Part-A

Introduction to NS-2:

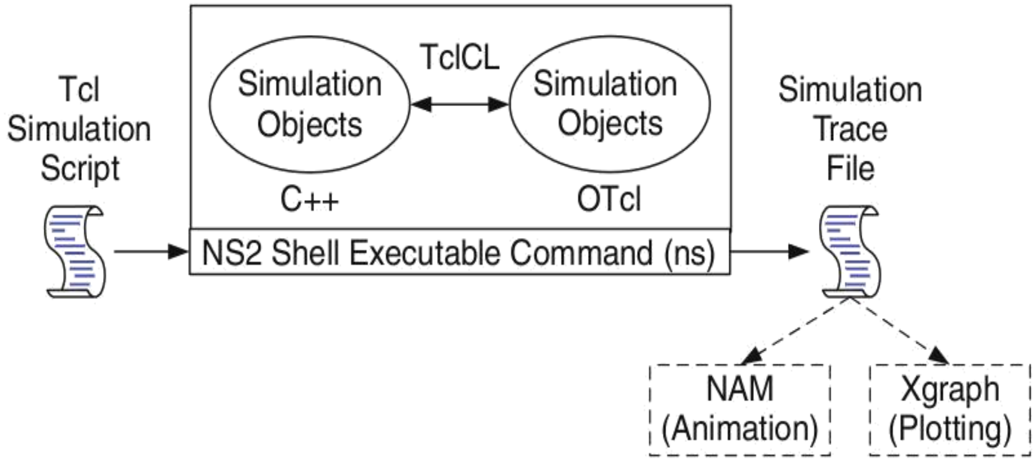
Widely known as NS2, is simply an event driven simulation tool.

Useful in studying the dynamic nature of communication networks.

Simulation of wired as well as wireless network functions and protocols (e.g., routing algorithms, TCP, UDP) can be done using NS2.

In general, NS2 provides users with a way of specifying such network protocols and simulating their corresponding behaviors.

Basic Architecture of NS2



Tcl scripting

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

Basics of TCL

Syntax: command arg1 arg2 arg3

Hello World!

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

Variables Command Substitution

```
set a 5        set len [string length foobar]
```

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

Simple Arithmetic

```
expr 7.2 / 4
```

Procedures

```
    proc Diag {a b} {
    set c [expr sqrt($a * $a + $b * $b)]
        return $c }
```

puts —Diagonal of a 3, 4 right triangle is [Diag 3 4]
Output: Diagonal of a 3, 4 right triangle is 5.0

Loops

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

---------                              ----------
}
```

Wired TCL Script Components

Create the event scheduler
Open new files & turn on the tracing
Create the nodes
Setup the links
Configure the traffic type (e.g., TCP, UDP, etc)
Set the time of traffic generation (e.g., CBR, FTP)
Terminate the simulation

NS Simulator Preliminaries.

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

Initialization and Termination of TCL Script in NS-2

An ns simulation starts with the command

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

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

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

#Open the Trace file

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

#Open the NAM trace file

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

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

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

at by the pointer "$namfile",i.e the file "out.tr".

The termination of the program is done using a ―finish‖ procedure

#Define a finish‟ procedure

```
Proc finish { } {

global ns tracefile1 namfile

$ns flush-trace

Close $tracefile1

Close $namfile

Exec nam out.nam &

Exit 0
```

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

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

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

will be used to call finish at time 125sec. Indeed,the at method of the simulator allows us to schedule events The simulation can then begin using the command

```
$ns run
```

Definition of a network of links and nodes

The way to define a node is

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

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

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

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

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

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

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

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

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

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

Agents and Applications

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

FTP over TCP

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

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

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

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

The command

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

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

#Setup a UDP connection

```
set udp [new Agent/UDP]

$ns attach-agent $n1 $udp

set null [new Agent/Null]

$ns attach-agent $n5 $null

$ns connect $udp $null

$udp set fid_2
```

#setup a CBR over UDP connection

The below shows the definition of a CBR application using a UDP agent

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

```
set cbr [new Application/Traffic/CBR]

$cbr attach-agent $udp

$cbr set packetsize_ 100

$cbr set rate_ 0.01Mb

$cbr set random_ false
```

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

$tcp set packetSize_ 552.

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

CBR over UDP

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

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

```
$cbr set interval_ 0.005
```

The packet size can be set to some value using

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

Scheduling Events

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

```
$ns at <time> <event>
```

The scheduler is started when running ns that is through the command $ns run.

The beginning and end of the FTP and CBR application can be done through the following command

```
$ns at 0.1 “$cbr start”

$ns at 1.0 “ $ftp start”

$ns at 124.0 “$ftp stop”

$ns at 124.5 “$cbr stop”
```

Structure of Trace Files

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

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

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

XGRAPH

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

Syntax:

```
Xgraph [options] file-name
```

Options are listed here

/-bd <color> (Border)

This specifies the border color of the xgraph window.

/-bg <color> (Background)

This specifies the background color of the xgraph window.

/-fg<color> (Foreground)

This specifies the foreground color of the xgraph window.

/-lf <fontname> (LabelFont)

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

/-t<string> (Title Text)

This string is centered at the top of the graph.

/-x <unit name> (XunitText)

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

/-y <unit name> (YunitText)

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

Awk- An Advanced

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

Syntax:

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

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

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

Variables

Awk allows the user to use variables of 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 {
kount =kount+1
printf “ %3f %20s %-12s %d\n”, kount,$2,$3,$6 }‟ empn.lst

THE –f OPTION: STORING awk PROGRAMS IN A 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 = "|"}

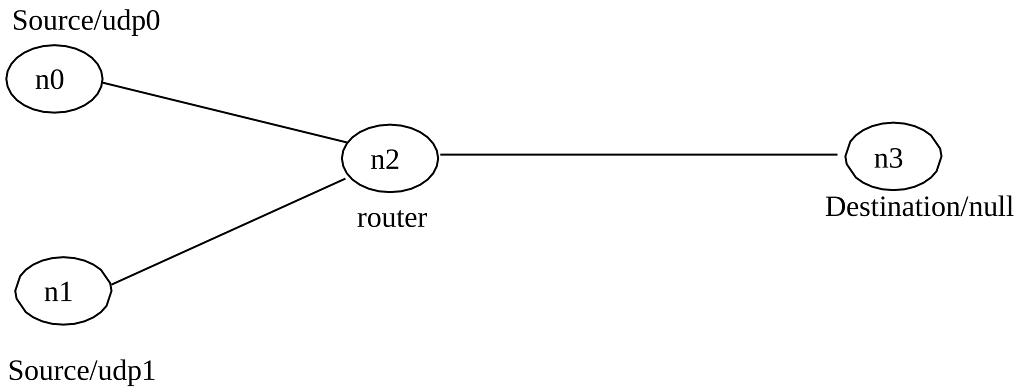
NF! =6 {

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

Lab Experiment 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.

Topology-



Code –

```
#Create Simulator object
set ns [new Simulator]

#Open trace file
set nt [open lab1.tr w]
$ns trace-all $nt

#Open namtrace file
set nf [open lab1.nam w]
$ns namtrace-all $nf

#Create nodes
set n0 [$ns node]
set n1 [$ns node]
set n2 [$ns node]
set n3 [$ns node]

#Assign color to the packet
$ns color 1 Blue
$ns color 2 Red

#label nodes
$n0 label "Source/udp0"
$n1 label "Source/udp1"
$n2 label "Router"
$n3 label "Destination/null"

#create links, specify the type, nodes, bandwidth, delay and ARQ algorithm for it
$ns duplex-link $n0 $n2 10Mb 300ms DropTail
$ns duplex-link $n1 $n2 10Mb 300ms DropTail
```

```
 $ns duplex-link $n2 $n3 100Kb 300ms DropTail

 #set queue size between the nodes
 $ns queue-limit $n0 $n2 10
 $ns queue-limit $n1 $n2 10
 $ns queue-limit $n2 $n3 5

#create and attach UDP agent to n0, n1 and Null agent to n3
set udp0 [new Agent/UDP]
$ns attach-agent $n0 $udp0

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


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

 #attach Application cbr to udp
 set cbr0 [new Application/Traffic/CBR]
 $cbr0 attach-agent $udp0

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

 #set udp0 packet to red color and udp1 packet to blue color
 $udp0 set class_ 1
 $udp1 set class_ 2

 #connect the agents
 $ns connect $udp0 $null3
 $ns connect $udp1 $null3

 #set packet size and interval for cbr1
 $cbr1 set packetSize_ 500Mb
 $cbr1 set interval_ 0.005

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

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

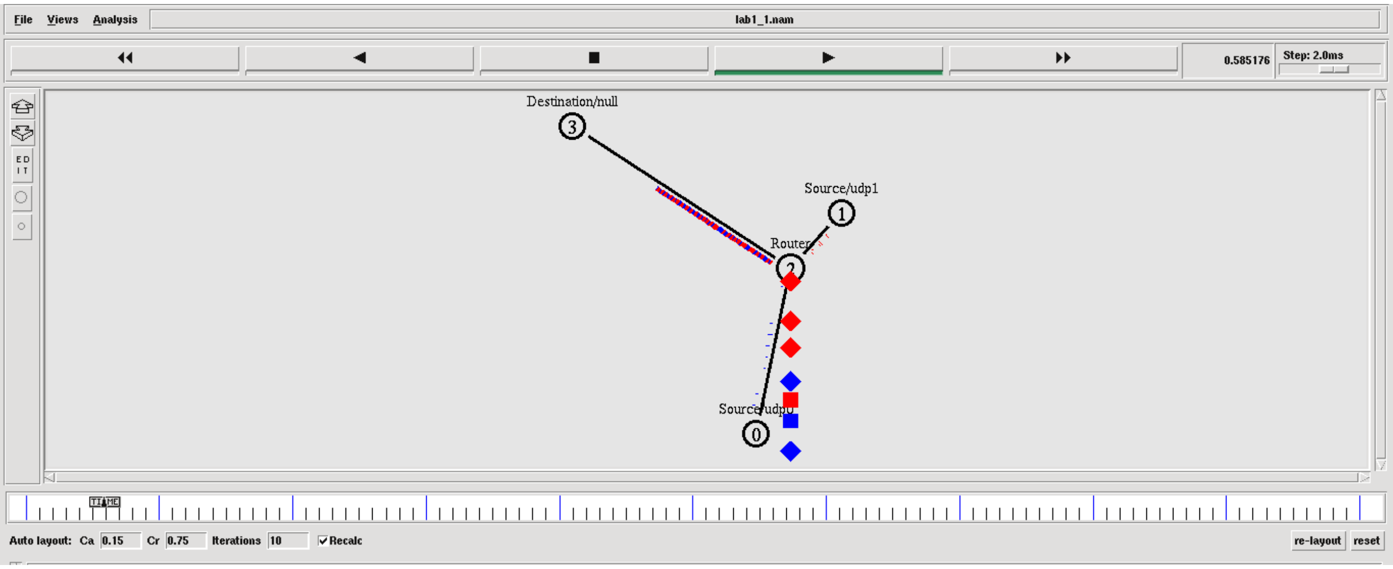
Awk file-

```
BEGIN { count=0;
}
{
        if($1=="d")
        count++
}
END{
        printf("Number of packets dropped is = %d\n",count);
}
```

Output- Trace file (lab1.tr) needs to checked to see the data transfer

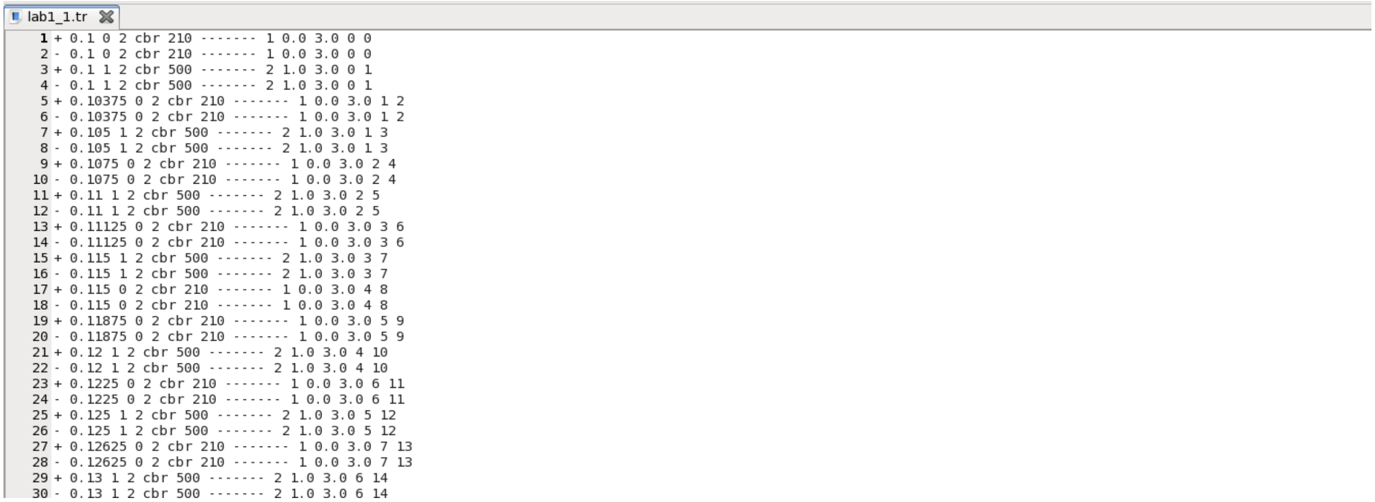
```
$ns lab1.tcl
$awk -f numDrop.awk lab1.tr
Number of packets dropped due to congestion is = 714
```

