VISVESVARAYA TECHNOLOGICAL UNIVERSITY
BELAGAVI-590 018,
KARNATAKA

COMPUTER NETWORK LABORATORY
Subject Code: 18CSL57

Scheme: 2018   **Semester:** V

Prepared by

Mrs. Roopa D E   **Ms. Shruthi N**

Assistant Professor   Assistant Professor

IS&E Department   IS&E Department

Department of Information Science & Engineering
Bapuji Institute of Engineering and Technology,
Davangere-04.

Vision and Mission of the Institute

Vision

"To be **center of excellence** recognized **nationally** and **internationally**, in distinctive areas of **engineering education** and **research**, based on a culture of **innovation** and **invention**."

Mission

"BIET contributes to the **growth** and **development** of its students by imparting a board based **engineering education** and **empowering** them to be successful in their chosen field by inculcating in them **positive approach**, **leadership qualities** and **ethical values**."

Vision and Mission of the Department

Vision

"To be the **center of excellence** by adopting technological **innovation** in **academics** and **research** to develop competent **man power** for emerging needs of **society** and i**ndustries."**

Mission

M1: To provide **quality education** to meet the challenges of technological changes to succeed in their **professional career** and **higher education.**

M2: To inculcate the culture of **research, innovation** and **entrepreneur skills** among the students**.**

M3: To groom our students with the quality of **team spirit, leadership skills** and **ethical values**, to share and apply their knowledge for the **benefit of the society.**

Program Outcomes (POs) defined by NBA:

PO1	**Engineering Knowledge:** Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems.
PO2	**Problem Analysis:** Identify, formulate, research literature, and analyze complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences.
PO3	**Design/development of Solutions:** Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for public health and safety and the cultural, societal and environmental considerations.
PO4	**Conduct Investigations of Complex Problems:** Use research-based knowledge and research Methods including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions.
PO5	**Modern Tool Usage:** Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools including prediction and modelling to complex engineering activities with an understanding of the limitations.

PO6	**The Engineer and Society:** Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to the professional engineering practice.
PO7	**Environment and Sustainability:** Understand the impact of the professional engineering solutions in societal and environmental contexts, and demonstrate the knowledge of, and need for sustainable development.
PO8	**Ethics:** Apply ethical principles and commit to professional ethics and responsibilities and norms of engineering practice.
PO9	**Individual and Team Work:** Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.
PO10	**Communication:** Communicate effectively on complex engineering activities with the engineering community and with society at large, such as, being able to comprehend and write effective reports and design documentation, make effective presentations, and give and receive clear instructions.
PO11	**Project Management and Finance:** Demonstrate knowledge and understanding of the engineering and management principles and apply these to one own work, as a member and leader in a team, to manage projects and in multidisciplinary environments.
PO12	**Life-long Learning:** Recognize the need for, and have the preparation and ability to engage in independent and life-long learning in the broadest context of technological change.

Program Educational Objectives (PEOs):

PEO1- The graduates of program will have **excellence** through **principles** and practices of **information technology** combined with **fundamentals of engineering.**

PEO2 - The graduates of program will be prepared in **diverse areas** of **information science** for their **successful careers**, **entrepreneurship** and **higher studies**.

PEO3 - The graduates of program will **work effectively** as an **individual** and in a **team,** exhibiting **leadership qualities**, **communication skills** to meet the **goals** of the **organization.**

PEO4 - The graduates of program will grove their **profession** with **ethics,** management principles to carry **societal responsibilities.**

Program Specific outcomes (PSOs):

PSO1- Problem Solving Skills - Ability to apply **standard principles** and **practices** of Information Technology to propose **feasible ideas** and **solutions** to computational tasks using **appropriate tools** and **techniques.**

PSO2- Knowledge of Information Technology – analyze, design, develop and **test** the computer based software in the areas related to **networks, cloud computing, web technology, data science and IoT.**

PSO3- Profession and Research Ability – Inculcate the knowledge to **excel** in **IT profession**, **entrepreneurship** and **research** with **ethical standards**.

COMPUTER NETWORK LABORATORY **[As per Choice Based Credit System (CBCS) scheme]** **(Effective from the academic year 2017 - 2018)** **SEMESTER – V**			
Subject Code	17CSL57	IA Marks	40
Number of Lecture Hours/Week	01I + 02P	Exam Marks	60
Total Number of Lecture Hours	40	Exam Hours	03
CREDITS – 02			

Description (If any):

For the experiments below modify the topology and parameters set for the experiment and take multiple rounds of reading and analyze the results available in log files. Plot necessary graphs and conclude. Use NS2/NS3.

Lab Experiments:

PART A

1. Implement three nodes point – to – point network with duplex links between them. Set the queue size, vary the bandwidth and find the number of packets dropped.
2. Implement transmission of ping messages/trace route over a network topology consisting of 6 nodes and find the number of packets dropped due to congestion.
3. Implement an Ethernet LAN using n nodes and set multiple traffic nodes and plot congestion window for different source / destination.
4. Implement simple ESS and with transmitting nodes in wire-less LAN by simulation and determine the performance with respect to transmission of packets.
5. Implement and study the performance of GSM on NS2/NS3 (Using MAC layer) or equivalent environment.
6. Implement and study the performance of CDMA on NS2/NS3 (Using stack called Call net) or equivalent environment.

PART B

Implement the following in Java:
7. Write a program for error detecting code using CRC-CCITT (16- bits).
8. Write a program to find the shortest path between vertices using bellman-ford algorithm.
9. Using TCP/IP sockets, write a client – server program to make the client send the file name and to make the server send back the contents of the requested file if present.
10. Write a program on datagram socket for client/server to display the messages on client side, typed at the server side.
11. Write a program for simple RSA algorithm to encrypt and decrypt the data.
12. Write a program for congestion control using leaky bucket algorithm.

Study Experiment / Project:

NIL

Course outcomes: The students should be able to:

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

Conduction of Practical Examination:

1. All laboratory experiments are to be included for practical examination.

2. Students are allowed to pick one experiment from part A and part B with lot.
3. Strictly follow the instructions as printed on the cover page of answer script
4. Marks distribution: Procedure + Conduction + Viva: 100
Part A: 8+35+7   =50
Part B: 8+35+7   =50
5. Change of experiment is allowed only once and marks allotted to the procedure part to be made zero.

Mapping of CO with PO and PSO's

Course Code	PO1	PO2	PO3	PO4	PO5	PO6	PO7	PO8	PO9	PO10	PO11	PO12	PSO1	PSO2	PSO3
C307.1	2	3	3	3	1				2	1		2	2	3	
C307.2	2	2	2	2	1				2	1		2	2	2	
C307.3	2	3	1	1				2				2	2	2	2
C307.4	2	3	3	3	1					1		2	2	2	
C307.5		2	3	3	3				1			2	2		
CO-PO-PSO LEVEL	2.00	2.60	2.40	2.40	1.50			2.00	1.67	1.00		2.00	2.00	2.25	2.00
%OF CO-PO-PSO	66.67	86.67	80	80	50			66.7	55.56	33.33		66.67	66.67	75.00	66.67

TABLE OF CONTENTS

BASICS OF NS2

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
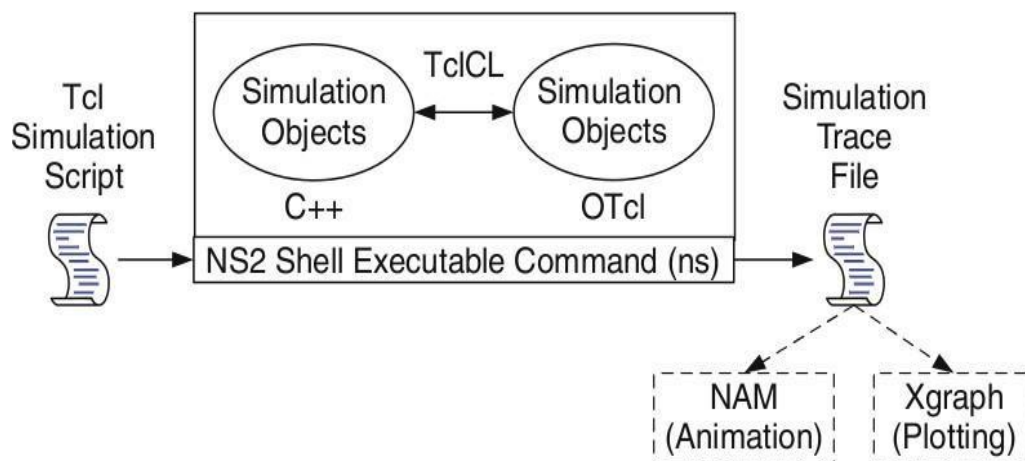
Tcl scripting

- 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.

1. Initialization and termination aspects of the ns simulator.

2. Definition of network nodes, links, queues and topology.

3. Definition of agents and of applications.

4. The nam visualization tool.

5. Tracing and random variables.

Initialization and Termination of TCL Script in NS-2

An ns simulation starts with the command

> **set ns [new Simulator]**

Which is thus the first line in the tcl script? This line declares a new variable as using the set command, you can call this variable as you wish, In general people declares it as ns because

it is an instance of the Simulator class, so an object the code[new Simulator] is indeed the installation of the class Simulator using the reserved word new.

