BENGALURU INDIA

Networks Laboratory

BTCS14F6600

SCHOOL OF COMPUTING AND INFORMATION TECHNOLOGY

Name	
Srn	
Branch	
Semester	
Section	
Academic Year	

Lab Requirements

Following are the required hardware and software for this lab, which is available in the laboratory.

- **Hardware:** Desktop system or Virtual machine in a cloud with OS installed. Presently in the Lab, Pentium IV Processor having 1 GB RAM and 250 GB Hard Disk is available. Desktop systems are dual boot having Windows as well as Linux OS installed on them.
- **Software:** UBUNTU 14.04 LTS operating system, NS-2 Simulator

STEPS FOR NS2 INSTALLATION IN UBUNTU operating system:

STEP 1: Download and Extract ns2 using terminal by using given commands:
cd ~/
sudo wget http://softlayer-sng.dl.sourceforge.net/project/nsnam/allinone/ns-allinone-2.35/ns-allinone-2.35.tar.gz
tar -xvzf ns-allinone-2.35.tar.gz

STEP 2: Ns2 requires some packages including GCC- version 4.4 to work correctly. So install all of them by using the following command:

sudo apt-get install build-essential autoconf automake libxmu-dev
sudo apt-get install gcc-4.4

STEP 3: After installation of packages, we have to make a change in the "ls.h" file. Navigate to the folder "linkstate" which is assumed be in the ns folder extracted and is in the home folder of your system.

cd ~/ns-allinone-2.35/ns-2.35/linkstate

STEP 4: Open the file named "ls.h" by using given command and scroll to the 137th line. You will find the word "error"; change that to "this->error". To open the file use the following command: *gedit ls.h*

STEP 5: Now, we need to inform ns which version of GCC will be used. To do so, go to your ns folder and type the following command. Here, change CC= @CC@ to CC=gcc-4.4, then save and exit.
Sudo gedit ns-allinone-2.34/otcl-1.13/Makefile.in

STEP 6: Now we are ready to install ns2. Run the install script using the following commands:
sudo su cd ~/ns-allinone-2.35/./install

STEP 7: The final step is to tell the system, where the files for ns2 are installed or present. Open the .bashrc file using the given command. Add the given lines but be sure that the path you are giving is the same where the folder ns is present. For example, if you have installed it in a folder "/home/abc", then replace "/home/pcquest/ns-allinone-2.35/otcl-1.14" with "/home/abc/ns-allinone-2.35/otcl-1.14".
sudo gedit ~/.bashrc

CONTENTS

	bandwidth and 10 ms of delay. Create two PING agents and attach them to the nodes n0 and n2. Connect the two agents and schedule the transmission of PING messages at an interval of 0.2 seconds and finish at 1.0 second. Write a Tcl script to observe the packet flow for the given network and observe the output in NAM for this network scenario.	
5.	A network consists of n nodes (say n =6). The duplex links between the nodes is as follows: • n0 and n2 has 2Mbps of bandwidth and 10 ms of delay, • n1 and n2 has 2Mbps of bandwidth and 10 ms of delay, The Simplex links between the nodes is as follows: • n2 and n3 has 0.3Mbps of bandwidth and 100 ms of delay. • n3 and n2 has 0.3Mbps of bandwidth and 100 ms of delay. The LAN is established between the nodes n3, n4 and n5 with 0.5 Mbps of band width and 40ms delay. Each node uses DropTail queue of which the maximum size is 10 . Write a Tcl script to observe the packet flow for the given network and observe the output in NAM for this network scenario.	44
6.	For a wireless network consisting of three mobile nodes (n0-n2), Nodes are configured with the specific parameters of a wireless node. Initial location of the node is fixed. Nodes are given mobility with fixed speed and fixed destination location. TCP agent is attached to node0 and TCP sink agent is attached to node1. Both the agents are connected and FTP application is attached to TCP agent. Write a Tcl script and make an ad-hoc simulation to analyze the output in the trace file. Use the routing protocol as Adhoc on demand distance vector (AODV).	59
	Part-B	
7.	Assume a message represented by the polynomial $M(x) = x^n + ... x2 + x + 1$, choosing 'n' in the range of 4 to 16. Message is sent from the sender with a checksum. When it arrives at the receiver's end, checksum is recalculated for detecting the error. Hence, write a program for error detecting code using CRC-CCITT (16-bits), considering 10001000000100001 as the standard divisor polynomial.	65
8.	A network topology consists of n nodes (varying from 4-10). Links connecting the nodes have some costs/weights associated with them. Links for unreachable nodes are indicated by infinity (say, 999).The sample cost adjacency matrix is given below for a network of 4 nodes: 0 2 99 99 2 0 1 99 99 1 0 4	81

	99 99 4 0 Write a program for distance vector algorithm to find suitable path for transmission by using the routing table for each node.	
9.	Using TCP/IP sockets, write a client-server program to make client send the file name and the server to send back the contents of the requested file file name "sample.txt" with the following contents:"HI I AM AT REVA UNIVERSITY. SEE YOU SOON". Steps for establishing a TCP socket on the client side are the following: • Create a socket using the socket() function; • Connect the socket to the address of the server using the connect() function; • Send and receive data by means of the read() and write() functions. The steps involved in establishing a TCP socket on the ***server side*** are as follows: • Create a socket with the socket() function; • Bind the socket to an address using the bind() function; • Listen for connections with the listen() function; • Accept a connection with the accept() function system call. • This call typically blocks until a client connects with the server. • Send and receive data by means of send() and receive(). Display suitable error message in case the the file is not present in the server.	91
10.	There is a single Server process which runs continuously in background, eventhough if there is no client to interact with it. Client processes runs in foreground and interacts with the server process. Both the client and server processes run on the same machine. The Client process accepts a command (a shell command) from the user and send's it to the Server via a FIFO which is a public channel between Client and Server for processing. Assume this FIFO name as **PUBLIC fifo** since its existence is known to all clients and the server. Once the command is received, the Server executes it using the **popen–pclose** sequence (which generates an unnamed pipe in the Server process). After execution Server process returns the output of the command executed to the client over a FIFO which is a private channel between the client and server. Let us name this FIFO as PRIVATE fifo. The Client, upon receipt, displays the output on the screen. Implement this scenario using message queues or FIFOs as interprocess communication channels by adopting the following procedure: **Functionality steps of a Client process:** • Create a unique name for the PRIVATE fifo and invoke it. • Open the PUBLIC fifo in write mode.	105

