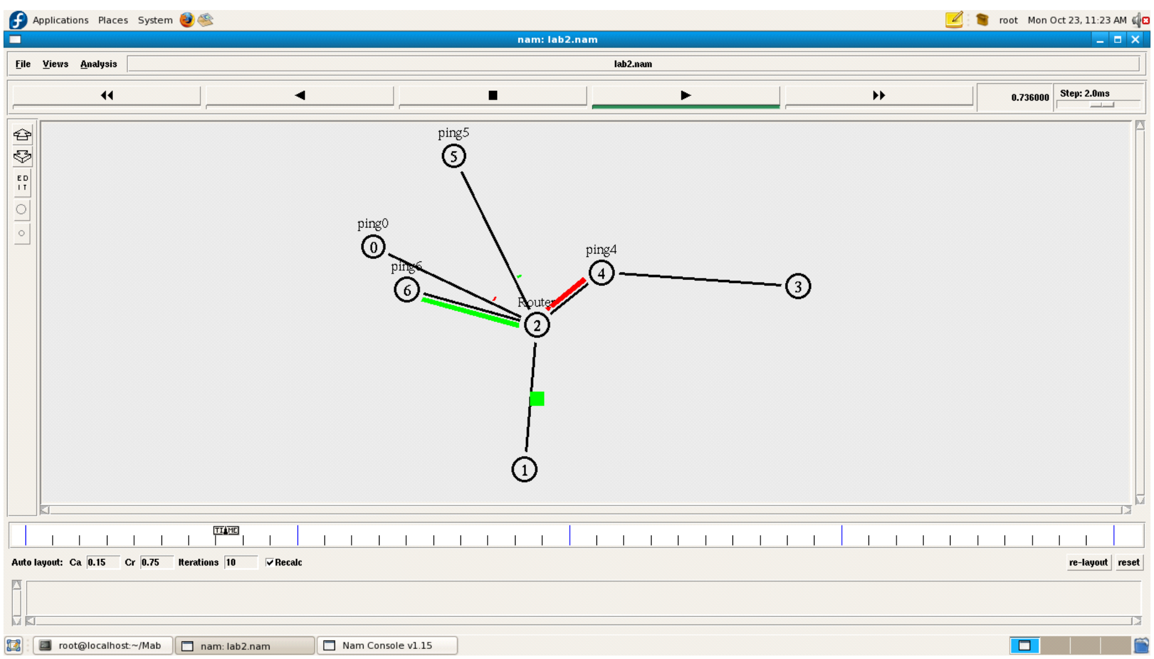
```
$ns at 1.7 "$ping5 send"
$ns at 1.8 "$ping5 send"

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

AWK SCRIPT

```
BEGIN {
#include<stdio.h>
count=0;
}
{
 if($1=="d")
 count++
}
END {
printf("Total no of packets dropped due to congestion is %d", count);
}
```

To Run:

ns lab2.tcl



awk –f lab2.awk lab2.tr

The node 0 received reply from 4 with rtt of 1604.5
The node 0 received reply from 4 with rtt of 1904.5
The node 0 received reply from 4 with rtt of 2204.5
The node 0 received reply from 4 with rtt of 2404.5
The node 5 received reply from 6 with rtt of 1685.5
The node 0 received reply from 4 with rtt of 2404.5
The node 6 received reply from 5 with rtt of 2065.3
The node 0 received reply from 4 with rtt of 2404.5
The node 5 received reply from 6 with rtt of 2145.5
The node 0 received reply from 4 with rtt of 2404.5

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

```
set ns [new Simulator]
set tf [open lab3.tr w]
$ns trace-all $tf
set nf [open lab3.nam w]
$ns namtrace-all $nf
set n0 [$ns node]
$n0 color "magenta"
$n0 label "src1"
set n1 [$ns node]
set n2 [$ns node]
$n2 color "magenta"
$n2 label "src2"
set n3 [$ns node]
$n3 color "blue"
$n3 label "dest2"
set n4 [$ns node]
set n5 [$ns node]
$n5 color "blue"
$n5 label "dest1"
$ns make-lan "$n0 $n1 $n2 $n3 $n4" 100Mb 100ms LL Queue/DropTail Mac/802_3
$ns duplex-link $n4 $n5 1Mb 1ms DropTail
set tcp0 [new Agent/TCP]
```

```
$ns attach-agent $n0 $tcp0
set ftp0 [new Application/FTP]
$ftp0 attach-agent $tcp0
$ftp0 set packetSize_ 500
$ftp0 set interval_ 0.0001
set sink5 [new Agent/TCPSink]
$ns attach-agent $n5 $sink5
$ns connect $tcp0 $sink5
set tcp2 [new Agent/TCP]
$ns attach-agent $n2 $tcp2
set ftp2 [new Application/FTP]
$ftp2 attach-agent $tcp2
$ftp2 set packetSize_ 600
$ftp2 set interval_ 0.001
set sink3 [new Agent/TCPSink]
$ns attach-agent $n3 $sink3
$ns connect $tcp2 $sink3
set file1 [open file1.tr w]
$tcp0 attach $file1
set file2 [open file2.tr w]
$tcp2 attach $file2
$tcp0 trace cwnd_
$tcp2 trace cwnd_
proc finish { } {
global ns nf tf
$ns flush-trace
close $tf
close $nf
exec nam lab3.nam &
exit 0
}
$ns at 0.1 "$ftp0 start"
$ns at 5 "$ftp0 stop"
$ns at 7 "$ftp0 start"
$ns at 0.2 "$ftp2 start"
$ns at 8 "$ftp2 stop"
$ns at 14 "$ftp0 stop"
$ns at 10 "$ftp2 start"
$ns at 15 "$ftp2 stop"
```

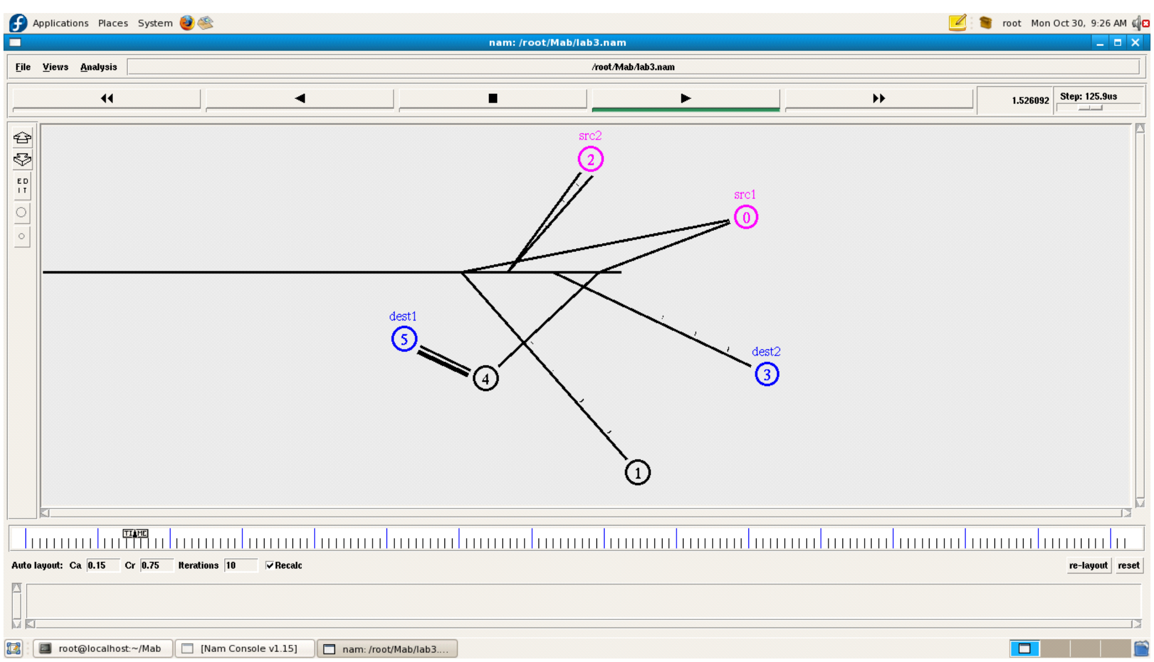
```
$ns at 16 "finish"
$ns run
```

AWK SCRIPT