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

#Open the Trace file

```
set tracefile1 [open out.tr w]

$ns trace-all $tracefile1
```

#Open the NAM trace file

```
set namfile [open out.nam w]

$ns namtrace-all $namfile
```

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

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

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

#Define a „finish" procedure

```
Proc finish { } {

global ns tracefile1 namfile

$ns flush-trace

Close $tracefile1

Close $namfile

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]

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

```
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 "$ftp stop"**
>
> **$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

Part-A

Experiment No: 1

THREE NODE POINT TO POINT NETWORK

Aim: *Implement three nodes point – to – point network with duplex links between them. Set the queue size, vary the bandwidth and find the number of packets dropped.*

```
set ns [new Simulator]                          # Letter S is capital
set nf [open lab1.nam w]                         # open a nam trace file in write mode
$ns namtrace-all $nf                             # nf  nam filename
set tf [open lab1.tr w]                          # tf  trace filename
$ns trace-all $tf

proc finish { } {
        global ns nf tf
        $ns flush-trace                          # clears trace file contents
        close $nf
        close $tf
        exec nam lab1.nam &
        exit 0
}
set n0 [$ns node]                                # creates 3 nodes
set n2 [$ns node]
set n3 [$ns node]

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

set udp0 [new Agent/UDP]                          # attaching transport layer protocols
$ns attach-agent $n0 $udp0
set cbr0 [new Application/Traffic/CBR]            # attaching application layer protocols
$cbr0 set packetSize_ 500
$cbr0 set interval_ 0.005
$cbr0 attach-agent $udp0

set null0 [new Agent/Null]                        # creating sink(destination) node
$ns attach-agent $n3 $null0
$ns connect $udp0 $null0

$ns at 0.1 "$cbr0 start"
$ns at 1.0 "finish"
$ns run
```

AWK file: *(Open a new editor using "vi command" and write awk file and save with ".awk" extension)*
#immediately after BEGIN should open braces „{„

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

Steps for execution

➢ *Open gedit editor and type program. Program name should have the extension " .tcl "*

> *[root@localhost ~]# gedit lab1.tcl*

➢ *Save the program and close the file.*

➢ *Open gedit editor and type **awk** program. Program name should have the extension ".awk "*

> *[root@localhost ~]# gedit lab1.awk*

➢ *Save the program and close the file.*

➢ *Run the simulation program*

> *[root@localhost~]# ns lab1.tcl*

➢ *Here **"ns"** indicates network simulator. We get the topology shown in the snapshot.*

➢ *Now press the play button in the simulation window and the simulation will begins.*

➢ *After simulation is completed run **awk file** to see the output ,*

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

➢ *To see the trace file contents open the file as ,*

> *[root@localhost~]# gedit lab1.tr*

Trace file contains 12 columns:

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

| Contents of Trace File | Topology | Output |

Experiment No: 2

TRANSMISSION OF PING MESSAGE

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

```
set ns [ new Simulator ]

set nf [ open lab2.nam w ]
$ns namtrace-all $nf

set tf [ open lab2.tr w ]
$ns trace-all $tf

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

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

set p1 [new Agent/Ping] # letters A and P should be capital
$ns attach-agent $n0 $p1
$p1 set packetSize_ 50000
$p1 set interval_ 0.0001

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

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


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

set p5 [new Agent/Ping] # letters A and P should be capital
$ns attach-agent $n5 $p5

$ns queue-limit $n0 $n4 5
$ns queue-limit $n2 $n4 3
```

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

$ns connect $p1 $p5
$ns connect $p3 $p4

proc finish { } {
global ns nf tf
$ns flush-trace
close $nf
close $tf
exec nam lab2.nam &
exit 0
}
$ns at 0.1 "$p1 send"
$ns at 0.2 "$p1 send"
$ns at 0.3 "$p1 send"
$ns at 0.4 "$p1 send"
$ns at 0.5 "$p1 send"
$ns at 0.6 "$p1 send"
$ns at 0.7 "$p1 send"
$ns at 0.8 "$p1 send"
$ns at 0.9 "$p1 send"
$ns at 1.0 "$p1 send"

$ns at 0.1 "$p3 send"
$ns at 0.2 "$p3 send"
$ns at 0.3 "$p3 send"
$ns at 0.4 "$p3 send"
$ns at 0.5 "$p3 send"
$ns at 0.6 "$p3 send"
$ns at 0.7 "$p3 send"
$ns at 0.8 "$p3 send"
$ns at 0.9 "$p3 send"
$ns at 1.0 "$p3 send"

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

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

```
BEGIN{
drop=0;
}
```

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

Steps for execution

> *Open gedit editor and type program. Program name should have the extension " .tcl "*

> ### [root@localhost ~]# gedit lab2.tcl

> *Save the program and close the file.*
> *Open gedit editor and type **awk** program. Program name should have the extension ".awk "*

> ### [root@localhost ~]# gedit lab2.awk

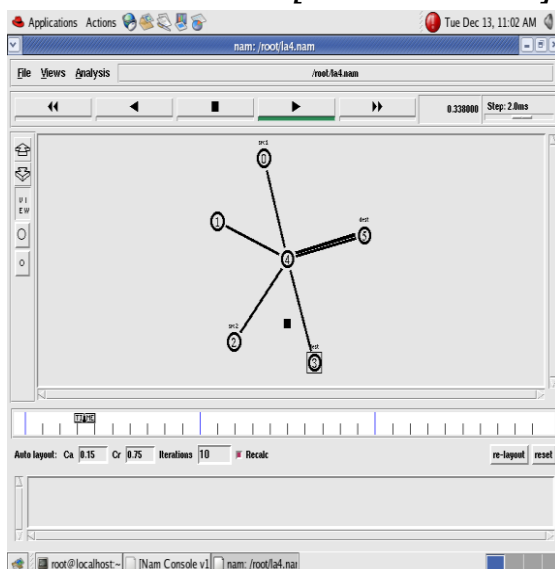> *Save the program and close the file.*
> *Run the simulation program*

> ### [root@localhost~]# ns lab2.tcl

> *Here **"ns"** indicates network simulator. We get the topology shown in the snapshot.*
> *Now press the play button in the simulation window and the simulation will begins.*
> *After simulation is completed run **awk file** to see the output ,*

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

> *To see the trace file contents open the file as ,*

> ### [root@localhost~]# gedit lab2.tr

Topology   **Output**

Output

Experiment No: 3

ETHERNET LAN USING N-NODES WITH MULTIPLE TRAFFIC

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

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

set n0 [$ns node]
$n0 color "magenta"
$n0 label "src1"
set n1 [$ns node]
set n2 [$ns node]
$n2 color "magenta"
$n2 label "src2"
set n3 [$ns node]
$n3 color "blue"
$n3 label "dest2"
set n4 [$ns node]
set n5 [$ns node]
$n5 color "blue"
$n5 label "dest1"

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

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

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

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

$ns connect $tcp0 $sink5

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

set ftp2 [new Application/FTP]
$ftp2 attach-agent $tcp2
```

$ftp2 set packetSize_ 600
$ftp2 set interval_ 0.001
set sink3 [new Agent/TCPSink]
$ns attach-agent $n3 $sink3
$ns connect $tcp2 $sink3

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

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

$tcp0 trace cwnd_ # must put underscore (_) after cwnd and no space between them
$tcp2 trace cwnd_

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

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

$ns at 16 "finish"
$ns run

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

cwnd:- means congestion window

