JAIN COLLEGE OF ENGINEERING & RESEARCH, BELAGAVI

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
(ACADEMIC YEAR 2020-21)

LABORATORY MANUAL

SUBJECT: COMPUTER NETWORK LABORATORY

SUB CODE: 18CSL57

SEMESTER: V

2018 CBCS Scheme

COMPUTER NETWORK LABORATORY
[As per Choice Based Credit System (CBCS) scheme]
(Effective from the academic year 2018 -2019)
SEMESTER – V

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

CREDITS – 02

Course objectives: This course will enable students to

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

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.

Installation procedure of the required software must be demonstrated, carried out in groups and documented in the journal.

Programs List:

PART A

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

PART B (Implement the following in Java)

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.

Course outcomes: The students should be able to:

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

Conduction of Practical Examination:

Experiment distribution

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

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

3. Change of experiment is allowed only once and marks allotted for procedure to be made zero of the changed part only.

4. Marks Distribution (Coursed to change in accordance with university regulations)
   - For laboratories having only one part – Procedure + Execution + Viva-Voce: 15+70+15 = 100 Marks
   - For laboratories having PART A and PART B
     Part A – Procedure + Execution + Viva = 6 + 28 + 6 = 40 Marks
     Part B – Procedure + Execution + Viva = 9 + 42 + 9 = 60 Marks

.

CONTENT LIST

Introduction

Introduction to NS-2

NS-2 stands for Network Simulator Version 2. It is an open-source event-driven simulator designed specifically for research in computer communication networks. Network Simulator-2 (NS2) is a popular open source network simulator for carrying out network experimentation. Way back when it was being designed, its primary usage was to analyze the performance of congestion control algorithms implemented in Transmission control protocol (TCP).

Even today, it remains the most widely used network simulator for TCP research. Over the period of time, it gained wide acceptance in industry, and now supports simulation of latest wired as well as wireless networking protocols (e.g., routing algorithms, TCP, User Data Protocol (UDP)) and paradigms such as Mobile Ad hoc Networks (MANETs) Vehicular Ad hoc Network (VANETs), etc.

Another simulator called ns-3 has gained a lot of popularity in the recent past. It is not a sequel of NS-2. NS-3 APIs are not compatible with those of NS-2 API. Both are completely different tools.

Features of NS-2:

- It is a discrete event simulator for networking research.
- It provides substantial support to simulate protocols like TCP, FTP, UDP & DSR.
- It simulates wired and wireless network.
- It is primarily UNIX based.
- Uses TCL as its scripting language.
- Otcl: Object oriented support Tcl
- TclCL: Tcl with Classes and OTcl linkage
- Discrete event scheduler

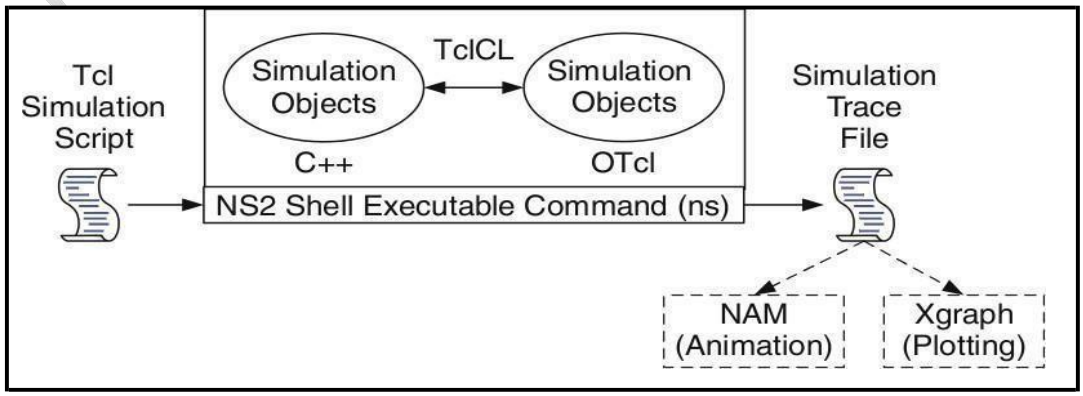


Fig. 1 Basic Architecture of Network Simulator

Why two languages? (TCL and C++)

- NS2 consists of two key languages: C++ and Object-oriented Tool Comman Language(OTcl).
- The C++ defines the internal mechanism (i.e.,a backend) of the simulation objects.
- The OTcl sets up simulation by configuring the objects as well as scheduling discrete events (i.e., a frontend).
- The C++ and the OTcl are linked together using TclCL.
- NS2 uses OTcl to create and configure a network, and uses C++ to run simulation.
- C++ is fast to run but slow to change.
- OTcl, on the other hand, is slow to run but fast to change.
- We write a Tcl simulation script and feeditas an input argument to NS2 when running simulation (e.g., executing "ns myfirst_ns.tcl").
- Here, "ns" is a C++ executable file obtained from the compilation.
- myfirst_ns.tcl is an input configuration file specifying system parameters and configuration such as nodes, link, and how they are connected.
- C++ is used for the creation of objects because of speed and efficiency.
- OTcl is used as a front-end to setup the simulator, configure objects and schedule event because of its ease of use.

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

Structure of NS-2 Program:

- Creating a Simulator Object
- Setting up files for trace & NAM
- Tracing files using their commands
- Closing trace file and starting NAM
- Creating LINK & NODE topology &Orientation of links

Working of NS-2

- NS2 provides users with executable command ns which takes an input argument, the name of a Tcl simulation scripting file.
- Users are feeding the name of a Tcl simulation script (which sets up a simulation) as an input argument of an NS2 executable command ns.
- In most cases, a simulation trace file is created, and is used to plot graph and/or to create animation.

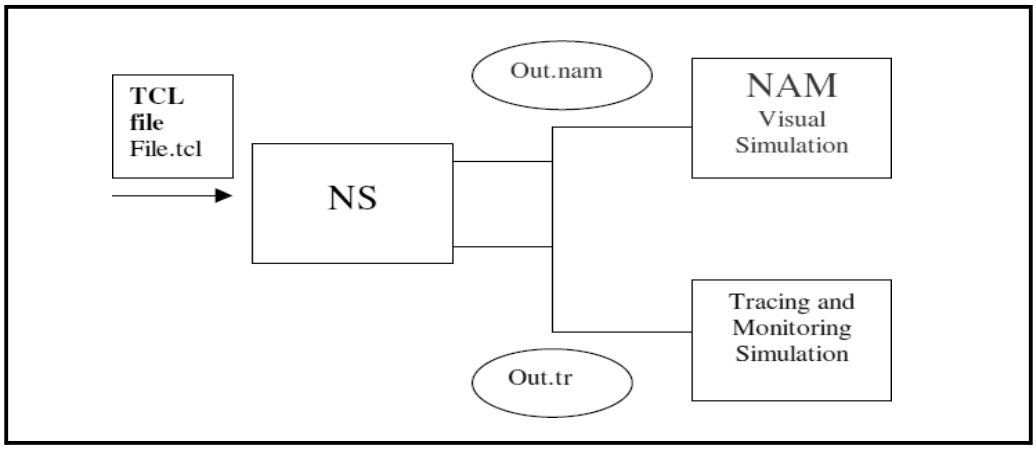


Fig 2. Working of Network Simulator 2

Trace file and NamTrace file:

- Once the simulation is complete, we can see two files: “trace.tr”, and “nam.out”.
- The trace file (trace.tr) is a standard format used by ns2.
- In ns2, each time a packet moves from one node to another, or onto a link, or into a buffer, etc., it gets recorded in this trace file.
- Each row represents one of these events and each column has its own meaning.
- Start nam with the command 'nam <nam-file>' where '<nam-file>' is the name of a nam trace file that was generated by ns.

Network Animator Window

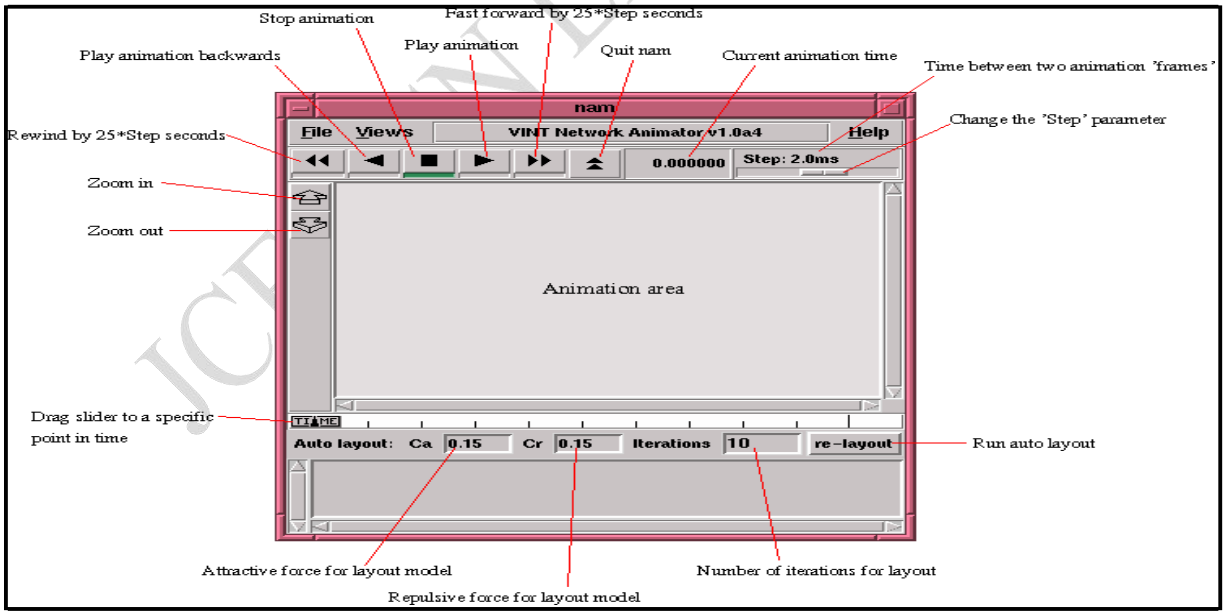


Fig. 3 Details of NAM Window

Advantages and Disadvantages of NS2:

Advantages

1. Open Source
2. Complex scenarios can be easily tested.

3. Results can be quickly obtained – more ideas can be tested in a smaller time frame.
4. Supported protocols
5. Supported platforms
6. Modularity

Disadvantages

1. Limitation in designing large scale systems
2. May be slow compared to real time network and computationallyexpensive
3. Does not reflect reality in large and complex networks.
4. Statistical uncertainty in results

Steps to create a scenario file:

Step1: Declare Simulator and setting output file

Step2: Setting Node and Link

Step3: Setting Agent

Step4: Setting Application

Step5: Setting Simulation time and schedules

Step6: Declare finish.