Simulation-



Auto layout: Ca 0.15 Cr 0.75 Iterations 10 ☑ Recalc

Trace File-

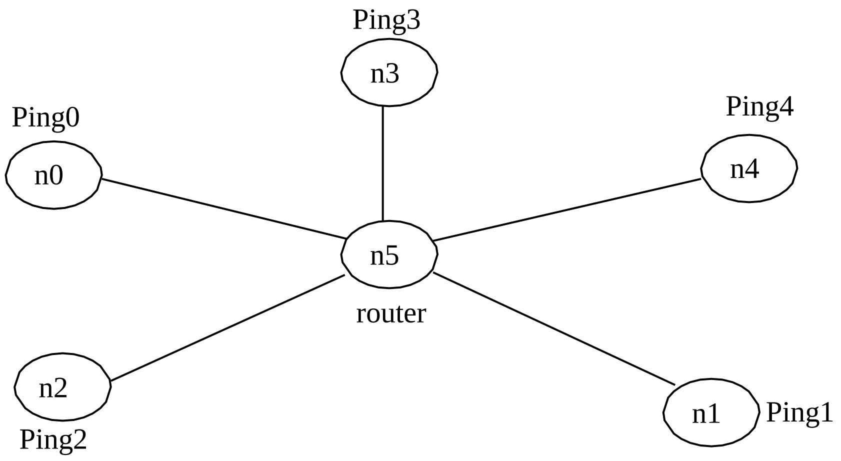


```
+ 0.1 0 2 cbr 210 ------- 1 0.0 3.0 0 0
- 0.1 0 2 cbr 210 ------- 1 0.0 3.0 0 0
+ 0.1 1 2 cbr 500 ------- 2 1.0 3.0 0 1
- 0.1 1 2 cbr 500 ------- 2 1.0 3.0 0 1
+ 0.10375 0 2 cbr 210 ------- 1 0.0 3.0 1 2
- 0.10375 0 2 cbr 210 ------- 1 0.0 3.0 1 2
+ 0.105 1 2 cbr 500 ------- 2 1.0 3.0 1 3
- 0.105 1 2 cbr 500 ------- 2 1.0 3.0 1 3
+ 0.1075 0 2 cbr 210 ------- 1 0.0 3.0 2 4
- 0.1075 0 2 cbr 210 ------- 1 0.0 3.0 2 4
+ 0.11 1 2 cbr 500 ------- 2 1.0 3.0 2 5
- 0.11 1 2 cbr 500 ------- 2 1.0 3.0 2 5
+ 0.11125 0 2 cbr 210 ------- 1 0.0 3.0 3 6
- 0.11125 0 2 cbr 210 ------- 1 0.0 3.0 3 6
+ 0.115 1 2 cbr 500 ------- 2 1.0 3.0 3 7
- 0.115 1 2 cbr 500 ------- 2 1.0 3.0 3 7
+ 0.115 0 2 cbr 210 ------- 1 0.0 3.0 4 8
- 0.115 0 2 cbr 210 ------- 1 0.0 3.0 4 8
+ 0.11875 0 2 cbr 210 ------- 1 0.0 3.0 5 9
- 0.11875 0 2 cbr 210 ------- 1 0.0 3.0 5 9
+ 0.12 1 2 cbr 500 ------- 2 1.0 3.0 4 10
- 0.12 1 2 cbr 500 ------- 2 1.0 3.0 4 10
+ 0.1225 0 2 cbr 210 ------- 1 0.0 3.0 6 11
- 0.1225 0 2 cbr 210 ------- 1 0.0 3.0 6 11
+ 0.125 1 2 cbr 500 ------- 2 1.0 3.0 5 12
- 0.125 1 2 cbr 500 ------- 2 1.0 3.0 5 12
+ 0.12625 0 2 cbr 210 ------- 1 0.0 3.0 7 13
- 0.12625 0 2 cbr 210 ------- 1 0.0 3.0 7 13
+ 0.13 1 2 cbr 500 ------- 2 1.0 3.0 6 14
- 0.13 1 2 cbr 500 ------- 2 1.0 3.0 6 14
```

Lab Experiment 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.

Topology-



Code-

```
#create Simulator object
set ns [new Simulator]

#open trace file
set nt [open prac2.tr w]
$ns trace-all $nt

#open namtrace file
set nf [open prac2.nam w]
$ns namtrace-all $nf


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

#label nodes
$n0 label "ping0"
$n1 label "ping1"
$n2 label "ping2"
$n3 label "ping3"
$n4 label "ping4"
$n5 label "router"
```

```
#create links, specify the type, nodes, bandwidth, delay and ARQ algorithm for it
$ns duplex-link $n0 $n5 1Mb 10ms DropTail
$ns duplex-link $n1 $n5 1Mb 10ms DropTail
$ns duplex-link $n2 $n5 1Mb 10ms DropTail
$ns duplex-link $n3 $n5 1Mb 10ms DropTail
$ns duplex-link $n4 $n5 1Mb 10ms DropTail

#set queue length
$ns queue-limit $n0 $n5 5
$ns queue-limit $n1 $n5 5
$ns queue-limit $n2 $n5 2
$ns queue-limit $n3 $n5 5
$ns queue-limit $n4 $n5 2

$ns color 2 Red
$ns color 3 Blue
$ns color 4 Green
$ns color 5 Yellow

#define 'recv' function for class Agent/Ping
Agent/Ping instproc recv {from rtt} {
        $self instvar node_
        puts "node [$node_ id] received ping answer from $from with round-trip time $rtt ms"
}

#create ping agent and attach them to node
set p0 [new Agent/Ping]
$ns attach-agent $n0 $p0
$p0 set class_ 1

set p1 [new Agent/Ping]
$ns attach-agent $n1 $p1
$p1 set class_ 2

set p2 [new Agent/Ping]
$ns attach-agent $n2 $p2
$p2 set class_ 3

set p3 [new Agent/Ping]
$ns attach-agent $n3 $p3
$p3 set class_ 4

set p4 [new Agent/Ping]
$ns attach-agent $n4 $p4
$p4 set class_ 5
#connect 2 agents
$ns connect $p2 $p4
$ns connect $p3 $p4

proc sendPingPacket { } {
        global ns p2 p3
```

```
                set intervalTime 0.001
                set now [$ns now]
                $ns at [expr $now + $intervalTime] "$p2 send"
                $ns at [expr $now + $intervalTime] "$p3 send"
                $ns at [expr $now + $intervalTime] "sendPingPacket"
        }

        proc finish { } {
                global ns nt nf
                $ns flush-trace

                close $nt
                close $nf
                exec nam prac2.nam &
                exit 0
        }

        $ns at 0.1 "sendPingPacket"
        $ns at 2.0 "finish"
        $ns run
```

Awk file-

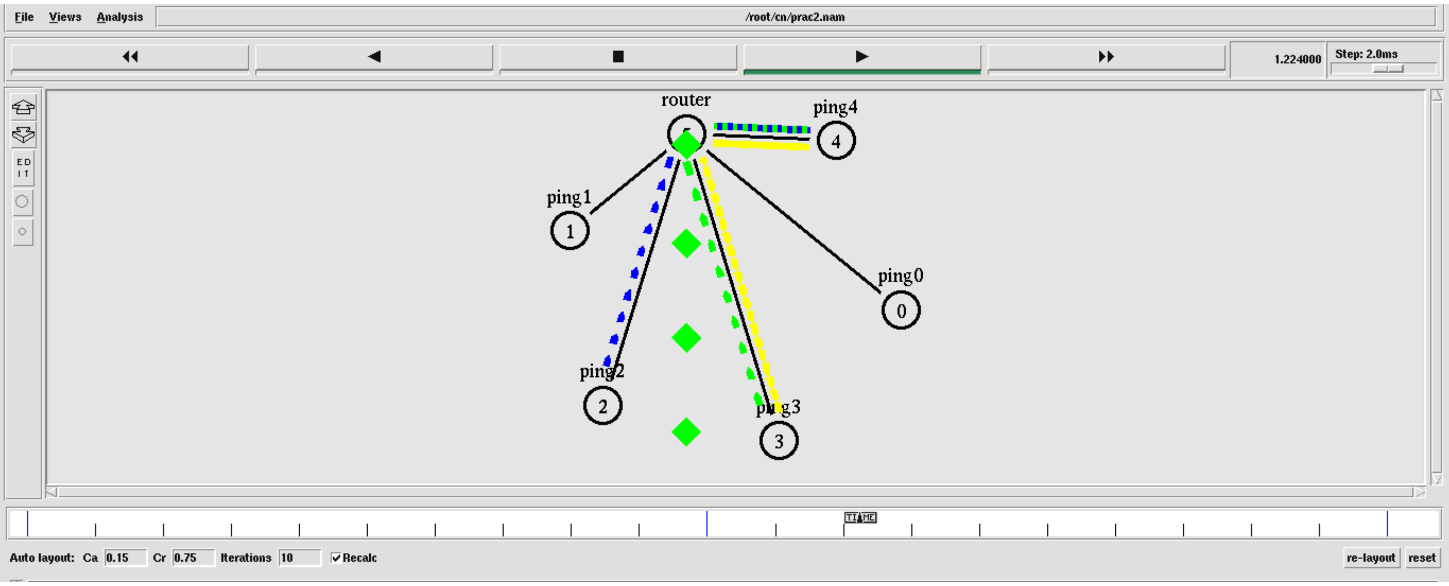
```
        BEGIN{
                count=0;
        }
        {
                if($1=="d")
                count++
        }
        END{
                printf ("Number of packets dropped is = %d\n",count);
        }
```

Output-

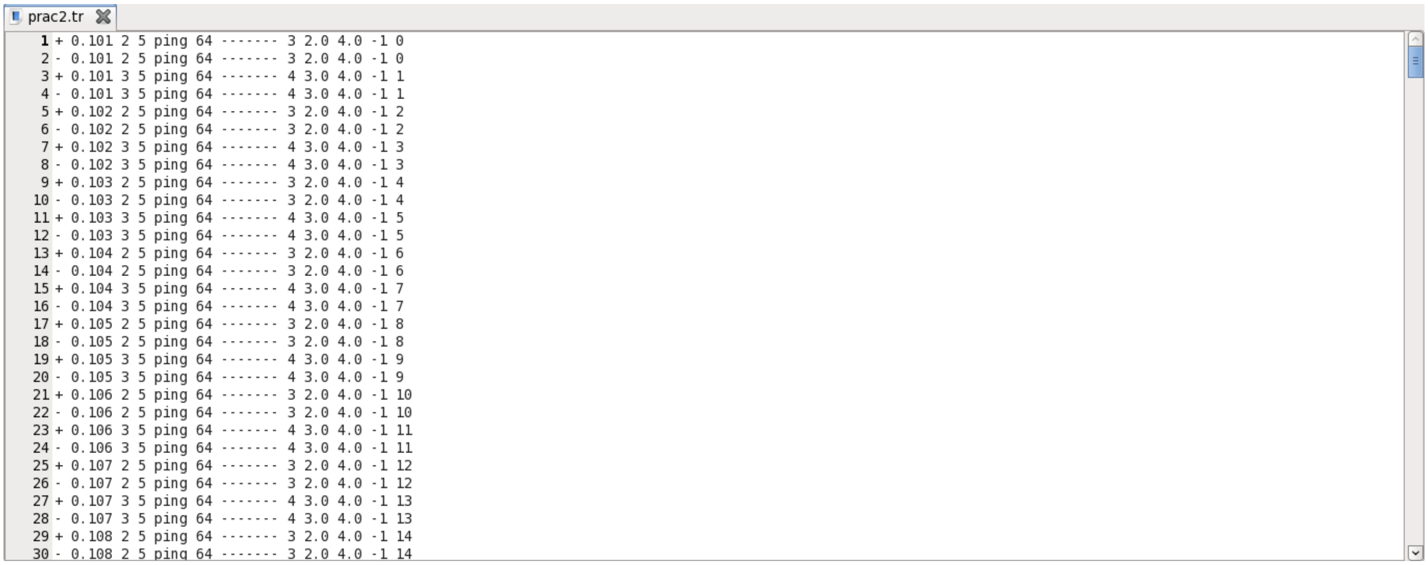
```
        $ns lab2.tcl
        node 3 received ping answer from 4 with round-trip time 66.3 ms
        node 3 received ping answer from 4 with round-trip time 66.8 ms
        node 3 received ping answer from 4 with round-trip time 66.3 ms
        node 3 received ping answer from 4 with round-trip time 66.9 ms
        node 3 received ping answer from 4 with round-trip time 66.4 ms
        node 3 received ping answer from 4 with round-trip time 66.9 ms
        node 3 received ping answer from 4 with round-trip time 66.4 ms
        node 3 received ping answer from 4 with round-trip time 66.9 ms
        node 3 received ping answer from 4 with round-trip time 66.4 ms
        node 3 received ping answer from 4 with round-trip time 66.9 ms
        node 3 received ping answer from 4 with round-trip time 66.4 ms
        node 3 received ping answer from 4 with round-trip time 67.0 ms
        node 3 received ping answer from 4 with round-trip time 66.5 ms
        node 3 received ping answer from 4 with round-trip time 67.0 ms
        node 3 received ping answer from 4 with round-trip time 66.5 ms
        node 3 received ping answer from 4 with round-trip time 67.0 ms
        node 3 received ping answer from 4 with round-trip time 66.5 ms
```

```
$awk -f numDrop.awk prac2.tr
Number of packets dropped is = 41
```

Simulation-



Trace File-

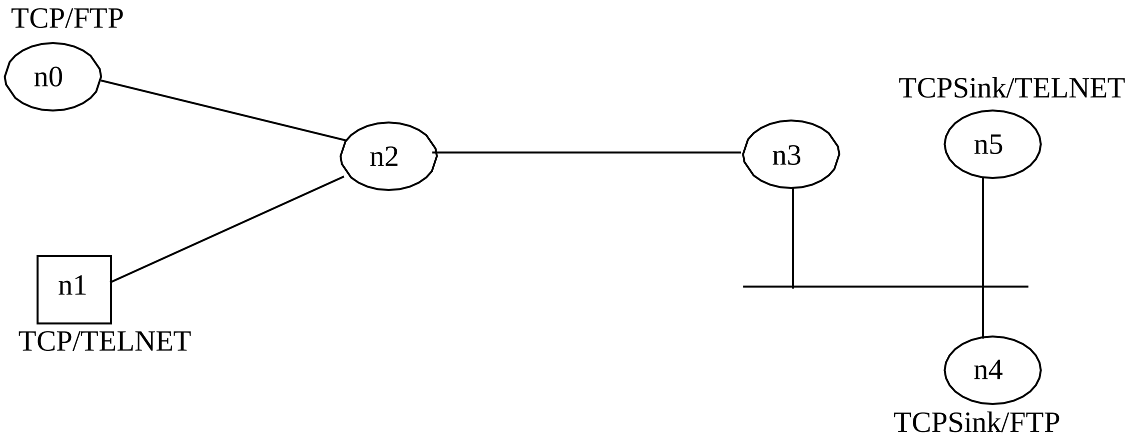


```
+ 0.101 2 5 ping 64 ------- 3 2.0 4.0 -1 0
- 0.101 2 5 ping 64 ------- 3 2.0 4.0 -1 0
+ 0.101 3 5 ping 64 ------- 4 3.0 4.0 -1 1
- 0.101 3 5 ping 64 ------- 4 3.0 4.0 -1 1
+ 0.102 2 5 ping 64 ------- 3 2.0 4.0 -1 2
- 0.102 2 5 ping 64 ------- 3 2.0 4.0 -1 2
+ 0.102 3 5 ping 64 ------- 4 3.0 4.0 -1 3
- 0.102 3 5 ping 64 ------- 4 3.0 4.0 -1 3
+ 0.103 2 5 ping 64 ------- 3 2.0 4.0 -1 4
- 0.103 2 5 ping 64 ------- 3 2.0 4.0 -1 4
+ 0.103 3 5 ping 64 ------- 4 3.0 4.0 -1 5
- 0.103 3 5 ping 64 ------- 4 3.0 4.0 -1 5
+ 0.104 2 5 ping 64 ------- 3 2.0 4.0 -1 6
- 0.104 2 5 ping 64 ------- 3 2.0 4.0 -1 6
+ 0.104 3 5 ping 64 ------- 4 3.0 4.0 -1 7
- 0.104 3 5 ping 64 ------- 4 3.0 4.0 -1 7
+ 0.105 2 5 ping 64 ------- 3 2.0 4.0 -1 8
- 0.105 2 5 ping 64 ------- 3 2.0 4.0 -1 8
+ 0.105 3 5 ping 64 ------- 4 3.0 4.0 -1 9
- 0.105 3 5 ping 64 ------- 4 3.0 4.0 -1 9
+ 0.106 2 5 ping 64 ------- 3 2.0 4.0 -1 10
- 0.106 2 5 ping 64 ------- 3 2.0 4.0 -1 10
+ 0.106 3 5 ping 64 ------- 4 3.0 4.0 -1 11
- 0.106 3 5 ping 64 ------- 4 3.0 4.0 -1 11
+ 0.107 2 5 ping 64 ------- 3 2.0 4.0 -1 12
- 0.107 2 5 ping 64 ------- 3 2.0 4.0 -1 12
+ 0.107 3 5 ping 64 ------- 4 3.0 4.0 -1 13
- 0.107 3 5 ping 64 ------- 4 3.0 4.0 -1 13
+ 0.108 2 5 ping 64 ------- 3 2.0 4.0 -1 14
- 0.108 2 5 ping 64 ------- 3 2.0 4.0 -1 14
```