```
BEGIN {
}
{
if($6= ="cwnd_")                    # don"t leave space after writing cwnd_
printf("%f\t%f\t\n",$1,$7);         # you must put \n in printf
}
END {
}
```

Steps for execution

- ➢ *Open gedit editor and type program. Program name should have the extension " .tcl "*

  [root@localhost ~]# gedit lab3.tcl

- ➢ *Save the program and close the file.*

- ➢ *Open gedit editor and type **awk** program. Program name should have the extension ".awk "*

  [root@localhost ~]# gedit lab3.awk

- ➢ *Save the program and close the file.*

- ➢ *Run the simulation program*

  [root@localhost~]# ns lab3.tcl

- ➢ *Here **"ns"** indicates network simulator. We get the topology shown in the snapshot.*

- ➢ *Now press the play button in the simulation window and the simulation will begins.*

- ➢ *After simulation is completed run **awk file** to see the output ,*

  [root@localhost~]# awk –f lab3.awk file1.tr > a1
  [root@localhost~]# awk –f lab3.awk file2.tr > a2
  [root@localhost~]# xgraph a1 a2

- ➢ *Here we are using the congestion window trace files i.e. **file1.tr** and **file2.tr** and we are redirecting the contents of those files to new files say **a1** and **a2** using **output redirection operator (>)**.*

- ➢ *To see the trace file contents open the file as ,*

Topolgy:
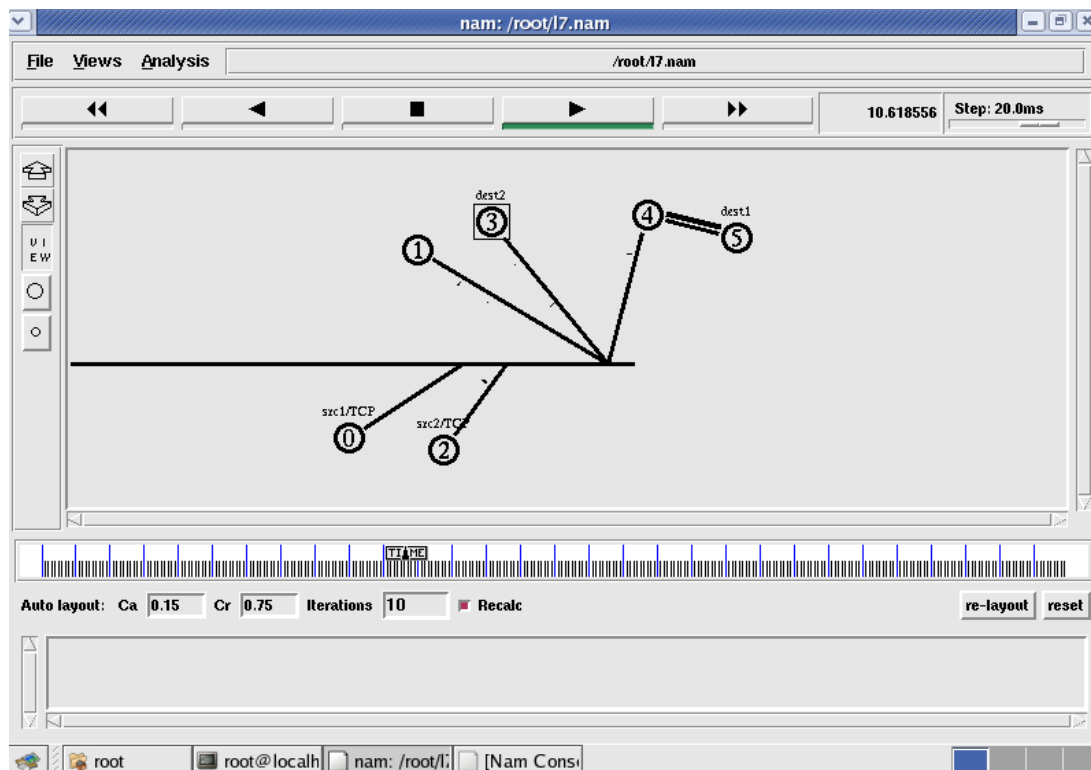
Output:
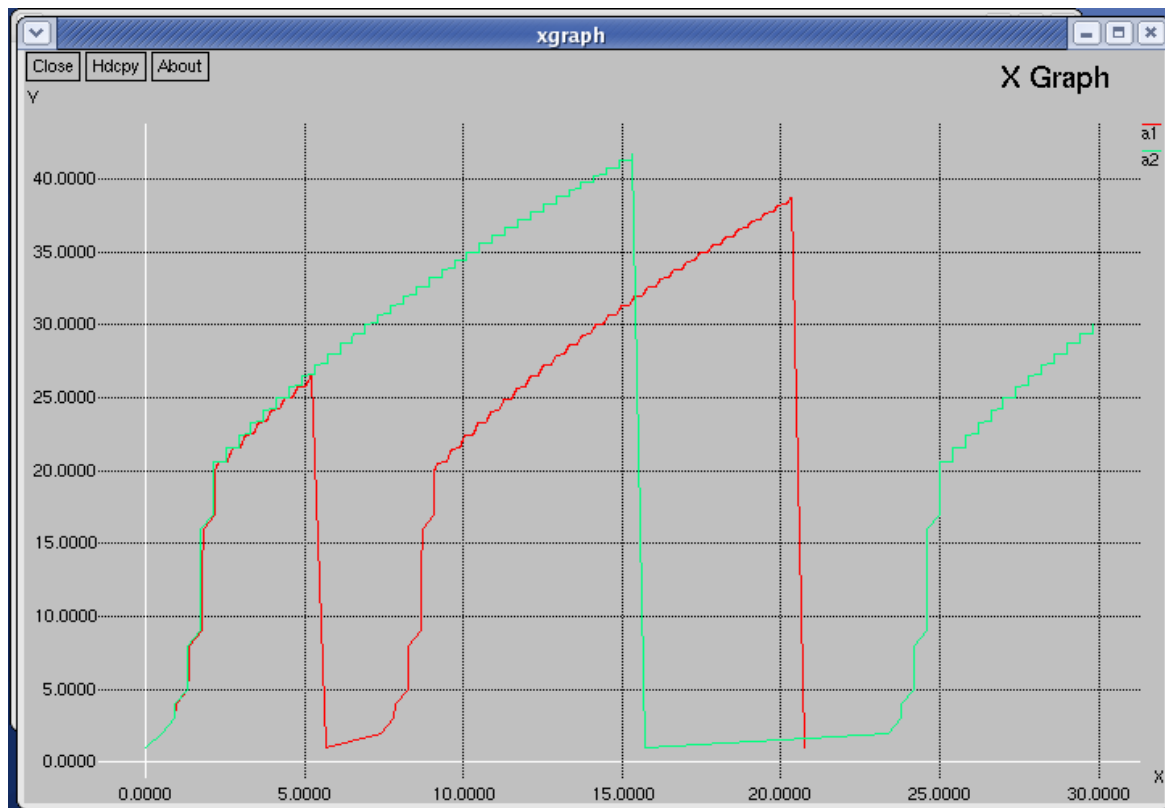
Experiment No: 4

<div align="center">

SIMPLE ESS WITH WIRELESS LAN

</div>

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

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

$ns node-config -adhocRouting DSDV \
            -llType LL \
            -macType Mac/802_11 \
            -ifqType Queue/DropTail \
            -ifqLen 50 \
            -phyType Phy/WirelessPhy \
            -channelType Channel/WirelessChannel \
            -prrootype 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 file: *(Open a new editor using "gedit command" and write awk file and save with ".awk" extension)*

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

```
        {
                count2++
                pack2=pack2+$8
                time2=$2
        }
}
```

END{
printf("The Throughput from n0 to n1: %f Mbps \n‖, ((count1*pack1*8)/(time1*1000000)));
printf("The Throughput from n1 to n2: %f Mbps", ((count2*pack2*8)/(time2*1000000)));
}

Steps for execution

> *Open gedit editor and type program. Program name should have the extension " .tcl "*
>> ***[root@localhost ~]# gedit lab4.tcl***
> *Save the program and close the file.*
> *Open gedit editor and type **awk** program. Program name should have the extension ".awk "*
>> ***[root@localhost ~]# gedit lab4.awk***
> *Save the program and close the file.*
> *Run the simulation program*
>> ***[root@localhost~]# ns lab4.tcl***
> *Here **"ns"** indicates network simulator. We get the topology shown in the snapshot.*
> *Now press the play button in the simulation window and the simulation will begins.*
> *After simulation is completed run **awk file** to see the output ,*
>> ***[root@localhost~]# awk –f lab4.awk lab4.tr***
> *To see the trace file contents open the file as ,*
>> ***[root@localhost~]# gedit lab4.tr***

Output:
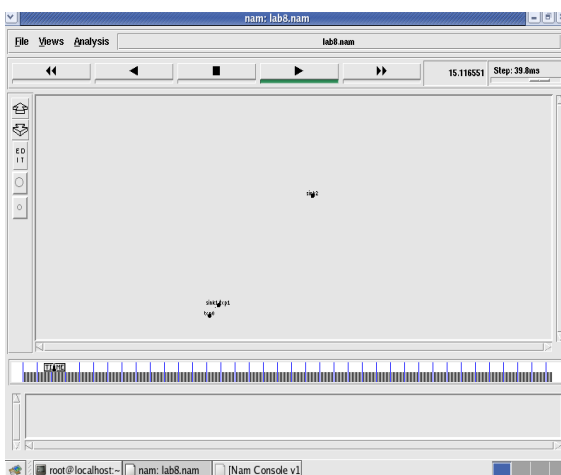
Node 1 and 2 are communicating   Node 2 is moving towards node 3