	• Prompt for command from user. • Write command to PUBLIC fifo for Server to process. • Open PRIVATE fifo in read mode to read the contents from Server. **Steps of a Server process:** • Generate a PUBLIC fifo and open it in both read and write mode. Wait for the message from a Client process. • Read the message from PUBLIC fifo. • Open the Client's PRIVATE fifo in write mode. • Execute the command from Client process using popen. • Write the output into Client's PRIVATE fifo. **NOTE:** Each and every Client process should have its own unique PRIVATE fifo to receive information from Server.	
11.	Choose the two prime numbers, p=17 and q=11. Write a program for public key encryption system using RSA algorithm to encrypt and decrypt the message. For a message M=”he12ll34o“, show the encryption and decryption	114
12.	Examine node transmitting/receiving packets to/from other nodes. Using a random function; vary the packet size, considering the following cases. Case i: Output rate of 5 Bytes and Bucket size of 10 Bytes. Case ii: Output rate of 10 Bytes and Bucket size of 5 Bytes. Write a program for congestion control implementing leaky bucket algorithm.	140

PART A Programs

INTRODUCTION

In communication and computer network research, **network simulation** is a technique where a program models the behavior of a network either by calculating the interaction between the different network entities (hosts/packets, etc.) using mathematical formulas, or actually capturing and playing back observations from a production network. The behavior of the network and the various applications and services it supports can then be observed in a test lab; various attributes of the environment can also be modified in a controlled manner to assess how the network would behave under different conditions.

Simulation is a meaningful evaluation approach only when it produces "trustable" results. Validation is needed to certify that the simulation models reproduce correctly the characteristics and dynamics of the system under study. The validation of a network model is done by:

1. Comparing Simulation vs Analytical Model, or
2. Comparing Simulation vs Real Measurements

Examples of network simulators

There are both free/open-source and proprietary network simulators available. Examples of notable network simulators / emulators include:

- ns (open source)
- OPNET (proprietary software)
- NetSim (proprietary software)

Uses of network simulators

Network simulators provide a cost effective method for

a. Network design validation for enterprises / data centers /sensor networks etc.
b. Analyzing Utilities distribution communication, railway signaling / communication etc
c. Network protocol R & D
d. Defense applications such as HF / UHF / VHF MANET networks, Tactical data links etc.

There are a wide variety of network simulators, ranging from the very simple to the very complex. Minimally, a network simulator must enable a user to

1. Model the network topology specifying the nodes on the network and the links between those nodes
2. Model the application flow (traffic) between the nodes
3. Providing network performance metrics as output
4. Visualization of the packet flow
5. Technology / protocol evaluation and device designs
6. Logging of packet / events for drill down analyses / debugging

Network emulation

A network emulator allows users to introduce real devices and applications into a test network (simulated) that alters packet flow in such a way as to mimic the behavior of a live network. Live traffic can pass through the simulator and be affected by objects within the simulation.

The typical methodology is that real packets from a live application reach the emulation server (where the virtual network is simulated. The real packet gets modulated into a simulation packet. The Simulation packet gets demodulated into real packet after experiencing effects of loss, errors, delay, jitter etc., thereby transferring these network effects into the real packet. Thus it is as-if the real packet flowed through the real networks but in reality it flowed through the simulated network.

Emulation is widely used in the design stage for validating communication networks prior to deployment.

Wireshark is a free and open source packet analyzer. It is used for network troubleshooting, analysis, software and communications protocol development, and education. Originally named Ethereal, the project was renamed Wireshark in May 2006 due to trademark issues.[4]

Wireshark is cross-platform, using the Qt widget toolkit in current releases to implement its user interface, and using pcap to capture packets; it runs on Linux, macOS, BSD, Solaris, some other Unix-like operating systems, and Microsoft Windows. There is also a terminal-based (non-GUI) version called TShark. Wireshark, and the other programs distributed with it such as TShark, are free software, released under the terms of the GNU General Public License.

Backend Environment of Network Simulator

Network Simulator is mainly based on two languages.They are C++ and OTcl. OTcl is the object oriented version of Tool Command language.The network simulator is a bank of of different network and protocol objects. C++ helps in the following way:

- It helps to increase the efficiency of simulation.
- Its is used to provide details of the protocols and their operation.
- It is used to reduce packet and event processing time.

OTcl helps in the following way:

- With the help of OTcl we can describe different network topologies
- It helps us to specify the protocols and their applications
- It allows fast development
- Tcl is compatible with many platforms and it is flexible for integration
- Tcl is very easy to use and it is available in free

Basics of Tcl Programming (w.r.t. ns2)

Before we get into the program we should consider the following things:

- Initialization and termination aspects of network simulator.
- Defining the network nodes,links,queues and topology as well.
- Defining the agents and their applications
- Network Animator(NAM)
- Tracing

Initialization

To start a new simulator we write

set ns [new Simulator]

From the above command we get that a variable ns is being initialized by using the set command. Here the code [new Simulator] is a instantiation of the class Simulator which uses the reserved word 'new'. So we can call all the methods present inside the class simulator by using the variable ns.

Creating the output files

```
#To create the trace files we write

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

#To create the nam files we write

 set namfile1 [open out.nam w]
 $ns namtrace-all $namfile
```

In the above we create a output trace file out.tr and a nam visualization file out.nam. But in the Tcl script they are not called by their names declared,while they are called by the pointers initialized for them such as tracefile1 and namfile1 respectively.The line which starts with '#' are commented.The next line opens the file 'out.tr' which is used for writing is declared 'w'.The next line uses a simulator method trace-all by which we will trace all the events in a particular format.

The termination program is done by using a 'finish' procedure

```
 # Defining the 'finish' procedure'

proc finish {} {
     global ns tracefile1 namfile1
     $ns flush-trace
     close $tracefile
     close $namfile
     exec nam out.nam &
     exit 0
    }
```

In the above the word 'proc' is used to declare a procedure called 'finish'.The word 'global' is used to tell what variables are being used outside the procedure.

'flush-trace' is a simulator method that dumps the traces on the respective files.the command 'close' is used to close the trace files and the command 'exec' is used to execute the nam visualization.The command 'exit' closes the application and returns 0 as zero(0) is default for clean exit.

In ns we end the program by calling the 'finish' procedure

```
1  #end the program
2  $ns at 125.0 "finish"
```

Thus the entire operation ends at 125 seconds.To begin the simulation we will use the command

```
1  #start the the simulation process
2  $ns run
```

Defining nodes,links,queues and topology

Way to create a node:
view source
print?
1 set n0 [ns node]

In the above we created a node that is pointed by a variable n0.While referring the node in the script we use $n0. Similarly we create another node n2.Now we will set a link between the two nodes.