Step 1: Declare Simulator and setting

$ns [new Simulator]	#first line of tcl script. Creates ns object
get file [open out.tr w] $ns trace-all $file	#open the trace file
get namfile [open out.nam w] $ns namtrace-all $namfile	#open the nam file

Step 2: Setting Node and Link

$ n0 [$ns node]	setting a node
ns duplex-link $n0 $n2 3Mb 5ms DropTail	#bidirectional link between n0 and n2 is declared bandwidth 3Mbps and delay 5ms. DropTail is a waiting queue type.
$ns duplex-link-op $n0 $n2 orient right-down	#Sets positions of node and link for Nam. It does not affect to the result of simulation
$ns queue-limit $n2 $n3 20	#The length of queue on the link from n2 to n3 is 20[packets].
$ns duplex-link-op $n2 $n3 queuePos 0.5	#The position of queue is set for Nam, 0.5 is the angle between link and queue, it equals to (0.5_).

Step 3: Setting Agent

UDP Agent : To use UDP in simulation, the sender sets the Agent as UDP Agent while the receiver sets to Null Agent. Null Agents do nothing except receiving the packets.

set udp [new Agent/UDP] $ns attach-agent $n0 $udp set null [new Agent/Null] $ns attach-agent $n3 $null	#udp and null Agent are set for n0 and n3, respectively
$ns connect $udp $null	Declares the transmission between udp and null.
$udp set fid_ 0	Sets the number for data flow of udp. This number will be recorded to all packets which are sent from udp
$ns color 0 blue	Mark the color to discrete packet for showing result on NAM.

TCP Agent: To use TCP in simulation, the sender sets the Agent as TCP Agent while the receiver sets to TCPSink Agent. When receiving a packet, TCPSink Agent will reply an acknowledgment packet (ACK). Setting Agent for TCP is similar to UDP.

set tcp [new Agent/TCP] $ns attach-agent $n1 $tcp set sink [new Agent/TCPSink] $ns attach-agent $n3 $sink	# tcp and sink Agent are set for n1 and n3, respectively
$ns connect $tcp $sink	#declares the transmission between tcp and sink.
$tcp set fid_ 1	#sets the number for data flow of tcp. This number will be recorded to all packet which are sent from tcp
$ns color 1 red	#mark the color to discrete packet for showing result on Nam.

Step 4: Setting Application

In general, UDP Agent uses CBR Application while TCP Agent uses FTP Application.

set cbr [new Application/Traffic/CBR] $cbr attach-agent $udp
set ftp [new Application/FTP] $ftp attach-agent $tcp

Step 5: Setting time schedule forsimulation

Time schedule of a simulation is set as below:

$ns at 1.0 "$cbr start" $ns at 3.5 "$cbr stop"	cbr transmits data from 1.0[sec] to 3.5[sec]
$ns at 1.5 "$ftp start" $ns at 3.0 "$ftp stop"	ftp transmits data from 1.5[sec] to 3.0[sec].

Step 6: Declare finish

After finish setting, declaration of finish is written at the end of file.

$ns at 4.0 "finish"

proc finish {} {
global ns file namfile tcpfile
$ns flush-trace
close $file
close $namfile
close $tcpfile
exit 0
}
	The finish function is used to output data file at the end of simulation.

Execute Simulation and start Nam

By executing below command line, simulation will be started and shows the animation of simulation

ns sample.tcl
nam out.nam

View trace file (out.tr)

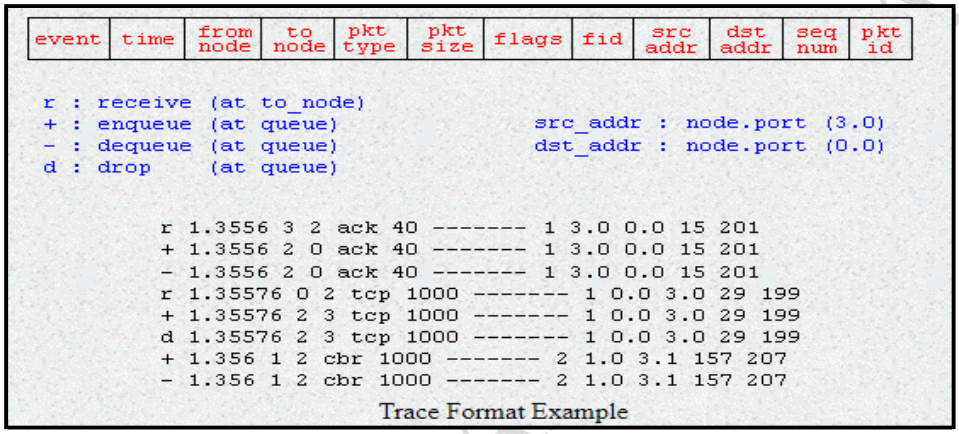


Figure 4 . Details of Trace Window

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.

AWK file:

- The basic function of awk is to search files for lines (or other units of text) that contain certain patterns. When a line matches one of the patterns, awk performs specified actions on that line. awk keeps processing input lines in this way until the end of the input files are reached.

- Programs in awk are different from programs in most other languages, because awk programs are **data-driven;** that is, we describe the data to work with, and then what to do when we find it. Most other languages are **procedural.** When working with procedural languages, it is usually much harder to clearly describe the data of our program will process.

- For this reason, awk programs are often refreshingly easy to both write and read. When we run awk, we can specify an awk **program** that tells awk what to do. The program consists of a series of **rules**. (It may also contain **function definitions**, an advanced feature which we will ignore for now.

- Each rule specifies one pattern to search for, and one action to perform when that pattern is found). Syntactically, a rule consists of a pattern followed by an action. The action is enclosed in curly braces to separate it from the pattern. Rules are usually separated by newlines. Therefore, an awk program looks like this:

***pattern* { *action* }** ***pattern* { *action* }**

- Since we are dealing with column oriented data, AWK is probably the easiest tool we can use to format our data. AWK is a simple scripting language that scans through a file line by line. It allows to access any column in the current line by using special variables $1,$2, $3, etc. for the first, second and third columns. The definition of each column of the trace file is shown above, so we can use the AWK script to check the value of each column and collect the data we need.

- The BEGIN and END sections are only executed once (before and after the file has been processed). The middle section is executed for each line of the file. The AWK script keeps three variables to store the throughput in Mb/s for flow 1, flow 2, and the total. For each line of the trace file, it checks to see if a TCP packet ($5 == "tcp") is received ($1 == "r") at node 3 ($4 == "3"), which is our destination node. If so, it increments the count for the appropriate flow, using the size of the particular packet (in bytes) from column 6. After each second, it prints the total while converting bytes to Mb.

BEGIN { print "START" } { print }END { print "STOP" }

To run awk script

awk -f <filename.awk> <input_file> <output_file>

XGRAPH:

- Plotting purposes.
- Comes together with NS2 installation package.
- Running Xgraph

Xgraph <inputfile1>...<inputfilen> -bg <color> -t <graph_title> -x <xtitle> -y <ytitle>

PART-A

Program 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.

Program Objective:

- Understand the Implementation of the Duplex link between the network.

Theory:

- Create a simulator object.
- We open a file for writing that is going to be used for the trace data.
- We now attach the agent to the nodes.
- Now we attach the application to run on top of these nodes
- We now connect the agent and the application for its working
- Set the simulation time
- The next step is to add a 'finish' procedure that closes the trace file and starts nam.

```
set ns [ new Simulator ]
set tf [ open lab1.tr w ]
$ns trace-all $tf
set nf [ open lab1.nam w ]
$ns namtrace-all $nf
# The below code is used to create the nodes.
set n0 [$ns node]
set n1 [$ns node]
set n2 [$ns node]
set n3 [$ns node]
#This is used to give color to the packets.
$ns color 1 "red"
$ns color 2 "blue"
$n0 label "Source/udp0"
$n1 label "Source/udp1"
$n2 label "Router"
$n3 label "Destination/Null"
#Vary the below Bandwidth and see the number of packets dropped.
$ns duplex-link $n0 $n2 10Mb 300ms DropTail
$ns duplex-link $n1 $n2 10Mb 300ms DropTail
$ns duplex-link $n2 $n3 1Mb 300ms DropTail
#The below code is used to set the queue size b/w the nodes
$ns set queue-limit $n0 $n2 10
$ns set queue-limit $n1 $n2 10
$ns set queue-limit $n2 $n3 5
#The below code is used to attach an UDP agent to n0, UDP agent to n1 and null agent to n3.
set udp0 [new Agent/UDP]
$ns attach-agent $n0 $udp0
set cbr0 [new Application/Traffic/CBR]
$cbr0 attach-agent $udp0 set null [new Agent/Null]
$ns attach-agent $n3 $null 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 $null
$ns connect $udp1 $null
#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, i.e., Data rate of the packets.
#if the data rate is high then packets drops are high.
$cbr1 set interval_ 0.005
proc finish { } {
global ns nf tf
$ns flush-trace
exec nam lab1.nam &
close $tf
close $nf
exit 0
}
$ns at 0.1 "$cbr0 start"
$ns at 0.1 "$cbr1 start"
$ns at 10.0 "finish"
$ns run
```

AWK file:

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

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

Steps for execution

1. Open gedit and type program. Program name should have the extension " .tcl
   [root@localhost ~] gedit lab1.tcl
2. Save the program.
3. Open gedit and type awk program. Program name should have the extension ".awk "
   [root@localhost ~] gedit lab1.awk
4. Save the program.
5. Run the simulation program [root@localhost~] **ns lab1.tcl**
6. Here "ns" indicates network simulator. We get the topology shown in the snapshot.
7. Now press the play button in the simulation window and the simulation will begin.
8. After simulation is completed run awk file to see the output ,
   [root@localhost~] awk –f lab1.awk lab1.tr
9. **To see the trace file contents open the file as ,**

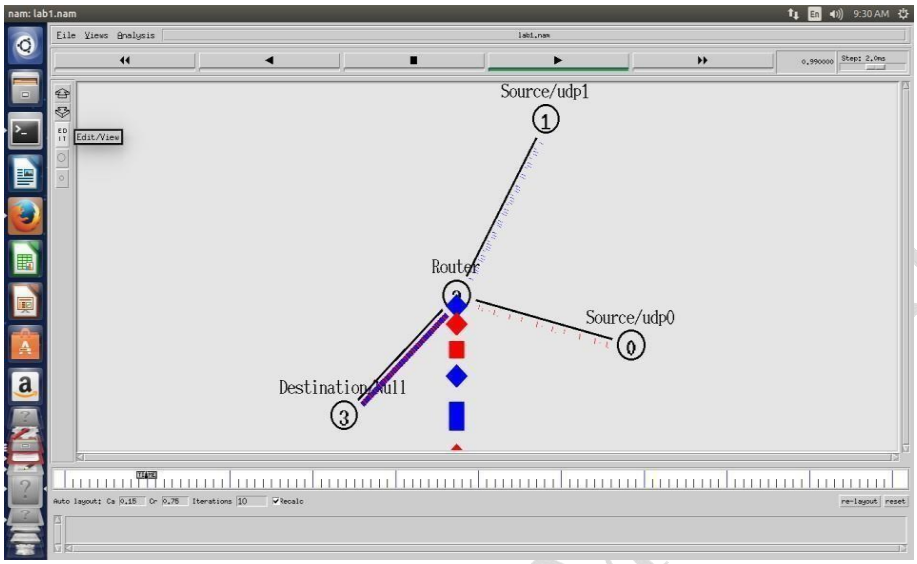
10.