Lab Experiment 3:

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

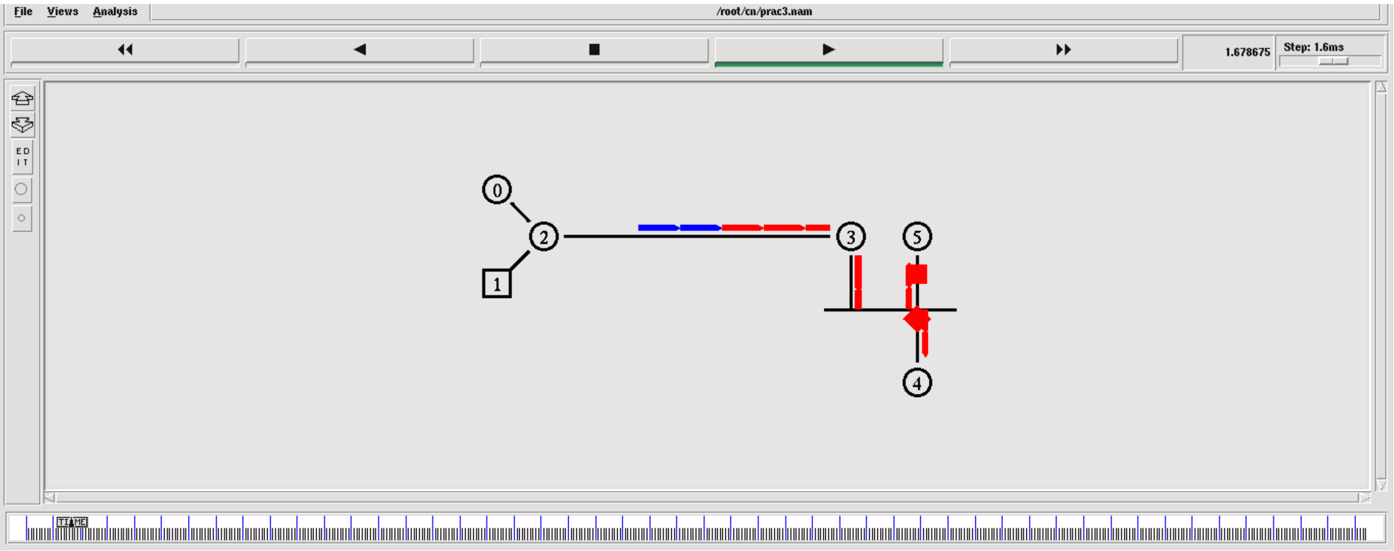
Topology-



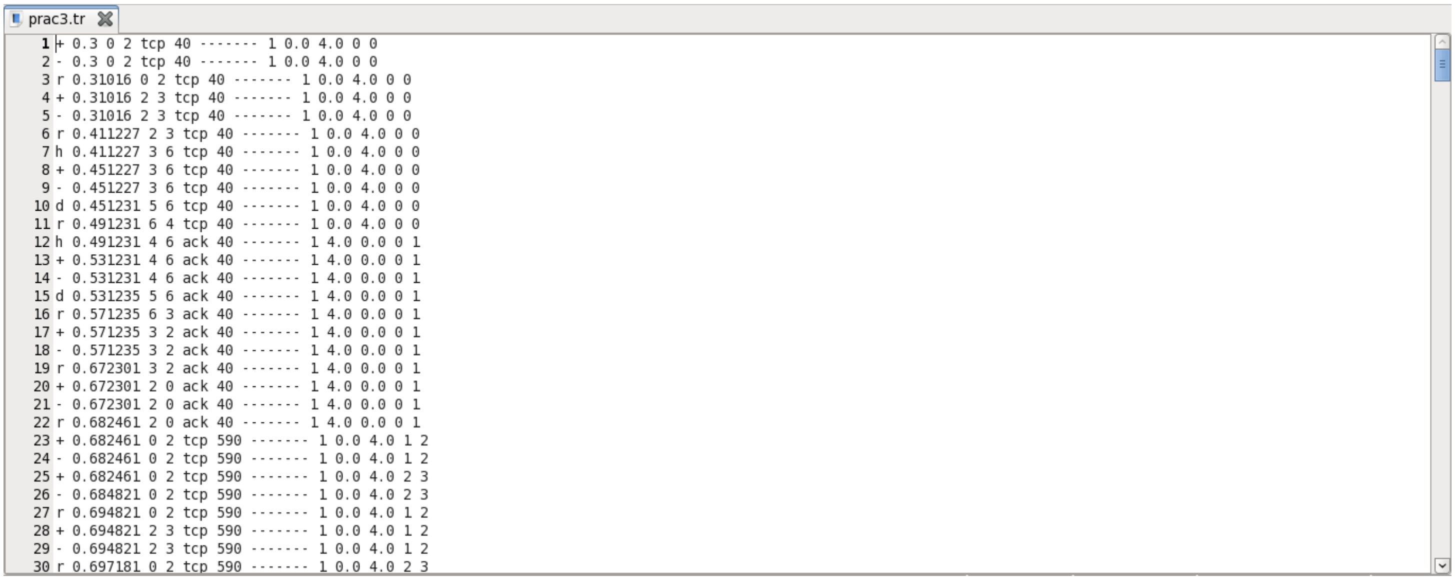
Code-

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

#define color for data flow
$ns color 1 Blue
$ns color 2 Red

#open trace file
set tracefile1 [open lab3.tr w]
set winfile [open winfile w]
$ns trace-all $tracefile1

#open namtrace file
set namfile [open lab3.nam w]
$ns namtrace-all $namfile

#define finish procedure
proc finish { } {
        global ns tracefile1 namfile
        $ns flush-trace
        close $tracefile1
        close $namfile
        exec nam lab3.nam &
        exit 0
}

#create 6 nodes
set n0 [$ns node]
set n1 [$ns node]
set n2 [$ns node]
set n3 [$ns node]
set n4 [$ns node]
set n5 [$ns node]

$n1 shape box
#create link  between nodes
$ns duplex-link $n0 $n2 2Mb 10ms DropTail
$ns duplex-link $n1 $n2 2Mb 10ms DropTail
$ns simplex-link $n2 $n3 0.3Mb 100ms DropTail
$ns simplex-link $n3 $n2 0.3Mb 100ms DropTail

set lan [$ns newLan "$n3 $n4 $n5" 0.5Mb 40ms LL Queue/DropTail MAC/802_3]

#give node position
$ns duplex-link-op $n0 $n2 orient right-down
$ns duplex-link-op $n1 $n2 orient right-up
$ns simplex-link-op $n3 $n2 orient left
$ns simplex-link-op $n2 $n3 orient right

#set queue size of link(n2-n3)
$ns queue-limit $n2 $n3 20

#setup tcp connection
set tcp [new Agent/TCP]
$ns attach-agent $n0 $tcp
set sink [new Agent/TCPSink]
$ns attach-agent $n4 $sink
$ns connect $tcp $sink
$tcp set fid_ 1
$tcp set packetSize_ 552

#set ftp over tcp connection
set ftp [new Application/FTP]
$ftp attach-agent $tcp

#setup a TCP1 connection
set tcp1 [new Agent/TCP]
$ns attach-agent $n1 $tcp1
set sink1 [new Agent/TCPSink]
$ns attach-agent $n5 $sink1
$ns connect $tcp1 $sink1
$tcp1 set fid_ 2
$tcp1 set packetSize_ 552

set telnet0 [new Application/Telnet]
$telnet0 attach-agent $tcp1

#title congestion window1
set outfile1 [open congestion1.xg w]
puts $outfile1 "TitleText: Congestion Window-- Source _tcp"
puts $outfile1 "xUnitText: Simulation Time(Secs)"
puts $outfile1 "yUnitText: Congestion WindowSize"

#title congestion window2
set outfile2 [open congestion2.xg w]
puts $outfile2 "TitleText: Congestion Window-- Source _tcp1"
puts $outfile2 "xUnitText: Simulation Time(Secs)"
puts $outfile2 "yUnitText: Congestion WindowSize"

proc plotWindow {tcpSource outfile} {
        global ns
        set time 0.1
        set now [$ns now]
        set cwnd [$tcpSource set cwnd_]
        puts $outfile "$now $cwnd"
        $ns at [expr $now+$time] "plotWindow $tcpSource $outfile"
}

$ns at 0.1 "plotWindow $tcp $winfile"
$ns at 0.0 "plotWindow $tcp $outfile1"
$ns at 0.1 "plotWindow $tcp1 $outfile2"
$ns at 0.3 "$ftp start"
$ns at 0.5 "$telnet0 start"
$ns at 49.0 "$ftp stop"

$ns at 49.1 "$telnet0 stop"
$ns at 50.0 "finish"

$ns run
```

Simulation-



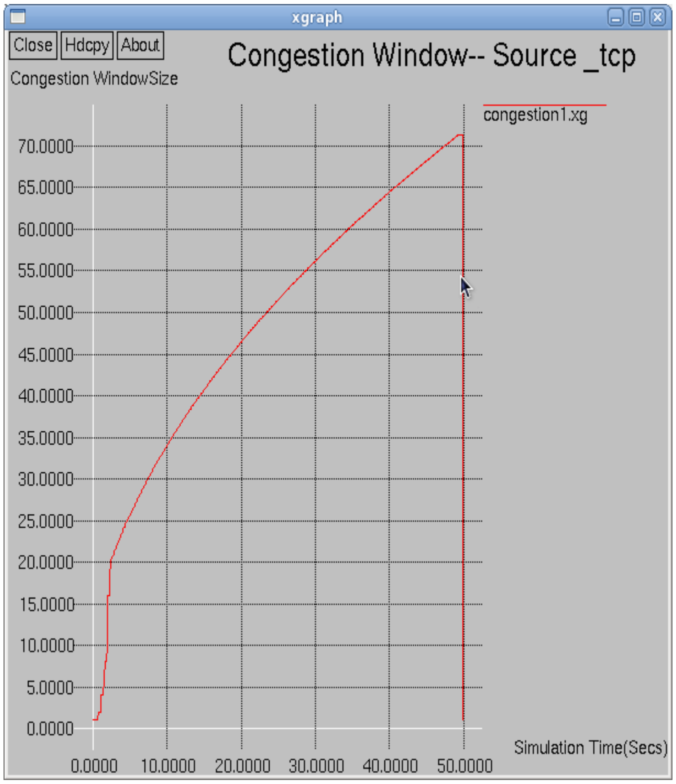
Trace File-



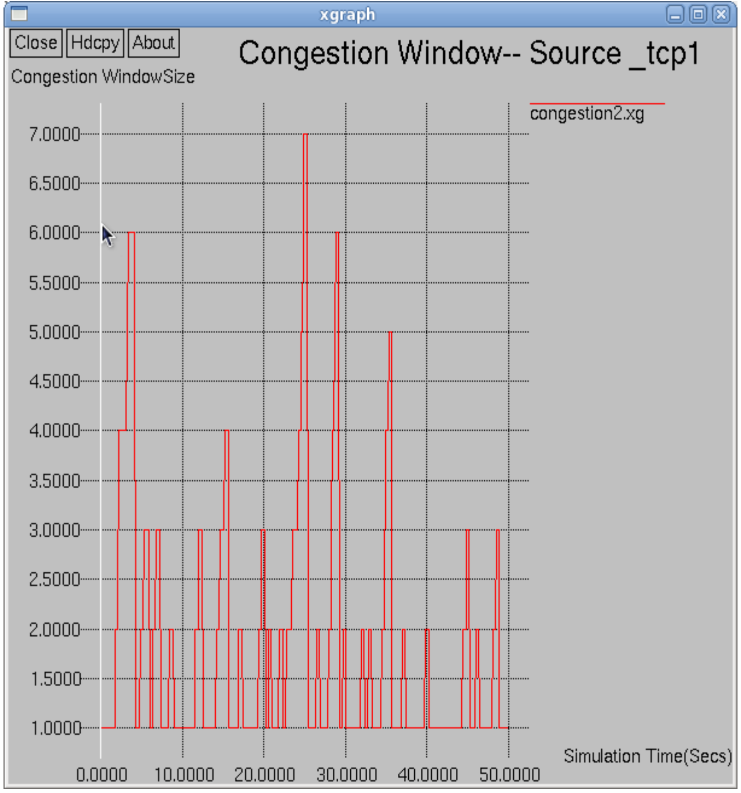
```
+ 0.3 0 2 tcp 40 ------- 1 0.0 4.0 0 0
- 0.3 0 2 tcp 40 ------- 1 0.0 4.0 0 0
r 0.31016 0 2 tcp 40 ------- 1 0.0 4.0 0 0
+ 0.31016 2 3 tcp 40 ------- 1 0.0 4.0 0 0
- 0.31016 2 3 tcp 40 ------- 1 0.0 4.0 0 0
r 0.411227 2 3 tcp 40 ------- 1 0.0 4.0 0 0
h 0.411227 3 6 tcp 40 ------- 1 0.0 4.0 0 0
+ 0.451227 3 6 tcp 40 ------- 1 0.0 4.0 0 0
- 0.451227 3 6 tcp 40 ------- 1 0.0 4.0 0 0
d 0.451231 5 6 tcp 40 ------- 1 0.0 4.0 0 0
r 0.491231 6 4 tcp 40 ------- 1 0.0 4.0 0 0
h 0.491231 4 6 ack 40 ------- 1 4.0 0.0 0 1
+ 0.531231 4 6 ack 40 ------- 1 4.0 0.0 0 1
- 0.531231 4 6 ack 40 ------- 1 4.0 0.0 0 1
d 0.531235 5 6 ack 40 ------- 1 4.0 0.0 0 1
r 0.571235 6 3 ack 40 ------- 1 4.0 0.0 0 1
+ 0.571235 3 2 ack 40 ------- 1 4.0 0.0 0 1
- 0.571235 3 2 ack 40 ------- 1 4.0 0.0 0 1
r 0.672301 3 2 ack 40 ------- 1 4.0 0.0 0 1
+ 0.672301 2 0 ack 40 ------- 1 4.0 0.0 0 1
- 0.672301 2 0 ack 40 ------- 1 4.0 0.0 0 1
r 0.682461 2 0 ack 40 ------- 1 4.0 0.0 0 1
+ 0.682461 0 2 tcp 590 ------- 1 0.0 4.0 1 2
- 0.682461 0 2 tcp 590 ------- 1 0.0 4.0 1 2
+ 0.682461 0 2 tcp 590 ------- 1 0.0 4.0 2 3
- 0.684821 0 2 tcp 590 ------- 1 0.0 4.0 2 3
r 0.694821 0 2 tcp 590 ------- 1 0.0 4.0 1 2
+ 0.694821 2 3 tcp 590 ------- 1 0.0 4.0 1 2
- 0.694821 2 3 tcp 590 ------- 1 0.0 4.0 1 2
r 0.697181 0 2 tcp 590 ------- 1 0.0 4.0 2 3
```

Congestion graph-