```
nam: lab8.nam
File  Views  Analysis                          lab8.nam
◀◀   ◀   ■   ▶   ▶▶   198.498787  Step: 398.1ms
```
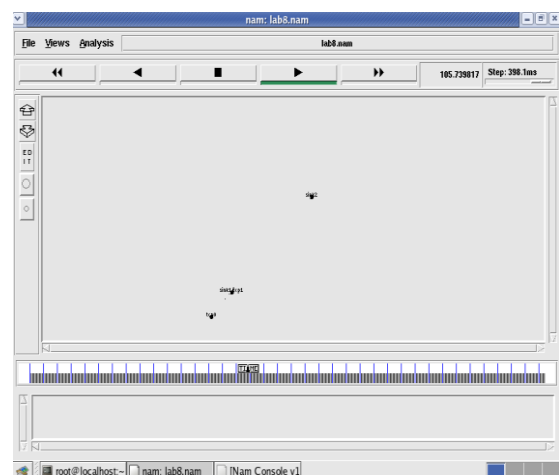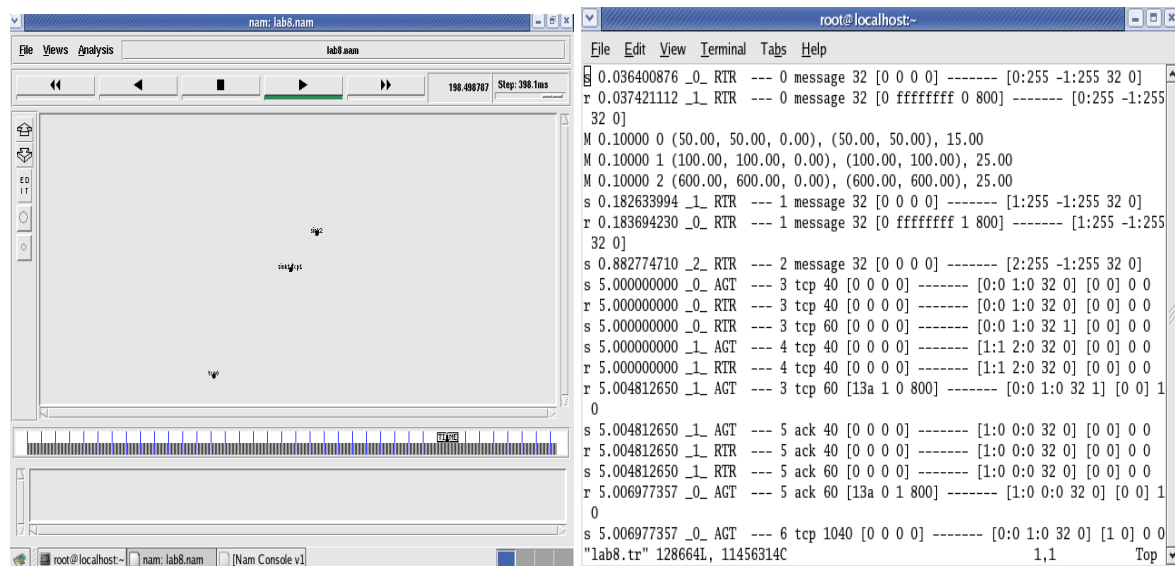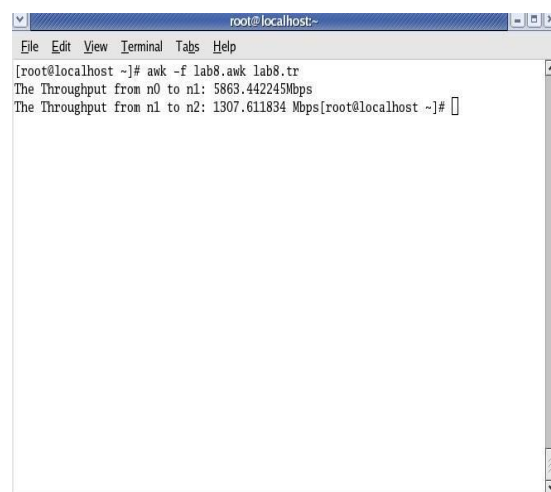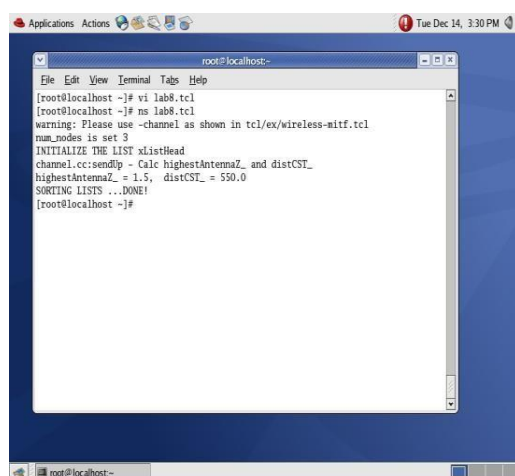
Node 2 is coming back from node 3 towards node1   Trace File

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

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


```
Applications  Actions                                    Tue Dec 14, 3:30 PM
root@localhost:~
File  Edit  View  Terminal  Tabs  Help
[root@localhost ~]# vi lab8.tcl
[root@localhost ~]# ns lab8.tcl
warning: Please use -channel as shown in tcl/ex/wireless-mitf.tcl
num_nodes is set 3
INITIALIZE THE LIST xListHead
channel.cc:sendUp - Calc highestAntennaZ_ and distCST_
highestAntennaZ_ = 1.5,  distCST_ = 550.0
SORTING LISTS ...DONE!
[root@localhost ~]#
```

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

Experiment No: 5

PERFORMANCE OF GSM USING MAC LAYER

Aim: *Implement and study the performance of GSM on NS2/NS3 (Using MAC layer) or*
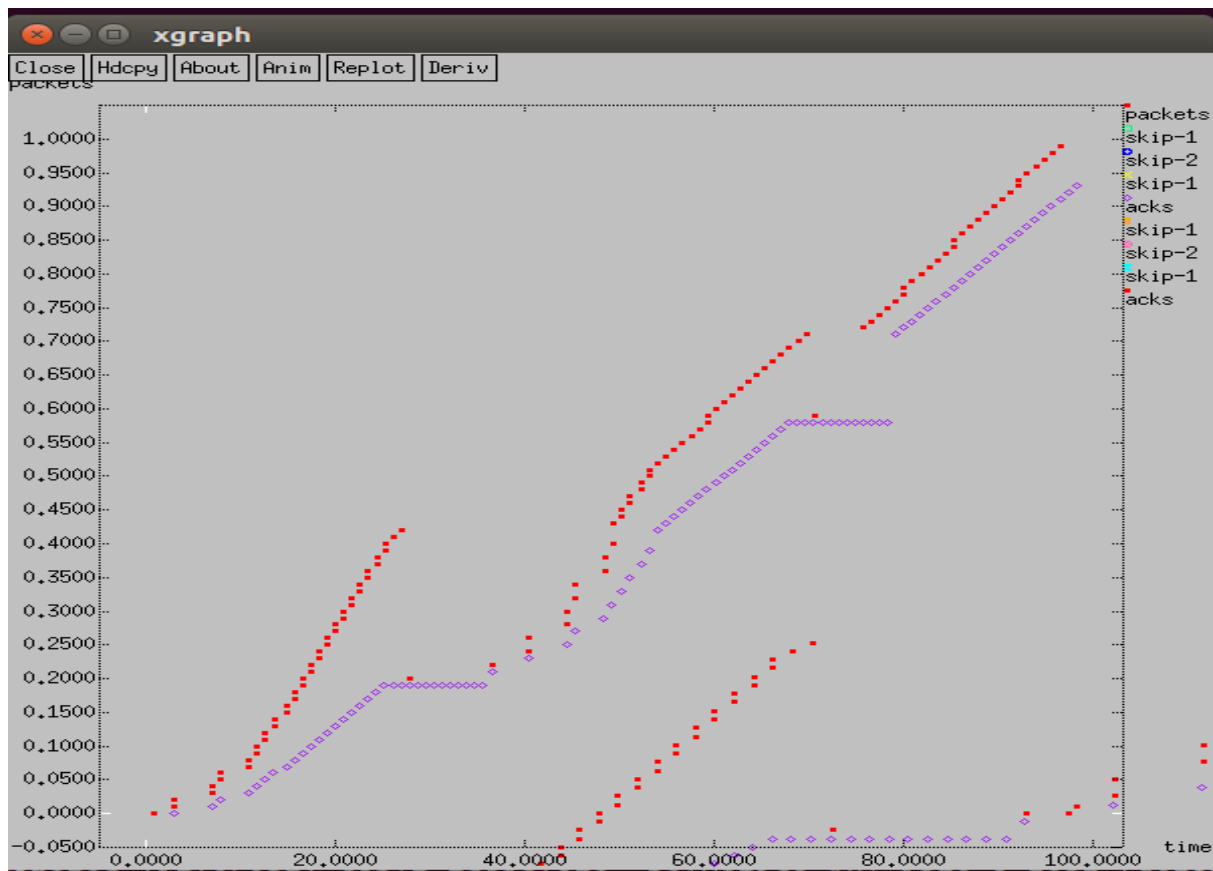
equivalent environment.
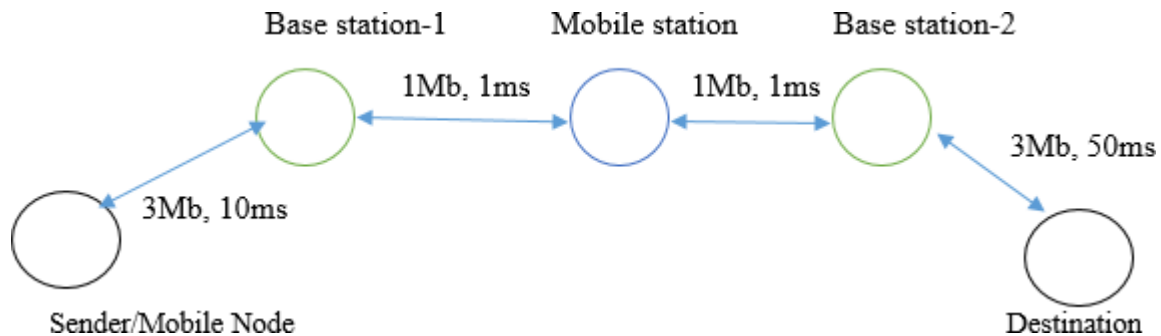
Source Code:
 # General Parameters
set opt(title) zero ;
set opt(stop) 100 ;# Stop time.
set opt(ecn) 0 ;
 # Topology
set opt(type) gsm ;#type of link:
set opt(secondDelay) 55 ;# average delay of access links in ms
 # AQM parameters
set opt(minth) 30 ;
set opt(maxth) 0 ;
set opt(adaptive) 1 ;# 1 for 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
```

```
+ 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
+ 3.05376 0 3 tcp 1040 ------- 0 0.0 4.0 2 3
- 3.056533 0 3 tcp 1040 ------- 0 0.0 4.0 2 3
r 3.106533 0 3 tcp 1040 ------- 0 0.0 4.0 1 2
+ 3.106533 3 1 tcp 1040 ------- 0 0.0 4.0 1 2
- 3.106533 3 1 tcp 1040 ------- 0 0.0 4.0 1 2
r 3.109307 0 3 tcp 1040 ------- 0 0.0 4.0 2 3
+ 3.109307 3 1 tcp 1040 ------- 0 0.0 4.0 2 3
- 3.9732 3 1 tcp 1040 ------- 0 0.0 4.0 2 3
r 4.4732 3 1 tcp 1040 ------- 0 0.0 4.0 1 2
+ 4.4732 1 2 tcp 1040 ------- 0 0.0 4.0 1 2
- 4.4732 1 2 tcp 1040 ------- 0 0.0 4.0 1 2
r 5.339867 3 1 tcp 1040 ------- 0 0.0 4.0 2 3
+ 5.339867 1 2 tcp 1040 ------- 0 0.0 4.0 2 3
- 5.339867 1 2 tcp 1040 ------- 0 0.0 4.0 2 3
```