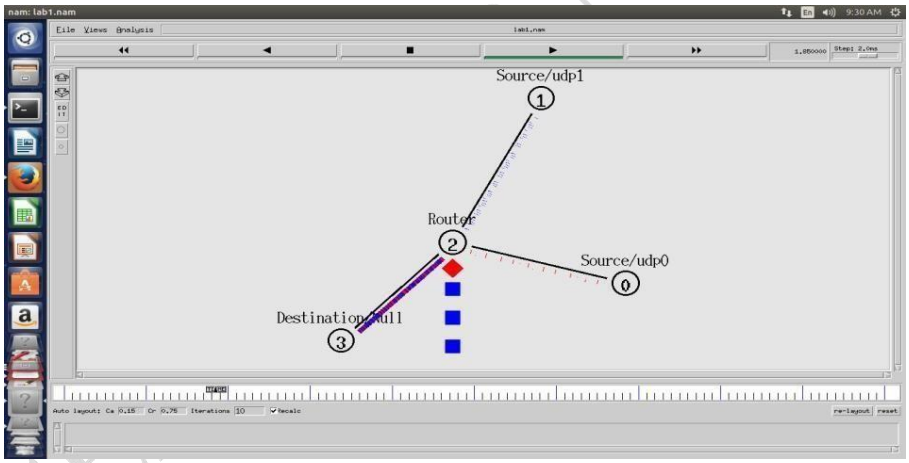
[root@localhost~] gedit lab1.tr

Output:

The Total no of packets Dropped due to congestion: 456

Topology





Program Outcome :

- Implement the Duplex link between the networks.

Viva Questions:

- Define Duplex link
- Define Bandwidth

Program No. 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

Program Objective:

- Understand the Implementation of the network topology consisting ofn nodes.

```
set ns [new Simulator]
set nf [open lab2.nam w]
$ns namtrace-all $nf
set nd [open lab2.tr w]
$ns trace-all $nd
proc finish {} {
global ns nf nd
$ns flush-trace
close $nf
close $nd
exec nam lab2.nam &
exit 0
}
set n0 [$ns node]
set n1 [$ns node]
set n2 [$ns node]
set n3 [$ns node]
set n4 [$ns node]
set n5 [$ns node]
set n6 [$ns node]

$ns duplex-link $n1 $n0 1Mb 10ms DropTail
$ns duplex-link $n2 $n0 1Mb 10ms DropTail
$ns duplex-link $n3 $n0 1Mb 10ms DropTail
$ns duplex-link $n4 $n0 1Mb 10ms DropTail
$ns duplex-link $n5 $n0 1Mb 10ms DropTail
$ns duplex-link $n6 $n0 1Mb 10ms DropTail
Agent/Ping instproc recv {from rtt} {
$self instvar node_
puts "node [$node_ id] recieved ping answer from \
$from with round-trip-time $rtt ms."
}

set p1 [new Agent/Ping]
set p2 [new Agent/Ping]
set p3 [new Agent/Ping]
set p4 [new Agent/Ping]
set p5 [new Agent/Ping]
set p6 [new Agent/Ping]

$ns attach-agent $n1 $p1
$ns attach-agent $n2 $p2
$ns attach-agent $n3 $p3
```

```
$ns attach-agent $n4 $p4
$ns attach-agent $n5 $p5
$ns attach-agent $n6 $p6

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

$ns connect $p1 $p4
$ns connect $p2 $p5
$ns connect $p3 $p6
$ns at 0.2 "$p1 send"
$ns at 0.4 "$p2 send"
$ns at 0.6 "$p3 send"
$ns at 1.0 "$p4 send"
$ns at 1.2 "$p5 send"
$ns at 1.4 "$p6 send"
$ns at 2.0 "finish"
$ns run
```

AWK file:

```
BEGIN {
count=0;
}
{
event=$1;
if(event=="d")
{
count++;
}
}
END {
printf("No of packets dropped : %d\n",count);
}
```

Output Commands:

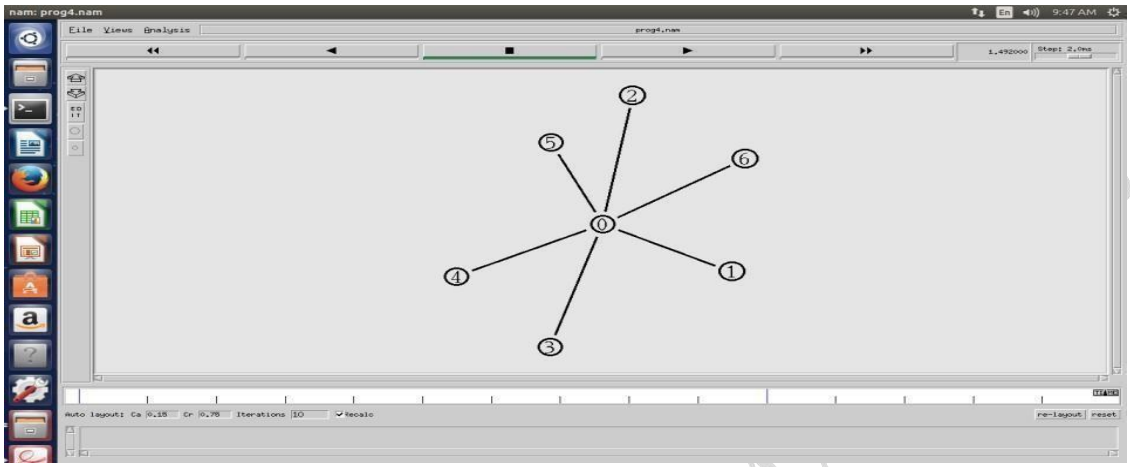
[root@localhost ~]# ns lab2.tcl

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

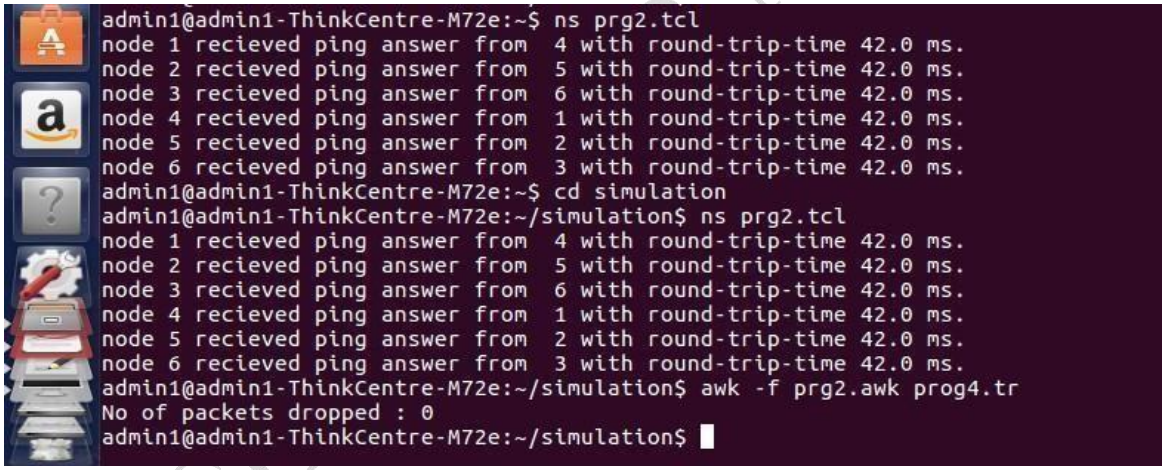
Output:

The Total no of packets dropped due to congestion: 6

Snapshot 1:



Snapshot 2:



Program Outcome :

- Implement network topology consisting of n nodes.

Viva Questions:

- What is network?
- Define Topology
- Explain different types of Topology
- What is congestion?

Program No. 3: Implement an Ethernet LAN using n nodes and set multiple traffic nodes and plot congestion window for different source / destination.

Program Objective:

- Understand the Implementation of the Ethernet LAN using n nodes.

```
set ns [new Simulator]
set nf [open lab3.nam w]
$ns namtrace-all $nf
set nd [open lab3.tr w]
$ns trace-all $nd
$ns color 1 Blue
$ns color 2 Red

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

set n0 [$ns node]
set n1 [$ns node]
set n2 [$ns node]
set n3 [$ns node]
set n4 [$ns node]
set n5 [$ns node]
set n6 [$ns node]
set n7 [$ns node]
set n8 [$ns node]

$n7 shape box
$n7 color Blue
$n8 shape hexagon
$n8 color Red

$ns duplex-link $n1 $n0 2Mb 10ms DropTail
$ns duplex-link $n2 $n0 2Mb 10ms DropTail
$ns duplex-link $n0 $n3 1Mb 20ms DropTail

$ns make-lan "$n3 $n4 $n5 $n6 $n7 $n8" 512Kb 40ms LL Queue/DropTail Mac/802_3
$ns duplex-link-op $n1 $n0 orient right-down
$ns duplex-link-op $n2 $n0 orient right-up
$ns duplex-link-op $n0 $n3 orient right
$ns queue-limit $n0 $n3 20

set tcp1 [new Agent/TCP/Vegas]
$ns attach-agent $n1 $tcp1
set sink1 [new Agent/TCPSink]
$ns attach-agent $n7 $sink1
```

```
$ns connect $tcp1 $sink1
$tcp1 set class_ 1
$tcp1 set packetSize_ 55

set ftp1 [new Application/FTP]
$ftp1 attach-agent $tcp1
set tfile [open cwnd.tr w]
$tcp1 attach $tfile
$tcp1 trace cwnd_
$ns at 0.5 "$ftp1 start"
$ns at 1.0 "$ftp2 start"
$ns at 5.0 "$ftp2 stop"
$ns at 5.0 "$ftp1 stop"
$ns at 5.5 "finish"
$ns run
```

AWK File:

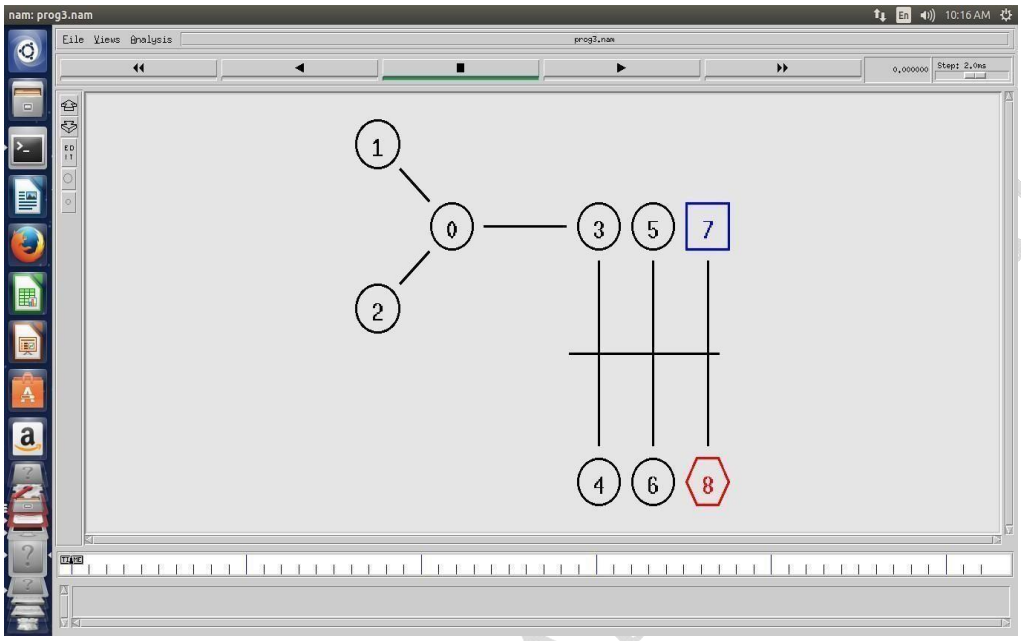
```
BEGIN {

}

{

if($6=="cwnd_")

 {
printf("%f\t%f\n",$1,$7);

}

}

END {

}
```

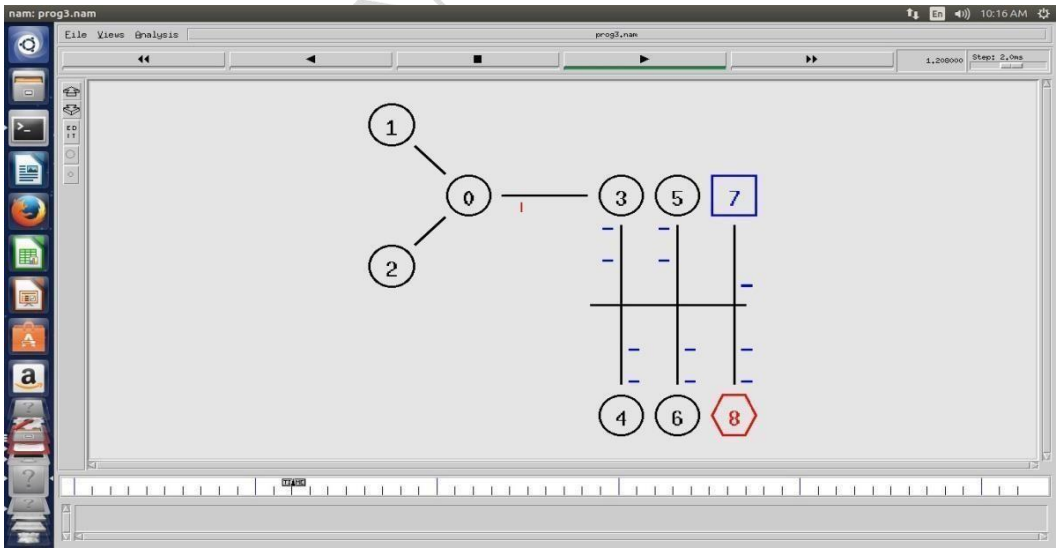
Output

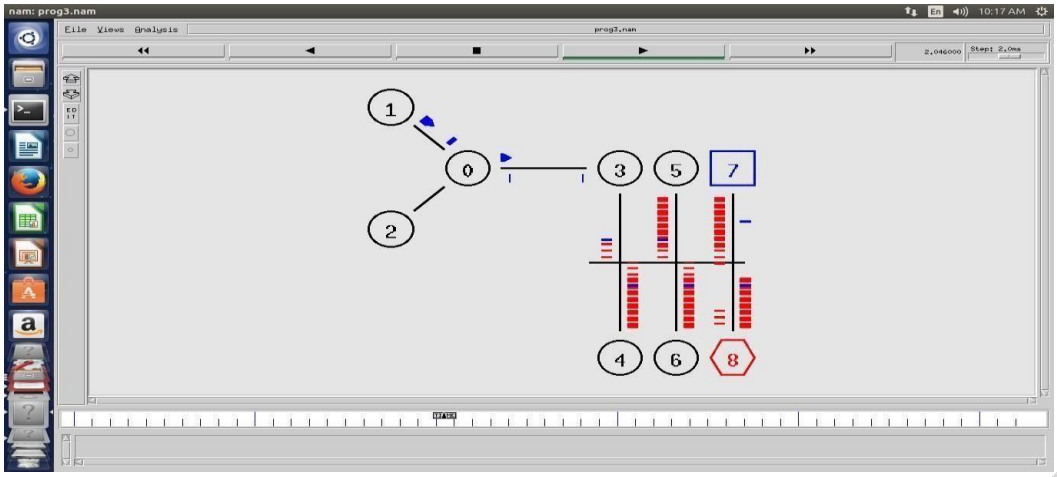
[root@localhost ~]# ns lab3.tcl

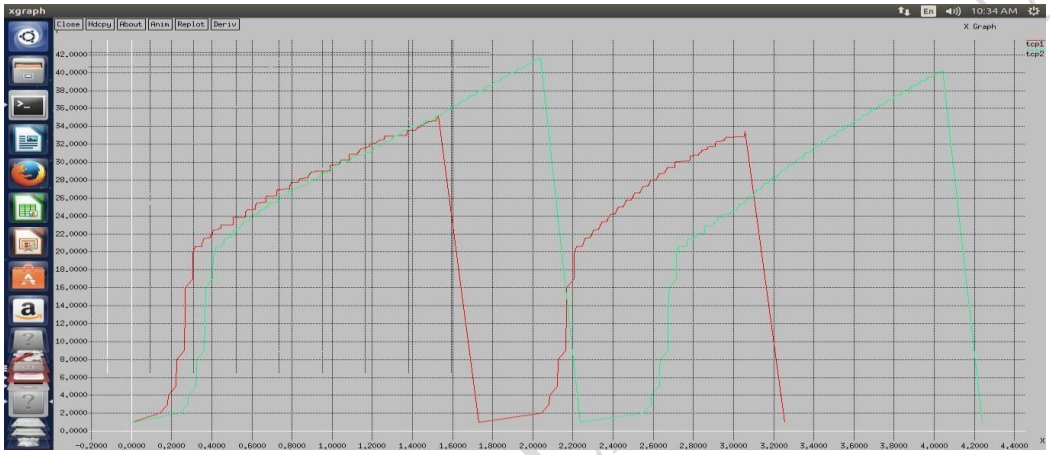
[root@localhost ~]# awk -f lab3.awk file1.tr>tcp1
[root@localhost ~]# awk -f lab3.awk file2.tr>tcp2
[root@localhost ~]# xgraph -x “time” -y “convalue” tcp1 tcp2

Topology









Program Outcome

- Implement Ethernet LAN using n nodes.

Viva Questions

- What is LAN? Explain different types of network?
- What do you mean by congestion window?

Program No. 4: Implement simple ESS and with transmitting nodes in wire-less LAN by simulation and determine the performance with respect to transmission of packets.

Program Objective:

- Understand the Implementation of the simple ESS in wireless LAN.

Simulation Parameters:

Area	: 1000m *1000m
Simulation Time	: 250 sim sec
Wireless nodes	: 3
Location of nodes	: As shown in above Figure
Routing Protocol	: DSDV (Distance Sequenced Distance vector)
Interface queue Type	: Queue/DropTail
MAC	: 802_11
Application	: FTP
Antenna	: omniantenna

```
set ns [new Simulator]

set val(chan) Channel/WirelessChannel;       # channel type
set val(prop) Propagation/TwoRayGround; # radio-propagation model
set val(netif) Phy/WirelessPhy;              # network interface type
set val(mac) Mac/802_11;                     # MAC type
set val(ifq) Queue/DropTail/PriQueue;        # interface queue type
set val(ll) LL;                              # link layer type
set val(ant) Antenna/OmniAntenna;            # antenna model
set val(ifqlen) 50;                          # max packet in ifq
set val(nn) 2;                               # number of mobilenodes
set val(rp) DSDV;                            # routing protocol set val(x) 1000.0;
set val(y) 1000.0;

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 $val(rp) \
                -llType $val(ll) \
                -macType $val(mac) \
                -ifqType $val(ifq) \
                -ifqLen $val(ifqlen) \
                -antType $val(ant) \
                -propType $val(prop) \
                -phyType $val(netif) \
                -channelType $val(chan) \
```

```
            -topoInstance $topo \
            -agentTrace ON \
            -routerTrace ON \
            -macTrace ON
create-god $val(nn)

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_ 250
$n0 set Y_ 250
$n0 set Z_ 0
$n1 set X_ 300
$n1 set Y_ 300
$n1 set Z_ 0
$n2 set X_ 600
$n2 set Y_ 600
$n2 set Z_ 0

$ns at 0.1 "$n0 setdest 250 250 15"
$ns at 0.1 "$n1 setdest 300 300 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"

#The below code is used to provide the node movements.
$ns at 100 "$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
```

Steps for execution

AWK FILE:

```
BEGIN {
#include<stdio.h>
count1=count2=pack1=pack2=time1=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)));
}
```

Output Commands:

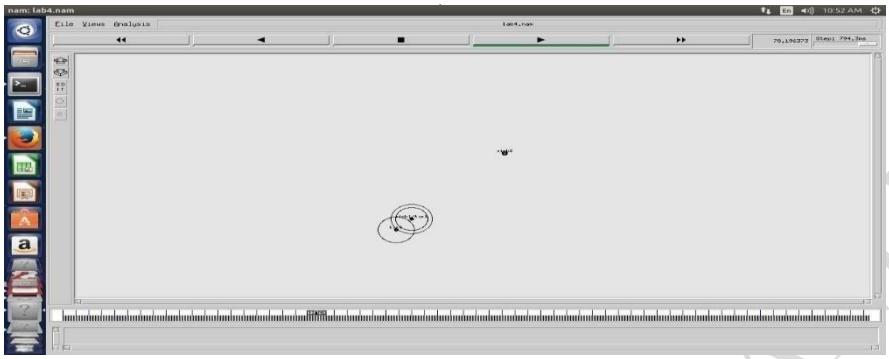
[root@localhost ~]# ns prg4.tcl

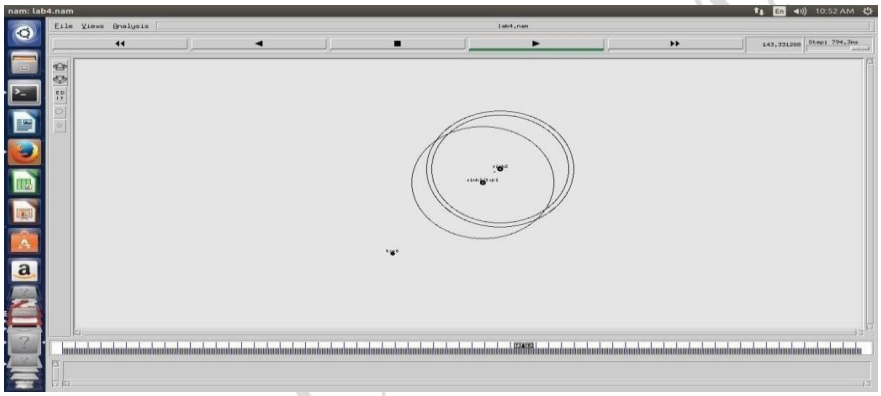
[root@localhost ~]# awk –f prg4.awk lab4.tr