```
BEGIN {
   #inculde<stdio.h>
}
{
if($6=="cwnd_")
printf("%f\t%f\t\n",$1,$7);
}
END {
}
```

To Run:

ns lab3.tcl



awk –f lab3.awk file1.tr>tcp1

awk –f lab3.awk file2.tr>tcp2

xgraph –x “time” –y “convalue” tcp1 tcp2

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

```
set ns [new Simulator]
set tf [open lab4.tr w]
$ns trace-all $tf
set topo [new Topography]
$topo load_flatgrid 1000 1000
set nf [open lab4.nam w]
$ns namtrace-all-wireless $nf 1000 1000
$ns node-config -adhocRouting DSDV \
        -llType LL \
        -macType Mac/802_11 \
        -ifqType Queue/DropTail \
        -ifqLen 50 \
        -phyType Phy/WirelessPhy \
```

```
        -channelType Channel/WirelessChannel \
        -propType Propagation/TwoRayGround \
        -antType Antenna/OmniAntenna \
        -topoInstance $topo \
        -agentTrace ON \
        -routerTrace ON
create-god 3
set n0 [$ns node]
set n1 [$ns node]
set n2 [$ns node]
$n0 label "tcp0"
$n1 label "sink1/tcp1"
$n2 label "sink2"
$n0 set X_ 50
$n0 set Y_ 50
$n0 set Z_ 0
$n1 set X_ 100
$n1 set Y_ 100
$n1 set Z_ 0
$n2 set X_ 600
$n2 set Y_ 600
$n2 set Z_ 0
$ns at 0.1 "$n0 setdest 50 50 15"
$ns at 0.1 "$n1 setdest 100 100 25"
$ns at 0.1 "$n2 setdest 600 600 25"
set tcp0 [new Agent/TCP]
$ns attach-agent $n0 $tcp0
set ftp0 [new Application/FTP]
$ftp0 attach-agent $tcp0
set sink1 [new Agent/TCPSink]
$ns attach-agent $n1 $sink1
$ns connect $tcp0 $sink1
set tcp1 [new Agent/TCP]
$ns attach-agent $n1 $tcp1
set ftp1 [new Application/FTP]
$ftp1 attach-agent $tcp1
set sink2 [new Agent/TCPSink]
$ns attach-agent $n2 $sink2
$ns connect $tcp1 $sink2
```

```
$ns at 5 "$ftp0 start"
$ns at 5 "$ftp1 start"
$ns at 100 "$n1 setdest 550 550 15"
$ns at 190 "$n1 setdest 70 70 15"
proc finish { } {
        global ns nf tf
        $ns flush-trace
        exec nam lab4.nam &
        close $tf
        exit 0
}
$ns at 250 "finish"
$ns run
```

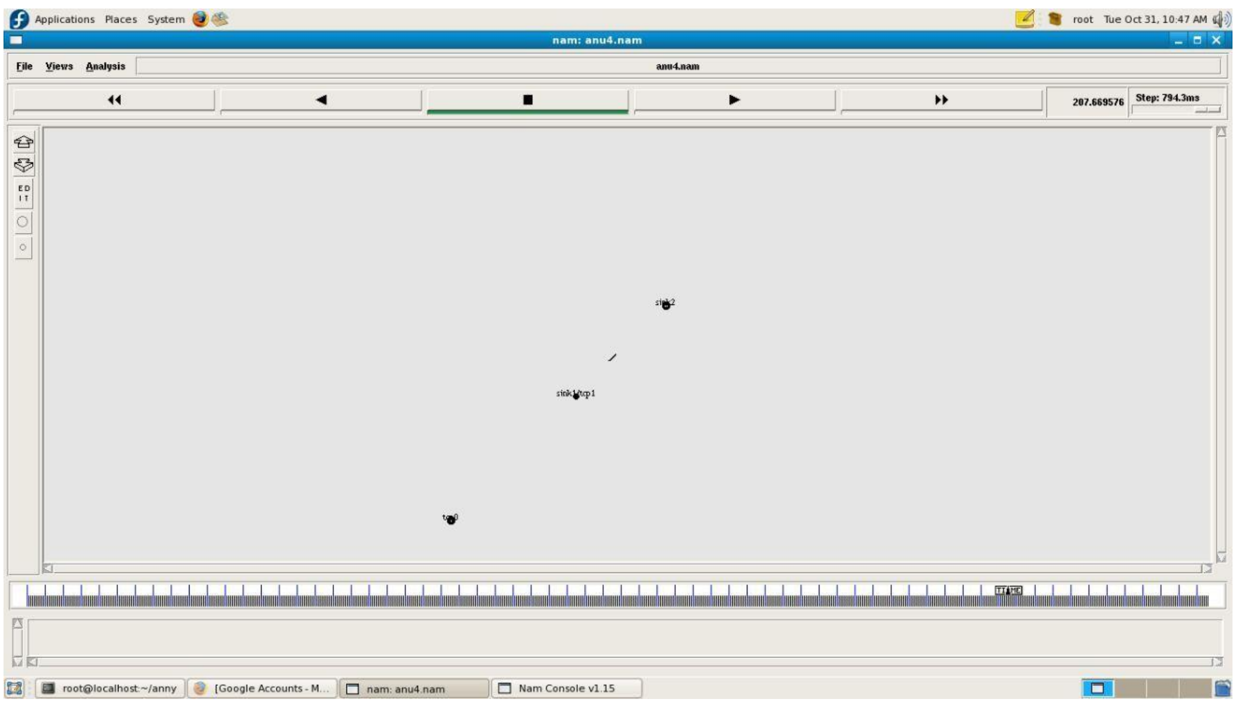
AWK SCRIPT