```
$ xgraph congestion1.xg
```



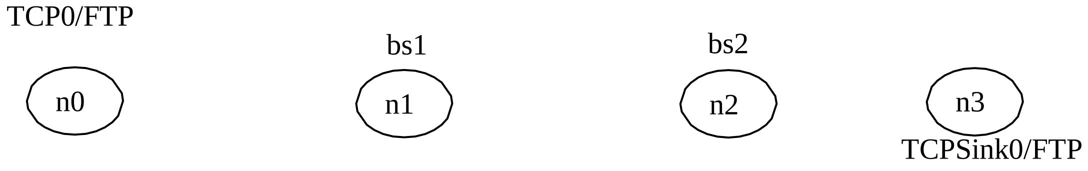
```
$ xgraph congestion2.xg
```



Lab Experiment 4 :

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

Topology-



Code-

```
#create Simulator class
set ns [new Simulator]

#open trace file
set nt [open lab42.tr w]
$ns trace-all $nt

#create Topography object
set topo [new Topography]
#define grid size
$topo load_flatgrid 1000 1000

#open namtrace file
set nf [open lab42.nam w]
$ns namtrace-all-wireless $nf 1000 1000

#specify node configuration
$ns node-config -adhocRouting DSDV \
                -llType LL \
                -macType Mac/802_11 \
                -ifqType Queue/DropTail \
                -ifqLen 20 \
                -phyType Phy/WirelessPhy \
                -channelType Channel/WirelessChannel \
                -propType Propagation/TwoRayGround \
                -antType Antenna/OmniAntenna \
                -topoInstance $topo \
                -agentTrace ON \
                -routerTrace ON

#create General Operation Director(god) object that stores total number of mobile nodes.
create-god 4
#create nodes and label them
set n0 [$ns node]
set n1 [$ns node]
set n2 [$ns node]
set n3 [$ns node]

$n0 label "tcp0"
$n1 label "sink0"
$n2 label "bs1"
$n3 label "bs2"

#give initial x, y, z coordinates to nodes
$n0 set X_ 110
$n0 set Y_ 500
$n0 set Z_ 0

$n1 set X_ 600
$n1 set Y_ 500
$n1 set Z_ 0

$n2 set X_ 300
$n2 set Y_ 500
$n2 set Z_ 0

$n3 set X_ 450
$n3 set Y_ 500
$n3 set Z_ 0

#attach agent and application to nodes and connect them
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

#schedule the event
$ns at 0.5 "$ftp0 start"

#set up destination for mobile nodes. They move to <x><y> coordinates at <s>m/s.
$ns at 0.3 "$n0 setdest 110 500 10"
$ns at 0.3 "$n1 setdest 600 500 20"
$ns at 0.3 "$n2 setdest 300 500 30"
$ns at 0.3 "$n3 setdest 450 500 30"

$ns at 10.0 "$n0 setdest 100 550 5"
$ns at 10.0 "$n1 setdest 630 450 5"

$ns at 70.0 "$n0 setdest 170 680 5"
$ns at 70.0 "$n1 setdest 580 380 5"
```

```
$ns at 120.0 "$n0 setdest 140 720 5"
$ns at 135.0 "$n0 setdest 110 600 5"
$ns at 140.0 "$n1 setdest 600 550 5"
$ns at 155.0 "$n0 setdest 89 500 5"
$ns at 190.0 "$n0 setdest 100 440 5"
$ns at 210.0 "$n1 setdest 700 600 5"
$ns at 240.0 "$n1 setdest 650 500 5"

proc finish { } {
        global ns nt nf
        $ns flush-trace
        exec nam lab42.nam &
        close $nt
        close $nf
        exit 0
}

$ns at 400 "finish"
$ns run
```

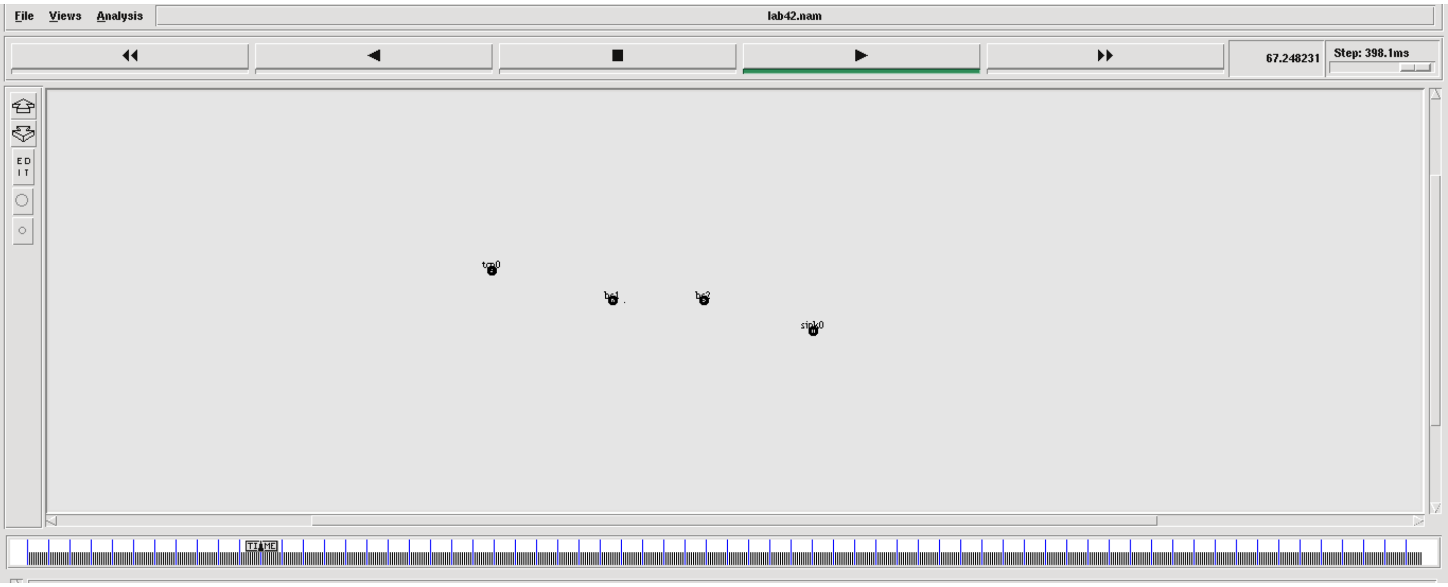
Awk file-

```
BEGIN{
        PktsSent=0;
        PktsRcvd=0;
        PktsAtRTR=0;
}

{
        if(($1=="s")&&($4=="RTR")&&($7=="tcp")) PktsAtRTR++;

        if(($1=="s")&&($4=="AGT")&&($7=="tcp")) PktsSent++;

        if(($1=="r")&&($4=="AGT")&&($7=="tcp")) PktsRcvd++;

}

END{
        print " Number of Packets Sent :" PktsSent

        print " Number of Packets Received :" PktsRcvd
        print " Pacjet Delivery Ratio :" PktsRcvd/PktsSent*100
        print " Routing Load :" PktsAtRTR/PktsRcvd
}
```

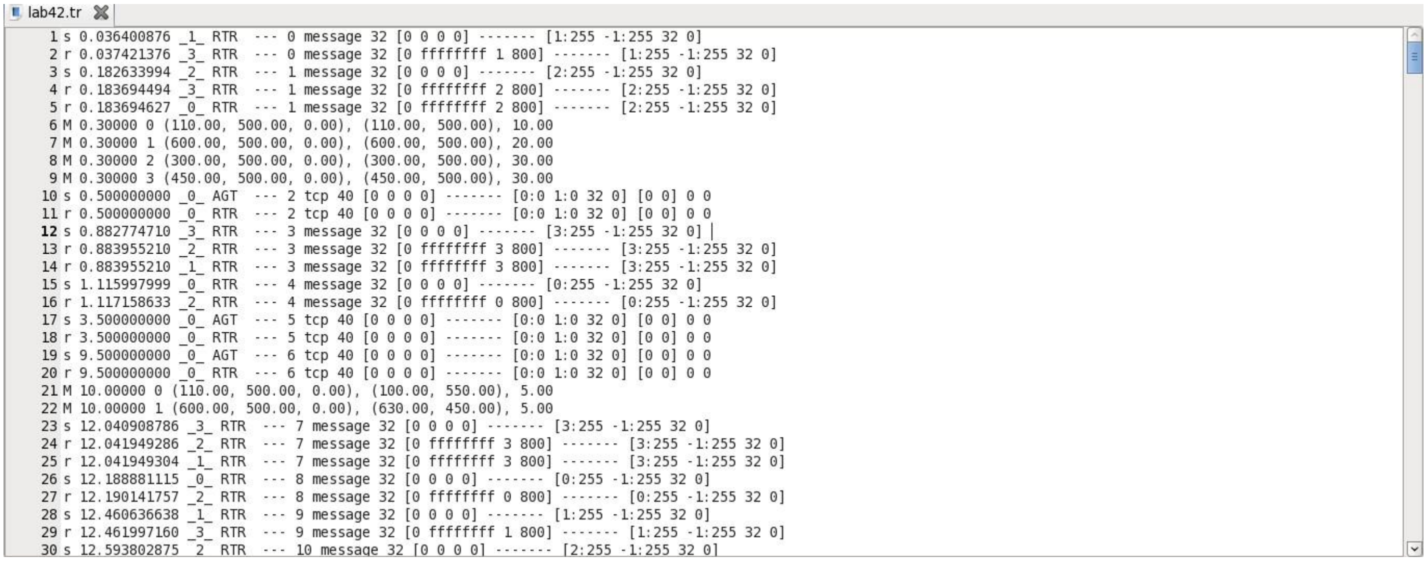
Output-

```
$ns pgm6.tcl
$awk -f count.awk lab42.tr
Number of Packets Sent :6819
Number of Packets Received :6685
Packet Delivery Ratio :98.0349
Routing Load :1.02004
```

Simulator-



Trace File-

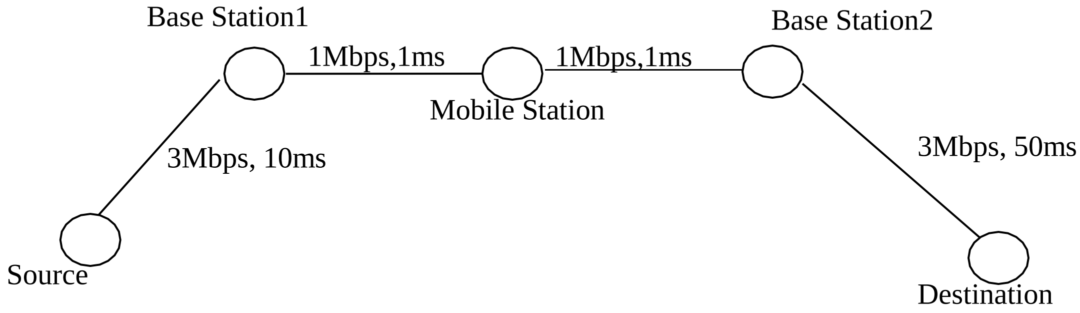


