VISVESVARAYA TECHNOLOGICAL UNIVERSITY

Belgaum, Karnataka-590 014

Laboratory Manual

Computer Networks Laboratory

[18CSL57]

Compiled by

1. Prof. Shrutika C Rampure **2. Prof. Sushmitha J**

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

(ACCREDITED BY NBA)

ACHARYA INSTITUTE OF TECHNOLOGY

Soldevanahalli, Bengaluru-560107

2020-2021

Table of contents

Vision, Mission, Motto of Institute	I
Vision, Mission of Department	I
Program Educational Objectives (PEOs)	I
Program Specific Outcomes (PSOs)	II
Program outcomes (POs)	II
Course outcomes of course (COs)	III

SL	Name of Program	Page No
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.	18
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.	21
3	Implement an Ethernet LAN using n nodes and set multiple traffic nodes and plot congestion window for different source/destination.	25
4	Implement simple ESS and with transmitting nodes in wire-less LAN by simulation and determine the performance with respect to transmission of packets.	29
5	Implement and study the performance of GSM on NS2/NS3(Using MAC layer) or equivalent environment.	32
6	Implement and study the performance of CDMA on NS2/NS3 (Using stack called Call net) or equivalent environment	35

7	Write a program for error detecting code using CRC-CCITT(16-bits).	39
8	Write a program to find the shortest path between vertices using bellman-ford algorithm	42
9	Using TCP/IP sockets, write a client – server program to make the client send the filename and to make the server send back the contents of the requested file if present.	46
10	Write a program on datagram socket for client/server to display the messages on client side, typed at the server side	49
11	Write a program for simple RSA algorithm to encrypt and decrypt the data.	51
12	Write a program for congestion control using leaky bucket algorithm	53

Vision of the Institute

Acharya Institute of Technology, committed to the cause of value-based education in all disciplines, envisions itself as a fountainhead of innovative human enterprise, with inspirational initiatives for Academic Excellence.

Mission of the institute

Acharya Institute of Technology strives to provide excellent academic ambiance to the students for achieving global standards of technical education, foster intellectual and personal development, meaningful research and ethical service to sustainable societal needs.

Vision, Mission of the Department

- **Vision of the Department**

Envisions to be recognized for quality education and research in the field of Computing, leading to creation of competent engineers, who are innovative and adaptable to the changing demands of industry and society

- **Mission of the Department:**
  - Act as a nurturing ground for young computing aspirants to attain the excellence by imparting quality education and professional ethics
  - Collaborate with industries and provide exposure to latest tools/ technologies.
  - Create an environment conducive for research and continuous learning.

Program Educational Objectives (PEOs)

Students shall

- Have a successful career in academia, R&D organizations, IT industry or pursue higher studies in specialized field of Computer Science and Engineering and allied disciplines.
- Be competent, creative and a valued professional in the chosen field
- Engage in life-long learning, professional development and adapt to the working environment quickly
- Become effective collaborators and exhibit high level of professionalism by leading or participating in addressing technical, business, environmental and societal challenges.

Program Specific Outcomes:

PSO	Statements
	Students shall
PSO-1:	Apply the knowledge of hardware, system software, algorithms, networking and data bases.
PSO-2:	Design, analyze and develop efficient, Secure algorithms using appropriate data structures, databases for processing of data.
PSO-3:	Be Capable of developing stand alone, embedded and web-based solutions having easy to operate interface using Software Engineering practices and contemporary computer programming languages.

Programme Outcomes

Engineering Graduates will be able to:

a. Engineering knowledge: Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems.
b. Problem analysis: Identify, formulate, review research literature, and analyze complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences.
c. 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 the public health and safety, and the cultural, societal, and environmental considerations.
d. 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.
e. Modern tool usage: Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools including prediction and modeling to complex engineering activities with an understanding of the limitations.

f. 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.
g. 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.
h. Ethics: Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.
i. Individual and team work: Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.
j. 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.
k. Project management and finance: Demonstrate knowledge and understanding of the engineering and management principles and apply these to one's own work, as a member and leader in a team, to manage projects and in multidisciplinary environments.
l. 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.

Course Outcomes:

After the completion of the course, the students will be able to,

CO1: Implement and study the performance of wired network by varying networking parameters like bandwidth and queue size using NS2/NS3.
CO2: Implement and study the performance of Simple ESS, GSM and CDMA on NS2/NS3.
CO3: Demonstrate the working of routing algorithm, error detection algorithm, data encryption and decryption algorithm and congestion control algorithm and concept of data transfer between nodes on sockets using java.
CO4: Communicate effectively through written records and viva.

PART-A

ACHARYA

INTRODUCTION TO NETWORK SIMULATOR - 2

NS (Version 2) is an open source network simulation tool. It is an object oriented, discrete event driven simulator written in C++ and Otcl. The primary use of NS is in network researches to simulate various types of wired/wireless local and wide area networks; to implement network protocols such as TCP and UPD, traffic source behavior such as FTP, Telnet, Web, CBR and VBR, router queue management mechanism such as Drop Tail, RED and CBQ, routing algorithms such as Dijkstra, and many more.

Ns2 is written in C++ and Otcl to separate the control and data path Implementations. The simulator supports a class hierarchy in C++ (the compiled hierarchy) and a corresponding hierarchy within the Otcl interpreter (interpreted hierarchy).

The reason why ns2 uses two languages is that different tasks have different requirements: For example simulation of protocols requires efficient manipulation of bytes and packet headers making the run-time speed very important. On the other hand, in network studies where the aim is to vary some parameters and to quickly examine a number of scenarios the time to change the model and run it again is more important.

In ns2, C++ is used for detailed protocol implementation and in general for such cases where every packet of a flow has to be processed. For instance, if you want to implement a new queuing discipline, then C++ is the language of choice. Otcl, on the other hand, is suitable for configuration and setup. Otcl runs quite slowly, but it can be changed very quickly making the construction of simulations easier. In ns2, the compiled C++ objects can be made available to the Otcl interpreter. In this way, the ready-made C++ objects can be controlled from the OTcl level.

Otcl basics

This chapter introduces the syntax and some basic commands of the Otcl language that are used by ns2. It is important to understand how Otcl works before moving to the part that deals with the creation of the actual simulation scenario.

Assigning values to variables

In tcl, values can be stored to variables and these values can be further used in commands:

```
set a 5
set b [expr $a/5]
```

In the first line, the variable a is assigned the value "5". In the second line, the result of the command [expr $a/5], which equals 1, is then used as an argument to another command, which in turn assigns a value to the variable b. The "$" sign is used to obtain a value contained in a variable and square brackets are an indication of a command substitution.

Procedures

One can define new procedures with the proc command. The first argument to proc is the name of the procedure and the second argument contains the list of the argument names to that procedure. For instance a procedure that calculates the sum of two numbers can be defined as follows:

```
proc sum {a b} {
        expr $a + $b
}
```

The next procedure calculates the factorial of a number:

```
proc factorial a {
        if {$a <= 1}
        {
            return 1
        }
        #here the procedure is called again
        expr $x * [factorial [expr  $x-1]]
}
```

It is also possible to give an empty string as an argument list. However, in this case the variables that are used by the procedure have to be defined as global. For instance:

```
proc sum {} {
    global a b
    expr $a + $b
}
```

Files and lists

In tcl, a file can be opened for reading with the command:

```
set testfile [open test.dat r]
```

The first line of the file can be stored to a list with a command:

```
gets $testfile list
```

Now it is possible to obtain the elements of the list with commands (numbering of elements starts from 0) :

```
set first [lindex $list 0]
set second [lindex $list 1]
```

Similarly, a file can be written with a puts command:

```
set testfile [open test.dat w]
puts $testfile"testi"
```

Calling sub processes

The command exec creates a sub process and waits for it to complete. The use of exec is similar to giving a command line to a shell program. For instance, to remove a file:

```
exec rm $testfile
```

The exec command is particularly useful when one wants to call a tcl-script from within another tcl script. For instance, in order to run the tcl-script example.tcl multiple times with the value of the parameter "test" ranging from 1 to 10, one can type the following lines to another tcl-script:

```
for {set ind 1} {$ind <= 10} {incr ind} {
    set test $ind
    exec ns example.tcl test
}
```

Creating the topology

To be able to run a simulation scenario, a network topology must first be created. In ns2, the topology consists of a collection of nodes and links.

Before the topology can be set up, a new simulator object must be created at the beginning of the script with the command:

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

The simulator object has member functions that enable creating the nodes and the links, connecting agents etc. All these basic functions can be found from the class Simulator. When using functions belonging to this class, the command begins with "$ns", since ns was defined to be a handle to the Simulator object.

Nodes

New node objects can be created with the command:

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

The member function of the Simulator class, called "node" creates four nodes and assigns them to the handles n0, n1, n2 and n3. These handles can later be used when referring to the nodes. If the node is not a router but an end system, traffic agents (TCP, UDP etc.) and traffic sources (FTP,CBR etc.) must be set up, i.e, sources need to be attached to the agents and the agents to the nodes, respectively.

Agents, applications and traffic sources

The most common agents used in ns2 are UDP and TCP agents. In case of a TCP agent, several types are available. The most common agent types are:

- Agent/TCP – a Tahoe TCP sender
- Agent/TCP/Reno – a Reno TCP sender
- Agent/TCP/Sack1 – TCP with selective acknowledgement

The most common applications and traffic sources provided by ns2 are:

- Application/FTP – produces bulk data that TCP will send
- Application/Traffic/CBR – generates packets with a constant bit rate
- Application/Traffic/Exponential – during off-periods, no traffic is sent. During on-periods, packets are generated with a constant rate. The length of both on and off-periods is exponentially distributed.
- Application/Traffic/Trace – Traffic is generated from a trace file, where the sizes and interarrival times of the packets are defined.

In addition to these ready-made applications, it is possible to generate traffic by using the methods provided by the class Agent. For example, if one wants to send data over UDP, the method

send(int nbytes)

can be used at the tcl-level provided that the udp-agent is first configured and attached to some node.