```
BEGIN {
     count1=0
     count2=0
     pack1=0
     pack2=0
     time1=0
     time2=0
}
{
     if($1=="r"&& $3=="_1_" && $4=="AGT")
     {
            count1++
            pack1=pack1+$8
            time1=$2
     }
     if($1=="r" && $3=="_2_" && $4=="AGT")
     {
            count2++
            pack2=pack2+$8
            time2=$2
        }
```

```
}
END {
printf("The Throughput from n0 to n1: %f Mbps\n",((count1*pack1*8)/(time1*1000000)));

printf("The Throughput from n1 to n2: %f Mbps\n",((count2*pack2*8)/(time2*1000000)));
}
```

<u>To Run:</u>

ns lab4.tcl



awk –f lab4.awk lab4.tr

The Through put from n0 to n1 is:5683.44245Mbps

The Through put from n1 to n2 is:1307.611834Mbps.

NOTE : Type 5th and 6th simulation in the path specified below

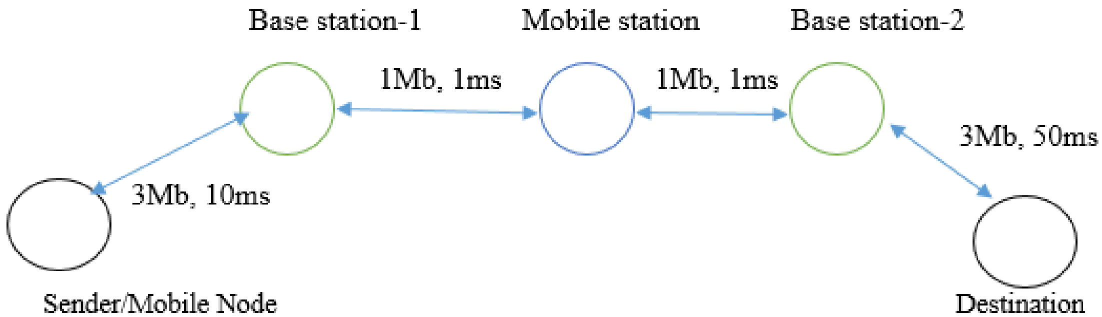
ns-allinone-2.35/ns-2.35/tcl/ex/wireless-scripts

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

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

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

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



General Parameters
set opt(title) zero ;

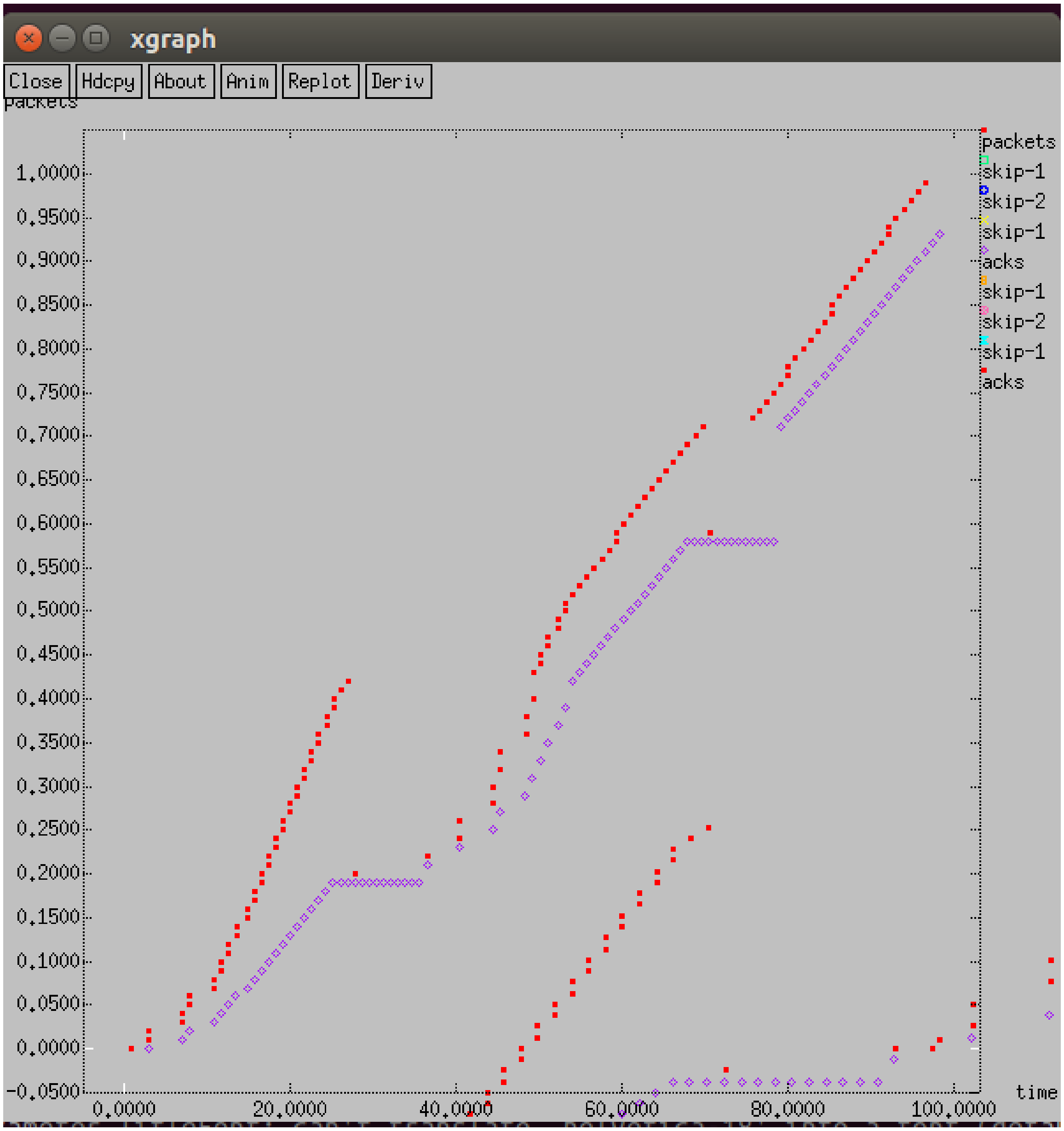
```
set opt(stop) 100 ;# Stop time.
set opt(ecn) 0 ;
# Topology
set opt(type) gsm ;#type of link:
set opt(secondDelay) 55 ;# average delay of access links in ms
# AQM parameters
set opt(minth) 30 ;
set opt(maxth) 0 ;
set opt(adaptive) 1 ;# 1 for Adaptive RED, 0 for plain RED
# Traffic generation.
set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