```
s 0.036400876 _1_ RTR  --- 0 message 32 [0 0 0 0] ------- [1:255 -1:255 32 0]
r 0.037421376 _3_ RTR  --- 0 message 32 [0 ffffffff 1 800] ------- [1:255 -1:255 32 0]
s 0.182633994 _2_ RTR  --- 1 message 32 [0 0 0 0] ------- [2:255 -1:255 32 0]
r 0.183694494 _3_ RTR  --- 1 message 32 [0 ffffffff 2 800] ------- [2:255 -1:255 32 0]
r 0.183694627 _0_ RTR  --- 1 message 32 [0 ffffffff 2 800] ------- [2:255 -1:255 32 0]
M 0.30000 0 (110.00, 500.00, 0.00), (110.00, 500.00), 10.00
M 0.30000 1 (600.00, 500.00, 0.00), (600.00, 500.00), 20.00
M 0.30000 2 (300.00, 500.00, 0.00), (300.00, 500.00), 30.00
M 0.30000 3 (450.00, 500.00, 0.00), (450.00, 500.00), 30.00
s 0.500000000 _0_ AGT  --- 2 tcp 40 [0 0 0 0] ------- [0:0 1:0 32 0] [0 0] 0 0
r 0.500000000 _0_ RTR  --- 2 tcp 40 [0 0 0 0] ------- [0:0 1:0 32 0] [0 0] 0 0
s 0.882774710 _3_ RTR  --- 3 message 32 [0 0 0 0] ------- [3:255 -1:255 32 0]
r 0.883955210 _2_ RTR  --- 3 message 32 [0 ffffffff 3 800] ------- [3:255 -1:255 32 0]
r 0.883955210 _1_ RTR  --- 3 message 32 [0 ffffffff 3 800] ------- [3:255 -1:255 32 0]
s 1.115997999 _0_ RTR  --- 4 message 32 [0 0 0 0] ------- [0:255 -1:255 32 0]
r 1.117158633 _2_ RTR  --- 4 message 32 [0 ffffffff 0 800] ------- [0:255 -1:255 32 0]
s 3.500000000 _0_ AGT  --- 5 tcp 40 [0 0 0 0] ------- [0:0 1:0 32 0] [0 0] 0 0
r 3.500000000 _0_ RTR  --- 5 tcp 40 [0 0 0 0] ------- [0:0 1:0 32 0] [0 0] 0 0
s 9.500000000 _0_ AGT  --- 6 tcp 40 [0 0 0 0] ------- [0:0 1:0 32 0] [0 0] 0 0
r 9.500000000 _0_ RTR  --- 6 tcp 40 [0 0 0 0] ------- [0:0 1:0 32 0] [0 0] 0 0
M 10.00000 0 (110.00, 500.00, 0.00), (100.00, 550.00), 5.00
M 10.00000 1 (600.00, 500.00, 0.00), (630.00, 450.00), 5.00
s 12.040908786 _3_ RTR  --- 7 message 32 [0 0 0 0] ------- [3:255 -1:255 32 0]
r 12.041949286 _2_ RTR  --- 7 message 32 [0 ffffffff 3 800] ------- [3:255 -1:255 32 0]
r 12.041949304 _1_ RTR  --- 7 message 32 [0 ffffffff 3 800] ------- [3:255 -1:255 32 0]
s 12.188881115 _0_ RTR  --- 8 message 32 [0 0 0 0] ------- [0:255 -1:255 32 0]
r 12.190141757 _2_ RTR  --- 8 message 32 [0 ffffffff 0 800] ------- [0:255 -1:255 32 0]
s 12.460636638 _1_ RTR  --- 9 message 32 [0 0 0 0] ------- [1:255 -1:255 32 0]
r 12.461997160 _3_ RTR  --- 9 message 32 [0 ffffffff 1 800] ------- [1:255 -1:255 32 0]
s 12.593802875 _2_ RTR  --- 10 message 32 [0 0 0 0] ------- [2:255 -1:255 32 0]
```

Lab Experiment 5 :

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

Topology-

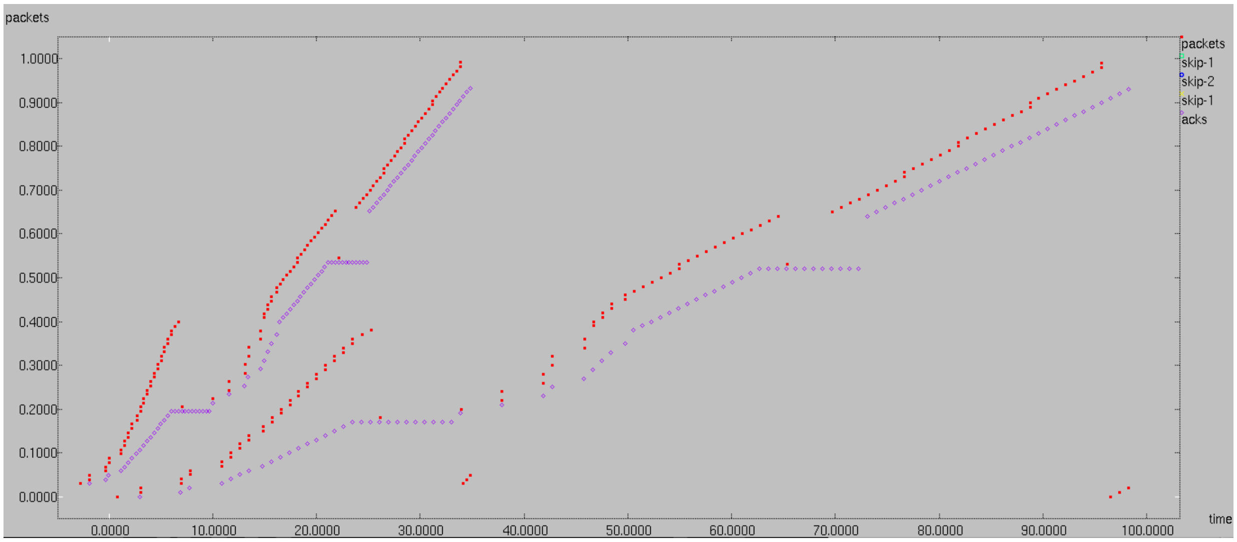


Code –

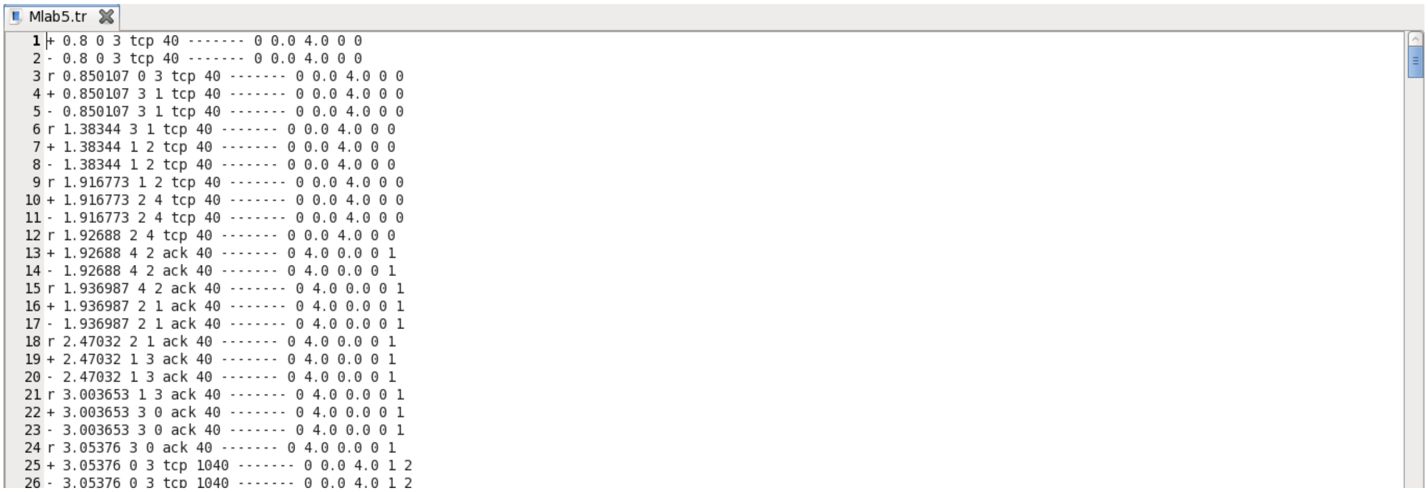
```
#General parameters
set stop 100
set type gsm

#AQM parameters
set minth 30
set maxth 0
set adaptive 1

#traffic generation
set flows 0
set window 30

#plotting statistics
set opt(wrap) 100
set opt(srcTrace) is
set opt(dstTrace) bs2

#default downlink bandwidth in bps
set bwDL(gsm) 9600
#default propogation delay in sec
set propDL(gsm) .500

set ns [new Simulator]

set tf [open Mlab5.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 "GSM Cell Topology"
        }
proc set_link_params {t} {

global ns nodes bwDL propDL
$ns bandwidth $nodes(bs1) $nodes(ms) $bwDL($t) duplex
$ns bandwidth $nodes(bs2) $nodes(ms) $bwDL($t) duplex

$ns delay $nodes(bs1) $nodes(ms) $propDL($t) duplex
$ns delay $nodes(bs2) $nodes(ms) $propDL($t) duplex

$ns queue-limit $nodes(bs1) $nodes(ms) 10
$ns queue-limit $nodes(bs2) $nodes(ms) 10
}

#RED and TCP parameters
Queue/RED set adaptive_ $adaptive
Queue/RED set thresh_ $minth
Queue/RED set maxthresh_ $maxth
Agent/TCP set window_ $window

#create topology
switch $type {
        gsm -
        umts {cell_topo}
}

set_link_params $type
$ns insert-delayer $nodes(ms) $nodes(bs1) [new Delayer]
$ns insert-delayer $nodes(ms) $nodes(bs2) [new Delayer]

# set up TCP connection
if {$flows == 0} {
        set tcp1 [$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"
         }

 proc stop {} {
        global nodes opt tf
        set wrap $opt(wrap)
        set sid [$nodes($opt(srcTrace)) id]
        set did [$nodes($opt(dstTrace)) id]
        set a "Mlab5.tr"

        set GETRC "../bin/getrc"
        set RAW2XG "../bin/raw2xg"

        exec $GETRC -s $sid -d $did -f 0 Mlab5.tr | \
        $RAW2XG -s 0.01 -m $wrap -r > plot.xgr

        exec $GETRC -s $did -d $sid -f 0 Mlab5.tr | \
        $RAW2XG -a -s 0.01 -m $wrap >> plot.xgr

        exec xgraph -x time -y packets plot.xgr &
        exit 0
        }
         $ns at $stop "stop"
         $ns run
```

Graph-



Trace File-

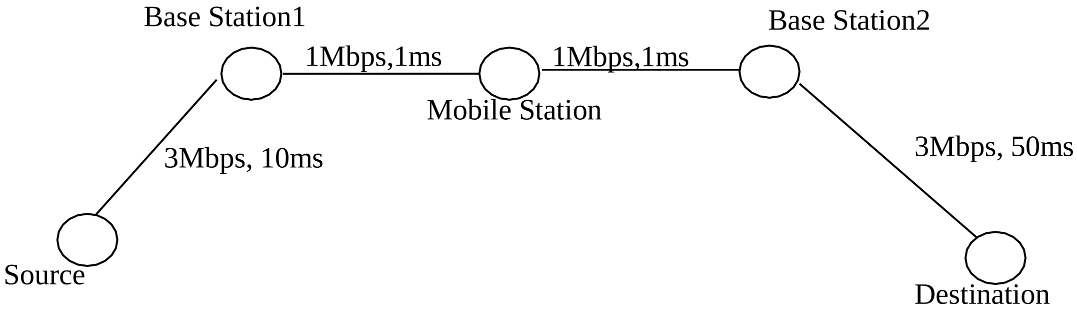


```
+ 0.8 0 3 tcp 40 ------- 0 0.0 4.0 0 0
- 0.8 0 3 tcp 40 ------- 0 0.0 4.0 0 0
r 0.850107 0 3 tcp 40 ------- 0 0.0 4.0 0 0
+ 0.850107 3 1 tcp 40 ------- 0 0.0 4.0 0 0
- 0.850107 3 1 tcp 40 ------- 0 0.0 4.0 0 0
r 1.38344 3 1 tcp 40 ------- 0 0.0 4.0 0 0
+ 1.38344 1 2 tcp 40 ------- 0 0.0 4.0 0 0
- 1.38344 1 2 tcp 40 ------- 0 0.0 4.0 0 0
r 1.916773 1 2 tcp 40 ------- 0 0.0 4.0 0 0
+ 1.916773 2 4 tcp 40 ------- 0 0.0 4.0 0 0
- 1.916773 2 4 tcp 40 ------- 0 0.0 4.0 0 0
r 1.92688 2 4 tcp 40 ------- 0 0.0 4.0 0 0
+ 1.92688 4 2 ack 40 ------- 0 4.0 0.0 0 1
- 1.92688 4 2 ack 40 ------- 0 4.0 0.0 0 1
r 1.936987 4 2 ack 40 ------- 0 4.0 0.0 0 1
+ 1.936987 2 1 ack 40 ------- 0 4.0 0.0 0 1
- 1.936987 2 1 ack 40 ------- 0 4.0 0.0 0 1
r 2.47032 2 1 ack 40 ------- 0 4.0 0.0 0 1
+ 2.47032 1 3 ack 40 ------- 0 4.0 0.0 0 1
- 2.47032 1 3 ack 40 ------- 0 4.0 0.0 0 1
r 3.003653 1 3 ack 40 ------- 0 4.0 0.0 0 1
+ 3.003653 3 0 ack 40 ------- 0 4.0 0.0 0 1
- 3.003653 3 0 ack 40 ------- 0 4.0 0.0 0 1
r 3.05376 3 0 ack 40 ------- 0 4.0 0.0 0 1
+ 3.05376 0 3 tcp 1040 ------- 0 0.0 4.0 1 2
- 3.05376 0 3 tcp 1040 ------- 0 0.0 4.0 1 2
```

Lab Experiment 6 :

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

Topology-

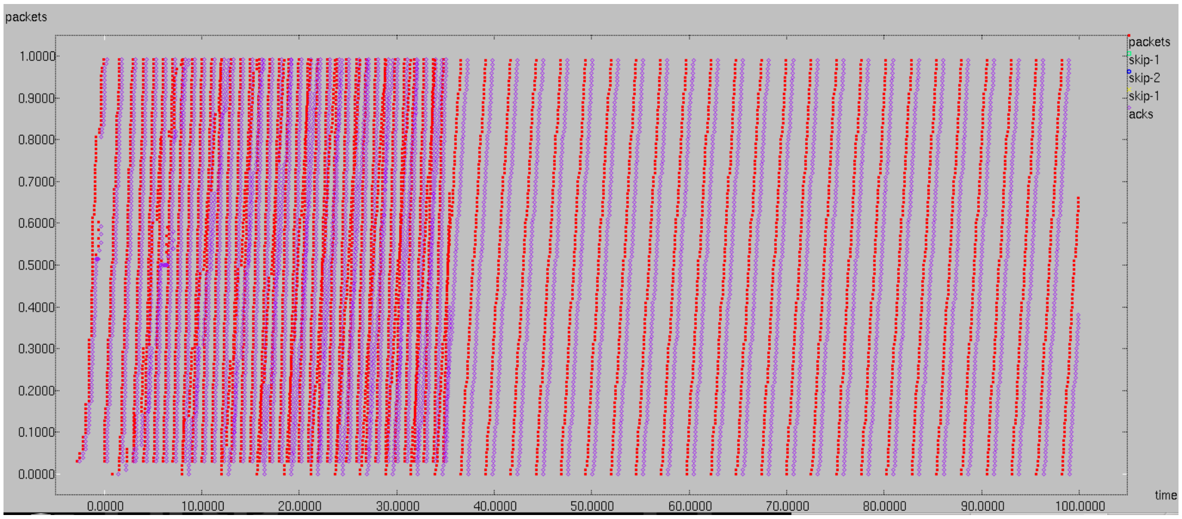


Code –

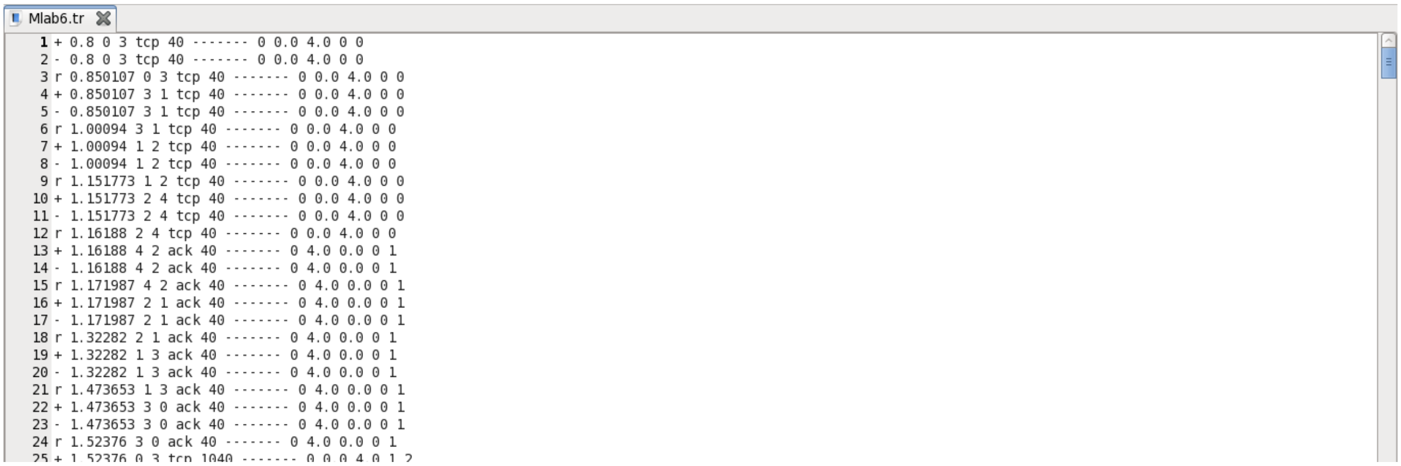
```
#General parameters
set stop 100
set type umts