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

So we are creating a bi-directional link between n0 and n2 with a capacity of 10Mb/sec and a propagation delay of 10ms. In NS an output queue of a node is implemented as a part of a link whose input is that node to handle the overflow at the queue.But if the buffer capacity of the output queue is exceeded then the last packet arrived is dropped and here we will use a 'DropTail' option.Many other options such as RED(Random Early Discard) mechanism, FQ(Fair Queuing), DRR(Deficit Round Robin), SFQ(Stochastic Fair Queuing) are available.

So now we will define the buffer capacity of the queue related to the above link

```
1  #Set queue size of the link
2  $ns queue-limit $n0 $n2 20
```

If we summarize the above three things we get

```
01  #create nodes
02
03  set n0 [$ns node]
04  set n1 [$ns node]
05  set n2 [$ns node]
06  set n3 [$ns node]
07  set n4 [$ns node]
08  set n5 [$ns node]
09
10  #create links between the nodes
11
12  $ns duplex-link $n0 $n2 10Mb 10ms DropTail
13  $ns duplex-link $n1 $n2 10Mb 10ms DropTail
14  $ns simplex-link $n2 $n3 0.3Mb 100ms DropTail
15  $ns simplex-link $n3 $n2 0.3Mb 100ms DropTail
16  $ns duplex-link $n0 $n2 0.5Mb 40ms DropTail
17  $ns duplex-link $n0 $n2 0.5Mb 40ms DropTail
18
19  #set queue-size of the link (n2-n3) to 20
20  $ns queue-limit $n2 $n3 20
```

Agents and applications

TCP

TCP is a dynamic reliable congestion protocol which is used to provide reliable transport of packets from one host to another host by sending acknowledgements on proper transfer or loss of packets.Thus TCP requires bi-directional links in order for acknowledgements to return to the source.

Now we will show how to set up tcp connection between two nodes

```
1  #setting a tcp connection
2
3  set tcp [new Agent/TCP]
4  $ns attach-agent $n0 $tcp
5  set sink [new Agent/TCPSink]
6  $ns attach-agent $n4 $sink
7  $ns connect $tcp $sink
8  $tcp set fid_1
9  $tcp set packetSize_552
```

The command 'set tcp [new Agent/TCP]' gives a pointer called 'tcp' which indicates the tcp agent which is a object of ns.Then the command '$ns attach-agent $n0 $tcp' defines the source node of tcp connection. Next the command 'set sink [new Agent/TCPSink]' defines the destination of tcp by a pointer called sink. The next command '$ns attach-agent $n4 $sink' defines the destination node as n4.Next, the command '$ns connect $tcp $sink' makes the TCP connection between the source and the destination.i.e n0 and n4.When we have several flows such as TCP, UDP etc in a network. So, to identify these flows we mark these flows by using the command '$tcp set fid_1'. In the last line we set the packet size of tcp as 552 while the default packet size of tcp is 1000.

FTP over TCP

File Transfer Protocol(FTP) is a standard mechanism provided by the Internet for transferring files from one host to another. Well this is the most common task expected from a networking or a inter networking . FTP differs from other client server applications in that it establishes between the client and the server. One connection is used for data transfer and other one is used for providing control information. FTP uses the services of the TCP. It needs two connections. The well Known port 21 is used for control connections and the other port 20 is used for data transfer.

Well here we will learn in how to run a FTP connection over a TCP

```
1  #Initiating FTP over TCP
2
3  set ftp [new Application/FTP]
4  $ftp attach-agent $tcp
```

In above,the command 'set ftp [new Application/FTP]' gives a pointer called 'ftp' which indicates the FTP application.Next, we attach the ftp application with tcp agent as FTP uses the services of TCP.

UDP

The User datagram Protocol is one of the main protocols of the Internet protocol suite.UDP helps the host to send send messages in the form of datagrams to another host which is present in a Internet protocol network without any kind of requirement for channel transmission setup. UDP provides a unreliable service and the datagrams may arrive out of order,appear duplicated, or go missing without notice. UDP assumes that error checking and correction is either not necessary or performed in the application, avoiding the overhead of such processing at the network interface level. Time-sensitive applications often use UDP because dropping packets is preferable to waiting for delayed packets, which may not be an option in a real-time system.

Now we will learn how to create a UDP connection in network simulator.

```
1  # setup a UDP connection
2  set udp [new Agent/UDP]
3  $ns attach-agent $n1 $udp
4  $set null [new Agent/Null]
5  $ns attach-agent $n5 $null
6  $ns connect $udp $null
7  $udp set fid_2
```

Similarly,the command 'set udp [new Agent/UDP]' gives a pointer called 'udp' which indicates the udp agent which is a object of ns.Then the command '$ns attach-agent $n1 $udp' defines the source node of udp connection. Next the command 'set null [new Agent/Null]' defines the destination of udp by a pointer called null. The next command '$ns attach-agent $n5 $null' defines the destination node as n5.Next, the command '$ns connect $udp $null' makes the UDP connection between the source and the destination.i.e n1 and n5.When we have several flows such as TCP,UDP etc in a network. So, to identify these flows we mark these flows by using the command '$udp set fid_2

Constant Bit Rate(CBR)

Constant Bit Rate (CBR) is a term used in telecommunications, relating to the quality of service.When referring to codecs, constant bit rate encoding means that the rate at which a codec's output data should be consumed is constant. CBR is useful for streaming multimedia content on limited capacity channels since it is the maximum bit rate that matters, not the average, so CBR would be used to take advantage of all of the capacity. CBR would not be the optimal choice for storage as it would not allocate enough data for complex sections (resulting in degraded quality) while wasting data on simple sections.

CBR over UDP Connection

```
1  #setup cbr over udp
2
3  set cbr [new Application/Traffic/CBR]
4  $cbr attach-agent $udp
5  $cbr set packetSize_1000
6  $cbr set rate_0.01Mb
7  $cbr set random _false
```

In the above we define a CBR connection over a UDP one. Well we have already defined the UDP source and UDP agent as same as TCP. Instead of defining the rate we define the time interval between the transmission of packets in the command '$cbr set rate_0.01Mb'. Next, with the help of the command '$cbr set random _false' we can set random noise in cbr traffic.we can keep the noise by setting it to 'false' or we can set the noise on by the command '$cbr set random _1'. We can set by packet size by using the command '$cbr set packetSize_(packetsize).We can set the packet size up to sum value in bytes.