To Run:

```
ns lab5.tcl
```

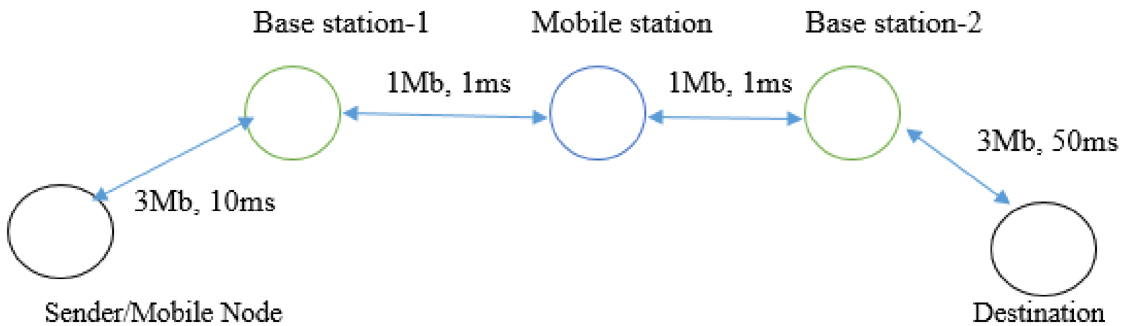

xgraph
Close
Hdcpy
About
Anim
Replot
Deriv
packets
1.0000
0.9500
0.9000
0.8500
0.8000
0.7500
0.7000
0.6500
0.6000
0.5500
0.5000
0.4500
0.4000
0.3500
0.3000
0.2500
0.2000
0.1500
0.1000
0.0500
0.0000
-0.0500
0.0000
20.0000
40.0000
60.0000
80.0000
100.0000
time
packets
skip-1
skip-2
skip-1
acks
skip-1
skip-2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

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)



Source Code:

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

```
set opt(flows) 0 ;# number of long-lived TCP flows
set opt(window) 30 ;# window for long-lived traffic
set opt(web) 2 ;# number of web sessions
# Plotting statistics.
set opt(quiet) 0 ;# popup anything
set opt(wrap) 100 ;# wrap plots
set opt(srcTrace) is ;# where to plot traffic
set opt(dstTrace) bs2 ;# where to plot traffic
set opt(umtsbuf) 10 ; # buffer size for umts
#default downlink bandwidth in bps
set bwDL(umts)  384000
#default uplink bandwidth in bps
set bwUL(umts) 64000
#default downlink propagation delay in seconds
set propDL(umts) .150
#default uplink propagation delay in seconds
set propUL(umts) .150
#default buffer size in packets
set buf(umts) 20
set ns [new Simulator]
set tf [open out.tr w]
$ns trace-all $tf
set nodes(is) [$ns node]
set nodes(ms) [$ns node]
set nodes(bs1) [$ns node]
set nodes(bs2) [$ns node]
set nodes(lp) [$ns node]
proc cell_topo {} {
global ns nodes
$ns duplex-link $nodes(lp) $nodes(bs1) 3Mbps 10ms DropTail
$ns duplex-link $nodes(bs1) $nodes(ms) 1 1 RED
$ns duplex-link $nodes(ms) $nodes(bs2) 1 1 RED
$ns duplex-link $nodes(bs2) $nodes(is) 3Mbps 50ms DropTail
puts "Cell Topology"
}

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

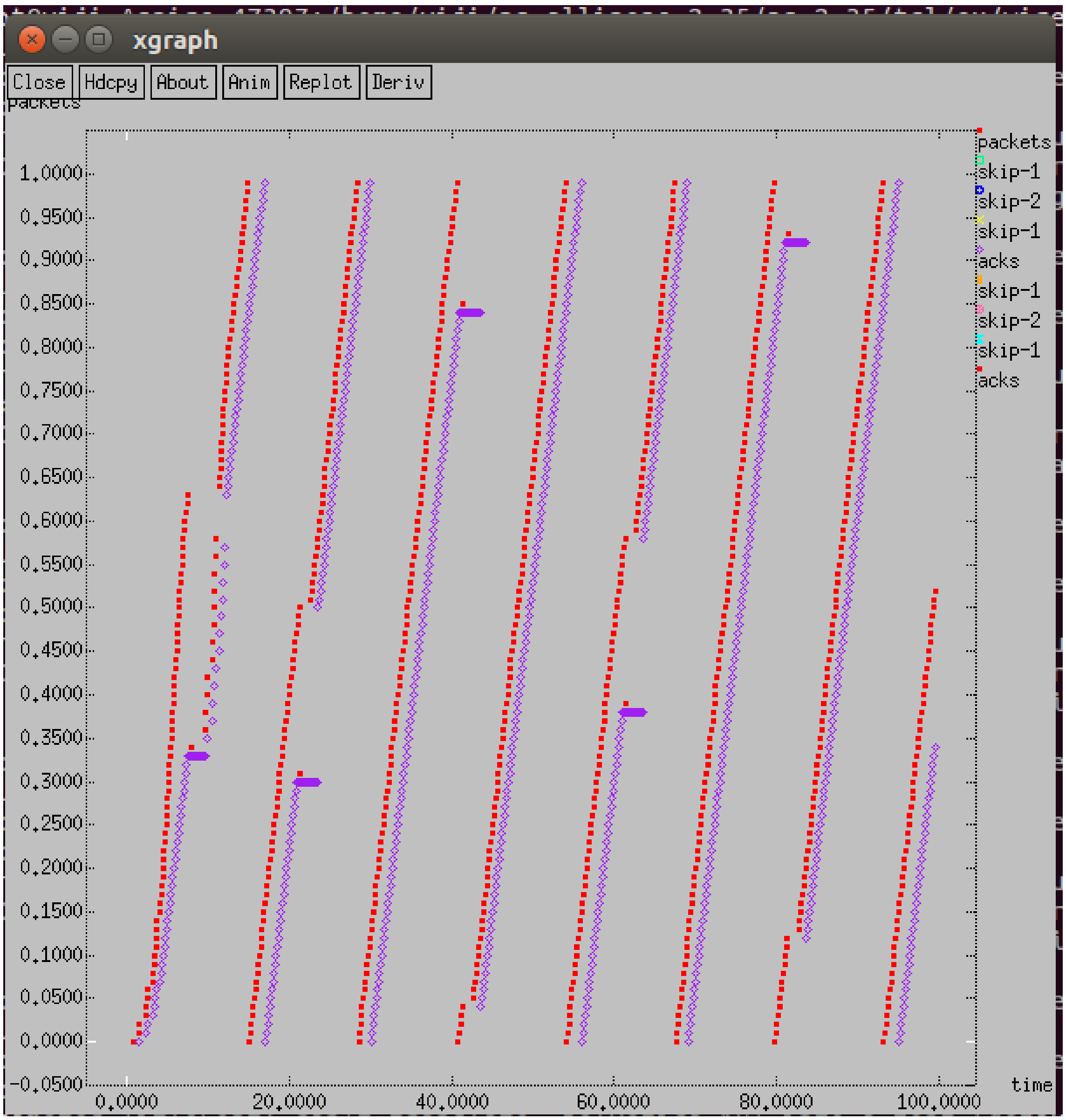
```
$ns bandwidth $nodes(ms) $nodes(bs1) $bwUL($t) simplex
$ns delay $nodes(bs1) $nodes(ms) $propDL($t) simplex
$ns delay $nodes(ms) $nodes(bs1) $propDL($t) simplex
$ns queue-limit $nodes(bs1) $nodes(ms) $buf($t)
$ns queue-limit $nodes(ms) $nodes(bs1) $buf($t)
$ns bandwidth $nodes(bs2) $nodes(ms) $bwDL($t) simplex
$ns bandwidth $nodes(ms) $nodes(bs2) $bwUL($t) simplex
$ns delay $nodes(bs2) $nodes(ms) $propDL($t) simplex
$ns delay $nodes(ms) $nodes(bs2) $propDL($t) simplex
$ns queue-limit $nodes(bs2) $nodes(ms) $buf($t)
$ns queue-limit $nodes(ms) $nodes(bs2) $buf($t)
}
# RED and TCP parameters
Queue/RED set summarystats_ true
Queue/DropTail set summarystats_ true
Queue/RED set adaptive_ $opt(adaptive)
Queue/RED set q_weight_ 0.0
Queue/RED set thresh_ $opt(minth)
Queue/RED set maxthresh_ $opt(maxth)
Queue/DropTail set shrink_drops_ true
Agent/TCP set ecn_ $opt(ecn)
Agent/TCP set window_ $opt(window)
DelayLink set avoidReordering_ true
source web.tcl
#Create topology
switch $opt(type) {
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
```

To Run:

ns lab6.tcl



<u>PART B</u>

<u>IMPLEMENT THE FOLLOWING IN JAVA</u>

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

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 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

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

SOURCE CODE:

```
import java.io.*;
public class CRC {
   public static void main(String args[]) throws IOException
   {
      BufferedReader br=new BufferedReader(new InputStreamReader(System.in));
      int[] data;
      int[] div;
      int[] divisor;
      int[] rem;
      int[] crc;
      int data_bits, divisor_bits, tot_length;
      System.out.println("Enter number of data bits : ");
      data_bits=Integer.parseInt(br.readLine());
      data=new int[data_bits];
      System.out.println("Enter data bits : ");
      for(int i=0; i<data_bits; i++)
```

```
    data[i]=Integer.parseInt(br.readLine());
System.out.println("Enter number of bits in divisor : ");
divisor_bits=Integer.parseInt(br.readLine());
divisor=new int[divisor_bits];
System.out.println("Enter Divisor bits : ");
for(int i=0; i<divisor_bits; i++)
    divisor[i]=Integer.parseInt(br.readLine());
tot_length=data_bits+divisor_bits-1;
div=new int[tot_length];
rem=new int[tot_length];
crc=new int[tot_length];
/*..........................CRC GENERATION..................................*/
System.out.println("data length is "+data.length);
for(int i=0;i<data.length;i++)
    div[i]=data[i];
System.out.print("Dividend (after appending 0's) are : ");
for(int i=0; i< div.length; i++)
    System.out.print(div[i]);
System.out.println();
for(int j=0; j<div.length; j++){
      rem[j] = div[j];
}
rem=divide(div, divisor, rem);
for(int i=0;i<div.length;i++)          // append dividend and remainder
{
    crc[i]=(div[i]^rem[i]);
}
System.out.println();
System.out.println("CRC code : ");
for(int i=0;i<crc.length;i++)
System.out.print(crc[i]);

 /*..........................ERROR DETECTION..............................*/
System.out.println();
System.out.println("Enter CRC code of "+tot_length+" bits : ");
for(int i=0; i<crc.length; i++)
    crc[i]=Integer.parseInt(br.readLine());

 for(int j=0; j<crc.length; j++){
      rem[j] = crc[j];
}
rem=divide(crc, divisor, rem);
for(int i=0; i< rem.length; i++)
{
    if(rem[i]!=0)
```