Loading file '/home/viji/ns-allinone-2.35/ns-2.35/tcl/ex/wireless-scripts/out.tr'...   Plain Text ▼   Tab Width: 8 ▼   Ln 1, Col 1   ▼   INS
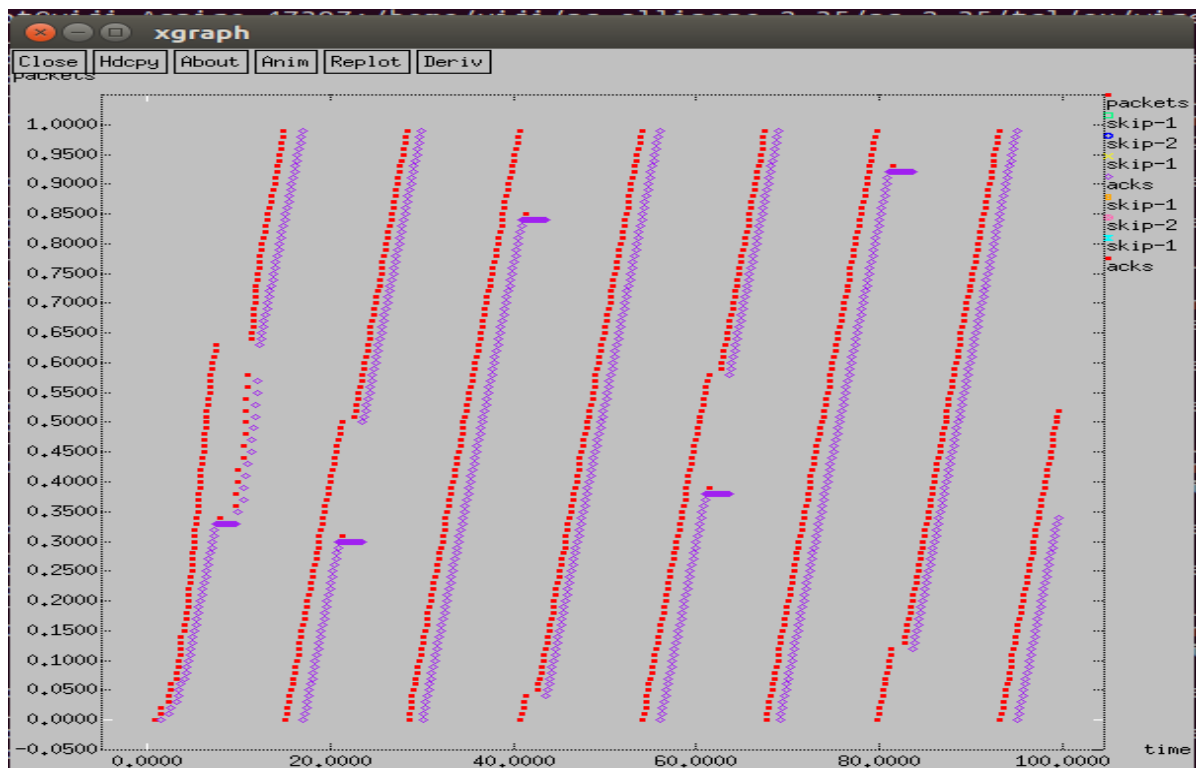
Experiment No: 6

PERFORMANCE OF CDMA

Aim: Implement and study the performance of CDMA on NS2/NS3 (Using stack called Call net) or equivalent environment.
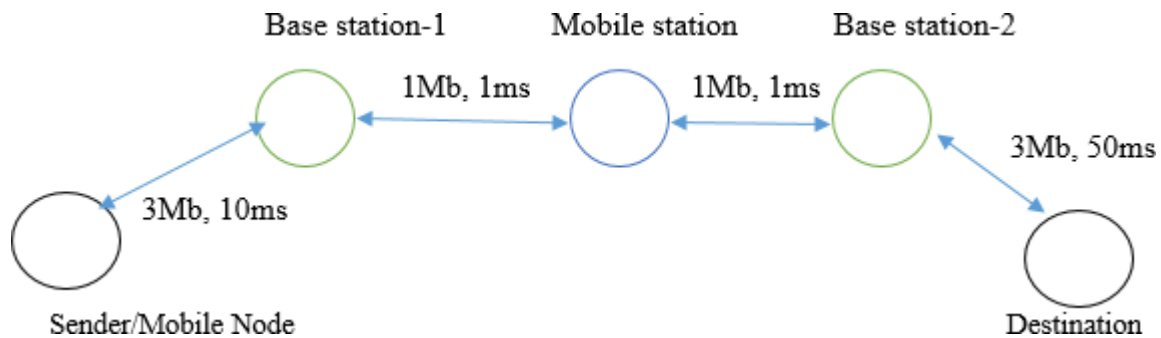
Source Code:
```
# General Parameters
set opt(title) zero ;
set opt(stop) 100 ;# Stop time.
set opt(ecn) 0 ;
# Topology
set opt(type) umts ;#type of link:
set opt(secondDelay) 55 ;# average delay of access links in ms
# AQM parameters
set opt(minth) 30 ;
set opt(maxth) 0 ;
set opt(adaptive) 1 ;# 1 for 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
```

Output:

CDMA Trace file:

Part-B

Experiment No: 1

Error Detecting Code Using CRC-CCITT (16-bit)

Aim: Write a Program for ERROR detecting code using CRC-CCITT (16bit).

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

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

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

young. It might only look a little bit strange. Also notations differ between countries, but the method is similar.

With decimal calculations you can quickly check that 109 divided by 19 gives a quotient of 5 with 14 as the remainder. But what we also see in the scheme is that every bit extra to check only costs one binary comparison and in 50% of the cases one binary subtraction. You can easily increase the number of bits of the test data string—for example to 56 bits if we use our example value "*Lammert*"—and the result can be calculated with 56 binary comparisons and an average of 28 binary subtractions. This can be implemented in hardware directly with only very few transistors involved. Also software algorithms can be very efficient.

All of the CRC formulas you will encounter are simply checksum algorithms based on modulo-2 binary division where we ignore carry bits and in effect the subtraction will be equal to an *exclusive or* operation. Though some differences exist in the specifics across different CRC formulas, the basic mathematical process is always the same:

- The message bits are appended with *c* zero bits; this *augmented message* is the dividend
- A predetermined *c+1*-bit binary sequence, called the *generator polynomial*, is the divisor
- The checksum is the *c*-bit remainder that results from the division operation

Table 1 lists some of the most commonly used generator polynomials for 16- and 32-bit CRCs. Remember that the width of the divisor is always one bit wider than the remainder. So, for example, you'd use a 17-bit generator polynomial whenever a 16-bit checksum is required.

Table 1: International Standard CRC Polynomials

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

Error detection with CRC

Consider a message represented by the polynomial M(x)

Consider a *generating polynomial* G(x)

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

Note this G(x) is prime.

Steps:

1. Multiply M(x) by highest power in G(x). i.e. Add So much zeros to M(x).

2. Divide the result by G(x). The remainder = C(x).

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

3. If: x div y gives remainder c

    that means: x = n y + c

    Hence (x-c) = n y

    (x-c) div y gives remainder 0

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

    Hence (x+c) div y gives remainder 0

4. Transmit: T(x) = M(x) + C(x)

5. Receiver end: Receive T(x). Divide by G(x), should have remainder 0.

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

then G(x) will cover (n+1) bits

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

    o   Used in: 802.16 (along with error *correction*).

*   CRC-CCITT:

    $x^{16}+x^{12}+x^5+1$

    o   Used in: HDLC, SDLC, PPP default

*   IBM-CRC-16 (ANSI):

    $x^{16}+x^{15}+x^2+1$

- 802.3:

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

- Used in: Ethernet, PPP rootion

Source Code:

```java
import java.util.*;
class crc
{       void div(int a[],int k)
        {       int gp[]={1,0,0,0,1,0,0,0,0,0,0,1,0,0,0,0,1};
                int count=0;
                for(int i=0;i<k;i++)
                {
                        if(a[i]==gp[0])
                        {
                                for(int j=i;j<17+i;j++)
                                {
                                        a[j]=a[j]^gp[count++];
                                }
                                count=0;
                        }
                }
        }
public static void main(String args[])
{
        int a[]=new int[100];
        int b[]=new int[100];
        int len,k;
        crc ob=new crc();
        System.out.println("Enter the length of Data Frame:");
        Scanner sc=new Scanner(System.in);
        len=sc.nextInt();
        int flag=0;
        System.out.println("Enter the Message:");
        for(int i=0;i<len;i++)
        {       a[i]=sc.nextInt();
        }
        for(int i=0;i<16;i++)
        {       a[len++]=0;
        }
        k=len-16;
        for(int i=0;i<len;i++)
        {       b[i]=a[i];
        }
        ob.div(a,k);
                for(int i=0;i<len;i++)
                a[i]=a[i]^b[i];
        System.out.println("Data to be transmitted: ");
        for(int i=0;i<len;i++)
```

```
        {
                System.out.print(a[i]+" ");
        }
        System.out.println();
        System.out.println("Enter the Reveived Data: ");
        for(int i=0;i<len;i++)
        {
                a[i]=sc.nextInt();
        }
        ob.div(a, k);

        for(int i=0;i<len;i++)
        {
                if(a[i]!=0)
                {
                        flag=1;
                        break;
                }
        }
        if(flag==1)
                System.out.println("error in data");
        else
                System.out.println("no error");
                }
        }
```

Output:

Enter the length of Data Frame: 4

Enter the Message: 1 0 1 1

Data to be transmitted: 1 0 1 1 1 0 1 1 0 0 0 1 0 1 1 0 1 0 1 1

Enter the Reveived Data: 1 0 1 1 1 0 1 1 0 0 0 0 0 1 1 0 1 0 1 1

ERROR in Recived Data

****

Experiment No: 2

<div align="center">

Bellman-ford Algorithm

</div>

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

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

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

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

Implementation Algorithm:

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

ii.   if I have an entry for D with metric M", add (D, P, M') to my routing
      table if M' < M"

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

Source Code:

```java
import java.util.Scanner;
public class BellmanFord
{       private int D[];
        private int num_ver;
        public static final int MAX_VALUE = 999;
        public BellmanFord(int n)
        {       this.n=n;
                D = new int[n+1];
        }
        public void shortest(int s,int A[][])
        {       for (int i=1;i<=n;i++)
                {       D[i]=MAX_VALUE;
                }       D[s] = 0;
                for(int k=1;k<=n-1;k++)
                {       for(int i=1;i<=n;i++)
                        {       for(int j=1;j<=n;j++)
                                {       if(A[i][j]!=MAX_VALUE)
                                        {       if(D[j]>D[i]+A[i][j])
                                                D[j]=D[i]+A[i][j];
                                        }
                                }
                        }
                }
                for(int i=1;i<=n;i++)
                        {       for(int j=1;j<=n;j++)
                                {       if(A[i][j]!=MAX_VALUE)
                                        {       if(D[j]>D[i]+A[i][j])
                                                {
System.out.println("The Graph contains negative egde cycle");
                                                return;
                                        }       }
                                }
                        }
for(int i=1;i<=n;i++)
{
System.out.println("Distance of source " + s + " to "+ i + " is " + D[i]);
}
}
public static void main(String[ ] args)
{       int n=0,s;
        Scanner sc = new Scanner(System.in);
        System.out.println("Enter the number of vertices");
        n = sc.nextInt();
```

```
        int A[][] = new int[n+1][n+1];
        System.out.println("Enter the Weighted matrix");
        for(int i=1;i<=n;i++)
        {       for(int j=1;j<=n;j++)
                {      A[i][j]=sc.nextInt();
                       if(i==j)
                       {       A[i][j]=0;
                               continue;
                       }
                       if(A[i][j]==0)
                       {       A[i][j]=MAX_VALUE;
                       }
                }
        }
        System.out.println("Enter the source vertex");
        s=sc.nextInt();
        BellmanFord b = new BellmanFord(n);
        b.shortest(s,A);
        sc.close();
}
}
```

Output:
Enter the number of vertices
4
Enter the adjacency matrix
0	**5**	**0**	**0**
5	**0**	**3**	**4**
0	**3**	**0**	**2**
0	**4**	**2**	**0**

Enter the source vertex
2
Distance of source 2 to 1 is 5
Distance of source 2 to 2 is 0
Distance of source 2 to 3 is 3
Distance of source 2 to 4 is 4

Experiment No: 3

Client-server using TCP/IP sockets

Aim: *Using TCP/IP Sockets, write a client-server program to make client sending the file name and the server to send back the contents of the requested file if present. Implement the above program using as message queues or FIFOs as IPC channels.*

Socket is an interface which enables the client and the server to communicate and pass on information from one another. Sockets provide the communication mechanism between two computers using TCP. A client program creates a socket on its end of the communication and attempts to connect that socket to a server. When the connection is made, the server creates a socket object on its end of the communication. The client and the server can now communicate by writing to and reading from the socket.

Source Code:
TCP Client

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

class Client
{
public static void main(String args[])throws Exception
{
String address = "";
Scanner sc=new Scanner(System.in);
System.out.println("Enter Server Address: ");
address=sc.nextLine();
//create the socket on port 5000
Socket s=new Socket(address,5000);
DataInputStream din=new DataInputStream(s.getInputStream());
DataOutputStream dout=new DataOutputStream(s.getOutputStream());
BufferedReader br=new BufferedReader(new InputStreamReader(System.in));
System.out.println("Send Get to start...");
String str="",filename="";
try
{
while(!str.equals("start"))
str=br.readLine();
dout.writeUTF(str);
dout.flush();
filename=din.readUTF();
```

```
System.out.println("Receving file: "+filename);
filename="client"+filename;
System.out.println("Saving as file: "+filename);
long sz=Long.parseLong(din.readUTF());
System.out.println ("File Size: "+(sz/(1024*1024))+" MB");
byte b[]=new byte [1024];
System.out.println("Receving file..");
FileOutputStream fos=new FileOutputStream(new File(filename),true);
long bytesRead;
do
{
bytesRead = din.read(b, 0, b.length);
fos.write(b,0,b.length);
}
while(!(bytesRead<1024));
System.out.println("Comleted");
fos.close();
dout.close();
s.close();
}
catch(EOFException e)
{
//do nothing
}

}
}
```
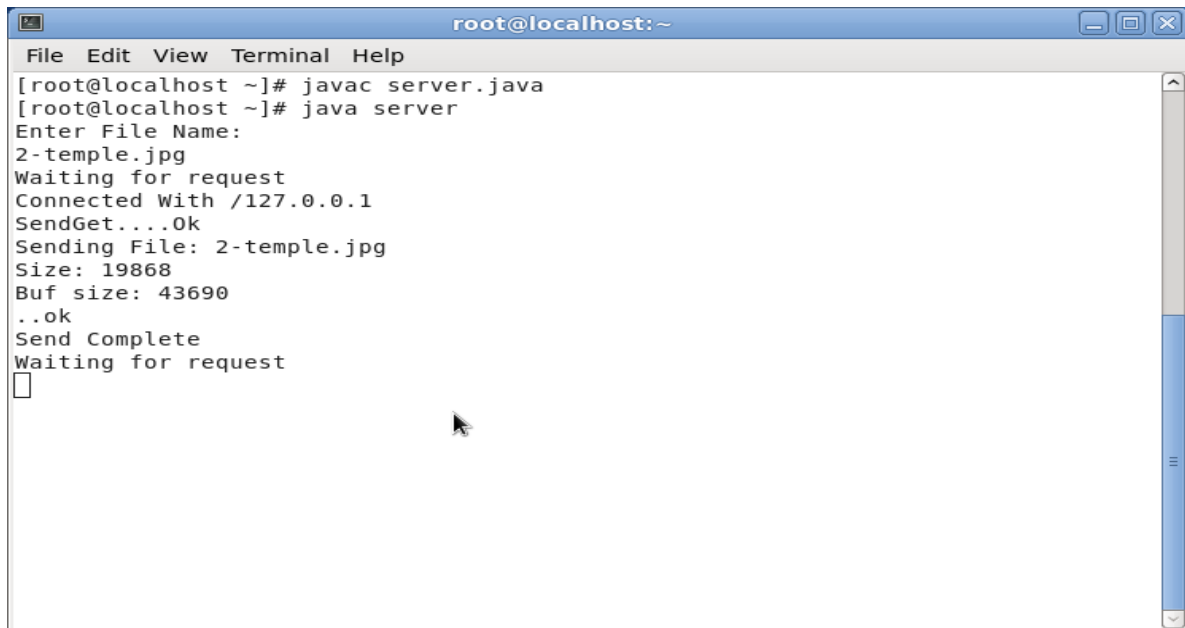
TCP Server

```
import java.io.DataInputStream;
import java.io.DataOutputStream;
import java.io.File;
import java.io.FileInputStream;
import java.net.ServerSocket;
import java.net.Socket;
import java.util.Scanner;
class Server
{
public static void main(String args[])throws Exception
{
String filename;
System.out.println("Enter File Name: ");
Scanner sc=new Scanner(System.in);
filename=sc.nextLine();
sc.close();
while(true)
{
//create server socket on port 5000
ServerSocket ss=new ServerSocket(5000);
System.out.println ("Waiting for request");
Socket s=ss.accept();
System.out.println ("Connected With "+s.getInetAddress().toString());
DataInputStream din=new DataInputStream(s.getInputStream());
DataOutputStream dout=new DataOutputStream(s.getOutputStream());
```

```
try
{
String str="";
str=din.readUTF();
System.out.println("SendGet ...Ok");
if(!str.equals("stop"))
{ System.out.println("Sending File: "+filename);
dout.writeUTF(filename);
dout.flush();
File f=new File(filename);
FileInputStream fin=new FileInputStream(f);
long sz=(int) f.length();
byte b[]=new byte [1024];
 int read;
dout.writeUTF(Long.toString(sz));
dout.flush();
System.out.println ("Size: "+sz);