#AQM parameters
set minth 30
set maxth 0
set adaptive 1

#traffic generation
set flows 0
set window 30

#plotting statistics
set opt(wrap) 100
set opt(srcTrace) is
set opt(dstTrace) bs2

#default downlink bandwidth in bps
set bwDL(umts) 38400
#default propogation delay in sec
set propDL(umts) .150


set ns [new Simulator]

set tf [open Mlab6.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 "umts Cell Topology"
            }
proc set_link_param {t} {

global ns nodes bwDL propDL
$ns bandwidth $nodes(bs1) $nodes(ms) $bwDL($t) duplex
$ns bandwidth $nodes(bs2) $nodes(ms) $bwDL($t) duplex

$ns delay $nodes(bs1) $nodes(ms) $propDL($t) duplex
$ns delay $nodes(bs2) $nodes(ms) $propDL($t) duplex

$ns queue-limit $nodes(bs1) $nodes(ms) 20
$ns queue-limit $nodes(bs2) $nodes(ms) 20
}

#set RED and TCP parameters
Queue/RED set adaptive_ $adaptive
Queue/RED set thresh_ $minth
Queue/RED set maxthresh_ $maxth
Agent/TCP set window_ $window

#create topology
switch $type {
            umts {cell_topo}
}

set_link_param $type
$ns insert-delayer $nodes(ms) $nodes(bs1) [new Delayer]
$ns insert-delayer $nodes(ms) $nodes(bs2) [new Delayer]

#set up TCP connection
if {$flows == 0 } {
            set tcp1 [$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"
              }

  proc stop {} {
            global nodes opt tf
            set wrap $opt(wrap)
            set sid [$nodes($opt(srcTrace)) id]
            set did [$nodes($opt(dstTrace)) id]

            set a "Mlab6.tr"
            set GETRC "../bin/getrc"
            set RAW2XG "../bin/raw2xg"

            exec $GETRC -s $sid -d $did -f 0 Mlab6.tr | \
            $RAW2XG -s 0.01 -m $wrap -r > plot6.xgr

            exec $GETRC -s $did -d $sid -f 0 Mlab6.tr | \
            $RAW2XG -a -s 0.01 -m $wrap >> plot6.xgr

            exec xgraph -x time -y packets plot6.xgr &
            exit 0
            }
             $ns at $stop "stop"
            $ns run
```

Graph-



Trace File-



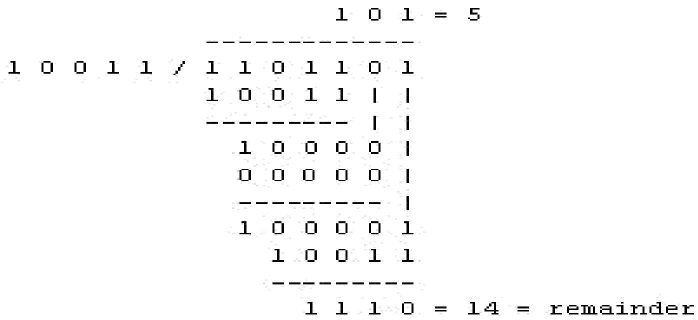
```
+ 0.8 0 3 tcp 40 ------- 0 0.0 4.0 0 0
- 0.8 0 3 tcp 40 ------- 0 0.0 4.0 0 0
r 0.850107 0 3 tcp 40 ------- 0 0.0 4.0 0 0
+ 0.850107 3 1 tcp 40 ------- 0 0.0 4.0 0 0
- 0.850107 3 1 tcp 40 ------- 0 0.0 4.0 0 0
r 1.00094 3 1 tcp 40 ------- 0 0.0 4.0 0 0
+ 1.00094 1 2 tcp 40 ------- 0 0.0 4.0 0 0
- 1.00094 1 2 tcp 40 ------- 0 0.0 4.0 0 0
r 1.151773 1 2 tcp 40 ------- 0 0.0 4.0 0 0
+ 1.151773 2 4 tcp 40 ------- 0 0.0 4.0 0 0
- 1.151773 2 4 tcp 40 ------- 0 0.0 4.0 0 0
r 1.16188 2 4 tcp 40 ------- 0 0.0 4.0 0 0
+ 1.16188 4 2 ack 40 ------- 0 4.0 0.0 0 1
- 1.16188 4 2 ack 40 ------- 0 4.0 0.0 0 1
r 1.171987 4 2 ack 40 ------- 0 4.0 0.0 0 1
+ 1.171987 2 1 ack 40 ------- 0 4.0 0.0 0 1
- 1.171987 2 1 ack 40 ------- 0 4.0 0.0 0 1
r 1.32282 2 1 ack 40 ------- 0 4.0 0.0 0 1
+ 1.32282 1 3 ack 40 ------- 0 4.0 0.0 0 1
- 1.32282 1 3 ack 40 ------- 0 4.0 0.0 0 1
r 1.473653 1 3 ack 40 ------- 0 4.0 0.0 0 1
+ 1.473653 3 0 ack 40 ------- 0 4.0 0.0 0 1
- 1.473653 3 0 ack 40 ------- 0 4.0 0.0 0 1
r 1.52376 3 0 ack 40 ------- 0 4.0 0.0 0 1
+ 1.52376 0 3 tcp 1040 ------- 0 0.0 4.0 1 2
```

Lab Program 7 :

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

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

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



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

With decimal calculations you can quickly check that 109 divided by 19 gives a quotient of 5 with 14 as the remainder. But what we also see in the scheme is that every bit extra to check only costs one binary comparison and in 50% of the cases one binary subtraction. You can 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	100000100110000010001110110110111

Table 1. International Standard CRC Polynomials

Error detection with CRC

Consider a message represented by the polynomial M(x)

Consider a *generating polynomial* G(x)

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

Note this G(x) is prime.

Steps:

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

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

   that means: x = n y + c

   Hence (x-c) = n y

(x-c) div y gives remainder 0

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

Hence (x+c) div y gives remainder 0

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

Note if G(x) has order n - highest power is x^n ,

and the *remainder* will cover n bits.

i.e. Add n bits (Zeros) to message.

Some CRC polynomials that are actually used

Some CRC polynomials

- CRC-8:
  x^8+x^2+x+1
  - Used in: 802.16 (along with error *correction*).
- CRC-CCITT:
  $x^{16}+x^{12}+x^5+1$
  - Used in: HDLC, SDLC, PPP default
- IBM-CRC-16 (ANSI):
  $x^{16}+x^{15}+x^2+1$
- 802.3:
  $x^{32}+x^{26}+x^{23}+x^{22}+x^{16}+x^{12}+x^{11}+x^{10}+x^8+x^7+x^5+x^4+x^2+x+1$

Code-

```
import java.util.Scanner;

public class CRC {

    public static int n;

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

                CRC crc=new CRC();

                String copy,rec,code,zero="0000000000000000";

                System.out.println("enter the dataword to be sent");
                code=sc.nextLine();

                n=code.length();

                copy=code;
                code+=zero;
                code=crc.divide(code);

                System.out.println("dataword="+copy);

                copy=copy.substring(0,n)+code.substring(n);

                System.out.print("CRC=");
                System.out.println(code.substring(n));

                System.out.println("transmitted frame is="+copy);

                System.out.println("enter received data:");
                rec=sc.nextLine();

                if(zero.equals(crc.divide(rec).substring(n)))
                System.out.println("correct bits received");
                else
                    System.out.println("received frame contains one or more error");

                sc.close();
            }

            public String divide(String s)
            {
                String div="10001000000100001";


                int i,j;
                char x;
                for(i=0;i<n;i++)
                {
                        x=s.charAt(i);

                        for(j=0;j<17;j++)
                        {
                                if(x=='1')
                                {
                                        if(s.charAt(i+j)!=div.charAt(j))
                                                s=s.substring(0,i+j)+"1"+s.substring(i+j+1);
                                        else
                                                s=s.substring(0,i+j)+"0"+s.substring(i+j+1);
                                }
                        }
                }
                return s;
        }
}
```

Output 1 –

```
enter the dataword to be sent
1100
dataword =1100
CRC=1100000110001100
transmitted frame is=11001100000110001100
enter received data:
11001100000010001100
received frame contains one or more error
```

Output 2 –

```
enter the dataword to be sent
1100
dataword =1100
CRC=1100000110001100
transmitted frame is=11001100000110001100
enter received data:
11001100000110001100
correct bits received
```

Output 3 –

```
enter the dataword to be sent
1101
dataword=1101
CRC=1101000110101101
transmitted frame is=11011101000110101101
enter received data:
11011001000110110010
received frame contains one or more error
```

Lab Program 8 :

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

Bellman-Ford algorithm is a procedure used to find all shortest path in a graph from one source to all other nodes. The algorithm requires that the graph does not contain any cycles of negative length, but if it does, the algorithm is able to detect it.

The algorithm was introduced by American mathematicians *Richard Bellman* and *Lester Ford*.

It is similar to Dijkstra's algorithm but it can work with graphs in which edges can have negative weights.

Why would one ever have edges with negative weights in real life?

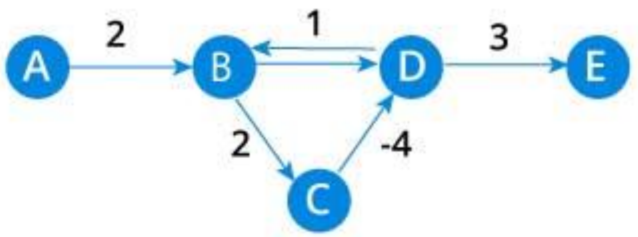
Negative weight edges might seem useless at first but they can explain a lot of phenomena like cashflow, heat released/absorbed in a chemical reaction etc.

For instance, if there are different ways to reach from one chemical A to another chemical B, each method will have sub-reactions involving both heat dissipation and absorption.

If we want to find the set of reactions where minimum energy is required, then we will need to be able to factor in the heat absorption as negative weights and heat dissipation as positive weights.

Why we need to be careful with negative weights?

Negative weight edges can create negative weight cycles i.e. a cycle which will reduce the total path distance by coming back to the same point.

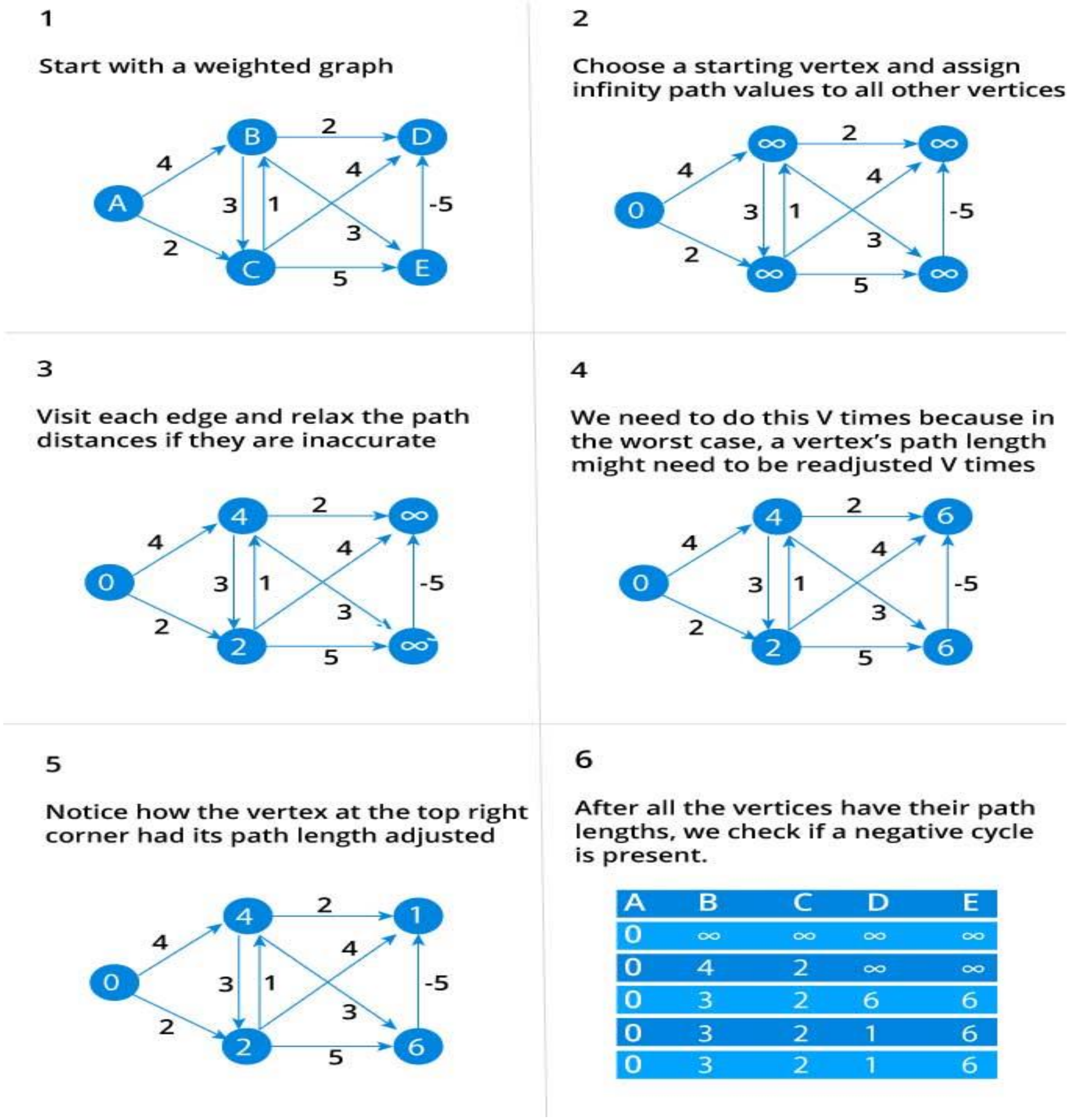


Shortest path algorithms like Dijkstra's Algorithm that aren't able to detect such a cycle can give an incorrect result because they can go through a negative weight cycle and reduce the path length.

How Bellman Ford's algorithm works

Bellman Ford algorithm works by overestimating the length of the path from the starting vertex to all other vertices. Then it iteratively relaxes those estimates by finding new paths that are shorter than the previously overestimated paths.

By doing this repeatedly for all vertices, we are able to guarantee that the end result is optimized.



1

Start with a weighted graph

2

Choose a starting vertex and assign infinity path values to all other vertices

3

Visit each edge and relax the path distances if they are inaccurate

4

We need to do this V times because in the worst case, a vertex's path length might need to be readjusted V times

5

Notice how the vertex at the top right corner had its path length adjusted

6

After all the vertices have their path lengths, we check if a negative cycle is present.

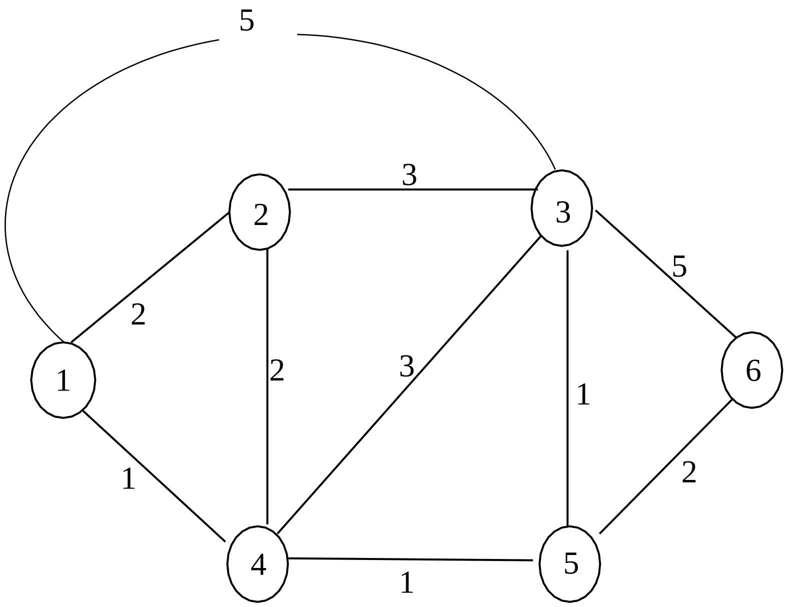
A	B	C	D	E
0	∞	∞	∞	∞
0	4	2	∞	∞
0	3	2	6	6
0	3	2	1	6
0	3	2	1	6

Code -