```
            {
               System.out.println("Data is incorrect!");
               break;
            }
            if (i==rem.length-1)
               System.out.println("Data is correct");
        }
    }
 static int[] divide(int div[],int divisor[], int rem[])
 {
        int cur=0;
        while(true)
        {
            for(int i=0;i<divisor.length;i++)
               rem[cur+i]=(rem[cur+i]^divisor[i]);

            while(rem[cur]==0 && cur!=rem.length-1)
               cur++;

            if((rem.length-cur)<divisor.length)
               break;
        }
        return rem;
}
}
```

OUTPUT

Case 1:

Enter number of bits
16
Enter the data
1
1
1
1
1
1
1
0
0
0
1
0
0

0
1
0
Enter the divisor bits
1
1
0
1
Data length is 16
Dividend after appending 0’s are
1
1
1
1
1
1
1
0
0
0
1
0
0
0
1
0
0
0
0

CRC CODE
1111111000100010101
Enter CRC Code of 19 bits
1
1
1
1
1
1
1
0
0
0
1
0

0
0
1
0
1
0
1
Data is correct

Case-2
Enter number of bits
16
Enter the data
1
1
1
1
1
1
1
0
0
0
1
0
0
0
1
0
Enter the divisor bits
1
1
0
1
Data length is 16
Dividend after appending 0's are
1
1
1
1
1
1
1
0
0

0
1
0
0
0
1
0
0
0
0

CRC CODE
1111111000100010101
Enter CRC Code of 19 bits
1
0
1
1
1
1
1
0
0
0
1
0
0
0
1
0
1
0
1
Data is incorrect

2.Write a program to find the shortest path between vertices using Bellman Ford algorithm

Distance Vector Algorithm is a decentralized routing algorithm that requires that each router simply inform its neighbors of its routing table. For each network path, the receiving routers pick the neighbor advertising the lowest cost, then add this entry into its routing table for re-advertisement. To find the shortest path, Distance Vector Algorithm is based on one of two basic algorithms: the Bellman-Ford and the Dijkstra algorithms.

Routers that use this algorithm have to maintain the distance tables (which is a one-dimension array -- "a vector"), which tell the distances and shortest path to sending packets to each node in the network. The information in the distance table is always up date by exchanging information with the neighboring nodes. The number of data in the table equals to that of all nodes in networks (excluded itself). The columns of table represent the directly attached neighbors whereas the rows represent all destinations in the network. Each data contains the path for sending packets to each destination in the network and distance/or time to transmit on that path (we call this as "cost"). The measurements in this algorithm are the number of hops, latency, the number of outgoing packets, etc.

The Bellman–Ford algorithm is an algorithm that computes shortest paths from a single source vertex to all of the other vertices in a weighted digraph. It is slower than Dijkstra's algorithm for the same problem, but more versatile, as it is capable of handling graphs in which some of the edge weights are negative numbers. Negative edge weights are found in various applications of graphs, hence the usefulness of this algorithm. If a graph contains a "negative cycle" (i.e. a cycle whose edges sum to a negative value) that is reachable from the source, then there is no cheapest path: any path that has a point on the negative cycle can be made cheaper by one more walk around the negative cycle. In such a case, the Bellman–Ford algorithm can detect negative cycles and report their existence

SOURCE CODE:

```
import java.io.* ;
public class FordPgm {
   public static void main(String[] args) throws IOException{
      int i,j,m,k,x,y,min,n;
      int a[][] = new int[10][10] ;
      int h[][] = new int[10][10] ;

      DataInputStream in=new DataInputStream(System.in);
      System.out.println("Enter number of nodes !") ;

      n = Integer.parseInt(in.readLine());

      System.out.println("Enter Distance Matrix") ;
      for (i = 1 ; i<=n; i++)
      {
         for (j=1; j<=n; j++)
         {
            a[i][j] = Integer.parseInt(in.readLine());
            h[i][j] = 0 ;
         }
      }
      /* -  calculate the hop for the nodes */
      for (i = 1 ; i<=n; i++)
      {
         for (j=1; j<=n; j++)
         {
```

```
            if ( a[i][j]!=0 && a[i][j]!=999)
                h[i][j] = 1 ;
        }
    }
    /* - display the node details */
    for (i = 1 ; i<=n; i++)
    {
        System.out.println("The routing table for " + i + "th node is") ;
        System.out.println( "NODE " + "\t" + "DISTANCE" + "\t" + "HOPS") ;
        for (j=1; j<=n; j++)
        {
          System.out.println( " " + j + "\t" + a[i][j] + "\t" + h[i][j]) ;
        }
    }

     System.out.println("After calculation of path") ;
     for (m = 1 ; m<=n; m++)
     {
        for (i=1; i<=n; i++)
        {
            for (j = 1 ; j<=n; j++)
             {
                min = a[i][j] ;
                for (k=1; k<=n; k++)
                {
                    if ( min > a[i][k]+ a[k][j])
                    {
                        a[i][j] = a[i][k] + a[k][j] ;
                        h[i][j] = h[i][k] + h[k][j] ;
                    }
                }
             }
        }
     }

    /*  display the result */
    for (i = 1 ; i<=n; i++)
    {
        System.out.println("The routing table for " + i + "th node is" ) ;
        System.out.println( "NODE " + "\t" + "DISTANCE" + "\t" + "HOPS") ;
        for (j=1; j<=n; j++)
        {
          System.out.println( " " + j + "\t" + a[i][j] + "\t" +"\t"+ h[i][j]) ;
        }
        System.out.println();
```

```
        }
        System.out.println("Enter the node whose shortest path is to be found") ;
        System.out.println(" From ") ;
        x = Integer.parseInt(in.readLine());
        System.out.println(" To ") ;
        y = Integer.parseInt(in.readLine());
        System.out.println("Shortest Path is " +a[x][y] + " with " +h[x][y] + " Hops.") ;

    }
}
```

Output

```
Enter number of nodes !
3
Enter Distance Matrix
0
8
999
999
0
3
4
999
0
The routing table for 1th node is
NODE          DISTANCE    HOPS
 1      0       0
 2      8       1
 3      999     0
The routing table for 2th node is
NODE          DISTANCE    HOPS
 1      999     0
 2      0       0
 3      3       1
The routing table for 3th node is
NODE          DISTANCE    HOPS
 1      4       1
 2      999     0
 3      0       0
After calculation of path
The routing table for 1th node is
NODE          DISTANCE    HOPS
1      0              0
 2      8              1
```

```
 3        11                  2

The routing table for 2th node is
NODE            DISTANCE    HOPS
 1        7                   2
 2        0                   0
 3        3                   1

The routing table for 3th node is
NODE            DISTANCE    HOPS
 1        4                   1
 2        12                  2
 3        0                   0

Enter the node whose shortest path is to be found
 From
1
 To
3
Shortest Path is 11 with 2 Hops.
```

3. 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 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.

CLIENT

```
package client;
import java.io.DataInputStream;
import java.io.DataOutputStream;
import java.net.Socket;
import java.util.Scanner;

public class clientpgm {
    public static void main(String[] args) throws Exception{
        // TODO code application logic here
```