Below is a complete example of how to create a CBR traffic source using UDP as transport protocol and attach it to node n0:

```
set udp0 [newAgent/UDP]
$ns attach-agent $n0 $udp0
set cbr0 [new Application/Traffic/CBR]
$cbr0 attach-agent $udp0
$cbr0 set packet_size_ 1000
$udp0 set packet_size_ 1000
$cbr0 set rate_ 1000000
```

An FTP application using TCP as a transport protocol can be created and attached to node n1 in much the same way:

```
set tcp1 [new Agent/TCP]
$ns attach-agent $n1 $tcp1
set ftp1 [new Application/FTP]
$ftp1 attach-agent $tcp1
$tcp1 set packet_size_ 1000
```

The UDP and TCP classes are both child-classes of the class Agent. With the expressions [new Agent/TCP] and [new Agent/UDP] the properties of these classes can be combined to the new objects udp0 and tcp1. These objects are then attached to nodes n0 and n1. Next, the application is defined and attached to the transport protocol. Finally, the configuration parameters of the traffic source are set. In case of CBR, the traffic can be defined by parameters rate_ (or equivalently interval_, determining the interarrival time of the packets), packetSize_ and random_ . With the random_ parameter it is possible to add some randomness in the interarrival times of the packets. The default value is 0, meaning that no randomness is added.

Traffic Sinks

If the information flows are to be terminated without processing, the udp and tcp sources have to be connected with traffic sinks. A TCP sink is defined in the class Agent/TCPSink and an UDP sink is defined in the class Agent/Null.

A UDP sink can be attached to n2 and connected with udp0 in the following way:

```
set null [new Agent/Null]
$ns attach-agent $n2$null
$ns connect $udp0 $null
```

A standard TCP sink that creates one acknowledgement per a received packet can be attached to n3 and connected with tcp1 with the commands:

```
set sink [new Agent/Sink]
$ns attach-agent $n3 $sink
$ns connect $tcp1 $sink
```

There is also a shorter way to define connections between a source and the destination with thecommand:

```
$ns create-connection <srctype> <src> <dsttype> <dst> <pktclass>
```

For example, to create a standard TCP connection between n1 and n3 with a class ID of 1:

```
$ns create-connection TCP $n1 TCPSink $n3 1
```

One can very easily create several tcp-connections by using this command inside a for-loop.

Links

Links are required to complete the topology. In ns2, the output queue of a node is implemented as part of the link, so when creating links the user also has to define the queue-type.

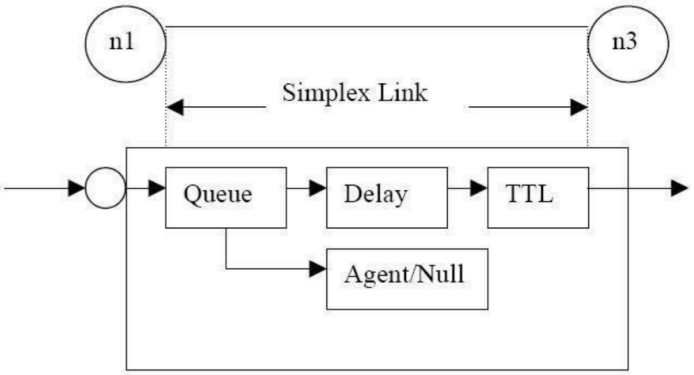


Figure 2 Link in ns2

Figure 2 shows the construction of a simplex link in ns2. If a duplex-link is created, two simplex links will be created, one for each direction. In the link, packet is first enqueued at the queue. After this, it is either dropped, passed to the Null Agent and freed there, or dequeued and passed to the Delay object which simulates the link delay. Finally, the TTL (time to live) value is calculated and updated.

Links can be created with the following command:

$ns duplex/simplex-link endpoint1 endpoint2 bandwidth delay queue-type

For example, to create a duplex-link with DropTail queue management between n0 and n2:

$ns duplex-link $n0 $n2 15Mb 10ms DropTail

Creating a simplex-link with RED queue management between n1 and n3:

$ns simplex-link $n1 $n3 10Mb 5ms RED

The values for bandwidth can be given as a pure number or by using qualifiers k (kilo), M (mega), b (bit) and B (byte). The delay can also be expressed in the same manner, by using m (milli) and u (mikro) as qualifiers. There are several queue management algorithms implemented in ns2, but in this exercise only DropTail and RED will be needed.

Tracing and monitoring

In order to be able to calculate the results from the simulations, the data has to be collected somehow. NS2 supports two primary monitoring capabilities: traces and monitors. The traces enable recording of packets whenever an event such as packet drop or arrival occurs in a queue or a

link. The monitors provide a means for collecting quantities, such as number of packet drops or number of arrived packets in the queue. The monitor can be used to collect these quantities for all packets or just for a specified flow (a flow monitor).

Traces

All events from the simulation can be recorded to a file by using the commands:

```
set trace_all [open all.dat w]
$ns trace-all $trace_all
$ns flush-trace
close $trace_all
```

First, the output file is opened and a handle is attached to it. Then the events are recorded to the file specified by the handle. Finally, at the end of the simulation the trace buffer has to be flushed and the file has to be closed. This is usually done with a separate finish procedure. If links are created after these commands, additional objects for tracing (EnqT, DeqT, DrpT and RecvT) will be inserted into them.

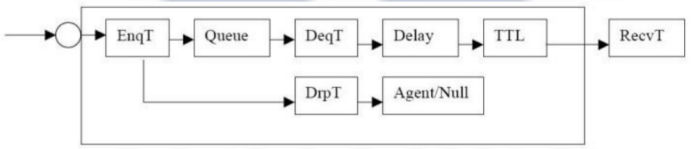


Figure 3 Link in ns2 when tracing is enabled

These new objects will then write to a trace file whenever they receive a packet. The format of the trace file is following:

```
+ 1.84375 0 2 cbr 210 ------- 0 0.0 3.1 225 610
- 1.84375 0 2 cbr 210 ------- 0 0.0 3.1 225 610
r 1.84471 2 1 cbr 210 -------1 3.0 1.0 195 600
r 1.84566 2 0 ack 40 ------2 3.2 0.1 82 602
+ 1.84566 0 2 tcp 1000 -------2 0.1 3.2 102 611
- 1.84566 0 2 tcp 1000 -------2 0.1 3.2 102 611
```

+ : enqueue

- : dequeue

d : drop

r : receive

The fields in the trace file are: type of the event, simulation time when the event occurred, source and destination nodes, packet type (protocol, action or traffic source), packet size, flags, flow id, source and destination addresses, sequence number and packet id. In additiontotracing all events of the simulation, it is also possible to create a trace object between a particular source and a destination with the command:

$ns create-trace <type> <file> <src> <dest>

where the type can be, for instance,

- Enque – a packet arrival (for instance at a queue)
- Deque – a packet departure (for instance at a queue)
- Drop – packet drop
- Recv – packet receive at the destination

Tracing all events from a simulation to a specific file and then calculating the desired quantities from this file for instance by using perl or awk and Matlab is an easy way and suitable when the topology is relatively simple and the number of sources is limited. However, with complex topologies and many sources this way of collecting data can become too slow. The trace files will also consume a significant amount of disk space.

Monitors

With a queue monitor it is possible to track the statistics of arrivals, departures and drops in either bytes or packets. Optionally the queue monitor can also keep an integral of the queue size over time.

For instance, if there is a link between nodes n0 and n1, the queue monitor can be set up as follows:

set qmon0 [$ns monitor-queue $n0 $n1]

The packet arrivals and byte drops can be tracked with the commands:

set parr [$qmon0 set parrivals_]

set bdrop [$qmon0 set bdrops_]

Besides assigning a value to a variable the set command can also be used to get the value of a variable. For example here the set command is used to get the value of the variable “parrivals” defined in the queue monitor class.

A flow monitor is similar to the queue monitor but it keeps track of the statistics for a flow rather than for aggregated traffic. A classifier first determines which flow the packet belongs to and then passes the packet to the flow monitor.

The flowmonitor can be created and attached to a particular link with the commands:

```
set fmon [$ns makeflowmon Fid]
$ns attach-fmon [$ns link $n1 $n3] $fmon
```

Notice that since these commands are related to the creation of the flow-monitor, the commands are defined in the Simulator class, not in the Flow monitor class. The variables and commands in the Flow monitor class can be used after the monitor is created and attached to a link. For instance, to dump the contents of the flow monitor (all flows):

```
$fmon dump
```

If you want to track the statistics for a particular flow, a classifier must be defined so that it selects the flow based on its flow id, which could be for instance 1:

```
set fclassifier [$fmon classifier]
set flow [$fclassifier lookup auto 0 0 1]
```

Controlling the simulation

After the simulation topology is created, agents are configured etc., the start and stop of the simulation and other events have to be scheduled. The simulation can be started and stopped with the commands

```
$ns at $simtime "finish"
$ns run
```

The first command schedules the procedure finish at the end of the simulation, and the second command actually starts the simulation. The finish procedure has to be defined to flush the trace buffer, close the trace files and terminate the program with the exit routine. It can optionally start NAM (a graphical network animator), post process information and plot this information.

The finish procedure has to contain at least the following elements:

```
proc finish {}{
```

```
    global ns trace_all
    $ns  flush-trace
    close $trace_all
    exit 0
}
```

Other events, such as the starting or stopping times of the clients can be scheduled in the following way:

```
$ns at 0.0 "cbr0 start"
$ns at 50.0 "ftp1start"
$ns at $simtime "cbr0 stop"
$ns at $simtime "ftp1 stop"
```

If you have defined your own procedures, you can also schedule the procedure to start for example every 5 seconds in the following way:

```
proc example {} {
    global ns
    set interval 5
    ....
    ...
    $ns at [expr $now + $interval] "example"
}
```

Simple Simulation Example

This section presents a simple NS simulation script and explains what each line does. Example 3 is an OTcl script which creates the simple network configuration and runs the simulation scenario in Figure 4. To run this simulation, copy the code in Example 3 in a file named "ns-simple.tcl" and type "ns ns-simple.tcl" at shell prompt.