```
import java.util.Scanner;
public class bellmanford
{

    public int distance[];
    public int numb_vert;
    public static final int MAX_VALUE=999;

    public bellmanford(int numb_vert)
    {
        this.numb_vert = numb_vert;
        distance = new int[numb_vert+1];
    }

    public void BellmanfordpEvaluation(int source,int adj_matrix[][])
    {

        for(int node=1;node<=numb_vert;node++)
            distance[node]=MAX_VALUE;
        distance[source]=0;

        for(int node=1;node<=numb_vert-1;node++)
        {
            for(int src_node=1;src_node<=numb_vert;src_node++)
            {
                for(int dest_node=1;dest_node<=numb_vert;dest_node++)
                {
                    if(adj_matrix[src_node][dest_node]!=MAX_VALUE)
                    {
                        if(distance[dest_node] > distance[src_node] +
                                    adj_matrix[src_node][dest_node])

                            distance[dest_node] = distance[src_node] +
                                    adj_matrix[src_node][dest_node];
                    }
                }
            }
        }

        for(int src_node=1;src_node<=numb_vert;src_node++)
        {
            for(int dest_node=1;dest_node<=numb_vert;dest_node++)
            {
                if(adj_matrix[src_node][dest_node]!=MAX_VALUE)
                {
                    if(distance[dest_node] > distance[src_node] +
                                    adj_matrix[src_node][dest_node])
                    {
                        System.out.println("The graph contains negative edge cycle");

                    }
```

```
                                }
                        }
                }

                System.out.println("Routing Table for Router " + source+" is");

                System.out.println("Destination        Distance\t");
                for(int vertex=1;vertex<=numb_vert;vertex++)
                        System.out.println(+vertex+"\t\t\t"+distance[vertex]);

        }

        public static void main(String args[])
        {

                int numb_vert=0;
                int source;
                Scanner scan = new Scanner(System.in);

                System.out.println("Enter the number of vertices");
                numb_vert = scan.nextInt();

                int adj_matrix[][] = new int[numb_vert+1][numb_vert+1];
                System.out.println("Enter the adjacency matrix");
                for(int src_node=1;src_node<=numb_vert;src_node++)
                        for(int dest_node=1;dest_node<=numb_vert;dest_node++)
                        {
                                adj_matrix[src_node][dest_node] = scan.nextInt();
                                if(src_node==dest_node)
                                {
                                        adj_matrix[src_node][dest_node]=0;
                                        continue;
                                }

                                if(adj_matrix[src_node][dest_node]==0)
                                        adj_matrix[src_node][dest_node]=MAX_VALUE;
                        }





                for(int i=1;i<=numb_vert;i++)
                {
                        bellmanford bellmanford = new bellmanford(numb_vert);
                        bellmanford.BellmanfordpEvaluation(i,adj_matrix);
                }
                scan.close();
        }
}
```

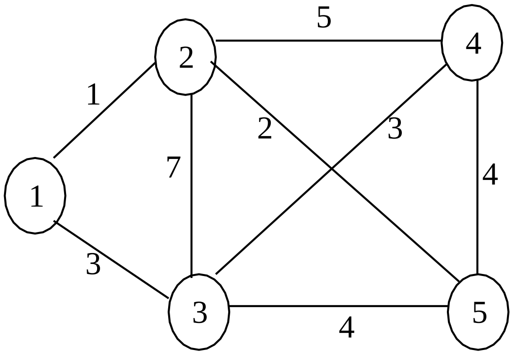
Output 1 –



```
Enter the number of vertices
6
Enter the adjacency matrix
0    2    5   1   999 999
2    0    3   2    999 999
5    3    0   3   1     5
1    2    3   0   1     999
999 999 1 1     0     2
999 999 5 999 2     0
Routing Table for Router 1 is
Destination        Distance
1                              0
2                              2
3                              3
4                              1
5                              2
6                              4
Routing Table for Router 2 is
Destination        Distance
1                              2
2                              0
3                              3
4                              2
5                              3
6                              5
Routing Table for Router 3 is
Destination        Distance
1                              3
2                              3
3                              0
4                              2
5                              1
6                              3
Routing Table for Router 4 is
Destination        Distance
1                              1
2                              2
3                              2
4                              0
5                              1
6                              3
Routing Table for Router 5 is
Destination        Distance
1                              2
2                              3
3                              1
4                              1
5                              0
6                              2
Routing Table for Router 6 is
```

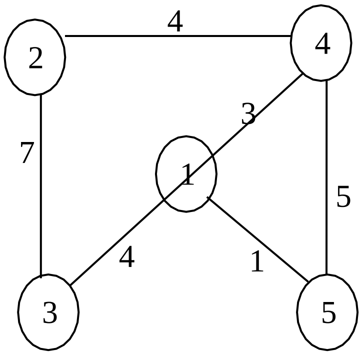
```
Destination        Distance
1                          4
2                          5
3                          3
4                          3
5                          2
6                          0
```

Output 2 –



```
Enter the number of vertices
5
Enter the adjacency matrix
0    1    3   999 999
1    0    7   5     2
3    7    0   3     4
999 5    3   0     4
999 2    4   4     0
Routing Table for Router 1 is
Destination        Distance
1                          0
2                          1
3                          3
4                          6
5                          3
Routing Table for Router 2 is
Destination        Distance
1                          1
2                          0
3                          4
4                          5
5                          2
Routing Table for Router 3 is
Destination        Distance
1                          3
2                          4
3                          0
4                          3
5                          4
Routing Table for Router 4 is
Destination        Distance
1                          6
2                          5
3                          3
4                          0
5                          4
Routing Table for Router 5 is
Destination        Distance
1                          3
2                          2
3                          4
4                          4
5                          0
```

Output 3 –



```
Enter the number of vertices
5
Enter the adjacency matrix
0    999 3    1    4
999 0    4    999 7
3    4    0    5    999
1    999 5    0    999
4    7    999 999 0
Routing Table for Router 1 is
Destination       Distance
1                      0
2                      7
3                      3
4                      1
5                      4
Routing Table for Router 2 is
Destination       Distance
1                      7
2                      0
3                      4
4                      8
5                      7
Routing Table for Router 3 is
Destination       Distance
1                      3
2                      4
3                      0
4                      4
5                      7
Routing Table for Router 4 is
Destination       Distance
1                      1
2                      8
3                      4
4                      0
5                      5
Routing Table for Router 5 is
Destination       Distance
1                      4
2                      7
3                      7
4                      5
5                      0
```

Lab Program 9 :

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

Sockets are a protocol independent method of creating a connection between processes. Sockets can be either

Connection based or connectionless: Is a connection established before communication or does each packet describe the destination?

Packet based or streams based: Are there message boundaries or is it one stream?
Reliable or unreliable: Can messages be lost, duplicated, reordered, or corrupted?

Socket characteristics

Sockets are characterized by their domain, type and transport protocol. Common domains are:

AF_UNIX: address format is UNIX pathname

AF_INET: address format is host and port number Common types are:
virtual circuit: received in order transmitted and reliably
datagram: arbitrary order, unreliable

Each socket type has one or more protocols. Ex:

TCP/IP (virtual circuits)
UDP (datagram)

Use of sockets:

Connection–based sockets communicate client-server: the server waits for a connection from the client
Connectionless sockets are peer-to-peer: each process is symmetric.

Socket APIs

socket: creates a socket of a given domain, type, protocol (buy a phone)

bind: assigns a name to the socket (get a telephone number)

listen: specifies the number of pending connections that can be queued for a server socket. (call waiting allowance)

accept: server accepts a connection request from a client (answer phone)

connect: client requests a connection request to a server (call)

send, *sendto*: write to connection (speak)

ecv, recvfrom: read from connection (listen)

shutdown: end the call

Connection-based communication

Server performs the following actions

socket: create the socket

bind: give the address of the socket on the server

listen: specifies the maximum number of connection requests that can be pending for this process

accept: establish the connection with a specific client

send, recv: stream-based equivalents of read and write (repeated)

shutdown: end reading or writing

close: release kernel data structures

socket: create the socket

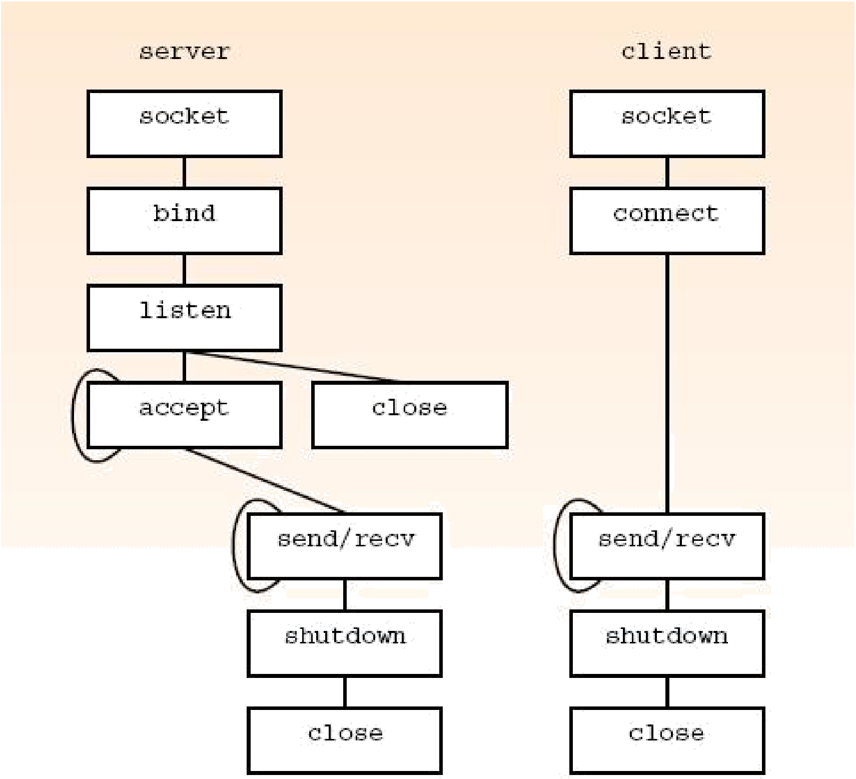
connect: connect to a server

send, recv: (repeated)

shutdown

close

TCP-based sockets



socket API

#include<sys/types.h>

#include<sys /socket.h>

int socket(int domain, int type, int protocol) ;

Returns a file descriptor (called a socket ID) if successful, -1 otherwise. Note that the socket returns a socket descriptor which is the same as a file descriptor.

The *domain* is AF_INET.

The *type* argument can be:

SOCK_STREAM: Establishes a virtual circuit for stream

SOCK_DGRAM: Establishes a datagram for communication

SOCK_SEQPACKET: Establishes a reliable, connection based, two way communication with maximum message size. (This is not available on most machines.)

protocol is usually zero, so that type defines the connection within domain.

bind

#include <sys / types.h>

#include<sys / socket.h>

int bind(int sid, struct sockaddr *addrPtr, int len)

Where

sid: is the socket id

addrPtr: is a pointer to the address family dependent address structure

len: is the size of *addrPtr

Associates a socket id with an address to which other processes can connect. In internet protocol the address is [ipNumber, portNumber]

sockaddr

For the internet family:

```
struct sockaddr_in {
sa_family_t    sin_family;  // = AF INET
in_port_t      sin_port;  // is a port number
struct in_addr   sin_addr; // an IP address
}
```

listen

#include <sys / types.h>

#include <sys / socket.h>

int listen (int sid, int size) ;

Where size it the number of pending connection requests allowed (typically limited by Unix kernels to 5).

Returns the 0 on success, or -1 if failure.

accept

#include <sys / types.h>

#include <sys / socket.h>

int accept(int sid ,struct sockaddr *addrPtr , int *lenPtr)
Returns the socketId and address of client connecting to socket.

if lenPtr or addrPtr equal zero, no address structure is returned.

lenPtr is the maximum size of address structure that can be called, returns the actual value.

Waits for an incoming request, and when received creates a socket for it.

send

#include <sys / types.h>

#include <sys / socket.h>

int send(int sid ,const char *bufferPtr ,int len ,int flag)

Send a message. Returns the number of bytes sent or -1 if failure.

(Must be a bound socket).

flag is either

0: default

MSG OOB: Out-of-band high priority communication

recv

#include <sys / types.h>

#include <sys / socket.h>

int recv (int sid , char *bufferPtr , int len , int flags)

Receive up to len bytes in bufferPtr. Returns the number of bytes received or -1 on failure.

flags can be either

0: default

MSG OOB: out-of-bound message

MSG PEEK: look at message without removing

Shutdown

#include <sys / types.h>

#include <sys / socket.h>

int shutdown (int sid , int how)

Disables sending (how=1 or how=2) or receiving (how=0 or how=2). Returns -1 on failure.

Connect

-this is the first of the client calls

#include <sys / types.h>

#include <sys / socket.h>

int connect (int sid , struct sockaddr *addrPtr , int len)

Specifies the destination to form a connection with (addrPtr), and returns a 0 if successful, -1 otherwise.

Port usage

Note that the initiator of communications needs a fixed port to target communications.

This means that some ports must be reserved for these ―well known‖ ports.

Port usage:

0-1023: These ports can only be binded to by root

1024-5000: well known ports

5001-64K-1: ephemeral ports

Code – Server Program

```
import java.util.*;
import java.net.*;
import java.io.*;
public class tcpclient
{
        public static void main(String args[])
        {

            try
            {
                Scanner ser=new Scanner(System.in);
                Socket s=new Socket("localhost",998);

                DataInputStream dis=new DataInputStream(s.getInputStream());
                DataOutputStream dos=new DataOutputStream (s.getOutputStream());

                dos.writeUTF("connected to 127.0.0.1 \n");

                System.out.println(dis.readUTF());