```
 String address ;
Scanner sc=new Scanner(System.in);
System.out.println("Enter Server Address: ");
address=sc.nextLine();
 //create the socket to connect to server port
Socket s = new Socket(address,1108);
DataInputStream din=new DataInputStream(s.getInputStream());
DataOutputStream dout=new DataOutputStream(s.getOutputStream());
//BufferedReader br=new BufferedReader(new InputStreamReader(System.in));
String filename, fdata, serverdata;
System.out.println("Enter File Name: ");
filename=sc.nextLine();
sc.close();

try
{
    System.out.println("Sending File: "+filename);
    dout.writeUTF(filename);
    dout.flush();
     // read from server
    serverdata = din.readUTF();
    if (serverdata.equals("No File"))
    {
        System.out.println("File does not exist!") ;
    }
    else
    {
        System.out.println("File contents are -") ;
        while (true)
        {
            fdata = din.readUTF() ;
            if (fdata.equals("Stop"))
                break ;
            System.out.println(fdata) ;

        }
        System.out.println("File contents are transferred completly");
        din.close();
    }
    dout.close();
    s.close();
}
catch(Exception e)
{
   System.out.println(e.getMessage());
```

```
        }
    }
}
```

SERVER

```
package server;
import java.io.BufferedReader;
import java.io.DataInputStream;
import java.io.DataOutputStream;
import java.io.FileReader;
import java.io.File ;
import java.net.ServerSocket;
import java.net.Socket;

public class srvpgm {
    public static void main(String[] args) throws Exception{
        while (true)
        {
            ServerSocket ss=new ServerSocket(1108);
            System.out.println ("Waiting for request");
            Socket s=ss.accept();
            System.out.println ("Connected With "+s.getInetAddress().toString());
            DataInputStream din=new DataInputStream(s.getInputStream());
            DataOutputStream dout=new DataOutputStream(s.getOutputStream());
            try
            {
                String filename, fdata ;
                filename = din.readUTF();
                System.out.println("Filename received from client is \n" +filename);

                File file = new File(filename);
                if (file.exists())
                {
                    dout.writeUTF("File contents are \n");
                    BufferedReader br = new BufferedReader(new FileReader(filename));
                    while ((fdata = br.readLine()) != null)
                    {
                        dout.writeUTF(fdata) ;
                    }
                    dout.writeUTF("Stop") ;
                    dout.flush() ;
                }
                else
                {
                    dout.writeUTF("No File") ;
```

```
            }
        }
        catch(Exception e)
        {
            System.out.println(e.getMessage());
        }
        din.close();
        dout.close();
        s.close();
        ss.close();
      }
   }
}
```

OUTPUT:

Case 1

SERVER	CLIENT
Waiting for request	Enter the server address
Connected with /127.0.0.1	127.0.0.1
	Enter the file name
	new.txt
	sending file:new.txt
Filename received from client is new.txt	
	File contents are:
	Computer Networks
Waiting for request	

Case 2:

SERVER	CLIENT
Waiting for request	Enter the server address
Connected with /127.0.0.1	127.0.0.1
	Enter the file name
	new.txt
	sending file:file.txt
Filename received from client is file.txt	
	File Doesn’t Exists.
Waiting for request	

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

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

SERVER

```
package server;
import java.io.*;
import java.net.*;
public class srvpgm {
   public static void main(String[] args) throws Exception{
   DatagramSocket serverSocket = new DatagramSocket(9876);
   byte[] receivebuffer = new byte[1024];
     byte[] sendbuffer = new byte[1024];
     System.out.println("Waiting for client request!") ;
     while(true)
       {
         DatagramPacket recvdpkt = new DatagramPacket(receivebuffer, 0, receivebuffer.length);
         serverSocket.receive(recvdpkt);
         InetAddress IP = recvdpkt.getAddress();
         int portno = recvdpkt.getPort();
         System.out.println("Connected to Client " +IP ) ;
         String clientdata = new String(recvdpkt.getData(),
recvdpkt.getOffset(),recvdpkt.getLength());
         System.out.println("\nClient Message: "+ clientdata);
         System.out.print("\nServer sending..: ");
         BufferedReader serverRead = new BufferedReader(new InputStreamReader (System.in) );
         String serverdata = serverRead.readLine();
         sendbuffer = serverdata.getBytes();
         DatagramPacket sendPacket = new DatagramPacket(sendbuffer, sendbuffer.length,
IP,portno);
         serverSocket.send(sendPacket);
         System.out.println("Message sent to client!") ;
         System.out.println() ;
       }
   }
}
```

CLIENT

```
package client;
import java.io.*;
import java.net.*;
public class clientpgm {
  public static void main(String[] args) throws Exception{
    BufferedReader clientRead =new BufferedReader(new InputStreamReader(System.in));
    DatagramSocket clientSocket = new DatagramSocket();
    InetAddress IP = InetAddress.getByName("127.0.0.1");
    byte[] sendbuffer = new byte[1024];
    byte[] receivebuffer = new byte[1024];
    System.out.print("\nClient: ");
    String clientData = clientRead.readLine();
    sendbuffer = clientData.getBytes();
    DatagramPacket sendPacket =
    new DatagramPacket(sendbuffer, sendbuffer.length, IP, 9876);
    clientSocket.send(sendPacket);
    System.out.println("Waiting for Server response!") ;
    DatagramPacket receivePacket =
    new DatagramPacket(receivebuffer, receivebuffer.length);
    clientSocket.receive(receivePacket);
    String serverData = new String(receivePacket.getData());
    System.out.println("\nServer Message received: " + serverData);
    System.out.println("Client terminated.") ;
    clientSocket.close();
  }
}
```

OUTPUT:

SERVER END	**CLIENT END**
Waiting for Client request	Client: Hello I am client Waiting for server response!
Connected to client 127.0.0.1 Client message: Hello I am client Hello I am server Message sent to client!	
	Server message received hello I am server

Client Terminated.

5. 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 mod 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 mod n.
2. Extracts the plaintext from the integer representative m.

SOURCE CODE:

```
import java.math.BigInteger;
import java.util.Random;
import java.io.*;

public class RSA {
        BigInteger p, q, N, phi, e, d;
        int bitlength = 100;
```

```
        Random r;

      public RSA() {
                r = new Random();
                // returns a prime number with in the specified bit length
                p = BigInteger.probablePrime(bitlength, r);
                q = BigInteger.probablePrime(bitlength, r);
                N = p.multiply(q);

                // In c equivalent – phi = (p-1)*(q-1)
                phi = p.subtract(BigInteger.ONE).multiply(q.subtract(BigInteger.ONE));
                e = BigInteger.probablePrime(bitlength/2, r);