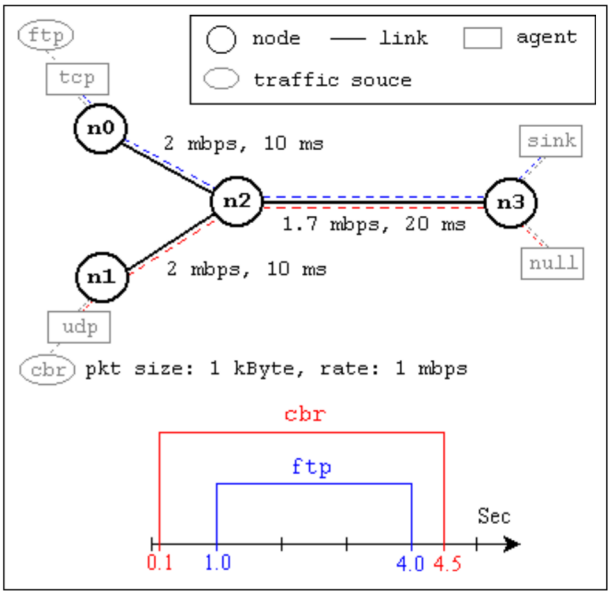


Figure 4. A Simple Network Topology and Simulation Scenario

The network consists of 4 nodes (n0, n1, n2, n3) as shown in above figure. The duplex links between n0 and n2, and n1 and n2 have 2 Mbps of bandwidth and 10 ms of delay. The duplex link between n2 and n3 has 1.7 Mbps of bandwidth and 20 ms of delay. Each node uses a DropTail queue, of which the maximum size is 10. A "tcp" agent is attached to n0, and a connection is established to a tcp "sink" agent attached to n3. As default, the maximum size of a packet that a "tcp" agent can generate is 1KByte. A tcp "sink" agent generates and sends ACK packets to the sender (tcp agent) and frees the received packets. A "udp" agent that is attached to n1 is connected to a "null" agent attached to n3. A "null" agent just frees the packets received. A "ftp" and a "cbr" traffic generator are attached to "tcp" and "udp" agents respectively, and the "cbr" is configured to generate 1 KByte packets at the rate of 1 Mbps. The "cbr" is set to start at 0.1 sec and stop at 4.5 sec, and "ftp" is set to start at 1.0 sec and stop at 4.0 sec.

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

#Define different colors for data flows (for NAM)
$ns color 1 Blue
$ns color 2 Red

#Open the NAM trace file
set nf [open out.nam w]
$ns namtrace-all $nf

#Define a 'finish' procedure
proc finish {} {
	global ns nf
	$ns flush-trace
	#Close the NAM trace file
	close $nf
	#Execute NAM on the trace file
	exec nam out.nam &
	exit 0
}

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

#Create links between the nodes
$ns duplex-link $n0 $n2 2Mb 10ms DropTail
$ns duplex-link $n1 $n2 2Mb 10ms DropTail
$ns duplex-link $n2 $n3 1.7Mb 20ms DropTail

#Set Queue Size of link (n2-n3) to 10
$ns queue-limit $n2 $n3 10

#Give node position (for NAM)
$ns duplex-link-op $n0 $n2 orient right-down
$ns duplex-link-op $n1 $n2 orient right-up
$ns duplex-link-op $n2 $n3 orient right
```

Example 5. A Simple NS Simulation Script

The following is the explanation of the script above. In general, an NS script starts with making a Simulator object instance.

- set ns [new Simulator]: generates an NS simulator object instance, and assigns it to variable ns (italics is used for variables and values in this section). What this line does is the following:
    - Create a scheduler (default is calendar scheduler)

  The "Simulator" object has member functions that do the following:
    - Create compound objects such as nodes and links (described later)
    - Connect network component objects created (ex. attach-agent)
    - Set network component parameters (mostly for compound objects)
    - Create connections between agents (ex. make connection between a "tcp" and "sink")
    - Specify NAM display options
    - Etc.

Most of member functions are for simulation setup and scheduling, however some of them are for the NAM display. The "Simulator" object member function implementations are located in the "ns-2/tcl/lib/ns-lib.tcl" file.

- $ns color fid color: is to set color of the packets for a flow specified by the flow id (fid). This is for the NAM display, and has no effect on the actual simulation.
- $ns namtrace-all file-descriptor: It tells the simulator to record 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. Similarly, the member function trace-all is for recording the simulation trace in a general format.
- proc finish {}: is called after this simulation is over by the command $ns at 5.0 "finish". In this function, post-simulation processes are specified.
- set n0 [$ns node]: The member function node creates a node.

- $ns duplex-link node1 node2 bandwidth delay queue-type: creates two simplex links of specified bandwidth and delay, and connects the two specified nodes. In NS, the output queue of a node is implemented as a part of a link, therefore users should specify the queue-type when creating links. In the above simulation script, DropTail queue is used.
- $ns queue-limit node1 node2 number: This line sets the queue limit of the two simplex links that connect node1 and node2 to the number specified.
- $ns duplex-link-op node1 node2 ...: The next couple of lines are used for the NAM display. This line of code is used to monitor the queue for the link between the nodes node1 and node2. User can see the packets in the queue, and after a while the user can even see how the packets are being dropped.

Now that the basic network setup is done, the next thing to do is to setup traffic agents such as TCP and UDP, traffic sources such as FTP and CBR, and attach them to nodes and agents respectively.

- set tcp [new Agent/TCP]: This line creates a TCP agent. To create agents or traffic sources, a user should know the class names these objects (Agent/TCP, Agnet/TCPSink, Application/FTP and so on).
- $ns attach-agent node agent: The attach-agent member function attaches an agent object created to a node object. Actually, this function calls the attach member function of specified node, which attaches the given agent to itself. Therefore, a user can do the same thing by, for example, $n0 attach $tcp.
- $ns connect agent1 agent2: After two agents that will communicate with each other are created, the next thing is to establish a logical network connection between them.

Assuming that all the network configuration is done, the next thing to do is write a simulation scenario (i.e. simulation scheduling). The Simulator object has many scheduling member functions. However, the one that is mostly used is thefollowing:

- $ns at time "string": This member function of a Simulator object makes the scheduler (scheduler_ is the variable that points the scheduler object created by [new Scheduler] command at the beginning of the script) to schedule the execution of the specified string at given simulation time. For example, $ns at 0.1 "$cbr start" will make the scheduler call a start member function of the CBR traffic source object, which starts the CBR to transmit data.

After all network configurations, scheduling and post-simulation procedure specifications are done, the only thing left is to run the simulation. This is done by $ns run.

ACHARYA

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.

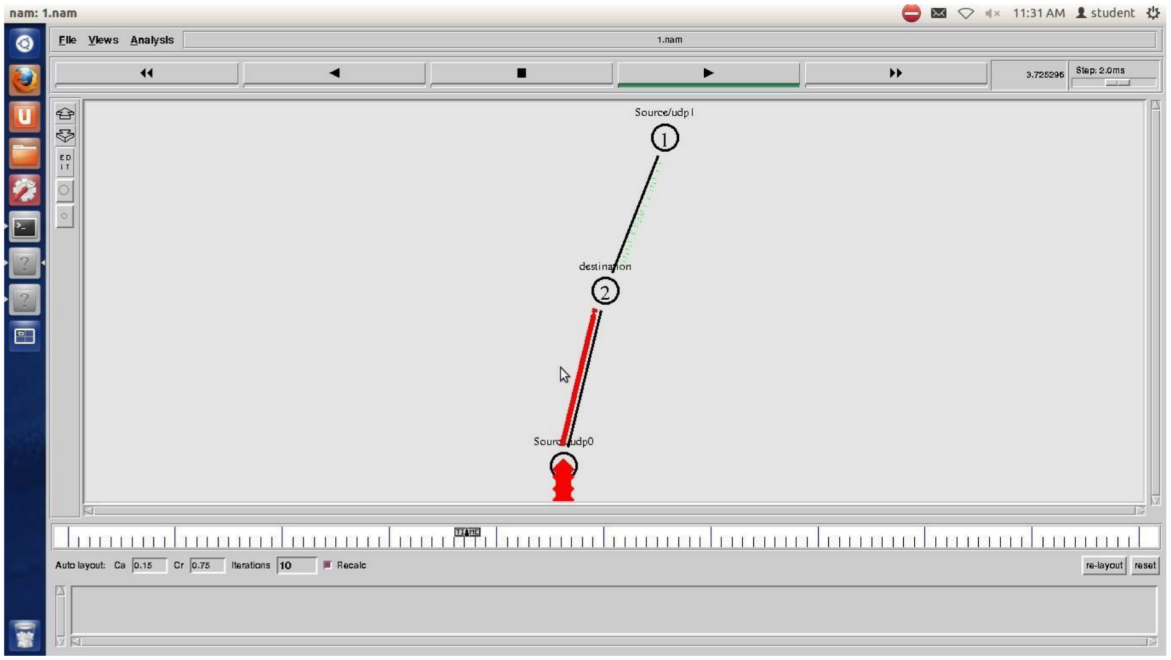
```
setns[new Simulator]
set trf [ open 1.tr w ]
$ns trace-all $trf
set namf [ open 1.nam w ]
$ns namtrace-all $namf
# The below code is used to create the nodes.
set n0 [$ns node]
set n1 [$ns node]
set n2 [$ns node]

#This is used to give color to the flow of packets.
$ns color 1 "red"
$ns color 2 "green"
$n0 label "Source/udp0"
$n1 label "Source/udp1"
$n2 label“destination”

#providing the link
$ns duplex-link $n0 $n2 10Kb 100ms DropTail
$ns duplex-link $n1 $n2 5Mb 200ms DropTail

# set the queue size b/w the nodes
$ns set queue-limit $n0 $n2 10
$ns set queue-limit $n1 $n2 10

set udp0 [new Agent/UDP]
$ns attach-agent $n0 $udp0
set cbr0 [new Application/Traffic/CBR]
$cbr0 attach-agent $udp0
set null3 [new Agent/Null]
$ns attach-agent $n2 $null3
set udp1 [new Agent/UDP]
$ns attach-agent $n1 $udp1
set cbr1 [new Application/Traffic/CBR]
$cbr1 attach-agent $udp1
```

```
#The below code sets the udp0 packets to red and udp1 #packets to blue
color
$udp0 set class_ 1
$udp1 set class_ 2

#The below code is used to connect the agents.
$ns connect $udp0 $null3
$ns connect $udp1 $null3
#The below code is used to set the packet size to 500

$cbr1 set packetSize_ 500Mb

#The below code is used to set the interval of the packets,
$cbr1 set interval_ 0.001

proc finish { } {
global ns namftrf
$ns flush-trace
exec nam1.nam
& close $trf
close $namf
exit 0
}