Scheduling Events

In ns the tcl script defines how to schedule the events or in other words at what time which event will occur and stop. This can be done using the command

$ns at .

So here in our program we will schedule the ftp and cbr.

```
1  # scheduling the events

2

3  $ns at 0.1 "cbr start"

4  $ns at 1.0 "ftp start"

5  $ns at 124.0 "ftp stop"
```

```
6   $ns at 124.5 "cbr stop"
```

Network Animator(NAM)

When we will run the above program in ns then we can can visualize the network in the NAM. But instead of giving random positions to the nodes, we can give suitable initial positions to the nodes and can form a suitable topology. So, in our program we can give positions to the nodes in NAM in the following way

```
1   #Give position to the nodes in NAM
2
3   $ns duplex-link-op $n0 $n2 orient-right-down
4   $ns duplex-link-op $n1 $n2 orient-right-up
5   $ns simplex-link-op $n2 $n3 orient-right
6   $ns simplex-link-op $n3 $n2 orient-left
7   $ns duplex-link-op $n3 $n4 orient-right-up
8   $ns duplex-link-op $n3 $n5 orient-right-down
```

We can also define the color of cbr and tcp packets for identification in NAM.For this we use the following command

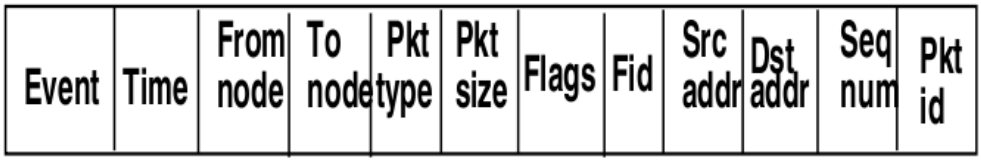
```
1   #Marking the flows
2   $ns color1 Blue
3   $ns color2 Red
```

To view the network animator we need to type the command: nam

Tracing

Tracing Objects

NS simulation can produce visualization trace as well as ASCII file corresponding to the events that are registered at the network. While tracing ns inserts four objects: EnqT,DeqT,RecvT & DrpT. EnqT registers information regarding the arrival of packet and is queued at the input queue of the link. When overflow of a packet occurs, then the information of thye dropped packet is registered in DrpT.DeqT holds the information abut the packet that is dequeued instantly.RecvT hold the information about the packet that has been received instantly.



Structure of Trace files

The first field is event.It gives you four possible symbols '+' '-' 'r' 'd'.These four symbols correspond respectively to enqueued, dequeued, received and dropped.

The second field gives the time at which the event occurs

The third field gives you the input node of the link at which the event occurs

The fourth field gives you the the output node at which the event occurs

The fifth field shows the information about the packet type.i.e whether the packet is UDP or TCP

The sixth field gives the packet size

The seventh field give information about some flags

The eight field is the flow id(fid) for IPv6 that a user can set for each flow in a tcl script.It is also used for specifying the color of flow in NAM display

The ninth field is the source address

The tenth field is the destination address

The eleventh field is the network layer protocol's packet sequence number

The last field shows the unique id of packet

Following are trace of two events:

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

The trace file can be viewed with the cat command:

cat out.tr

PART-A

PROGRAM 1: Point-to-Point network

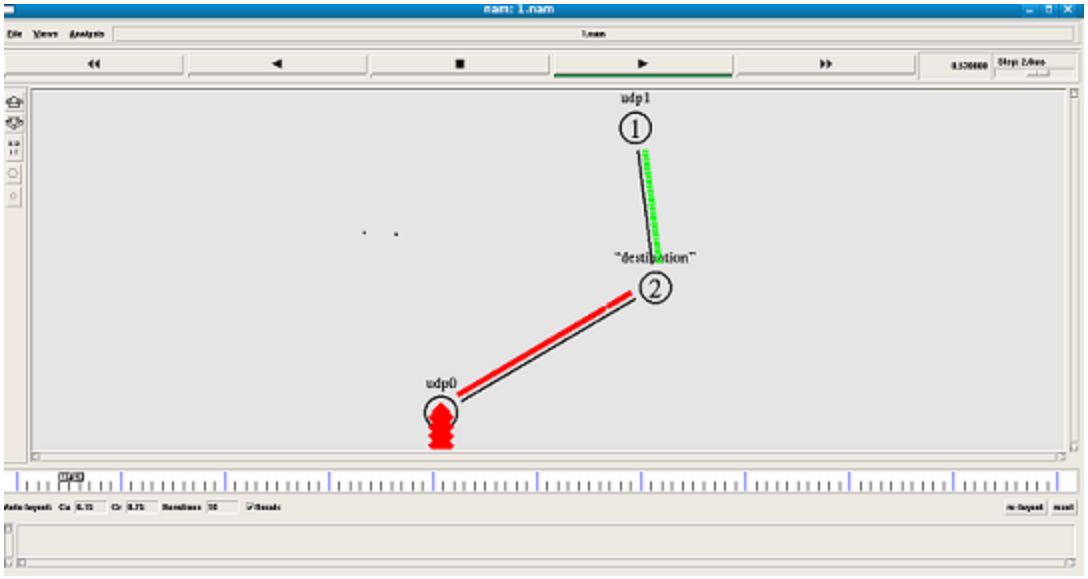
1	**Problem Statement**
	A network consists of three nodes (n0-n2). Link existence (duplex in nature) between the nodes is as follows: n0-n1 and n1-n2. The link no-n1 has 10 Kbps of bandwidth and 100 ms of delay. The link n1-n2 has 5 Mbps of bandwidth and 200 ms of delay. Node "n0" is having some data to send to node "n2" through node "n1", which is a hub device. Each node uses Drop Tail queue of which the maximum size is 10.Write a Tcl script to observe the packet flow for the given network in network animator (NAM).
2	**Student Learning Outcomes**
	After successful completion of this lab, the students will be able to • Familiarize the need and advantages of Simulation • Learn to simulate a point-to-point network with duplex link • Interpret the effect of varying Queue size and Bandwidth • Analyze the packet drop
3	**Design of the Program**
3.1	**Theory** A point-to-point connection refers to a communications connection between two nodes or endpoints. An example is a telephone call, in which one telephone is connected with one other, and what is said by one caller can only be heard by the other. This is contrasted with a point-to-multipoint or broadcast communication topology, in which many nodes can receive information transmitted by one node. Other examples of point-to-point communications links are leased lines, microwave relay links, and two way radio. Examples of point-to-multipoint communications systems are radio and television broadcasting. The term is also used in computer networking and computer architecture to refer to a wire or other connection that links only two computers or circuits, as opposed to other network topologies such as buses or crossbar switches which can connect many communications devices. Point-to-point is sometimes abbreviated as P2P, Pt2Pt.