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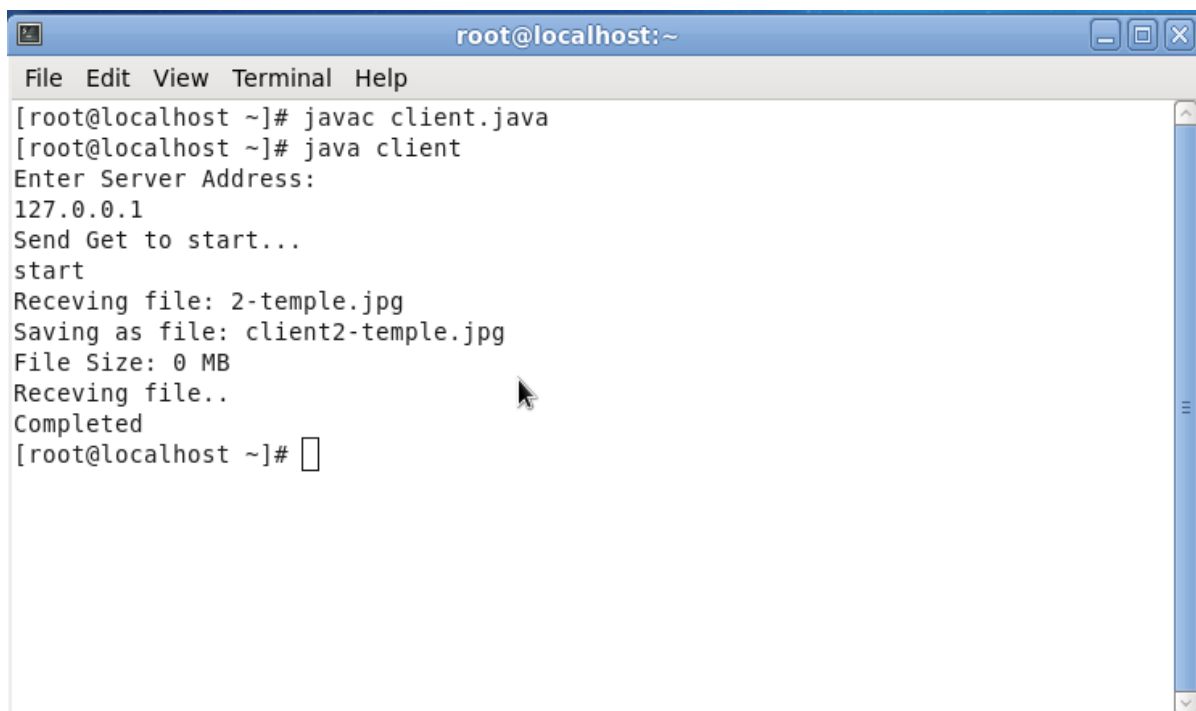
Output:

At server side:

```
root@localhost:~
File  Edit  View  Terminal  Help
[root@localhost ~]# javac server.java
[root@localhost ~]# java server
Enter File Name:
2-temple.jpg
Waiting for request
Connected With /127.0.0.1
SendGet....Ok
Sending File: 2-temple.jpg
Size: 19868
Buf size: 43690
..ok
Send Complete
Waiting for request
```

At Client Side:

```
root@localhost:~
File  Edit  View  Terminal  Help
[root@localhost ~]# javac client.java
[root@localhost ~]# java client
Enter Server Address:
127.0.0.1
Send Get to start...
start
Receiving file: 2-temple.jpg
Saving as file: client2-temple.jpg
File Size: 0 MB
Receving file..
Completed
[root@localhost ~]#
```

Experiment No: 4

Client-Server Communication

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

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

Source Code:
UDP Client

```java
import java.io.*;
import java.net.*;
public class UDPC
{
public static void main(String[] args)
{
DatagramSocket skt; try
{
skt=new DatagramSocket();
String msg= "text message ";
byte[] b = msg.getBytes();
InetAddress host=InetAddress.getByName("127.0.0.1");
 int serverSocket=6788;
DatagramPacket request =new DatagramPacket (b,b.length,host,serverSocket);
skt.send(request);
byte[] buffer =new byte[1000];
DatagramPacket reply= new DatagramPacket(buffer,buffer.length);
 skt.receive(reply);
System.out.println("client received:" +new String(reply.getData()));
skt.close();
}
catch(Exception ex)
{
}
}
}
```

UDP Server

```java
import java.io.*;
import java.net.*;
public class UDPS
{
public static void main(String[] args)
{
DatagramSocket skt=null;
try
{
```

```
skt=new DatagramSocket(6788);
byte[] buffer = new byte[1000];
while(true)
{
DatagramPacket request = new DatagramPacket(buffer,buffer.length);
skt.receive(request);
String[] message = (new String(request.getData())).split(" ");
byte[] sendMsg= (message[1]+ " server processed").getBytes();
DatagramPacket reply = new DatagramPacket(sendMsg,sendMsg.length,request.getAddress
(),request.getPort());
skt.send(reply);
}
}
catch(Exception ex)
{
}
}
}
```

AT SERVER SIDE

[root@localhost]# cc prg6s.c

[root@localhost]# ./a.out

SERVER online!

CLIENT online!

    Waiting for request. ..SERVER: /Test.txt found!

    Transfering the contents...

SERVER transfer Completed!

SERVER transfer Completed!

AT CLIENT SIDE

[root@localhost]# cc prg6c.c

[root@localhost]# ./a.out

Waiting for SERVER...

SERVER online!

    CLIENT: Enter the path: /Test.txt

Wating for reply...

File recieved! Displaying the contents:

    Sockets are a mechanism for exchanging data between processes. These processes can

either be on the same machine, or on different machines connected via a network. Once a

socket connection is established, data can be sent in both directions until one of the endpoints closes the connection.

I needed to use sockets for a project I was working on, so I developed and refined a few C++ classes to encapsulate the raw socket API calls. Generally, the application requesting the data is called the client, and the application servicing the request is called the server. I created two primary classes, C ZlientSocket and ServerSocket, that the client and server could use to exchange data.

Output:
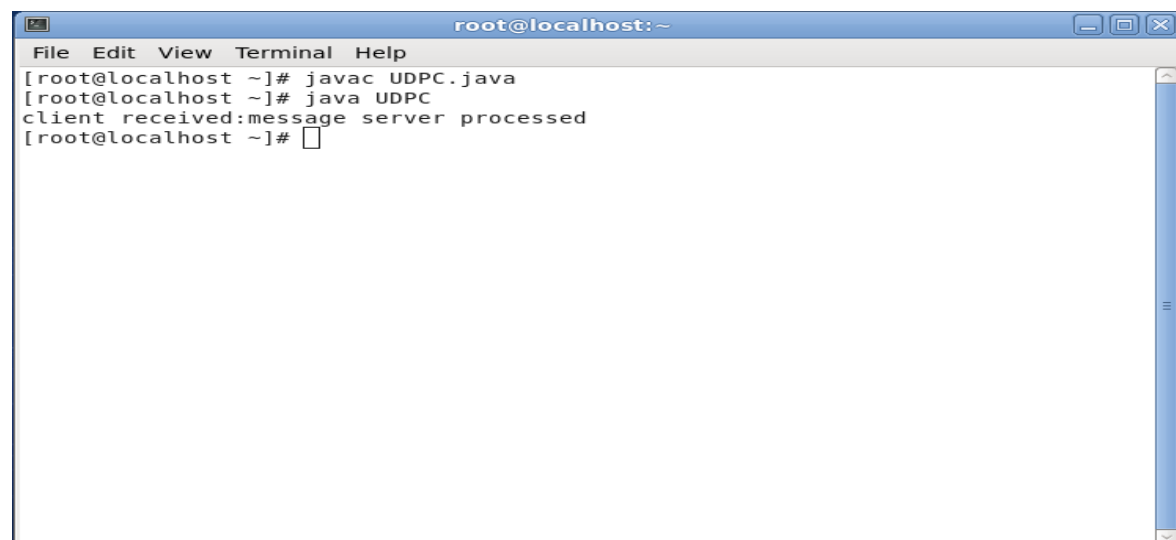
At Server Side:

```
[root@localhost ~]# javac UDPS.java
[root@localhost ~]# java UDPS
```

At Client Side:

```
[root@localhost ~]# javac UDPC.java
[root@localhost ~]# java UDPC
client received:message server processed
[root@localhost ~]#
```

Experiment No: 5

RSA Algorithm to Encrypt and Decrypt the Data

Aim: C 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^{-1} mod phi = 3^{-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^e mod n = 7^3 mod 33 = 343 mod 33 = 13.

Hence the ciphertext c = 13.

To check decryption we compute

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

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

One way of calculating m' is as follows:-

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

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

= (2197 mod 33).(2197 mod 33).(13 mod 33) mod 33

= 19.19.13 mod 33 = 4693 mod 33

= 7.

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

m 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16

c 0 1 8 27 31 26 18 13 17 3 10 11 12 19 5 9 4

m 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32

c 29 24 28 14 21 22 23 30 16 20 15 7 2 6 25 32

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

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

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

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

Remember that calculating m^e mod n is easy, but calculating the inverse c^-e mod n is very difficult, well, for large n's anyway. However, if we can factor n into its prime factors

p and q, the solution becomes easy again, even for large n's. Obviously, if we can get hold of the secret exponent d, the solution is easy, too.

Key Generation Algorithm

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

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

Encryption

Sender A does the following:-

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

Decryption

Recipient B does the following:-

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

Source Code:
```
import java.util.*;
import java.io.*;
public class rsa
{       static int gcd(int m,int n)
        {       while(n!=0)
                {       int r=m%n;
                        m=n;
                        n=r;
                }
                return m;
        }
```

```java
public static void main(String args[])
{
        int p=0,q=0,n=0,e=0,d=0,phi=0;
        int nummes[]=new int[100];
        int encrypted[]=new int[100];
        int decrypted[]=new int[100];

        int i=0,j=0,nofelem=0;
        Scanner sc=new Scanner(System.in);
        String message ;
        System.out.println("Enter the Message tobe encrypted:");
        message= sc.nextLine();
        System.out.println("Enter value of p and q\n");
                p=sc.nextInt();
        q=sc.nextInt();
        n=p*q;
        phi=(p-1)*(q-1);

        for(i=2;i<phi;i++)
        if(gcd(i,phi)==1) break;
        e=i;

        for(i=2;i<phi;i++)
        if((e*i-1)%phi==0)
                break;
        d=i;

        for(i=0;i<message.length();i++)
        {
                char c = message.charAt(i);
        int a =(int)c;
        nummes[i]=c-96;

        }
        nofelem=message.length();
        for(i=0;i<nofelem;i++)
        {
                encrypted[i]=1;
                for(j=0;j<e;j++)
                encrypted[i] =(encrypted[i]*nummes[i])%n;
        }
        System.out.println("\n Encrypted message\n");
        for(i=0;i<nofelem;i++)
        {
                System.out.print(encrypted[i]);
                System.out.print((char)(encrypted[i]+96));
        }
        for(i=0;i<nofelem;i++)
        {       decrypted[i]=1;
                for(j=0;j<d;j++)
```

```
                decrypted[i]=(decrypted[i]*encrypted[i])%n;
        }

        System.out.println("\n Decrypted message\n ");
        for(i=0;i<nofelem;i++)
        System.out.print((char)(decrypted[i]+96));
        return;
    }

    }
```

#**
                    RESULT

Enter the text:
hello
Enter the value of P and Q :
5
7
Encrypted Text is: 8 h 10 j 17 q 17 q 15 o
Decrypted Text is: hello

Experiment No: 6

<div align="center">

Congestion Control Using Leaky Bucket Algorithm

</div>

Aim: *C 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)).