          // In C equivalent - while ( phi.gcd(e) > 1 && e < phi )
        while (phi.gcd(e).compareTo(BigInteger.ONE) > 0 && e.compareTo(phi) < 0 ) {
                        e.add(BigInteger.ONE);  // In C equivalent- e+1
                }

                d = e.modInverse(phi);
     }

    // RSA Main
    public static void main(String[] args) throws IOException{

      RSA rsa = new RSA();
        DataInputStream in=new DataInputStream(System.in);
        String teststring ;

      System.out.println("Enter the plain text:");

        teststring=in.readLine();

        System.out.println("Encrypting String: " + teststring);
        System.out.println("String in Bytes: " + bytesToString(teststring.getBytes()));
         // encrypt
        byte[] encrypted = rsa.encrypt(teststring.getBytes());
        System.out.println("Encrypted String in Bytes: " + bytesToString(encrypted));

         // decrypt
        byte[] decrypted = rsa.decrypt(encrypted);
        System.out.println("Decrypted String in Bytes: " + bytesToString(decrypted));
        System.out.println("Decrypted String: ");
        System.out.println(new String(decrypted));
    }
```

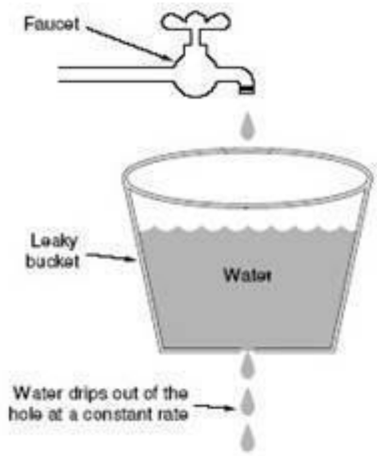
```
    private static String bytesToString(byte[] encrypted) {
        String test = "";
        for (byte b : encrypted) {
            test += Byte.toString(b);
        }
        return test;
    }

    // Encrypt message
    public byte[] encrypt(byte[] message) {
        return (new BigInteger(message)).modPow(e, N).toByteArray();
    }

    // Decrypt message
    public byte[] decrypt(byte[] message) {
        return (new BigInteger(message)).modPow(d, N).toByteArray();
    }
}
```

OUTPUT

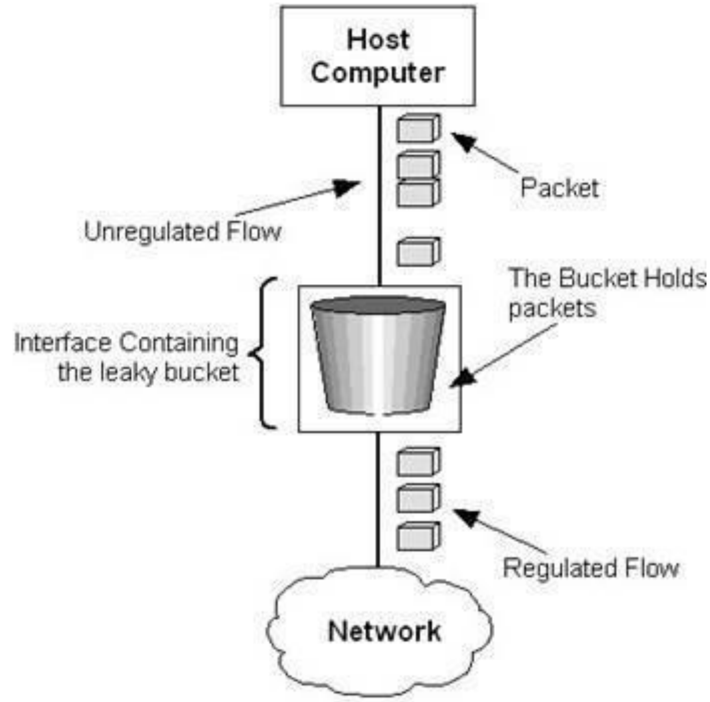
Enter the plain text:
Sapthagiri
Encrypting String: Sapthagiri
String in Bytes: 8397112116104971031051114105
Encrypted String in Bytes: 91-2-10491-456154-32-71-122-35-187884101-3125-1856102-17356-582
Decrypted String in Bytes: 8397112116104971031051114105
Decrypted String:
Sapthagiri

6. 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.

SOURCE CODE:

```
import java.io.*;
public class Leakybucket {
   public static void main(String[] args) throws IOException {
        int b_size, out_rate, p_size,discard, cnt, outcnt, clkcnt=0;
        BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
        System.out.println("Enter the bucket size") ;
        b_size=Integer.parseInt(br.readLine());
        System.out.println("Enter the output rate");
       out_rate = Integer.parseInt(br.readLine());
       System.out.println("Enter the number of packets") ;
       p_size = Integer.parseInt(br.readLine());
       discard = b_size - p_size ;
       if (discard < 0)
       {
          discard = p_size - b_size ;
          p_size = b_size ;
          System.out.println("Packets are more so " +discard+" packets are discarded");
       }
```

```
        System.out.println("Input packet");
        int packets[] = new int[p_size];
        for (cnt =0; cnt<p_size; cnt++)
        {
           packets[cnt] = Integer.parseInt(br.readLine());
        }
              for (cnt=0; cnt<p_size; )
        {
           System.out.println("At time tick "+(clkcnt++));
           for (outcnt=0; outcnt<out_rate; outcnt++)
           {
              if (cnt < p_size)
              {
                 System.out.print("packet " +packets[cnt++] + " drained out");
                 System.out.println();
              }
              else
                 break ;

           }
        }
    }
```

Output:

Case 1:

Enter the bucket size
5
Enter the output rate
2
Enter the number of packets
7
Packets are more so 2 packets are discarded
Input packet
10
20
30
40
50
60
70
Packets are more so 2 packets are discarded
At time tick 0
packet 10 drained out
packet 20 drained out

At time tick 1
packet 30 drained out
packet 40 drained out
At time tick 2
packet 50 drained out

Case 2:
Enter the bucket size
7
Enter the output rate
2
Enter the number of packets
5
Input packet
10
20
30
40
50
At time tick 0
packet 10 drained out
packet 20 drained out
At time tick 1
packet 30 drained out
packet 40 drained out
At time tick 2
packet 50 drained out

Case 3:
Enter the bucket size
5
Enter the output rate
2
Enter the number of packets
5
Input packet
10
20
30
40
50
At time tick 0
packet 10 drained out
packet 20 drained out
At time tick 1

packet 30 drained out
packet 40 drained out
At time tick 2
packet 50 drained out