	A duplex communication system is a point-to-point system composed of two connected parties or devices that can communicate with one another in both directions. "Duplex" comes from "duo" that means "double", and "plex" that means "structure" or "parts of"; thus, a duplex system has two clearly defined data transmissions, with each path carrying information in only one direction: A to B over one path, and B to A over the other. There are two types of duplex communication systems: full-Duplex and half-Duplex. In a full duplex system, both parties can communicate with each other simultaneously. In a half-duplex system, there are still two clearly defined paths/channels, and each party can communicate with the other but not simultaneously; the communication is one direction at a time. Duplex systems are employed in many communications networks, either to allow for a communication "two-way street" between two connected parties or to provide a "reverse path" for the monitoring and remote adjustment of equipment in the field.
3.2	**TCL SCRIPT:**

```
# To create simulator
1.  set ns [ new Simulator ]
2.  set trf [ open 1.tr w ]
3.  $ns trace-all $trf
4.  set namf [ open 1.nam w ]
5.  $ns namtrace-all $namf

# The below code is used to create the nodes.
6.  set n0 [$ns node]
7.  set n1 [$ns node]
8.  set n2 [$ns node]

#This is used to give color to the flow of packets.
9.      $ns color 1 "red"
10.     $ns color 2 "green"

11.     $n0 label "udp0"
12.     $n1 label "udp1"
13.     $n2 label “destination”

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

```
    # set the queue size b/w the nodes
$ns set queue-limit $n0 $n2 10
set udp0 [new Agent/UDP]
$ns attach-agent $n0 $udp0
set cbr0 [new Application/Traffic/CBR]
$cbr0 attach-agent $udp0

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

set null3 [new Agent/Null]
$ns attach-agent $n2 $null3

    #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 namf trf
        $ns flush-trace
        exec nam 1.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
```

3.3	**Expected Results**



Program 2 : TCP & UDP

1	**Problem Statement**

A network consists of 4 nodes (n0-n3) . Here, n0 and n1 are source nodes, n2 is a router and n3 is the destination node. The duplex links between the nodes is as follows:

- n0 and n2 has 10 Mbps of bandwidth and 10 ms of delay,
- n1 and n2 has 10 Kbps of bandwidth and 100 ms of delay, and
- n2 and n3 has 10 Kbps of bandwidth and 100 ms of delay.

A TCP agent is attached to n0 and connection is established to a TCP sink agent attached to n3. An UDP agent that is attached to n1 is connected to a NULL agent attached to n3. An FTP and a CBR traffic generator are attached to a TCP and UDP agent, respectively. The TCP agent between n0-n3 has a packet size of 200 bytes with a time interval of 0.01 seconds, and the UDP agent between n1-n3 has a packet size of 300 bytes with the time interval of 0.001 seconds.
The CBR is set to start at 0.1 seconds, FTP is set to start at 0.3 seconds and both stop at 5 seconds. Write a Tcl script to observe the packet flow for the given network and observe the output in NAM for this network scenario.

2	**Student Learning Outcomes**

After successful completion of this lab, the student will be able to

- Determine the principle of UDP
- Determine the principle of TCP
- Differentiate the working of TCP and UDP

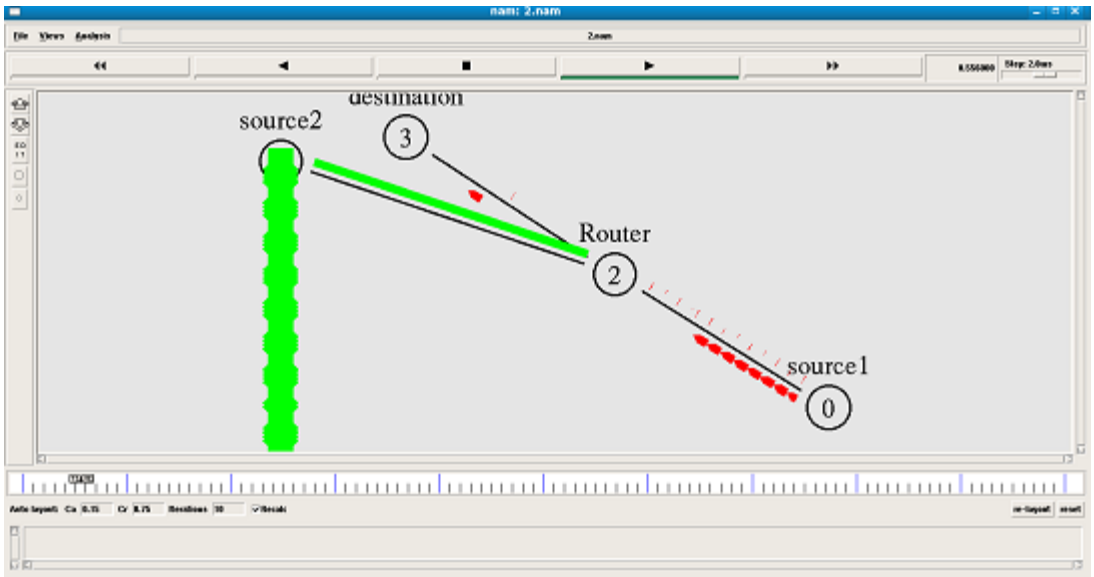
3	**Design of the Program**
3.1	**Theory** The Transmission Control Protocol (TCP) is a core protocol of the Internet protocol suite. It originated in the initial network implementation in which it complemented the Internet Protocol (IP). Therefore, the entire suite is commonly referred to as TCP/IP. TCP provides reliable, ordered, and error-checked delivery of a stream of octets between applications running on hosts communicating over an IP network. Major Internet applications such as the World Wide Web, email, remote administration and file transfer rely on TCP. UDP uses a simple connectionless transmission model with a minimum of protocol mechanism. It has no handshaking dialogues, and thus exposes the user's program to any unreliability of the underlying network protocol. There is no guarantee of delivery, ordering, or duplicate protection. UDP provides checksums for data integrity, and port numbers for addressing different functions at the source and destination of the datagram.

3.2	**TCL SCRIPT:**

```
# to create simulator
set ns [new Simulator]
set trf [open 2.tr w]
$ns trace-all $trf
set namf [open 2.nam w]
$ns namtrace-all $namf

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

# The below code is used to set the color and name's to the #nodes.
$ns color 1 "red"
$ns color 2 "green"
$n0 label "source1"
$n1 label "source2"
$n2 label "Router"
$n3 label "destination"