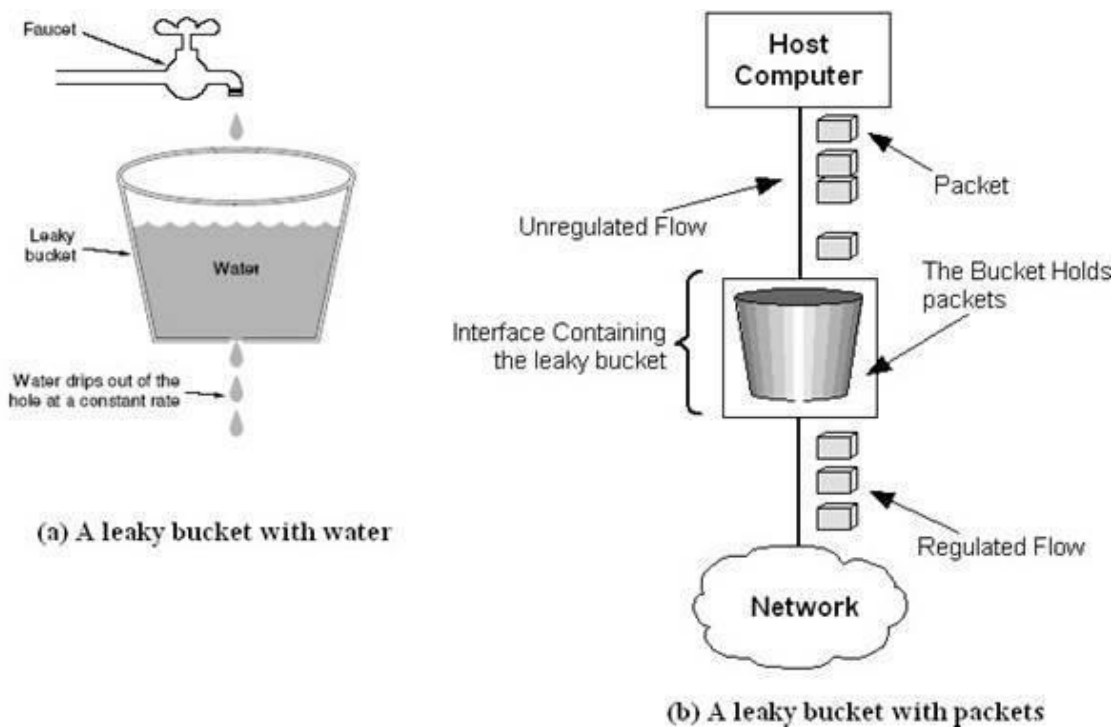*Figure 2.4* - The leaky bucket traffic shaping algorithm

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

Implementation Algorithm:

Steps:

1. Read The Data For Packets

2. Read The Queue Size

3. Divide the Data into Packets

4. Assign the random Propagation delays for each packets to input into the bucket (input_packet).

5. wlile((Clock++<5*total_packets)and

    (out_packets< total_paclets))

    a. if (clock == input_packet)

        i. insert into Queue

    b. if (clock % 5 == 0)

        i. Remove paclet from Queue

6. End

Source Code:

```
import java.util.*;
public class leaky
{
        static int min(int x,int y)
        {
        if(x<y)
        return x;
        else
        return y;
        }
        public static void main(String[] args)
        {       int drop=0,mini,nsec,cap,count=0,i,process;
                int inp[]=new int[25];
                Scanner sc=new Scanner(System.in);

                System.out.println("Enter The Bucket Size\n");
                cap= sc.nextInt();
                System.out.println("Enter The Operation Rate\n");
                process= sc.nextInt();
                System.out.println("Enter The No. Of Seconds You Want To Stimulate\n");
                nsec=sc.nextInt();
                for(i=0;i<nsec;i++)
                {       System.out.print("Enter The Size Of The Packet Entering At "+ i+1 + "
sec");
                        inp[i] = sc.nextInt();
```

```
                }
                System.out.println("\nSecond | Packet Recieved | Packet Sent | Packet Left |
Packet Dropped|\n");
                //System.out.println("
----\n");

                for(i=0;i<nsec;i++)
                {       count+=inp[i];
                        if(count>cap)
                        {               drop=count-cap;
                                count=cap;
                        }
                        System.out.print(i+1);
                        System.out.print("\t\t"+inp[i]);
                        mini=min(count,process);
                        System.out.print("\t\t"+mini);
                        count=count-mini;
                        System.out.print("\t\t"+count);
                        System.out.print("\t\t"+drop);
                        drop=0;
                        System.out.println();
                }
                for(;count!=0;i++)
                {
                if(count>cap)
                {
                drop=count-cap;
                count=cap;
                }
                System.out.print(i+1);
                System.out.print("\t\t0");
                mini=min(count,process);
                System.out.print("\t\t"+mini);
                count=count-mini;
                System.out.print("\t\t"+count);
                System.out.print("\t\t"+drop);
                System.out.println();
                }
                }


    }
```

Output:

Enter The Bucket Size
5
Enter The Operation Rate
2
Enter The No. Of Seconds You Want To Stimulate
3

Enter The Size Of The Packet Entering At 1 sec
5
Enter The Size Of The Packet Entering At 1 sec
4
Enter The Size Of The Packet Entering At 1 sec
3
Second|Packet Recieved|Packet Sent|Packet Left|Packet Dropped|

1	5	2	3	0
2	4	2	3	2
3	3	2	3	1
4	0	2	1	0
5	0	1	0	0

VIVA QUESTIONS

1. What are functions of different layers?

2. Differentiate between TCP/IP Layers and OSI Layers

3. Why header is required?

4. What is the use of adding header and trailer to frames?

5. What is encapsulation?

6. Why fragmentation requires?

7. What is MTU?

8. Which layer imposes MTU?

9. Differentiate between flow control and congestion control.

10. Differentiate between Point-to-Point Connection and End-to-End connections.

11. What are protocols running in different layers?

12. What is Protocol Stack?

13. Differentiate between TCP and UDP.

14. Differentiate between Connectionless and connection oriented connection.

15. Why frame sorting is required?

16. What is meant by subnet?

17. What is meant by Gateway?

18. What is an IP address?

19. What is MAC address?

20. Why IP address is required when we have MAC address?

21. What is meant by port?

22. What are ephemerical port number and well known port numbers?

23. What is a socket?

24. What are the parameters of socket()?

25. Describe bind(), listen(), accept(),connect(), send() and recv().

26. What are system calls? Mention few of them.

27. What is IPC? Name three techniques.

28. Explain mkfifo(), open(), close() with parameters.

29. What is meant by file descriptor?

30. What is meant by traffic shaping?

31. How do you classify congestion control algorithms?

32. Differentiate between Leaky bucket and Token bucket.

33. How do you implement Leaky bucket?

34. How do you generate busty traffic?

35. What is the polynomial used in CRC-CCITT?

36. What are the other error detection algorithms?

37. What is difference between CRC and Hamming code?

38. Why Hamming code is called 7,4 code?

39. What is odd parity and even parity?

40. What is meant by syndrome?

41. What is generator matrix?

42. What is spanning tree?

43. Differentiate between Prim's and Kruskal's algorithm.

44. What are Routing algorithms?

45. How do you classify routing algorithms? Give examples for each.

46. What are drawbacks in distance vector algorithm?

47. How routers update distances to each of its neighbor?

48. How do you overcome count to infinity problem?

49. What is cryptography?

50. How do you classify cryptographic algorithms?

51. What is public key?

52. What is private key?

53. What are key, ciphertext and plaintext?

54. What is simulation?

55. What are advantages of simulation?

56. Differentiate between Simulation and Emulation.

57. What is meant by router?

58. What is meant by bridge?

59. What is meant by switch?

60. What is meant by hub?

61. Differentiate between route, bridge, switch and hub.

62. What is ping and telnet?

63. What is FTP?

64. What is BER?

65. What is meant by congestion window?

66. What is BSS?

67. What is incoming throughput and outgoing throughput?

68. What is collision?

69. How do you generate multiple traffics across different sender-receiver pairs?

70. How do you setup Ethernet LAN?

71. What is meant by mobile host?

72. What is meant by NCTUns?

73. What are dispatcher, coordinator and nctunsclient?

74. Name few other Network simulators

75. Differentiate between logical and physical address.

76. Which address gets affected if a system moves from one place to another place?

77. What is ICMP? What are uses of ICMP? Name few.

78. Which layer implements security for data?