                System.out.println("\n enter the full path of the the file to be displayed");
                String path=ser.nextLine();

                dos.writeUTF(path);
                System.out.println(new String (dis.readUTF()));

                dis.close();
                dos.close();

                s.close();
                ser.close();
            }
            catch(IOException e)
            {
                System.out.println("IO: "+e.getMessage());
            }
        }
}
```

```
// Client Program
import java.util.*;
import java.net.*;
import java.io.*;
public class tcpserver
{
        public static void main (String args[])
        {
                try
                {
                        ServerSocket s=new ServerSocket(998);

                        System.out.println("server ready \n waiting for connection \n");

                        Socket s1=s.accept();

                        DataOutputStream dos=new DataOutputStream(s1.getOutputStream());
                        DataInputStream dis=new DataInputStream(s1.getInputStream());

                        System.out.println(dis.readUTF());

                        dos.writeUTF("connected to server \n");

                        String path=dis.readUTF();
                        System.out.println("\n request received \n processing.......");
                        try
                        {
                                File myfile=new File(path);
                                Scanner scr=new Scanner(myfile);
                                String st=scr.nextLine();
                                st="\n the context of file is \n "+st;
                                while(scr.hasNextLine())
                                {
                                        st=st + "\n" + scr.nextLine();
                                }
                                dos.writeUTF(st);
                                dos.close();
                                s1.close();
                                scr.close();
                        }
                        catch(FileNotFoundException e)
                        {
                                System.out.println("\n,,,error...\n file not found");
                                dos.writeUTF("...error \n file not found");
                        }
                }
                catch(IOException e)
                {
                        System.out.println("IO: "+e.getMessage());
                }
                finally
                {
                        System.out.println("\n connection terminated");
                    }
                }
            }
```

Output –

Client Side

```
$ javac tcpclient.java
$java tcpclient
connected to server


 enter the full path of the the file to be displayed
/root/cn/udp_tcp/text

 the context of file is
 Hi
how are you
tcp/ip program
112233
```

Server Side

```
$javac tcpserver.java
$ java tcpserver
server ready
 waiting for connection

connected to 127.0.0.1


 request received
 processing.......

 connection terminated
```

Lab Program 10:

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

Code –

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

public class UDPClient
{
	public static void main(String args[])
	{
		DatagramSocket aSocket=null;
		int clientPort=998;

		try
		{
			aSocket=new DatagramSocket(clientPort);
			byte[] buf=new byte[1000];

			DatagramPacket data=new DatagramPacket(buf,buf.length);
			System.out.println("Waiting for server\n");

			aSocket.receive(data);
			byte[] msg=new byte[1000];
			msg=data.getData();
			System.out.println("\n msg:"+(new String(msg,0,data.getLength())));

		}
		catch(SocketException e)
		{
			System.out.println("Socket:" +e.getMessage());
		}
		catch(IOException e)
		{
			System.out.println("IO:" +e.getMessage());
		}
		finally
		{
			if(aSocket!=null)
				aSocket.close();
		}

	}

}
```

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

public class UDPServer {

        public static void main(String args[])
        {

            DatagramSocket aSocket = null;
            Scanner scn=new Scanner(System.in);
            int serverPort =999;

            System.out.println("Server Ready\n Waiting for connection....\n");

            try
            {

                       aSocket=new DatagramSocket(serverPort);

                      byte[] buffer=new byte[1000];

                      System.out.println("\nEnter message to be sent:");
                     String str=scn.nextLine();

                      buffer=str.getBytes();

                      DatagramPacket data = new DatagramPacket(buffer,buffer.length,
                                                          InetAddress.getLocalHost(),998);
                      aSocket.send(data);

            }
            catch(SocketException e)
            {
                      System.out.println("Socket:"+e.getMessage());
            }
            catch(IOException e)
            {
               System.out.println("Io:"+e.getMessage());
            }
            finally
            {
                      System.out.println("\nMessage sent\nConnection terminated");

                       if(aSocket!=null)
                         aSocket.close();

                       scn.close();
            }

        }
}
```

Output –

Client Side

```
$ javac UDPClient.java
$ java UDPClient
Waiting for server


 msg:hello, this is server
```

Server Side

```
$ javac UDPServer.java
$ java UDPServer
Server Ready
 Waiting for connection....


Enter message to be sent:
hello, this is server

Message sent
Connection terminated
```

Lab Program 11:

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

The RSA algorithm can be used for both public key encryption and digital signatures. Its security is based on the difficulty of factoring large integers.

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

A very simple example of RSA encryption

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

1. Select primes p = 11, q = 3.
2. n = pq = 11.3 = 33

   phi = (p-1)(q-1) = 10.2 = 20
3. Choose e=3

   Check gcd(e, p-1) = gcd(3, 10) = 1 (i.e. 3 and 10 have no common factors except 1),

   and check gcd(e, q-1) = gcd(3, 2) = 1

   therefore gcd(e, phi) = gcd(e, (p-1)(q-1)) = gcd(3, 20) = 1
4. Compute d such that ed ≡ 1 (mod phi)

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

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

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

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

   Check: ed-1 = 3.7 - 1 = 20, which is divisible by phi.
5. Public key = (n, e) = (33, 3)

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

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

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

c = m$^{\wedge e}$ mod n = 7$^{\wedge 3}$ mod 33 = 343 mod 33 = 13.

To check decryption we compute

$m' = c^{\wedge d} \bmod n = 13^{\wedge 7} \bmod 33 = 7.$

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

$m' = 13^{\wedge 7} \bmod 33 = 13^{\wedge(3+3+1)} \bmod 33 = 13^{\wedge 3}.13^{\wedge 3}.13 \bmod 33$

$= (13^{\wedge 3} \bmod 33).(13^{\wedge 3} \bmod 33).(13 \bmod 33) \bmod 33$

$= (2197 \bmod 33).(2197 \bmod 33).(13 \bmod 33) \bmod 33$

$= 19.19.13 \bmod 33 = 4693 \bmod 33$

$= 7.$

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

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

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

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

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

{9,6,13,13,16,24,16,19,13,5}

Using our table above, we obtain ciphertext integers c1, c2, ...

{3,18,19,19,4,30,4,28,19,26}

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

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

Key Generation Algorithm

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

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

Encryption

Sender A does the following:-

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

Decryption

Recipient B does the following:-

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

Code –

```
import java.util.*;
import java.math.*;
public class RSA
{
    static BigInteger p,q,e,d,n,phi;
    static int bitLength=256;
    static Scanner S=new Scanner(System.in);
    static Random R=new Random();
    public static void main (String args[])
    {    p=BigInteger.probablePrime(bitLength,R);
         q=BigInteger.probablePrime(bitLength,R);
         n=p.multiply(q);
         e=BigInteger.probablePrime(bitLength/2,R);
         phi=p.subtract(BigInteger.ONE).multiply(q.subtract(BigInteger.ONE));
         while(phi.gcd(e).compareTo(BigInteger.ONE)!=0 && e.compareTo(phi)<0)
         e.add(BigInteger.ONE);
         d=e.modInverse(phi);
         String msg="";
         System.out.print("Enter The Msg : ");
         msg=S.nextLine();
         byte msg_arr[]=msg.getBytes();
         System.out.println("Msg Byte Array : "+display(msg_arr));
         byte en[]=encrypt(msg_arr);
         System.out.println("Encrypted Byte Array : "+display(en));
         byte de[]=decrypt(en);
         System.out.println("Decrypted Byte Array : "+display(de));
         System.out.println("Received Msg : "+ new String(de));
    }
static byte[] encrypt(byte a[])
{ return (new BigInteger(a).modPow(e,n)).toByteArray(); }
static byte[] decrypt(byte a[])
{ return (new BigInteger(a).modPow(d,n)).toByteArray(); }

static String display(byte a[])
    {String s="";
    for(int i=0;i<a.length;i++)
    s+=Byte.toString(a[i]);
    return s;}
  }
```

Output 1-

```
Enter The Msg : HELLO WORLD!2@
Msg Byte Array : 726976767932877982766833564
Encrypted Byte Array : 33-121-10895123127-921071446103-13-98-31-4012178-95-6233-5913106-65-49-41-60112-
556-49131010683-451390-5399-5835-1088589-31-7722-125104-101-114108419566-58-126-12733-1657-69
Decrypted Byte Array : 726976767932877982766833564
Received Msg : HELLO WORLD!2@
```

Output 2-

```
Enter The Msg :  This is a sample
Msg Byte Array:841041051153210511532973211597109112108101
Encrypted Byte Array:7-38-64-487597-725231-45-87-6981-29-17-73-34127-101108-1289-126-769143-126-56-22-
21-27-7819120852868-91-81-47-105-7937-75-48-10681-6651-43-74-126-28-10468-853610941-38-58-127-126-
10910936-63347-69127
Decrypted Byte Array:841041051153210511532973211597109112108101
Received Msg: This is a sample
```

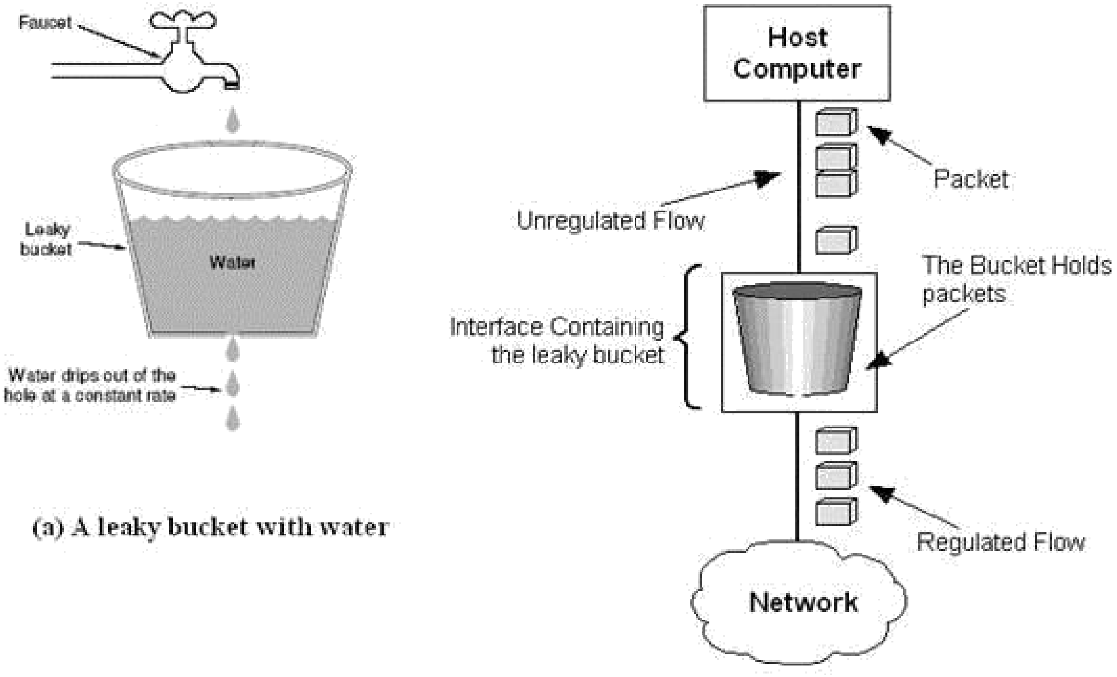
Output 3–

```
Enter The Msg: rsa algorithm
Msg Byte Array:114115973297108103111114105116104109
Encrypted Byte Array:3-56-1172220151939-1055135-16-4771-43127-58-2160117-3011961-46-323011771-125-
5612-175326-89480-23-102-111-94-239089983410156-12-113-128-50-9787-32-49-12033110-113-75-1611-23-
12671-86-852-62-70
Decrypted Byte Array:114115973297108103111114105116104109
Received Msg:rsa algorithm
```

Lab Program 12:

Write a program for congestion control using leaky bucket algorithm.

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



(a) A leaky bucket with water

(b) A leaky bucket with packets

Figure: The leaky bucket traffic shaping algorithm

While leaky bucket eliminates completely bursty traffic by regulating the incoming data flow its main drawback is that it drops packets if the bucket is full. Also, it doesn't take into account the idle process of the sender which means that if the host doesn't transmit data for some time the bucket becomes empty without permitting the transmission of any packet.

Implementation Algorithm:

Steps:

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

Code –

```
import java.util.Scanner;

public class bucket {

        public static void main(String[] args)
        {

                Scanner sc=new Scanner(System.in);

                int bucket=0;
                int op_rate,i,n,bsize;

                System.out.println("Enter the number of packets");
                n=sc.nextInt();

                System.out.println("Enter the output rate of the bucket");
                op_rate=sc.nextInt();

                System.out.println("Enter the bucket size");
                bsize=sc.nextInt();

                System.out.println("Enter the arriving packets(size)");
                int pkt[]=new int[n];
                for(i=0;i<n;i++)
                {
                        pkt[i]=sc.nextInt();
                }

                System.out.println("\nSec\tpsize\tBucket\tAccept/Reject\tpkt_send");
                System.out.println("---------------------------------------------------");
                for(i=0;i<n;i++)
                {
                        System.out.print(i+1+"\t"+pkt[i]+"\t");
                        if(bucket+pkt[i]<=bsize)
                        {
                              bucket+=pkt[i];
                              System.out.print(bucket+"\tAccept\t\t"+min(bucket,op_rate)+"\n" +"");
                              bucket=sub(bucket,op_rate);
                        }
                        else
                        {
                              int reject=(bucket+pkt[i]-bsize);
                              bucket=bsize;
                              System.out.print(bucket+"\tReject "+reject+"\t"+min(bucket,op_rate)+"\n");
                              bucket=sub(bucket,op_rate);
                        }
                }
                while(bucket!=0)
                {
```

```
            System.out.print((++i)+"\t0\t"+bucket+"\tAccept\t\t"+min(bucket,op_rate)+"\t");
            bucket=sub(bucket,op_rate);
        }
    }

    static int min(int a,int b)
    {
        return ((a<b)?a:b);
    }

    static int sub(int a,int b)
    {
        return (a-b)>0?(a-b):0;
    }
}
```

Output 1–

```
Enter the number of packets
4
Enter the output rate of the bucket
7
Enter the bucket size
8
Enter the arriving packets(size)
6 8 9 5

Sec     psize   Bucket  Accept/Reject   pkt_send
----------------------------------------------------
1       6       6       Accept          6
2       8       8       Accept          7
3       9       8       Reject 2        7
4       5       6       Accept          6
```

Output 2–

```
Enter the number of packets
4
Enter the output rate of the bucket
6
Enter the bucket size
8
Enter the arriving packets(size)
4 5 6 10
Sec     psize   Bucket  Accept/Reject   pkt_send
----------------------------------------------------
1       4       4       Accept          4
2       5       5       Accept          5
3       6       6       Accept          6
4       10      8       Reject 2        6
5       0       2       Accept          2
```

Output 3–

```
Enter the number of packets
5
Enter the output rate of the bucket
5
Enter the bucket size
5
Enter the arriving packets(size)
4 6 3 7 5

Sec     psize    Bucket  Accept/Reject   pkt_send
--------------------------------------------------
1       4        4       Accept          4
2       6        5       Reject 1        5
3       3        3       Accept          3
4       7        5       Reject 2        5
5       5        5       Accept          5
```