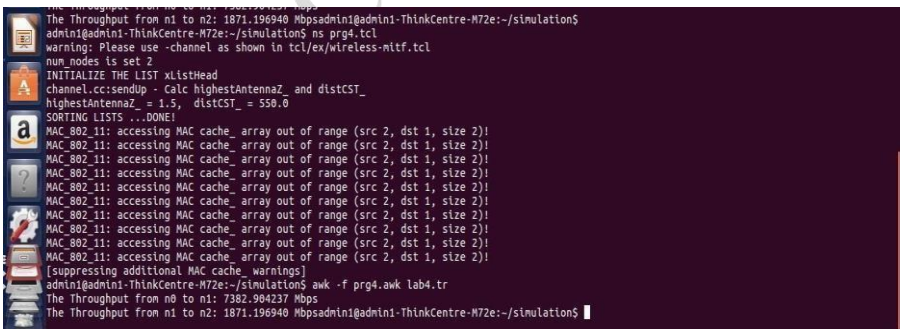
Output:

The Throughput from n0 to n1: 5444Mbps
The Throughput from n1 to n2: 345Mbps

Topology







Program Outcome

- Simulate and implement a simple ESS in wireless LAN.

Viva Questions

- What is ESS? Explain in detail
- What is simulation?

Program No. 5: Implement and study the performance of GSM on NS2/NS3 (Using MAC layer) or equivalent environment.

Program Objective:

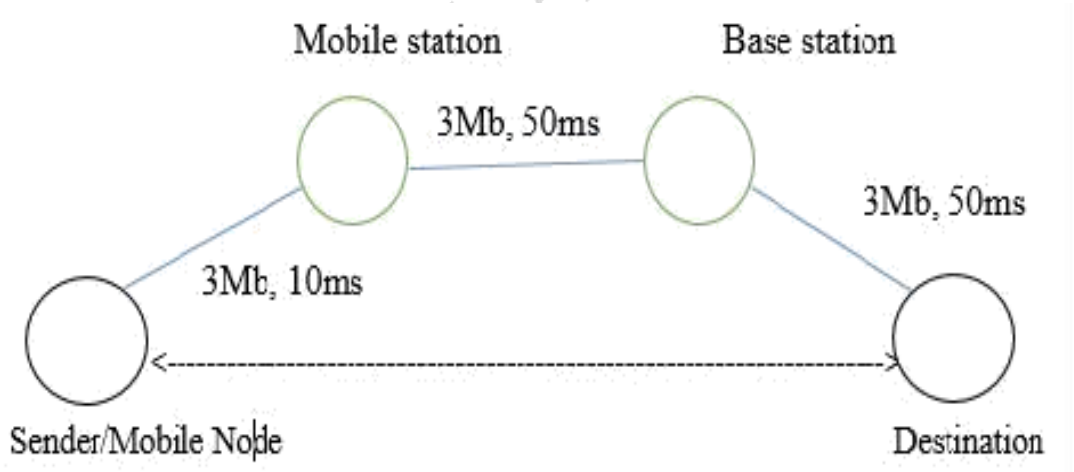
- Understand the Implementation of the performance of GSM.

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

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

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

Design:

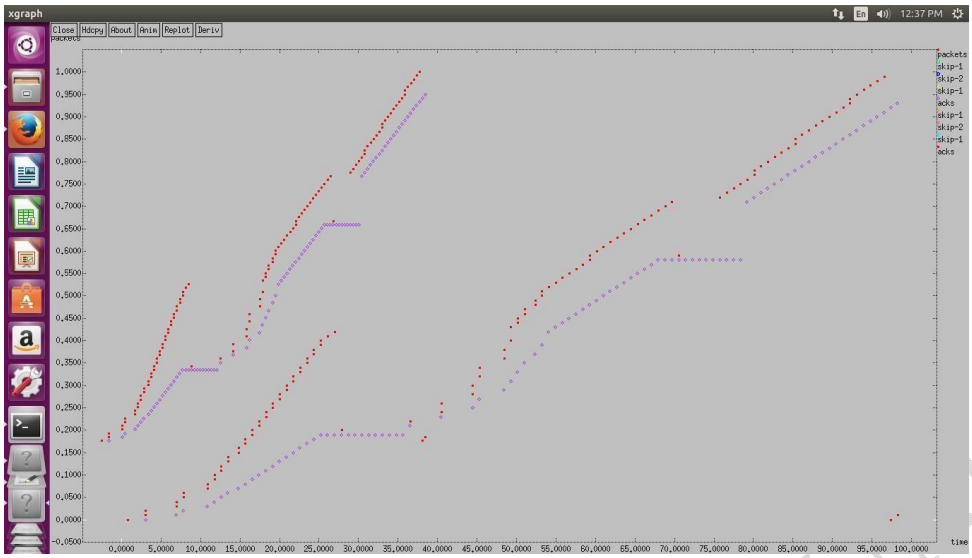


```
# General Parameters
set opt(title) zero;
set opt(stop) 100;
set opt(ecn) 0;
# Topology
set opt(type) umts;
set opt(secondDelay) 55;
# AQM parameters
set opt(minth) 30;
set opt(maxth) 0;
set opt(adaptive) 1 ;    # 1 for Adaptive RED, 0 for plain RED
# Traffic generation.
set opt(flows) 0;        # 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 inbps
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) duplex
$ns bandwidth $nodes(bs2) $nodes(ms) $bwDL($t) duplex
$ns delay $nodes(bs1) $nodes(ms) $propDL($t) duplex
```

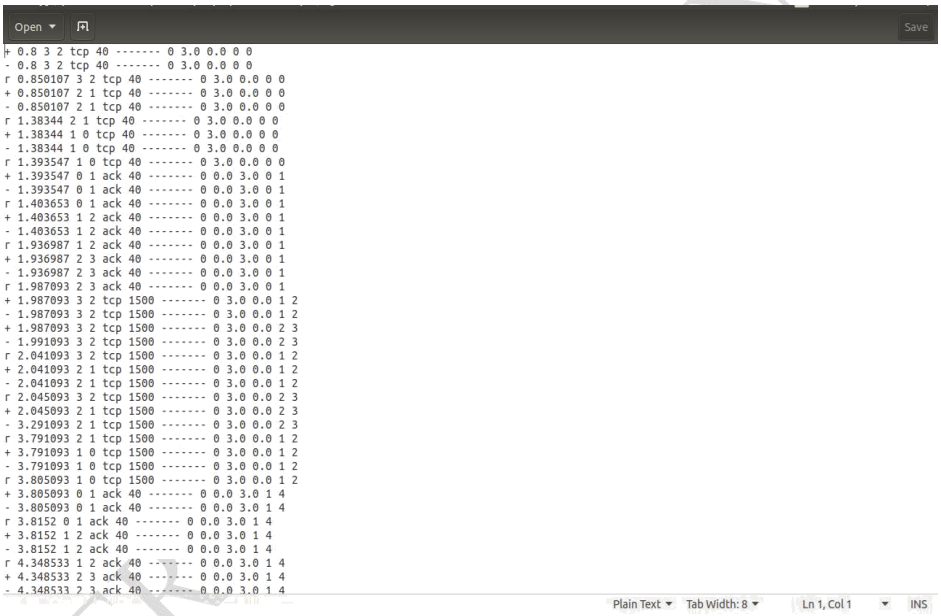
```
$ns delay $nodes(bs2) $nodes(ms) $propDL($t) duplex
$ns queue-limit $nodes(bs1) $nodes(ms) $buf($t)
$ns queue-limit $nodes(ms) $nodes(bs1) $buf($t)
$ns queue-limit $nodes(bs2) $nodes(ms) $buf($t)
$ns queue-limit $nodes(ms) $nodes(bs2) $buf($t)
}
# RED and TCP parameters
Queue/RED set summarystats_ 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) [newDelayer]
$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"
}
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:



GSM Trace file:



Program Outcome

- Simulate and demonstrate the performance of GSM.

Viva Questions

- Simulate and demonstrate the performance of GSM.
- Define GSM??
- What do you mean by network simulator2?
- What is the difference between network simulator 1 and 2 ?

Program No. 6: Implement and study the performance of CDMA on NS2/NS3 (Usingstack called Call net) or equivalent environment.

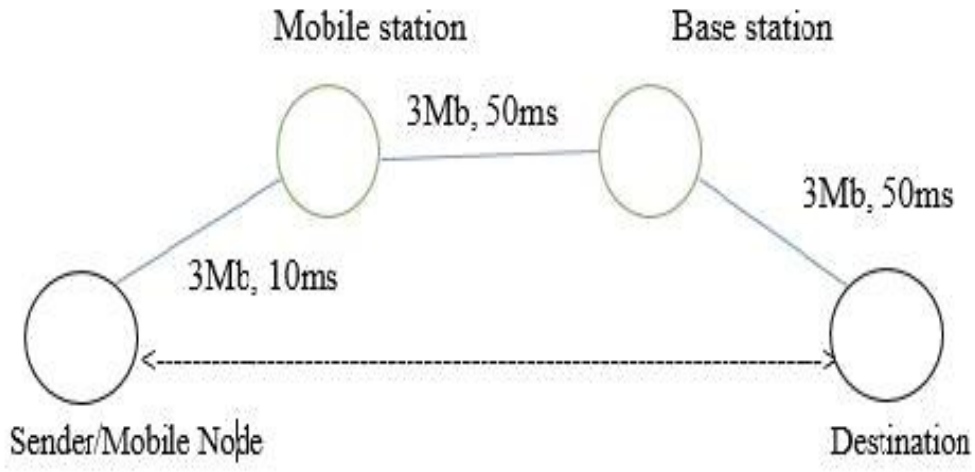
Program Objective:

- Understand the Implementation of the performance of CDMA

3G networks developed as a replacement for second generation (2G) GSM standard network with full duplex voice telephony. CDMA is used as the access method in many mobile phone standards. IS-95, also called cdmaOne, and its 3G evolution CDMA2000, are often simply referred to as CDMA, but UMTS(The Universal Mobile Telecommunications System is a third generation mobile cellular system for networks based on the GSM standard.), the 3G standard used by GSM carriers, also uses wideband CDMA. Long-Term Evolution (LTE) is a standard for high-speed wireless communication which uses CDMA network technology.

3G technology generally refers to the standard of accessibility and speed of mobile devices. The standards of the technology were set by the International Telecommunication Union (ITU). This technology enables use of various services like GPS (Global Positioning System), mobile television and video conferencing. It not only enables them to be used worldwide, but also provides with better bandwidth and increased speed. The main aim of this technology is to allow much better coverage and growth with minimum investment.

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

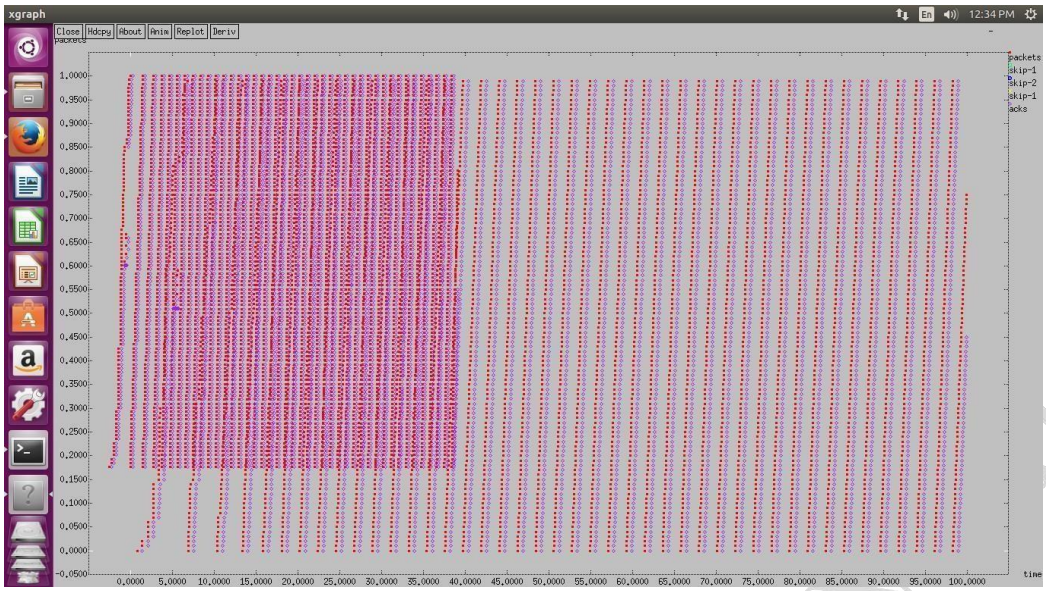