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

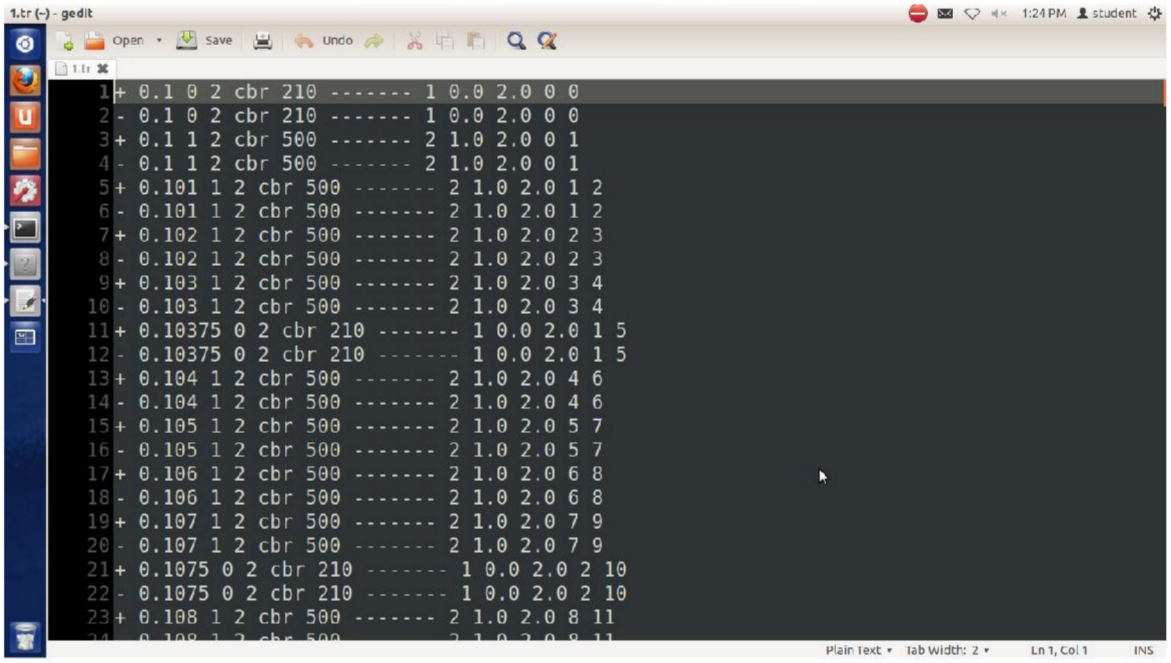
//AWK Script

```
BEGIN{
#include<stdio.h>
c=0;
}
{
if($1=="d") #d stands for the packets drops.
c++ ;
}
END{
printf("The Total no of Packets Dropped due to Congestion : %d\n", c);
}
```

Network Animator:



Trace File:



Output:

Bandwidth: 10Kb, 50Mb

Delay:

The Total no of Packets Dropped due to Congestion: 2532

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.

```
set ns [ new Simulator ]
setnf[open lab2.nam w]

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

$ns trace-all $tf
set n0 [$ns node]
set n1 [$ns node]
set n2 [$ns node]
set n3 [$ns node]
set n4 [$ns node]
set n5 [$ns node]

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

set p1 [new Agent/Ping]
$ns attach-agent $n0 $p1
$p1 set packetSize_ 50000
$p1 set interval_ 0.0001

set p2 [new Agent/Ping]
$ns attach-agent $n1 $p2

set p3 [new Agent/Ping]
$ns attach-agent $n2 $p3
$p3 set packetSize_ 30000
$p3 set interval_ 0.00001

set p4 [new Agent/Ping]
$ns attach-agen5t $n3 $p4

set p5 [new Agent/Ping]
$ns attach-agent $n5 $p5
```

```
$ns queue-limit $n0 $n4 4
$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
$nsconnect $p3 $p4
proc finish { } {
global ns nf tf
$ns flush-trace
close $nf
close $tf
exec nam lab2.nam &
exit 0
}

$ns at 0.1 "$p1 send"
$ns at 0.2 "$p1 send"
$ns at 0.3 "$p1 send"
$ns at 0.4 "$p1 send"
$ns at 0.5 "$p1 send"
$ns at 0.6 "$p1 send"
$ns at 0.7 "$p1 send"
$ns at 0.8 "$p1 send"
$ns at 0.9 "$p1 send"
$ns at 1.0 "$p1 send"
$ns at 1.1 "$p1 send"
$ns at 1.2 "$p1 send"
$ns at 1.3 "$p1 send"
$ns at 1.4 "$p1 send"
$ns at 1.5 "$p1 send"
$ns at 1.6 "$p1 send"
$ns at 1.7 "$p1 send"
$ns at 1.8 "$p1 send"
$ns at 1.9 "$p1 send"
```

```
$ns at 2.0 "$p1 send"
$ns at 2.1 "$p1 send"
$ns at 2.2 "$p1 send"
$ns at 2.3 "$p1 send"
$ns at 2.4 "$p1 send"
$ns at 2.5 "$p1 send"
$ns at 2.6 "$p1 send"
$ns at 2.7 "$p1 send"
$ns at 2.8 "$p1 send"
$ns at 2.9 "$p1 send"
$ns at 0.1 "$p3 send"
$ns at 0.2 "$p3 send"
$ns at 0.3 "$p3 send"
$ns at 0.4 "$p3 send"
$ns at 0.5 "$p3 send"
$ns at 0.6 "$p3 send"
$ns at 0.7 "$p3 send"
$ns at 0.8 "$p3 send"
$ns at 0.9 "$p3 send"
$ns at 1.0 "$p3 send"
$ns at 1.1 "$p3 send"
$ns at 1.2 "$p3 send"
$ns at 1.3 "$p3 send"
$ns at 1.4 "$p3 send"
$ns at 1.5 "$p3 send"
$ns at 3.0 "finish"
$ns run
```

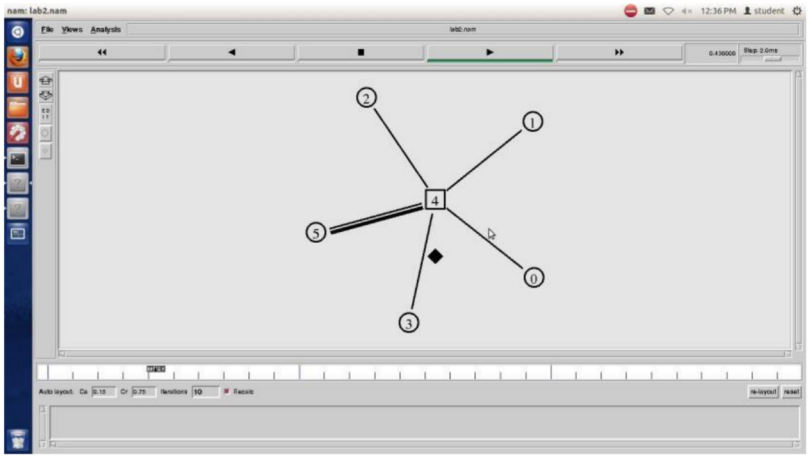
//AWK Script

```
BEGIN{
#include<stdio.h>
c=0;
}
{
if($1=="d") #d stands for the packets drops.
c++ ;
}
END{
printf("The Total no of Packets Dropped due to Congestion : %d\n", c);
```

Output:

```
student@CSELAB001-22:~$ awk -f ping1.awk lab2.tr
the total no of packets dropped due to congestion:20
student@CSELAB001-22:~$
```

Network Animator:



Round Trip Time:

```
student@CSELAB001-22:~$ gedit ping1.tcl
student@CSELAB001-22:~$ ns ping1.tcl
node 2 recieved answer from 3 with round trip time 5.3 mesc
node 2 recieved answer from 3 with round trip time 5.3 mesc
node 2 recieved answer from 3 with round trip time 5.3 mesc
node 2 recieved answer from 3 with round trip time 5.3 mesc
node 0 recieved answer from 5 with round trip time 404.9 mesc
node 2 recieved answer from 3 with round trip time 5.3 mesc
node 2 recieved answer from 3 with round trip time 5.3 mesc
node 2 recieved answer from 3 with round trip time 5.3 mesc
node 2 recieved answer from 3 with round trip time 5.3 mesc
node 0 recieved answer from 5 with round trip time 704.9 mesc
node 2 recieved answer from 3 with round trip time 5.3 mesc
node 2 recieved answer from 3 with round trip time 5.3 mesc
node 2 recieved answer from 3 with round trip time 5.3 mesc
node 2 recieved answer from 3 with round trip time 5.3 mesc
node 0 recieved answer from 5 with round trip time 804.9 mesc
node 2 recieved answer from 3 with round trip time 5.3 mesc
node 2 recieved answer from 3 with round trip time 5.3 mesc
node 2 recieved answer from 3 with round trip time 5.3 mesc
node 2 recieved answer from 3 with round trip time 5.3 mesc
node 0 recieved answer from 5 with round trip time 804.9 mesc
node 2 recieved answer from 3 with round trip time 5.3 mesc
node 2 recieved answer from 3 with round trip time 5.3 mesc
node 2 recieved answer from 3 with round trip time 5.3 mesc
node 2 recieved answer from 3 with round trip time 5.3 mesc
node 0 recieved answer from 5 with round trip time 804.9 mesc
node 2 recieved answer from 3 with round trip time 5.3 mesc
node 2 recieved answer from 3 with round trip time 5.3 mesc
node 2 recieved answer from 3 with round trip time 5.3 mesc
node 2 recieved answer from 3 with round trip time 5.3 mesc
node 0 recieved answer from 5 with round trip time 804.9 mesc
node 2 recieved answer from 3 with round trip time 5.3 mesc
node 2 recieved answer from 3 with round trip time 5.3 mesc
node 2 recieved answer from 3 with round trip time 5.3 mesc
node 2 recieved answer from 3 with round trip time 5.3 mesc
node 0 recieved answer from 5 with round trip time 804.9 mesc
node 2 recieved answer from 3 with round trip time 5.3 mesc
student@CSELAB001-22:~$
```

EXPERIMENT 3

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

```
set ns [new Simulator]
set tf [open lab3.tr w]
$ns trace-all $tf
set nf [open lab3.namw]
$ns namtrace-all $nf
set n0 [$ns node]
$n0 color "magenta"
$n0 label "src1"
set n1 [$ns node]
$n1 color "red"
set n2 [$ns node]
$n2 color "magenta"
$n2 label "src2"
set n3 [$ns node]
$n3 color "blue"
$n3 label "dest2"
set n4 [$ns node]
$n4 shape square
set n5 [$ns node]
$n5 color "blue"
$n5 label "dest1"

$ns make-lan "$n0 $n1 $n2 $n3 $n4" 50Mb 100ms LL Queue/DropTail
Mac/802_3
$ns duplex-link $n4 $n5 1Mb 1ms DropTail
$ns duplex-link-op $n4 $n5 orient right
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 sink0 [new Agent/TCPSink]
$ns attach-agent $n5 $sink0
$ns connect $tcp0 $sink0

set tcp1 [new Agent/TCP]
$ns attach-agent $n2 $tcp1
set ftp1 [new Application/FTP]
$ftp1 attach-agent $tcp1
```

```
$ftp1 set packetSize_ 600
$ftp1 set interval_ 0.001
set sink1 [new Agent/TCPSink]
$ns attach-agent $n3 $sink1
$ns connect $tcp1 $sink1

set file1 [open file1.tr w]
$tcp0 attach $file1
set file2 [open file2.tr w]
$tcp1 attach $file2
$tcp0 trace cwnd_
$tcp1tracecwnd_
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"
$ns at 0.2 "$ftp1 start"
$ns at 8 "$ftp1 stop"
$ns at 14"$ftp0stop"
$ns at 10"$ftp1start"
$ns at 15 "$ftp1 stop"
$ns at 16 "finish"
$ns run
```