# To provide link
$ns duplex-link $n0    $n2 10Mb 10ms DropTail
$ns duplex-link $n1    $n2 10Kb 100ms DropTail
$ns duplex-link $n2    $n3 100Mb 1ms DropTail

# The below code is used assign TCP and UDP agents and traffic
set tcp0 [new Agent/TCP]
$ns attach-agent $n0 $tcp0
set ftp0 [new Application/FTP]
$ftp0 attach-agent $tcp0
set sink3 [new Agent/TCPSink]
$ns attach-agent $n3 $sink3

set udp1 [new Agent/UDP]
$ns attach-agent $n1 $udp1
set nul3 [new Agent/Null]
$ns attach-agent $n3 $nul3
set cbr1 [new Application/Traffic/CBR]
$cbr1 attach-agent $udp1

#The below code is used to set the packet size of ftp and udp.
$ftp0 set packetSize_ 200
$ftp0 set interval_ 0.01
```

```
33. $cbr1 set packetSize_ 300
34. $cbr1 set interval_ 0.001

    #This code is used give color for links
35.     $tcp0 set class_ 1
36.     $udp1 set class_ 2

    #connect the agents
37. $ns connect $tcp0       $sink3
38. $ns connect $udp1        $nul3

39. proc finish { }
40.  {
41.       global ns naf trf
42. $ns flush-trace
43. exec nam 2.nam &
44. close $naf
45. close $trf
46. exit 0
47. }

48. $ns at 0.1 "$cbr1 start"
49. $ns at 0.3 "$ftp0 start"
50. $ns at 3.3 "$ftp0 stop"
51. $ns at 10.0 "finish"
52. $ns run
```

3.3	**Expected Results**



Program 3: FTP & TELNET

1	**Problem Statement**

A network consists of 4 nodes (n0-n3) . Here, n0 is the FTP source, and n1 is the TELNET source, n2 is a router and n3 is the common destination node. The duplex links between the nodes is as follows:

- n0 and n2 has 5 Mbps of bandwidth and 10 ms of delay,
- n1 and n2 has 10 Mbps of bandwidth and 10 ms of delay, and
- n2 and n3 has 15 Mbps of bandwidth and 10 ms of delay.

TCP agents are attached to n0 and n1 and connection is established to a sink agent attached to n3. With Telnet connection, packet size is set to 500 Mega bytes with a time interval of 0.001 seconds. The default maximum size of a packet that a TCP agent can generate is 1 kb.

Both FTP and TELNET are set to start at 0.3 seconds, and stop at 5 seconds. Write a Tcl script to observe the packet flow for the given network and observe the output in NAM for this network scenario.

2	**Student Learning Outcomes**

After successful completion of this lab, the student will able to:

- Understand significance of free Email services and FTP.
- Understand importance of data transmission protocol and TELNET.

3	**Design of the Program**
3.1	**Theory**

FTP and Telnet are two very old protocols that are used on networks to add certain functionalities. FTP is a File Transfer Protocol, and its only concern is to facilitate the transfer of files from one point to another, along with a few management capabilities like making and deleting directories. Telnet is a bit more like a 'jack of all trades', as it is simply a connection protocol that allows a user to connect to a remote server that is listening for Telnet commands.

Once the connection is established, the user can then issue commands to the server computer, and examine the responses that are sent back. Although both started out as command line tools, GUIs later appeared that greatly simplified the use of FTP. Instead of knowing all the commands and typing out all the filenames, some dedicated applications let you browse a local drive and a remote drive, as if you are using a file explorer. It keeps all the commands invisible to the user, thereby lessening the learning curve. This is not really possible with Telnet, as there are a wide range of commands and parameters that can be issued to the server.

Due to the age of both software, they do not have any built-in security measures. Even usernames and passwords are sent in plain text, making them vulnerable to sniffing. With later modifications, people can now use secure versions of FTP, called FTPS and SFTP. On the other hand, Telnet has been largely replaced by SSH, due to the addition of security measures. As Telnet has been superseded by SSH, making it secure seems redundant.

Currently, FTP is still in wide use, as it is an easy way to upload files to web servers. There's a wide array of applications that use FTP to achieve their purpose. The use of Telnet has been dwindling since the creation of SSH, but there are still people who use it mainly as a diagnostic tool. Telnet provides a good view of how certain network services work, by sending commands and examining the response to determine if it is proper or not.

3.2 Algorithm/Steps

1. set ns [new Simulator]
2. set trf [open 3.tr w]
3. $ns trace-all $trf
4. set naf [open 3.nam w]
5. $ns namtrace-all $naf

#To create nodes

6. set n0 [$ns node]
7. set n1 [$ns node]
8. set n2 [$ns node]
9. set n3 [$ns node]

Set colors and names to the #nodes

10. $n0 label "Source/FTP"
11. $n1 label "Source/Telnet"
12. $n3 label "Destination/FTP/Telnet"
13. $ns color 1 "red"
14. $ns color 2 "green"

To provide link

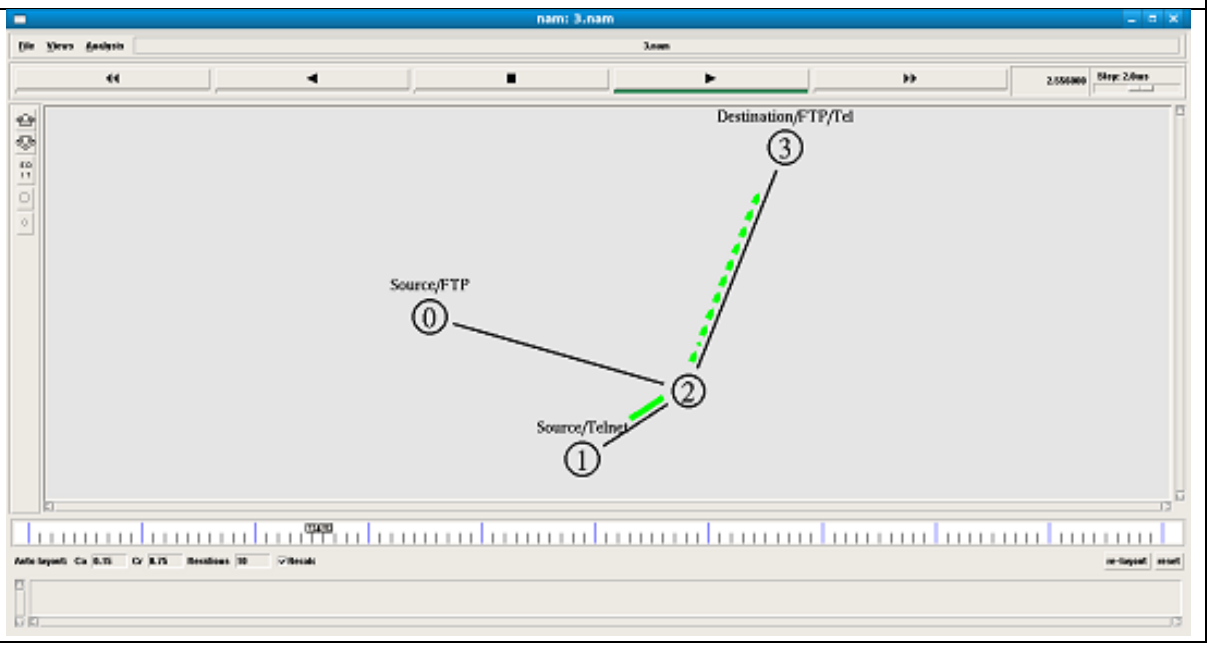
15. $ns duplex-link $n0 $n2 5Mb 10ms DropTail
16. $ns duplex-link $n1 $n2 10Mb 10ms DropTail
17. $ns duplex-link $n2 $n3 15Mb 10ms DropTail

Assign TCP ,FTP and Telnet agents and traffic

18. set tcp0 [new Agent/TCP]
19. $ns attach-agent $n0 $tcp0
20. set ftp0 [new Application/FTP]
21. $ftp0 attach-agent $tcp0
22. set sink3 [new Agent/TCPSink]
23. $ns attach-agent $n3 $sink3
24. set tcp1 [new Agent/TCP]