```
# General Parameters
set opt(title) zero;
set opt(stop) 100;
set opt(ecn) 0;
# Topology
set opt(type) umts;              #type of link:
set opt(secondDelay) 55;         # average delay of access links in ms
# AQM parameters
set opt(minth) 30;
set opt(maxth) 0;
set opt(adaptive) 1;     # 1 for Adaptive 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) duplex
$ns delay $nodes(bs1) $nodes(ms) $propDL($t) duplex
$ns bandwidth $nodes(bs2) $nodes(ms) $bwDL($t) duplex
$ns delay $nodes(bs2) $nodes(ms) $propDL($t) duplex
```

```
$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 A
gent/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"
}
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

Program Outcome :

- Simulate and demonstrate the performance of CDMA.

Viva Questions:

- Define CDMA?

PART-B

Java is a general-purpose computer programming language that is simple, concurrent, class-based, object-oriented language. The compiled Java code can run on all platforms that support Java without the need for recompilation hence Java is called as "write once, run anywhere" (WORA).

The Java compiled intermediate output called “byte-code” that can run on any Java virtual machine (JVM) regardless of computer architecture. The language derives much of its syntax from C and C++, but it has fewer low-level facilities than either of them.

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

1. Compile Java Program from Command Prompt
   [root@host ~]# javac Filename.java

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

2. Run Java program from Command Prompt
   [root@host ~]# java Filename

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

Program 7.Write a program for error detecting code using CRC-CCITT (16-bits).

Program Objective:

- Understand the operation of CRC-CCITT.

Whenever digital data is stored or interfaced, data corruption might occur. Since the beginning of computer science, developers have been thinking of ways to deal with this type of problem. For serial data they came up with the solution to attach a parity bit to each sent byte. This simple detection mechanism works if an odd number of bits in a byte changes, but an even number of false bits in one byte will not be detected by the parity check.

To overcome this problem developers have searched for mathematical sound mechanisms to detect multiple false bits. The **CRC** calculation or *cyclic redundancy check* was the result of this. Nowadays CRC calculations are used in all types of communications. All packets sent over a network connection are checked with a CRC. Also each data block on your hard disk has a CRC value attached to it.

Modern computer world cannot do without these CRC calculations. So let's see why they are so widely used. The answer is simple; they are powerful, detect many types of errors and are extremely fast to calculate especially when dedicated hardware chips are used.

The idea behind CRC calculation is to look at the data as one large binary number. This number is divided by a certain value and the remainder of the calculation is called the CRC. Dividing in the CRC calculation at first looks to cost a lot of computing power, but it can be performed very quickly if we use a method similar to the one learned at school. We will as an example calculate the remainder for the character 'm'—which is 1101101 in binary notation— by dividing it by 19 or 10011. Please note that 19 is an odd number.

This is necessary as we will see further on. Please refer to your schoolbooks as the binary calculation method here is not very different from the decimal method you learned when you were young. It might only look a little bit strange. Also notations differ between countries, but the method is similar.

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

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

You can easily increase the number of bits of the test data string—for example to 56 bits if we use our example value "*Lammert*"—and the result can be calculated with 56 binary comparisons and an average of 28 binary subtractions. This can be implemented in hardware directly with only very few transistors involved. Also software algorithms can be very efficient.

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

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

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

	CRC-CCITT	**CRC-16**	**CRC-32**
Checksum Width	16 bits	16 bits	32 bits
Generator Polynomial	10001000000100001	11000000000000101	10000010011000001000111011011 0111

International Standard CRC Polynomials

Algorithm:-

1. Given a bit string, append 0^S to the end of it (the number of 0^s is the same as the degree of the generator polynomial) let B(x) be the polynomial corresponding to B.

2. Divide B(x) by some agreed on polynomial G(x) (generator polynomial) and determine the remainder R(x). This division is to be done using Modulo 2 Division.

3. Define T(x) = B(x) –R(x)
   (T(x)/G(x) => remainder 0)

4. Transmit T, the bit string corresponding to T(x).
   Let T' represent the bit stream the receiver gets and T'(x) the associated polynomial. The receiver divides T1(x) by G(x). If there is a 0 remainder, the receiver concludes T = T' and no error occurred otherwise, the receiver concludes an error occurred and requires a retransmission

/* CRC */

//Source Code:

```
import java.io.*;
class Crc
{
public static void main(String args[]) throws IOException
{
BufferedReader br=new BufferedReader(new
InputStreamReader(System.in));
int[ ] data;
int[ ]div;
int[ ]divisor;
int[ ]rem;
int[ ] crc;
int data_bits, divisor_bits, tot_length;

System.out.println("Enter number of data bits : ");
data_bits=Integer.parseInt(br.readLine());
data=new int[data_bits];

System.out.println("Enter data bits : ");
for(int i=0; i<data_bits; i++)
data[i]=Integer.parseInt(br.readLine());

System.out.println("Enter number of bits in divisor : ");
divisor_bits=Integer.parseInt(br.readLine());
divisor=new int[divisor_bits];

System.out.println("Enter Divisor bits : ");
for(int i=0; i<divisor_bits; i++)
divisor[i]=Integer.parseInt(br.readLine());


tot_length=data_bits+divisor_bits-1; //9
div=new int[tot_length];//9
rem=new int[tot_length]; //9
crc=new int[tot_length];//9


/*.....CRC GENERATION.....*/
for(int i=0;i<data.length;i++)
div[i]=data[i];
System.out.print("Dividend (after appending 0's) are : ");
for(int i=0; i< div.length; i++)
System.out.print(div[i]);
System.out.println();

for(int j=0; j<div.length; j++)
{
rem[j] = div[j];
}
rem=divide(div, divisor, rem);
for(int i=0;i<div.length;i++)          //append dividend and ramainder
{
```

```
crc[i]=(div[i]^rem[i]);
}
System.out.println();
System.out.println("CRC code : ");
for(int i=0;i<crc.length;i++)
System.out.print(crc[i]);


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

/*
System.out.print("crc bits are : "); for(int i=0; i< crc.length; i++)
System.out.print(crc[i]); System.out.println();
*/

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

for(int i=0; i< rem.length; i++)
{
if(rem[i]!=0)
{
System.out.println("Error");
break;
}
if(i==rem.length-1)
System.out.println("No Error");
}
System.out.println("THANK YOU.....)");
}

static int[] divide(int div[],int divisor[], int rem[])
{
int cur=0;
while(true)
{
for(int i=0;i<divisor.length;i++)
rem[cur+i]=(rem[cur+i]^divisor[i]);

while(rem[cur]==0 && cur!=rem.length-1)
cur++;
if((rem.length-cur)<divisor.length)
break;
}
return rem;
}
}
```

Output1

```
enter number of data bits
7
enter data bits
1
0
1
1
0
0
1
enter number of bits in divisor
3
enter divisor bits
1
0
1
Dividend after appending 0's(1 0 1 1 0 0 1 0 0 )
CRC CODE
1 0 1 1 0 0 1 1 1
enter CRC code of 9 bits
1
0
1
1
0
0
1
1
1
NO error
Thank you
```

Program Outcome

- Identify and apply the operation of CRC-CCITT.

Viva Questions:

- Explain the features of JAVA?
- What is CRC-CCITT(16bits)?
- How CRC will detect error in a program?

Program 8:Write a program to find the shortest path between vertices using bellman-ford algorithm.

Program Objective:

- Understand the Implementation of the shortest path for bellman-ford algorithm.

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

Routers that use this algorithm have to maintain the distance tables (which is a one-dimension array -- "a vector"), which tell the distances and shortest path to sending packets to each node in the network. The information in the distance table is always up date by exchanging information with the neighboring nodes. The number of data in the table equals to that of all nodes in networks (excluded itself).

The columns of table represent the directly attached neighbors whereas the rows represent all destinations in the network. Each data contains the path for sending packets to each destination in the network and distance/or time to transmit on that path (we call this as "cost"). The measurements in this algorithm are the number of hops, latency, the number of outgoing packets, etc.

The Bellman–Ford algorithm is an algorithm that computes shortest paths from a single source vertex to all of the other vertices in a weighted digraph. It is slower than Dijkstra's algorithm for the same problem, but more versatile, as it is capable of handling graphs in which some of the edge weights are negative numbers. Negative edge weights are found in various applications of graphs, hence the usefulness of this algorithm.