//**AWK Script**

```
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 to execute Congestion Window

Note:

1. The congestion at the nodes can be traced by the following steps:

Step 1: Create a file1.tr and file2.tr set file1 [open file1.tr w]
set file2 [open file2.tr w]

Step 2: Attach these files (file1.tr & file2.tr) to the agents tcp0 and tcp1 as shown below.
$tcp0 attach $file1
$tcp1 attach $file2

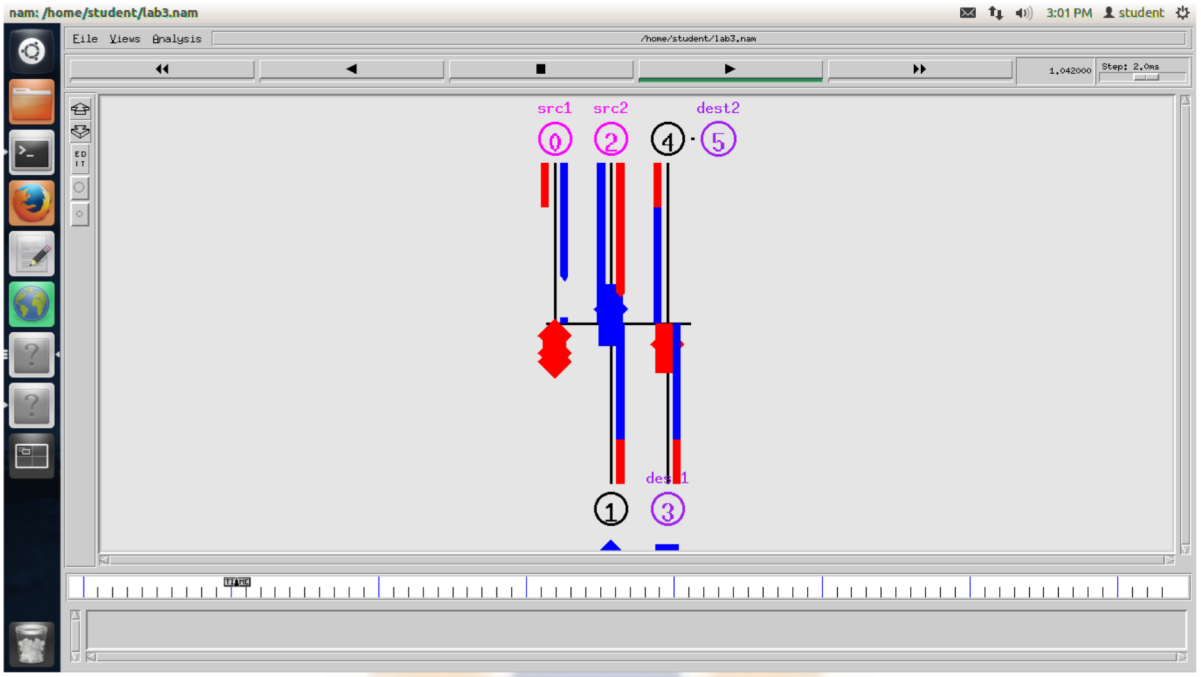
Step 3: To trace the congestion window value we use the trace command to trace the congestion window values. $tcp0 trace cwnd_
$tcp1 trace cwnd_
Then, the congestion window values are stored in the file1 and file2.

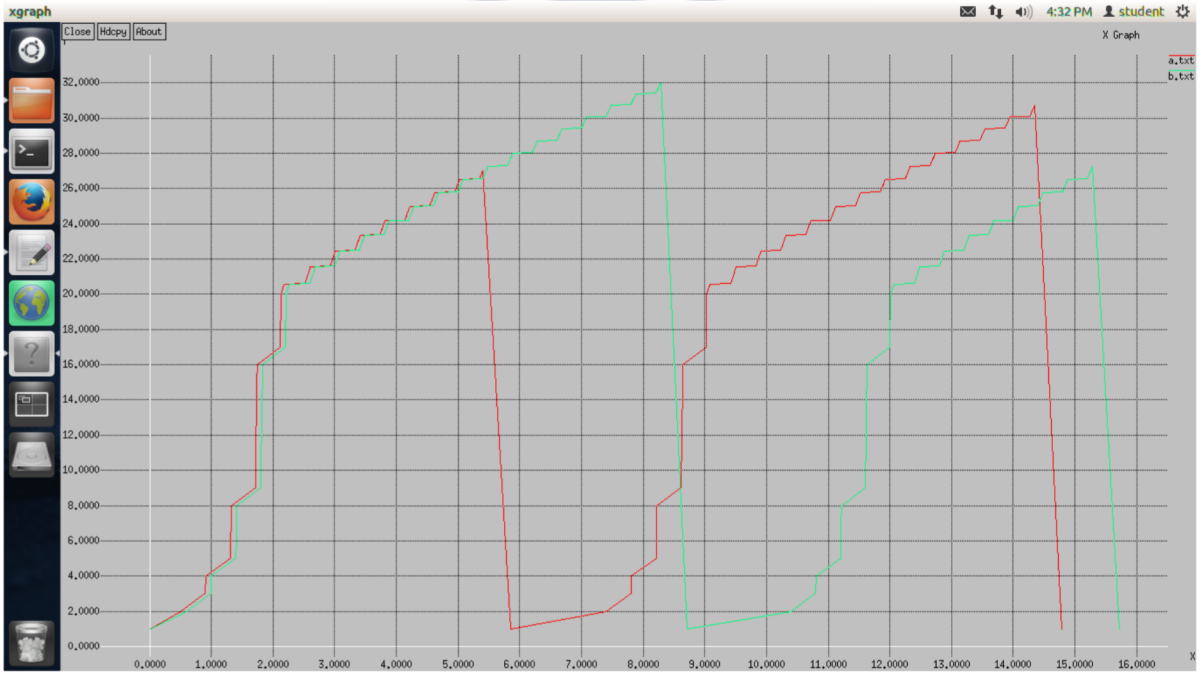
Step 4: To plot the graph the steps are given below

Commands to execute program

```
ns lab7.tcl
awk -f lab7.awk file1.tr > tcp1 awk -f lab7.awk file2.tr > tcp2
xgraph –x “Time” -y “cwnd” tcp1 tcp2
```

Network Animator:





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.

```
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.namw]
$ns namtrace-all-wireless $nf 1000 1000

$ns node-config -adhocRouting DSDV \
-llType LL \
-macType Mac/802_11 \
-ifqType Queue/DropTail \
-ifqLen 50 \
-phyType Phy/WirelessPhy \
-channelType Channel/WirelessChannel\
-propType Propagation/TwoRayGround\
-antType Antenna/OmniAntenna\
-topoInstance $topo \
-agentTrace ON \
-routerTrace ON
create-god 3

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

$n0 label "tcp0"
$n1 label "sink1/tcp1"
$n2 label "sink2"
$n0 set X_50
$n0 set Y_50
$n0 set Z_ 0
$n1 set X_100
$n1 set Y_100
$n1 set Z_ 0
$n2 set X_600
$n2 set Y_600
$n2 set Z_ 0
```

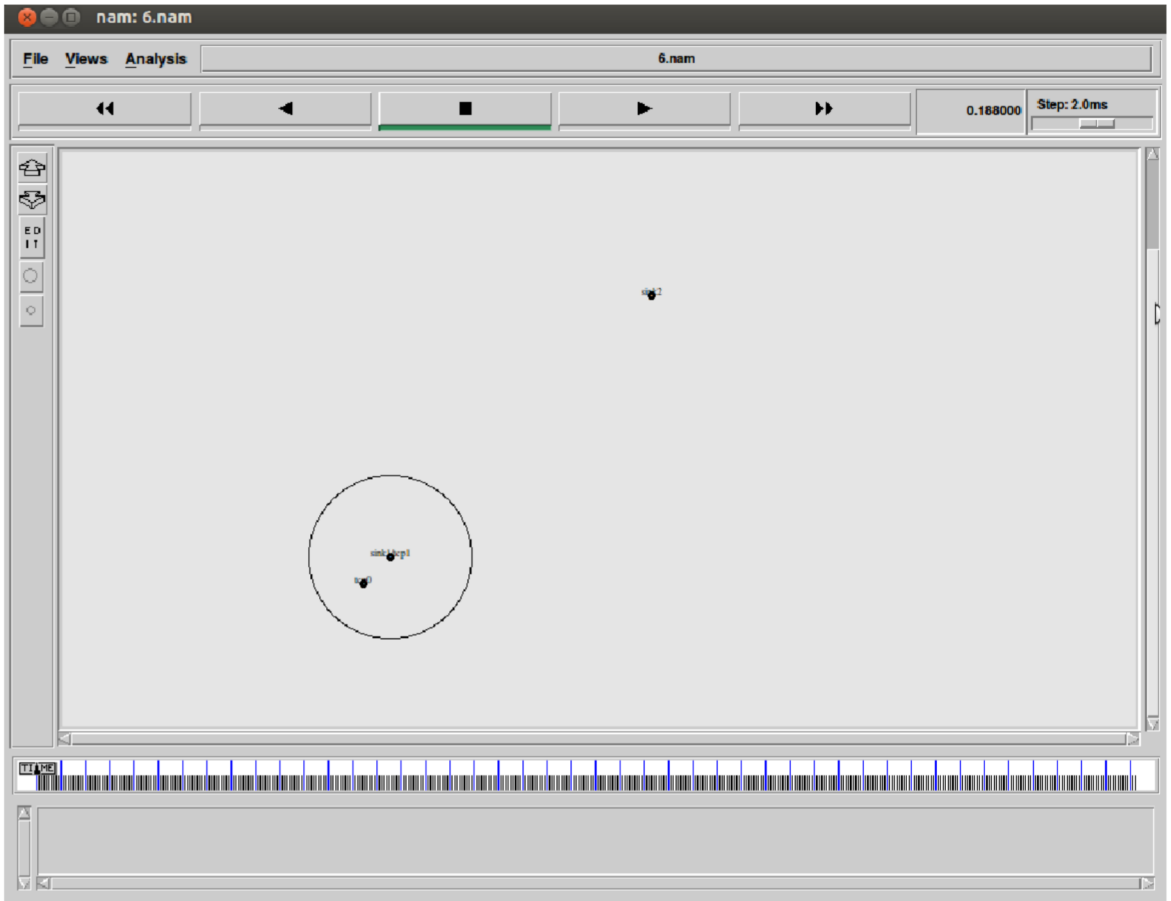
```
$ns at 0.1 "$n0 setdest 50 50 15"
$ns at 0.1 "$n1 setdest 100 100 25"
$ns at 0.1 "$n2 setdest 600 600 25"
set tcp0 [new Agent/TCP]