```
$ns attach-agent $n1 $tcp1
set telnet1 [new Application/Telnet]
$telnet1 attach-agent $tcp1
set sink3 [new Agent/TCPSink]
$ns attach-agent $n3 $sink3

    #Set the packet size
$telnet1 set packetSize_ 500Mb
$telnet1 set interval_ 0.001

    #The below code is used to connect the tcp agents & sink.
$ns connect $tcp0 $sink3
$ns connect $tcp1 $sink3


    #The below code is used to assign color to packets.
$tcp0 set class_ 1
$tcp1 set class_ 2
proc finish { } {
global ns naf trf
exec nam 3.nam &
close $naf
close $trf
exit 0 }

$ns at 0.3 "$ftp0 start"
$ns at 0.3 "$telnet1 start"
$ns at 10 "finish"
$ns run
```

3.3	**Expected Results**



Program 4: PING Agents

1	**Problem Statement**
A network consists of 6 nodes (n0-n5). The duplex links between n0 and n2, n1 and n2,n2 and n3, n3 and n4 & n4 and n5 with 0.1 Mbps of bandwidth and 10 ms of delay. Create two PING agents and attach them to the nodes n0 and n2. Connect the two agents and schedule the transmission of PING messages at an interval of 0.2 seconds and finish at 1.0 second. Write a Tcl script to observe the packet flow for the given network and observe the output in NAM for this network scenario.	
2	**Student Learning Outcomes**
After successful completion of this lab, the student will able to • Learn the software utility used to test the reachability of a host on an Internet Protocol (IP) network • Apply 'ping' concepts to find the number of packets dropped due to congestion.	
3	**Design of the Program**
3.1	**Theory**
	The ping command is a very common method for troubleshooting the accessibility of devices. It uses a series of Internet Control Message Protocol (ICMP) Echo messages to determine: • Whether a remote host is active or inactive. • The round-trip delay in communicating with the host, and • Packet loss. Ping operates by sending Internet Control Message Protocol (ICMP) Echo Requestpackets to the target host and waiting for an ICMP Echo Reply. The program measures the round-trip time from transmission to reception, reporting errors and packet loss. The results of the test usually include a statistical summary of the results, including the minimum, maximum, the mean round-trip times, and usually standard deviation of the mean. The command-line options for the ping utility and its output vary for its many implementations. Options may include the size of the payload, count of tests, limits for the number of network hops (TTL) that probes traverse, and interval between the requests. Many systems provide a companion utility ping6, for testing on Internet Protocol version 6 (IPv6) networks.

3.2	**Algorithm/Steps**

1. set ns [new Simulator]
2. set trf [open 4.tr w]
3. $ns trace-all $trf
4. set naf [open 4.nam w]
5. $ns namtrace-all $naf

To create nodes
6. set n0 [$ns node]
7. set n1 [$ns node]
8. set n2 [$ns node]
9. set n3 [$ns node]
10. set n4 [$ns node]
11. set n5 [$ns node]
12. set n6 [$ns node]

#The below code is used to set the color and name's to the #nodes
13. $n0 label "Ping0"
14. $n3 label "Ping3"
15. $n4 label "Ping4"
16. $n6 label "Ping6"
17. $n2 label "Router"

18. $ns color 1 "red"
19. $ns color 2 "blue"

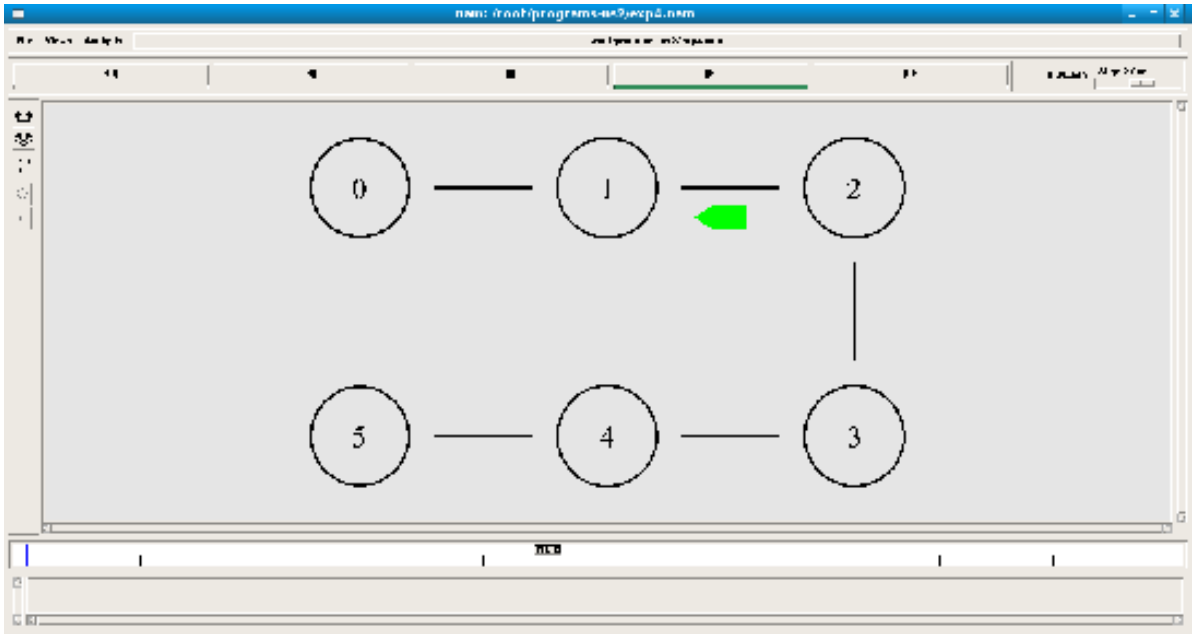
#To provide link
20. $ns duplex-link $n0 $n2 10Mb 30ms DoopTal
21. $ns duplex-link $n5 $n2 10Mb 30ms DoopTal
22. $ns duplex-link $n2 $n6 5Mb 50ms DropTail
23. $ns duplex-link $n3 $n2 10Kb 40ms DropTail
24. $ns duplex-link $n1 $n4 100Mb 100ms DropTail

Connect the ping agents
25. set ping0 [new Agent/Ping]
26. $ns attach-agent $n0 $ping0
27. set ping3 [new Agent/Ping]
28. $ns attach-agent $n3 $ping3
29. set ping4 [new Agent/Ping]
30. $ns attach-agent $n4 $ping4
31. set ping5 [new Agent/Ping]
32. $ns attach-agent $n6 $ping6

To set packet size