If a graph contains a "negative cycle" (i.e. a cycle whose edges sum to a negative value) that is reachable from the source, then there is no cheapest path: any path that has a point on the negative cycle can be made cheaper by one more walk around the negative cycle. In such a case, the Bellman– Ford algorithm can detect negative cycles and report their existence

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 neighbor

/* Bellman-Ford */

```
import java.util.*;
public class Belmanford
 {
   private int D[];
   private int n;
   public static final int max_value=999;
   public Belmanford(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 -ve edge 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 no.of values");
      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();
    Belmanford b=new Belmanford(n);
    b.shortest(s,a);
    sc.close();
    }

}
```

Output1

```
enter the no.of values
4
enter the weighted matrix:
0 999 999 999
5 0 3 4
999 999 0 2
999 999 999 0
enter the source vertex:
2
distance of source 2 to1 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
```

Output2:

```
enter the no.of values
4
enter the weighted matrix:
0 4 999 5
999 0 999 999
999 -10 0 999
999 999 3 0
enter the source vertex:
1
distance of source 1to 1 is 0
distance of source 1 to 2 is-2
distance of source 1 to 3 is 8
distance of source 1 to 4 is 5
```

Output3

```
enter the no.of values
4
enter the weighted matrix:
0 4 5 999
999 0 999 7
999 7 0 999
999 999 -15 0
enter the source vertex:
1
the graph contains -ve edge cycle
```


Program Outcome

- Implement the shortest path for bellman-ford algorithm.

Viva Questions:

- What is bellman ford algorithm?
- What are the advantages and applications of bell man ford algorithm?

Program 9:Using TCP/IP sockets, write a client – server program to make the client send the file name and to make the server send back the contents of the requested file if present. Implement the above program using as message queues or FIFOs as IPC channels.

Program Objective:

- Understand the Implementation of the transport layer protocols.

The term network programming refers to writing programs that execute across multiple devices (computers), in which the devices are all connected to each other using a network.

The java.net package of the J2SE APIs contains a collection of classes and interfaces that provide the low-level communication details, allowing you to write programs that focus on solving the problem at hand.

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

- **TCP** − TCP stands for Transmission Control Protocol, which allows for reliable communication between two applications. TCP is typically used over the Internet Protocol, which is referred to as TCP/IP.
- **UDP** − UDP stands for User Datagram Protocol, a connection-less protocolthat allows for packets of data to be transmitted between applications.

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

- Connection based or connectionless: Is a connection established before communication or does each packet describe the destination?
- Packet based or streams based: Are there message boundaries or is it one stream?
- Reliable or unreliable: Can messages be lost, duplicated, reordered, orcorrupted?

Socket characteristics

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

- AF_UNIX: address format is UNIX pathname
- AF_INET: address format is host and port number

Common types are:

- virtual circuit: received in order transmitted and reliably
- datagram: arbitrary order, unreliable

Each socket type has one or more protocols. Ex:

- TCP/IP (virtual circuits)
- UDP (datagram)

Use of sockets:

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

Socket is an interface which enables the client and the server to communicate and pass on information from one another. Sockets provide the communication mechanism between two computers using TCP.

A client program creates a socket on its end of the communication and attempts to connect that socket to a server. When the connection is made, the server creates a socket object on its end of the communication.

The client and the server can now communicate by writing to and reading from the socket. The java.net.Socket class represents a socket, and the java.net.ServerSocket class provides a mechanism for the server program to listen for clients and establish connections with them.

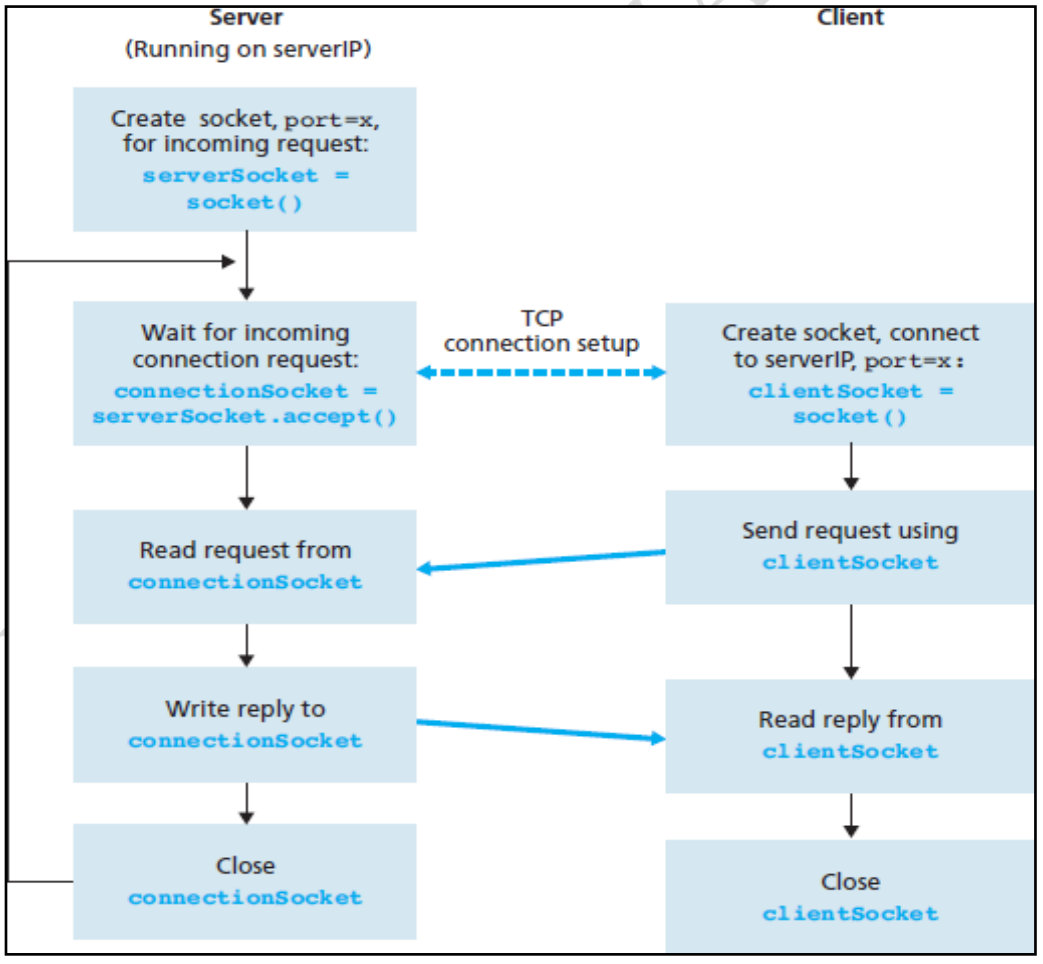


Fig. 6 The client-server application using TCP

As shown in the Fig. 1, the following steps occur when establishing a TCP connection between two computers using sockets –

- The server instantiates a ServerSocket object, denoting which port number communication is to occur on.
- The server invokes the accept() method of the ServerSocket class. This method waits until a client connects to the server on the given port.
- After the server is waiting, a client instantiates a Socket object, specifying the server name and the port number to connect to.
- The constructor of the Socket class attempts to connect the client to the specified server and the port number. If communication is established, the client now has a Socket object capable of communicating with the server.
- On the server side, the accept() method returns a reference to a new socket on the server that is connected to the client's socket.

Algorithm (Client Side)

1. Start.
2. Create a socket using socket() system call.
3. Connect the socket to the address of the server using connect() system call.
4. Send the filename of required file using send() system call.
5. Read the contents of the file sent by server by recv() system call.
6. Stop.

Algorithm (Server Side)

1. Start.
2. Create a socket using socket() system call.
3. Bind the socket to an address using bind() system call.
4. Listen to the connection using listen() system call.
5. accept connection using accept()
6. Receive filename and transfer contents of file with client.
7. Stop.

TCP Client

At client side:

```
/*TCPClient*/
import java.net.*;
import java.io.*;
public class TCPClient
{
        public static void main(String args[]) throws Exception {
                Socket sock=new Socket("127.0.0.1",4000);
                System.out.println("Enter the filename");
                BufferedReader keyRead=new BufferedReader(new InputStreamReader(System.in));
                String fname=keyRead.readLine();
                OutputStream ostream=sock.getOutputStream();
                PrintWriter pwrite=new PrintWriter(ostream,true);
                pwrite.println(fname);
                InputStream istream=sock.getInputStream();
                BufferedReader socketRead=new BufferedReader(new InputStreamReader(istream));
                String str;
                while((str=socketRead.readLine())!=null)
                {
```

```
                System.out.println(str);
            }
            pwrite.close();
            socketRead.close();
            keyRead.close();
        }
}
```

TCP Server

At server side:

```
/* TCPServer */
import java.net.*;
import java.io.*;
public class TCPServer
{
    public static void main(String args[]) throws Exception {
        ServerSocket sersock=new ServerSocket(4000);
        System.out.println("Server ready for Connection");
        Socket sock=sersock.accept();
        System.out.println("Connection is Successful and waiting for chatting");
        InputStream istream=sock.getInputStream();
        BufferedReader fileRead=new BufferedReader(new InputStreamReader(istream));
        String fname=fileRead.readLine();
        BufferedReader contentRead=new BufferedReader(new FileReader(fname));
        OutputStream ostream=sock.getOutputStream();
        PrintWriter pwrite=new PrintWriter(ostream,true);
        String str;
        while((str=contentRead.readLine())!=null)
        {
            pwrite.println(str);
        }
        sock.close();
        sersock.close();
        pwrite.close();
        fileRead.close();
        contentRead.close();

    }
}
```

Note: Create two different files TcpClient.java and TcpServer.java. Follow the steps given:

1. Open a terminal run the server program and provide the filename to send
2. Open one more terminal run the client program and provide the IP address of the server. We can give localhost address “127.0.0.1” as it is running on same machine or give the IP address of the machine.
3. Send any start bit to start sending file.

First Method of Executing TCP/IP sockets, write a client – server program

Output1

```
student@student:~/naveen$ javac TCPServer.java
student@student:~/naveen$ java TCPServer
Server ready for Connection
Connection is Successful and waiting for chatting

student@student:~/naveen$
student@student:~/naveen$ javacTCPClient.java
student@student:~/naveen$ java TCPClient
Enter the filename
abc.txt
atme college of engineering,mysuru
student@student:~/naveen$
```

Second Method of Executing TCP/IP sockets, write a client – server program

Output2

TCPServer

first run TCPServer program .you will get below message that server is started and ready to connect with client

Server ready for Connection

TCPClient

Next run TCPClient program

Enter the filename
/home/student/naveen/abc.txt
hello atme college of engineering

Program Outcome

- Implement transport layer protocols.

Viva Questions:

- Explain the client/server model in detail.
- What do you mean by TCP/IP Socket and what is its use?

Program 10:Write a program on datagram socket for client/server to display the messages on client side, typed at the server side.