$ns attach-agent $n0 $tcp0
set ftp0 [new Application/FTP]
$ftp0 attach-agent $tcp0
set sink1 [new Agent/TCPSink]

$ns attach-agent $n1 $sink1
$ns connect $tcp0 $sink1
set tcp1 [new Agent/TCP]

$ns attach-agent $n1 $tcp1
set ftp1 [new Application/FTP]
$ftp1 attach-agent $tcp1
set sink2 [new Agent/TCPSink]

$ns attach-agent $n2 $sink2
$ns connect $tcp1 $sink2
$ns at 5 "$ftp0 start"
$ns at 5 "$ftp1 start"
$ns at 100 "$n1 setdest 550 550 15"
$ns at 190 "$n1 setdest 70 70 15"

proc finish { } {
global ns nf tf
$ns flush-trace
exec nam lab4.nam &
close $tf
exit 0
}

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

//AWK SCRIPT

```
BEGIN{ #include<stdio.h>
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: %fMbps\n", ((count1*pack1*8)/
(time1*1000000))) printf("The Throughput from n1 to n2: %f Mbps",
((count2* pack2 * 8) /(time2*1000000)))
}
```

Network Animator:



Throughput:

```
student@CSELAB001-11:~$ awk -f 6.awk 6.tr
The Throughput from n0 to n1: 5863.442245Mbps
student@CSELAB001-11:~$
```

EXPERIMENT 5

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

```
# General Parameters
set stop 100                ;# Stop time.

# Topology
set type gsm                ;#type of link:

# AQM parameters
set minth 30
set maxth 0
set adaptive 1              ;# 1 for Adaptive RED, 0 for plain RED

# Traffic generation.
set flows 0                 ;# number of long-lived TCP flows
set window 30               ;# window for long-lived traffic

# Plotting statistics.
set opt(wrap) 100           ;# wrap plots?
set opt(srcTrace) is        ;# where to plot traffic
set opt(dstTrace) bs2       ;# where to plot traffic

#default downlink bandwidth in bps
set bwDL(gsm) 9600

#default downlink propagation delay in seconds
set propDL(gsm) .500

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
```

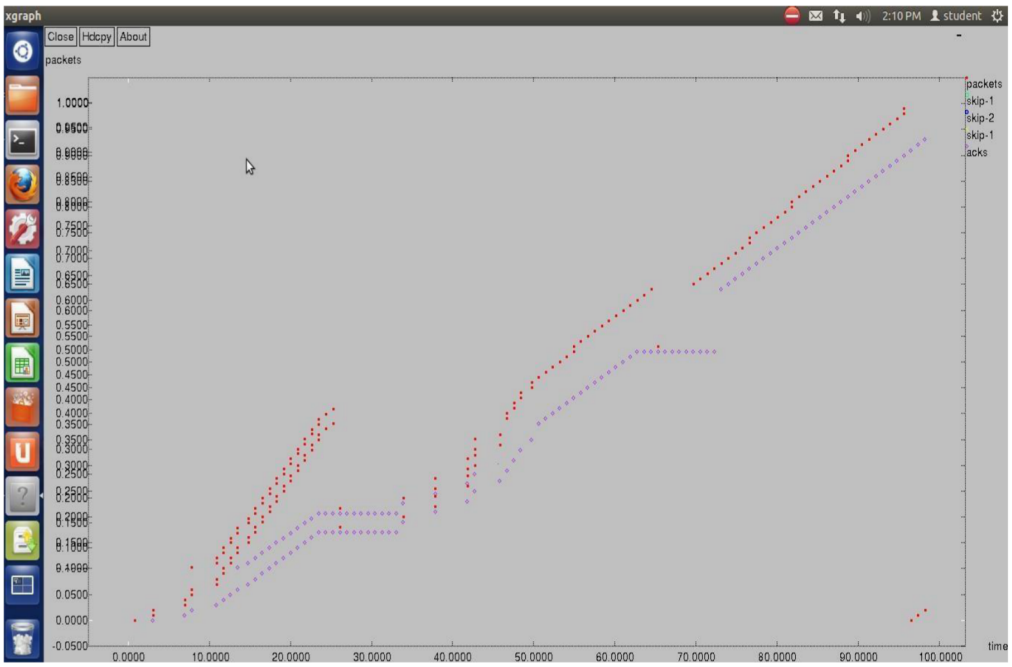
```
  $ns duplex-link $nodes(bs1) $nodes(ms) 1 1 RED
  $ns duplex-link $nodes(ms) $nodes(bs2) 1 1 RED
  $ns duplex-link $nodes(bs2) $nodes(is) 3Mbps 50ms DropTail
  puts "GSM Cell Topology"
}

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/REDsetadaptive_ $adaptive
Queue/RED set thresh_ $minth
Queue/RED set maxthresh_ $maxth
Agent/TCP set window_ $window


source web.tcl

#Create topology
switch $type {
gsm -
 cdma {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 forward 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 opttf
```

```
        set wrap $opt(wrap)
        set sid [$nodes($opt(srcTrace)) id]
        set did [$nodes($opt(dstTrace)) id]
        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 xgraph -x time -y packets plot.xgr &

        exit 0
}
$ns at $stop "stop"
$ns run
```

Steps to execute the simulation program

1) Save the tcl file in "ns-allinone-2.34/ns-2.34/tcl/ex/wireless-scripts".
2) In the terminal, change the directory to above mentioned path to access wireless-scripts folder and type "ns filename.tcl".

XGraph:



EXPERIMENT 6

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

```
# General Parameters
set stop 100                    ;# Stop time.

# Topology
set type cdma                   ;#type of link

# AQM parameters
set minth 30
set maxth 0
set adaptive 1                  ;# 1 for Adaptive RED, 0 for plain RED

# Traffic generation.
set flows 0                     ;# number of long-lived TCP flows
set window 30                   ;# window for long-lived traffic

# Plotting statics.
set opt(wrap) 100               ;# wrap plots?
set opt(srcTrace) is            ;# where to plot traffic
set opt(dstTrace) bs2           ;# where to plot traffic

#default downlink bandwidth in bps
set bwDL(cdma) 384000

#default downlink propagation delay in seconds
set propDL(cdma) .150

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
```

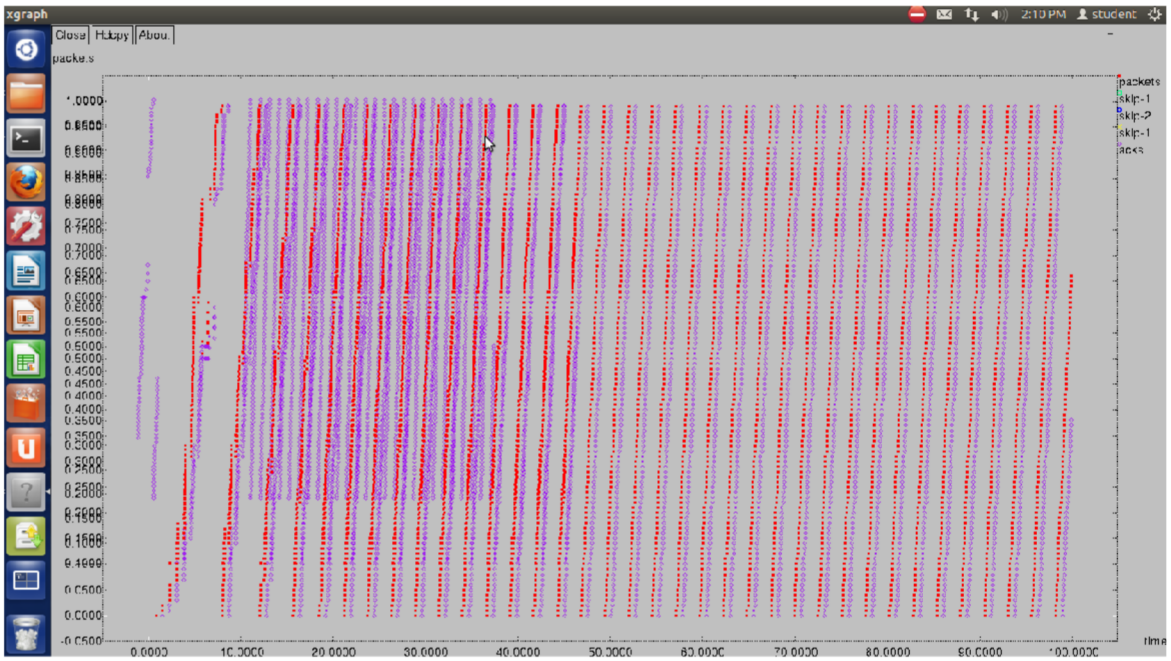
```
$ns duplex-link $nodes(bs1) $nodes(ms) 1 1 RED
$ns duplex-link $nodes(ms) $nodes(bs2) 1 1 RED
$ns duplex-link $nodes(bs2) $nodes(is) 3Mbps 50ms DropTail
puts " cdma Cell Topology"
}

proc set_link_para {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
}

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

source web.tcl

#Create topology
switch $type {
cdma {cell_topo}
}

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

# Set up forward 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 {} {
globalnodesopttf
```

```
set wrap $opt(wrap)
set sid [$nodes($opt(srcTrace)) id]
set did [$nodes($opt(dstTrace)) id]

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

Steps to execute the simulation program

1) Save the tcl file in "ns-allinone-2.34/ns-2.34/tcl/ex/wireless-scripts".
2) In the terminal, change the directory to above mentioned path to access wireless-scripts folder and type "ns filename.tcl".

XGraph:



PART-B

ACHARYA

Program No: 07

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

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