Program Objective:

- Understand the Implementation of the data link layer protocols.

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.

Datagram packets are used to implement a connectionless packet delivery service supported by the UDP protocol. Each message is transferred from source machine to destination based on information contained within that packet. That means, each packet needs to have destination address and each packet might be routed differently, and might arrive in any order. Packet delivery is not guaranteed.

Java supports datagram communication through the following classes:

- DatagramPacket
- DatagramSocket

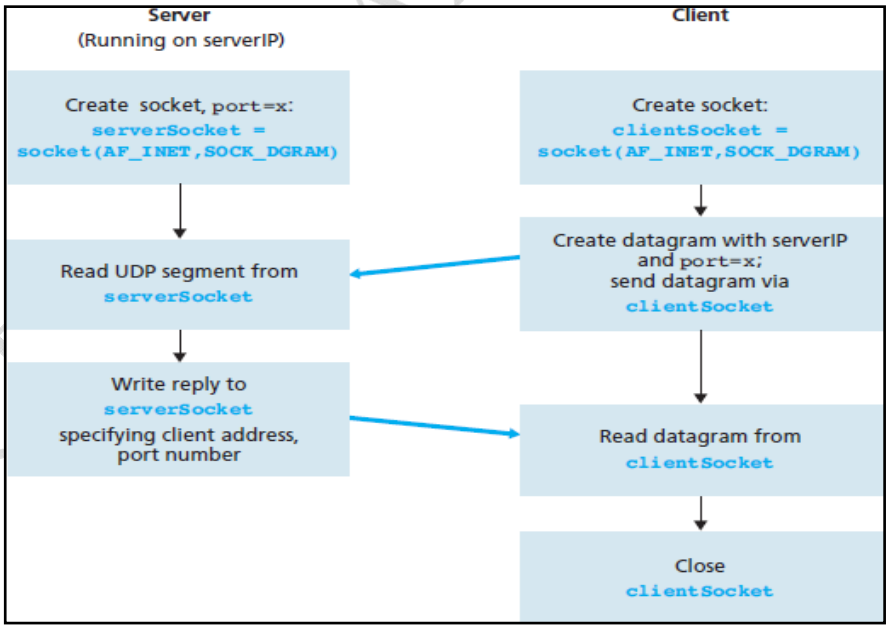


Fig -7-UDP client/server communication flow:

/* Datagram Socket Program */

UDP Server

```
import java.io.*;
import java.net.*;
public class UDPServer
{
  public static void main(String[] args)
 {
DatagramSocket skt=null;
try
{
System.out.println("server is started");
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].toUpperCase()+ " from server to client").getBytes();
DatagramPacket reply = new
DatagramPacket(sendMsg,sendMsg.length,request.getAddress(),request.getPort());
skt.send(reply);
}
}
catch(Exception ex)
{
System.out.println(ex.getMessage());
}
}
}
```

UDP Client

```
import java.io.*;
import java.net.*;
public class UDPClient
{
 public static void main(String[] args)
{
DatagramSocket skt;
try
{
skt=new DatagramSocket();
String msg= "atme college ";
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)
{
   System.out.println(ex.getMessage());
}
}
}
```

Output1:

client received : COLLEGE from server to client

Program Outcome

- Implement data link layer protocols.

Viva Questions:

- What do you mean by UDP Socket and what is its use.
- What is the difference between TCP/IP and UDP connection?
- Which connection is more reliable?

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

Program Objective:

- Understand the operation of RSA algorithm.

- **RSA** is algorithm used by modern computers to encrypt and decrypt messages. It is an asymmetric cryptographic algorithm. Asymmetric means that there are two different keys. This is also called *public key cryptography*, because one of them can be given to everyone. The other key must be kept private.
- It is based on the fact that finding the factors of an integer is hard (the factoring problem). RSA stands for **Ron Rivest, Adi Shamir** and **Leonard Adleman**, who first publicly described it in 1978. A user of RSA creates and then publishes the product of two large prime numbers, along with an auxiliary value, as their public key.
- The prime factors must be kept secret. Anyone can use the public key to encrypt a message, but with currently published methods, if the public key is large enough, only someone with knowledge of the prime factors can feasibly decode the message.
- RSA involves a public key and private key. The public key can be known to everyone; it is used to encrypt messages. Messages encrypted using the public key can only be decrypted with the private key. 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.
- Basically RSA is cryptographic algorithm which is meant to encrypt the data, generally used in network security applications while we are sending the data from one source to destination. The concept of RSA algorithm starts with a two keyconcepts, it uses two keys (asymmetric keys) one is considered as the public key and another is a private key.
- It was developed because using the symmetric encryption algorithm is easy but the key distribution is difficult, so the concept of two key concept appears to be more efficient. The whole algorithm depends on the fact that "It is not possible to judge another key when attacker gets one key" Here in two keys, one key is taken as the public key another as a private key.

- Public key is available for everyone to access, so whenever sender want to send the data to receiver, he uses the public key of receiver (as it is available for use to all) and encrypts the data using the key, this encrypted data is called cipher text, when receiver receives the cipher text, he can decrypt the data using his private key. Here even if theattacker knows the encryption algorithm, he can't do anything until the keys are available.

A very simple example of RSA encryption

1.Select primes p = 11, q = 3.

2. n = p*q = 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).

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

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

Hence the ciphertext c = 13.

To check decryption we compute

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

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

One way of calculating m' is as follows:-

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

= ($13^{\wedge 3}$ 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.

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.

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.

4. Compute the secret exponent d, 1 < d < phi, such that $ed \equiv 1 \pmod{phi}$.

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.

Note: It is possible to find a smaller d by using lcm(p-1,q-1) instead of phi, lcm(p-1,q-1) = phi /gcd(p-1,q-1)).

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.
3. Computes the ciphertext c = $m^{\wedge e}$ mod n.
4. Sends the ciphertext c to B.

Decryption

Recipient B does the following:-

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

/* RSA Key Generation */

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;
}
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;
}
}
```

Output

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

Program Outcome

- Implement data link layer protocols. Identify and apply the operation of RSA algorithm.

Viva Questions:

- What is RSA? Explain its algorithm.
- What do you mean by encryption and decryption of data?

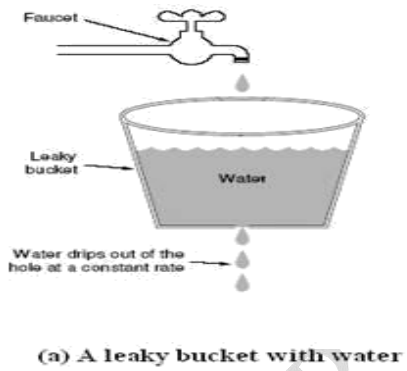
Program 12:Write a program for congestion control using leaky bucket algorithm.

Program Objective:

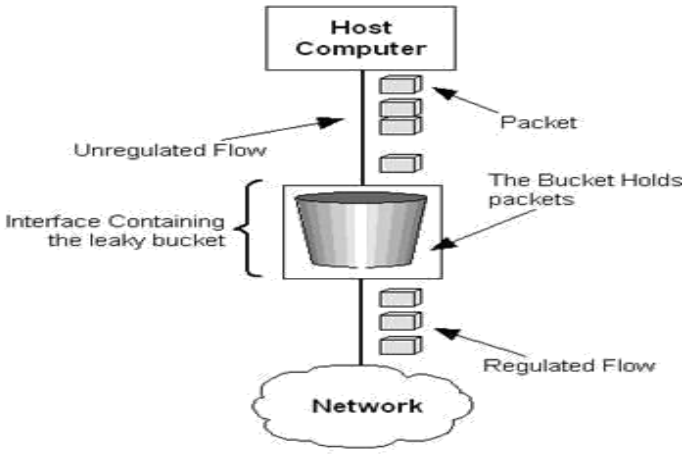
- Understand the operation of congestion control using leaky bucket algorithm.

The main concept of the leaky bucket algorithm is that the output data flow remains constant despite the variant input traffic, such as the water flow in a bucket with a small hole at the bottom. In case the bucket contains water (or packets) then the output flow follows a constant rate, while if the bucket is full any additional load will be lost because of spillover. In a similar way if the bucket is empty the output will be zero.

From network perspective, leaky bucket consists of a finite queue (bucket) where all the incoming packets are stored in case there is space in the queue, otherwise the packets are discarded. In order to regulate the output flow, leaky bucket transmits one packet from the queue in a fixed time (e.g. at every clock tick). In the following figure we can notice the main rationale of leaky bucket algorithm, for both the two approaches (e.g. leaky bucket with water (a) and with packets (b)).



(a) A leaky bucket with water



(b) A leaky bucket with packets

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

The leaky-bucket algorithm:

The algorithm can be conceptually understood as follows:

- Consider a bucket with a hole in the bottom.
- The empty space of the bucket represents an amount of credit available measured in bytes.

- The size of the bucket is *b* bytes. This means that if the bucket is empty, *b* bytes of credit is available.
- If a packet arrives and its size is less than the available credit, the packet can be forwarded. Otherwise, it is discarded or queued depending on the application.
- The bucket leaks through the hole in its bottom at a constant rate of *r* bytes per second, this indicates credit accumulation.

/* Leaky Bucket */

```
import java.util.Scanner;
import java.lang.*;
public class lab7
{
public static void main(String[] args)
{
int i;
int a[]=new int[20];
int buck_rem=0,buck_cap=4,rate=3,sent,recv;
Scanner in = new
Scanner(System.in);
System.out.println("Enter the number of packets");
int n = in.nextInt();
System.out.println("Enter the packets");
for(i=1;i<=n;i++)
a[i]= in.nextInt();
System.out.println("Clock \t packet size \t accept \t sent \t remaining");
for(i=1;i<=n;i++)
{
if(a[i]!=0)
{
if(buck_rem+a[i]>buck_cap)
recv=-1;
else
{
recv=a[i];
buck_rem+=a[i];
}
}
else
recv=0;
if(buck_rem!=0)
{
if(buck_rem<rate)
{sent=buck_rem;
buck_rem=0;
}
else
{
sent=rate;
buck_rem=buck_rem-rate;
```

```
}
}
else
sent=0;
if(recv==-1)
System.out.println(+i+ "\t\t" +a[i]+ "\t dropped \t" +  sent +"\t" +buck_rem);
else
System.out.println(+i+ "\t\t" +a[i] +"\t\t" +recv +"\t" +sent + "\t" +buck_rem);
}
}
}
```

Output

```
C:\Users\Admin>javac LeakyBucket.java

C:\Users\Admin>java LeakyBucket
Enter the number of packets
4
Enter the packets
1
2
3
5
```

Clock	packet size	accept	sent	remaining
1	1	1	1	0
2	2	2	2	0
3	3	3	3	0
4	5	dropped	0	0

Program Outcome

- Identify and apply the operation of congestion control using leaky bucket algorithm.

Viva Questions:

- What is congestion control?
- Explain leaky bucket algorit