```
import java.util.*;
class crc
{
	public String crc (String dividend, String divisor)
	{
		String str1, div;
		int shift;
		shift = dividend.length( ) - divisor.length( );
		while (shift >= 0)
		{
			//XORing the string
			dividend = Integer.toBinaryString (Integer.parseInt
(dividend, 2 ) ^ (Integer.parseInt (divisor, 2) << shift));
			shift = dividend.length( ) - divisor.length( );
		}
		if (dividend.length( ) < 16)
		{
			while (dividend.length( ) != 16)
			{
				dividend  = "0"  + dividend;
			}
		}
		System.out.println("Div="+dividend);
		return dividend;
	}
	public static void main (String args[ ])
	{
		String data, checksum, syn, dividend, Received_data;
		int padding;
		String polynomial = "10001000000100001";

		Scanner sc = new Scanner(System.in);
		//Data to be transmitted from sender side
		System.out.println("Enter the data to be encrypted\n");
		data = sc.next( );
		dividend = data;
		padding = polynomial.length( ) - 1;

		//Zero padding of data based on polynomial
```

```
            for(int i=0;i < padding; i++)
            {
                    dividend += "0";
            }

            crc obj = new crc( );
            checksum = obj.crc(dividend, polynomial);
            //Generated Codeword
            data = data + checksum;
            System.out.println("Sender Checksum="+checksum);
            System.out.println("Codeword transmitted over
            network="+data);

            //Data received at the receiver side
            System.out.println("Enter the received codeword\n");
            Received_data = sc.next( );
            syn = obj.crc(Received_data,polynomial);

            //Generated Syn bits after checking checksum
            if(Long.parseLong (syn) == 0)
                    System.out.println("No error in data transmission");
            else
                    System.out.println("Error in transmission");

        }
}
```

ACHARYA

Output:

Run 1:

```
student@CSELAB001-29: ~
student@CSELAB001-29:~$ javac crc.java
student@CSELAB001-29:~$ java crc
Enter the data to be encrypted

11001
Div=1000001100011000
Sender Checksum=1000001100011000
Codeword transmitted over network=11001100001100011000
Enter the received codeword

11001100001100011000
Div=0000000000000000
No error in data transmission
student@CSELAB001-29:~$
```

Run 2:

```
student@CSELAB001-29: ~
student@CSELAB001-29:~$ javac crc.java
student@CSELAB001-29:~$ java crc
Enter the data to be encrypted

11001
Div=1000001100011000
Sender Checksum=1000001100011000
Codeword transmitted over network=11001100001100011000
Enter the received codeword

11011100001100011000
Div=0010000001000010
Error in transmission
student@CSELAB001-29:~$
```

Program No: 08

Bellman Ford Algorithm

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

```
import java.util.Scanner;

public classBellmanFord
{
        private int distances[ ];
        private int numberofvertices;
        public static final int MAX_VALUE = 999;

        public BellmanFord (intnumberofvertices)
        {
                this.numberofvertices = numberofvertices;
                distances = new int[numberofvertices + 1];
        }

       public void BellmanFordEvaluation(int source, int adjacencymatrix[]
[])
        {
                for (int node = 1; node <= numberofvertices; node++)
                {
                                distances[node] = MAX_VALUE;
                }

           distances[source] = 0;
           for (int node = 1; node <= numberofvertices - 1; node++)
          {
              for(int sourcenode=1;sourcenode<=numberofvertices;
sourcenode++)
                      {
                 for(int destinationnode=1;destinationnode<=numberofvertices;
destinationnode++)
                 {
                         if(adjacencymatrix[sourcenode][destinationnode]!
=MAX_VALUE)
                      {
                           if (distances[destinationnode] > distances[sourcenode]
+ adjacencymatrix[sourcenode][destinationnode])
                     distances[destinationnode] = distances[sourcenode] +
adjacencymatrix[sourcenode][destinationnode];
                         }
```

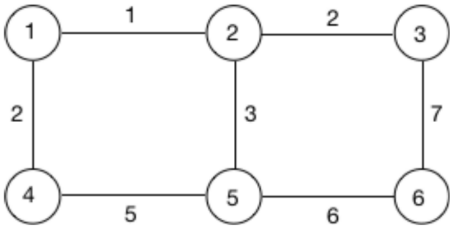
```
                }
            }
        }

    for (int sourcenode = 1; sourcenode <= numberofvertices; sourcenode++)
    {
            for (int destinationnode = 1; destinationnode <= numberofvertices;
destinationnode++)
        {
                        if (adjacencymatrix[sourcenode][destinationnode] !=
MAX_VALUE)
            {
                if (distances[destinationnode] > distances[sourcenode]
                    + adjacencymatrix[sourcenode][destinationnode])
                  System.out.println("The Graph contains negative egde cycle");
            }
        }
    }

    for (int vertex = 1; vertex <= numberofvertices; vertex++)
    {
        System.out.println("distance of source   " + source + " to“ + vertex + " is
" + distances[vertex]);
    }
  }

  public static void main(String arg[ ])
  {
    int numberofvertices = 0;
    int source;
    Scanner scanner = new Scanner(System.in);

    System.out.println("Enterthenumberofvertices");
    numberofvertices = scanner.nextInt( );

                int adjacencymatrix[ ][ ] = new int[numberofvertices + 1]
[numberofvertices + 1];
    System.out.println("Enter the adjacencymatrix");
    for (int sourcenode = 1; sourcenode <= numberofvertices; sourcenode++)
    {
      for (int destinationnode = 1; destinationnode <= numberofvertices;
destinationnode++)
        {
          adjacencymatrix[sourcenode][destinationnode] = scanner.nextInt(
);
```

```
            if (sourcenode == destinationnode)
          {
            adjacencymatrix[sourcenode][destinationnode] = 0;
            continue;
          }
          if (adjacencymatrix[sourcenode][destinationnode] == 0)
          {
            adjacencymatrix[sourcenode][destinationnode] = MAX_VALUE;
          }
        }
      }

      System.out.println("Enter the source vertex");
      source = scanner.nextInt( );

      BellmanFord bellmanford = new BellmanFord(numberofvertices);
      bellmanford.BellmanFordEvaluation(source, adjacencymatrix);
      scanner.close( );
    }
}
```

Output:

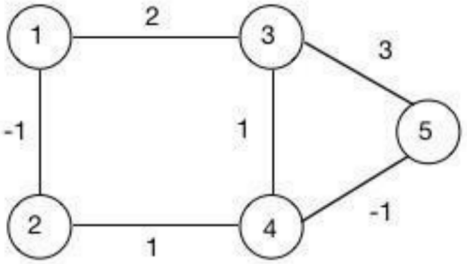
Run 1:

```
student@CSELAB00130:~$ javac BellmanFord.java
student@CSELAB00130:~$ java BellmanFord
Enter the number of vertices
6
Enter the adjacency matrix
0 1 999 4 999 999
1 0 2 999 3 999
999 2 0 999 999 7
4 999 999 0 5 999
999 3 999 5 0 6
999 999 7 999 6 0
Enter the source vertex
1
distance of source  1 to 1 is 0
distance of source  1 to 2 is 1
distance of source  1 to 3 is 3
distance of source  1 to 4 is 4
distance of source  1 to 5 is 4
distance of source  1 to 6 is 10
```



Run 2:

```
student@CSELAB001-22:~$ javac BellmanFord.java
student@CSELAB001-22:~$ java BellmanFord
Enter the number of vertices
5
Enter the adjacency matrix
0 -1 2 999 999
-1 0 999 1 999
2 999 0 1 3
999 1 1 0 -1
999 999 3 -1 0
Enter the source vertex
1
The Graph contains negative edge cycle
The Graph contains negative edge cycle
distance of source  1 to 1 is -8
distance of source  1 to 2 is -7
distance of source  1 to 3 is -5
distance of source  1 to 4 is -8
distance of source  1 to 5 is -7
student@CSELAB001-22:~$
```



Program No: 09

Client-server socket programming

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.

TCP Client Side:

```
import java.net.*;
import java.io.*;
public class ContentsClient
{
 public static void main( String args[ ] ) throws Exception
 {
   Socket sock = new Socket( "127.0.0.1", 4000);

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

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

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

Server Side:

```
import java.net.*;
import java.io.*;
public class ContentsServer
{
```

```
 public static void main(String args[ ]) throws Exception
 {                    // establishing the connection with the server
   ServerSocket sersock = new ServerSocket(4000);
   System.out.println("Server ready for connection");
   Socket sock = sersock.accept();        // binding with port: 4000
   System.out.println("Connection is successful and wating for chatting");

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

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

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

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

Output:

Run 1:

Server Terminal

```
student@CSELAB001-29: ~
student@CSELAB001-29:~$ javac ContentsServer.java
student@CSELAB001-29:~$ java ContentsServer
Server ready for connection
Connection is successful and wating for chatting
student@CSELAB001-29:~$
```

Client terminal

```
student@CSELAB001-29: ~
student@CSELAB001-29:~$ javac ContentsClient.java
student@CSELAB001-29:~$ java ContentsClient
Enter the file name:aki.txt
This is a text file
student@CSELAB001-29:~$
```

Run 2:

Server Terminal

```
student@CSELAB001-29: ~
student@CSELAB001-29:~$ gedit ContentsServer.java
student@CSELAB001-29:~$ javac ContentsServer.java
student@CSELAB001-29:~$ java ContentsServer
Server ready for connection
Connection is successful and wating for chatting
Exception in thread "main" java.io.FileNotFoundException:  aki.txt (No
such file or directory)
        at java.io.FileInputStream.open0(Native Method)
        at java.io.FileInputStream.open(FileInputStream.java:195)
        at java.io.FileInputStream.<init>(FileInputStream.java:138)
        at java.io.FileInputStream.<init>(FileInputStream.java:93)
        at java.io.FileReader.<init>(FileReader.java:58)
        at ContentsServer.main(ContentsServer.java:19)
student@CSELAB001-29:~$
```

Client terminal

```
student@CSELAB001-29: ~
student@CSELAB001-29:~$ gedit ContentsClient.java
student@CSELAB001-29:~$ javac ContentsClient.java
student@CSELAB001-29:~$ java ContentsClient
Enter the file name aki.txt
student@CSELAB001-29:~$
```

Program No: 10

Client-Server Communication using Datagram Socket

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

Server Side:

```
import java.io.*;
import java.net.*;
class UDPServer
{
    public static void main(String args[ ]) throws Exception
    {
            BufferedReader inFromUser=new BufferedReader(new
InputStreamReader(System.in));
        DatagramSocket serverSocket=new DatagramSocket(11111);
        byte[ ] receiveData=new byte[1024];
        byte[]sendData=newbyte[1024];
        while(true)
        {
            DatagramPacket receivePacket=new
 DatagramPacket(receiveData, receiveData.length);
            serverSocket.receive(receivePacket);
            String sentence=new String(receivePacket.getData( ));
            InetAddress IPAddress=receivePacket.getAddress( );
            int port=receivePacket.getPort( );
            Stringmessage=inFromUser.readLine( );
            sendData=message.getBytes( );
            DatagramPacket sendPacket=new DatagramPacket(sendData,
sendData.length, IPAddress, port);
            serverSocket.send(sendPacket);
        }
    }
}
```

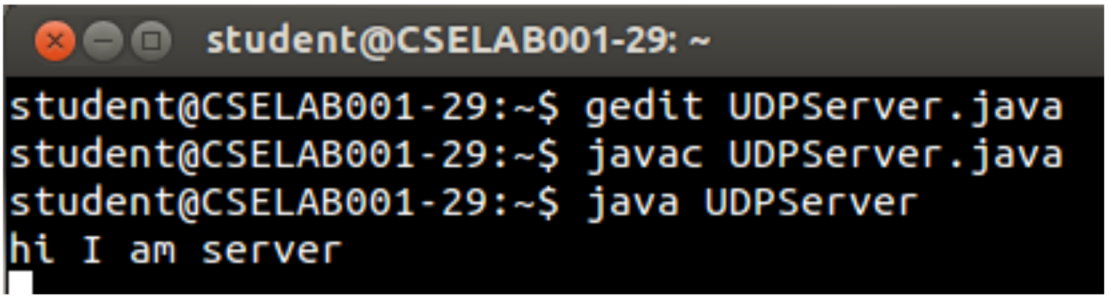
Client Side:

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

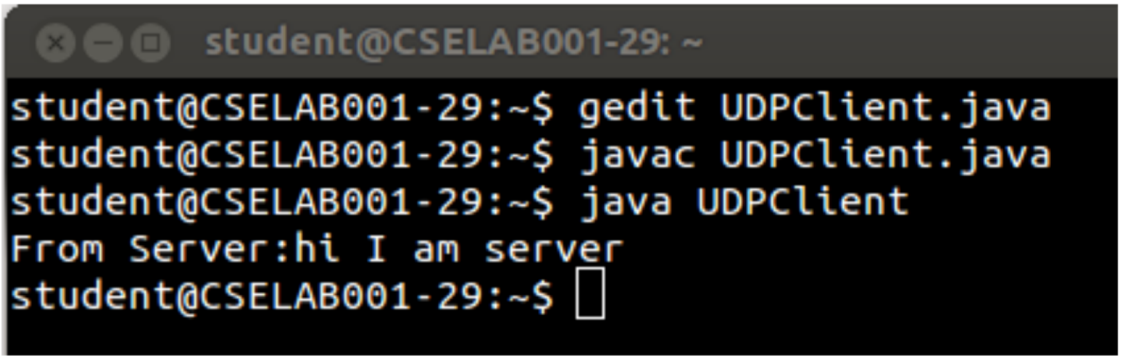
```
        DatagramSocket clientSocket=new DatagramSocket( );
        InetAddress IPAddress=InetAddress.getByName("127.0.0.1");
        byte[] sendData=new byte[1024];
        byte[] receiveData=new byte[1024];
        String sentence= "Hi, I am Client. Send me a message";

        sendData=sentence.getBytes( );
            DatagramPacket sendPacket=new DatagramPacket(sendData,
sendData.length, IPAddress, 11111);
        clientSocket.send(sendPacket);

      DatagramPacket receivePacket=new DatagramPacket(receiveData,
receiveData.length);
        clientSocket.receive(receivePacket);

        String reply=new String(receivePacket.getData( ));
        System.out.println("From Server:" + reply);

        clientSocket.close( );

    }
}
```

Output:

Server Terminal



Client Terminal



Program No: 11

RSA Algorithm to Encrypt and Decrypt the Data

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

```
import java.util.*;
import java.math.BigInteger;
import java.lang.*;

class RSA
{
	public static void main(String[ ] args)
	{
		Random rand1=new Random(System.currentTimeMillis( ));
		Random rand2=new Random(System.currentTimeMillis( )*10);
		int pubkey=2;
		 BigInteger p=BigInteger.probablePrime(32, rand1);
		 BigInteger q=BigInteger.probablePrime(32, rand2);
		 BigInteger n=p.multiply(q);
		 BigInteger p_1=p.subtract(new BigInteger("1"));
		 BigInteger q_1=q.subtract(new BigInteger("1"));
		 BigInteger z=p_1.multiply(q_1);
		 while(true)
		 {
			BigInteger GCD=z.gcd(new BigInteger(""+pubkey));
			if(GCD.equals(BigInteger.ONE))
			{
				break;
			}
			pubkey++;
		 }
		 BigInteger big_pubkey=new BigInteger(""+pubkey);
		 BigInteger prvkey=big_pubkey.modInverse(z);
		 System.out.println("publickey: "+big_pubkey+","+n);
		 System.out.println("private key : "+prvkey+","+n);

		 //RSA Encryption and Decryption
		 Scanner sc = new Scanner(System.in);
		 System.out.println("Enter the message to be encrypted");
		 String msg = sc.nextLine( );

		 byte[] bytes = msg.getBytes( );

		 for(int i=0;i<msg.length( );i++)
		 {
```

```
                    int asciiVal=bytes[i];
                    BigInteger val=new BigInteger(""+asciiVal);
                    BigInteger cipherVal=val.modPow(big_pubkey,n);
                    System.out.println("Cipher text: " +cipherVal);
                    BigInteger plainVal=cipherVal.modPow(prvkey,n);
                    int i_plainVal=plainVal.intValue();
            System.out.println("Plain text:"
+Character.toString((char)i_plainVal));
            }
        }
}
```

Output:

```
student@CSELAB001-22:~$ gedit RSA.java
student@CSELAB001-22:~$ javac RSA.java
student@CSELAB001-22:~$ java RSA
public key:7,10251256900127434987
private key:1464465270527025223,10251256900127434987
Enter the message to be encrypted
computer_network
Cipher text:93206534790699
plain text:c
Cipher text:207616015289871
plain text:o
Cipher text:182803912081669
plain text:m
Cipher text:221068140740608
plain text:p
Cipher text:300124211606973
plain text:u
Cipher text:282621973446656
plain text:t
Cipher text:107213535210701
plain text:e
Cipher text:250226879128704
plain text:r
Cipher text:69833729609375
plain text:_
Cipher text:194871710000000
plain text:n
Cipher text:107213535210701
plain text:e
Cipher text:282621973446656
plain text:t
Cipher text:337931541778439
plain text:w
Cipher text:207616015289871
plain text:o
Cipher text:250226879128704
plain text:r
Cipher text:160578147647843
plain text:k
```

Program No: 12

Congestion Control Using Leaky Bucket Algorithm

Write a program for congestion control using leaky bucket algorithm.

```
import java.util.*;

class congestioncontrol
{
	public static void main(String[ ] args)
	{
		int time, output_rate, max_buffer_size, num_of_pkts;
		int count=0, cur_buffer_size=0;
		Scanner in = new Scanner(System.in);

		System.out.println("Enter the maximum size of buffer");
		max_buffer_size = in.nextInt( );

		System.out.println("Enter the output rate of packets from the
buffer");
		output_rate = in.nextInt();

		System.out.println("Enter the number of arriving packets");
		num_of_pkts = in.nextInt( );

		int[] pkt_size = new int[num_of_pkts];
		int[] arr_time_of_pkts =newint[num_of_pkts];

		System.out.println("Enter the time of arrival of packets");
		for(count=0;count<num_of_pkts;count++)
		{
			arr_time_of_pkts[count] = in.nextInt( );
		}

		time=0;
		count=0;

		while(count < num_of_pkts)
		{
			if(time==arr_time_of_pkts[count])
			{
				Random rn = new Random();
				pkt_size[count] = (rn.nextInt(10)+1) * 10;
				System.out.println("Packet "+(count+1)+" has
arrived & its size is:" +pkt_size[count]);
```

```
                        System.out.println("Current Size of
buffer:"+cur_buffer_size);
                        if(cur_buffer_size+pkt_size[count] <
max_buffer_size)
                        {
                                cur_buffer_size += pkt_size[count];
                                System.out.println("Packet "+(count+1)+"
arriving at "+arr_time_of_pkts[count]+" is CONFORMING PACKET\n");
                        }
                        else
                        {
                                System.out.println("Packet  "+(count+1)+"
arriving at "+arr_time_of_pkts[count]+" is NON CONFORMING PACKET as it
exceeds the buffer limit\n");
                        }
                        count++;
                }
                time++;
                cur_buffer_size -= output_rate;
                if(cur_buffer_size < 0)
                        cur_buffer_size=0;
        }
    }
}
```

Output:

```
student@CSELAB001-29:~$ gedit congestioncontrol.java
student@CSELAB001-29:~$ clear

student@CSELAB001-29:~$ javac congestioncontrol.java
student@CSELAB001-29:~$ java congestioncontrol
Enter the maximum size of buffer
150
Enter the output rate of packets from the buffer
10
Enter the number of arriving packets
5
Enter the time of arrival of packets
2
5
6
7
10
Packet 1 has arrived & its size is:100
Current Size of buffer:0
Packet 1 arriving at 2 is CONFORMING PACKET

Packet 2 has arrived & its size is:30
Current Size of buffer:70
Packet 2 arriving at 5 is CONFORMING PACKET

Packet 3 has arrived & its size is:10
Current Size of buffer:90
Packet 3 arriving at 6 is CONFORMING PACKET

Packet 4 has arrived & its size is:60
Current Size of buffer:90
Packet 4 arriving at 7 is NON CONFORMING PACKET as it exceeds the buff
er limit

Packet 5 has arrived & its size is:100
Current Size of buffer:60
Packet 5 arriving at 10 is NON CONFORMING PACKET as it exceeds the buf
fer limit

student@CSELAB001-29:~$
```

ACHARYA