```
$ping0 set packetSize_ 500
$ping0 set interval_ 0.001
$ping4 set packetSize_ 500
$ping4 set interval_ 0.001
$ping0 set class_ 1
$ping4 set class_ 2
$ns connect $ping0 $ping3
$ns connect $ping4 $ping5


  #The below function is executed when the ping agent receives #a reply from the destination
Agent/Ping instproc recv {from rtt} { $self instvar node_
puts " The node [$node_ id] received an reply from $from
with round trip time of $rtt"
}
proc finish {} {
global ns naf trf
exec nam 4.nam &
$ns flush-trace
close $trf
close $naf
exit 0
}

  # to start sending the ping packets
$ns at 0.1 "$ping0 send"
$ns at 0.2 "$ping0 send"
$ns at 0.3 "$ping0 send"
$ns at 0.4 "$ping0 send"
$ns at 0.5 "$ping0 send"
$ns at 0.6 "$ping0 send"
$ns at 0.7 "$ping0 send"
$ns at 0.8 "$ping0 send"
$ns at 0.9 "$ping0 send"
$ns at 1.0 "$ping0 send"
$ns at 0.1 "$ping4 send"
$ns at 0.2 "$ping4 send"
$ns at 0.3 "$ping4 send"
$ns at 0.4 "$ping4 send"
$ns at 0.5 "$ping4 send"
$ns at 0.6 "$ping4 send"
$ns at 0.7 "$ping4 send"
$ns at 0.8 "$ping4 send"
$ns at 0.9 "$ping4 send"
$ns at 1.0 "$ping4 send"
$ns at 6.0 "finish"
$ns run
```

3.3	**Expected Results**



PROGRAM 5: Local Area Network

1	**Problem Statement**

A network consists of n nodes (say n =6). The duplex links between the nodes is as follows:

- n0 and n2 has 2Mbps of bandwidth and 10 ms of delay,
- n1 and n2 has 2Mbps of bandwidth and 10 ms of delay,

The Simplex links between the nodes is as follows:

- n2 and n3 has 0.3Mbps of bandwidth and 100 ms of delay.
- n3 and n2 has 0.3Mbps of bandwidth and 100 ms of delay.

The LAN is established between the nodes n3, n4 and n5 with 0.5 Mbps of band width and 40ms delay. Each node uses DropTail queue of which the maximum size is 10 . Write a Tcl script to observe the packet flow for the given network and observe the output in NAM for this network scenario.

2	**Student Learning Outcomes**

After successful completion of this lab, the students will be able to

- Evaluate the throughput of TCP and UDP.
- Insert error model in a LAN.
- Interpret the effect of error model in network traffic.

3	**Design of the Program**
3.1	**Theory** A local area network (LAN) is a group of computers that are connected together in a localized area to communicate with one another and share resources such as printers. Data is sent in the form of packets and to regulate the transmission of the packets, different technologies can be used. The most widely used LAN technology is the Ethernet and it is specified in a standard called IEEE 802.3. Other types of LAN networking technologies include token ring and FDDI. Ethernet uses a star topology in which the individual nodes (devices) are networked with one another via active networking equipment such as switches. The number of networked devices in a LAN can range from two to several thousand. Depending on the type of twisted pair or fiber optic cables used, data rates today can range from 100 Mbit/s to 10,000 Mbit/s.
3.2	**TCL SCRIPT:**

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

#Use colors to differentiate the traffic
$ns color 1 Blue
$ns color 2 Red

#Open trace and NAM trace file
set ntrace [open prog5.tr w]
$ns trace-all $ntrace
set namfile [open prog5.nam w]
$ns namtrace-all $namfile

#Finish Procedure
proc Finish {} {
global ns ntrace namfile

#Dump all trace data and close the files
$ns flush-trace
close $ntrace
close $namfile

#Execute the nam animation file
exec nam prog5.nam &

#Calculate the throughput = (number of packets received/time taken for
simulation)
set TcpSize [exec grep "^r" prog5.tr | grep "tcp" | tail -n 1 | cut -d " " -f 6]
set numTcp [exec grep "^r" prog5.tr | grep -c "tcp"]
set tcpTime 123.0
set UdpSize [exec grep "^r" prog5.tr | grep "cbr" | tail -n 1 | cut -d " " -f 6]
set numUdp [exec grep "^r" prog5.tr | grep -c "cbr"]
set udpTime 124.4
puts "The throughput of FTP is"
puts "[expr ($numTcp*$TcpSize)/$tcpTime] bytes per second"
puts "The throughput of CBR is"
puts "[expr ($numUdp*$UdpSize)/$udpTime] bytes per second"
exit 0
}

#Create 6 nodes
for {set i 0} {$i < 6} {incr i} {
set n($i) [$ns node]
}

#Create duplex links between the nodes
$ns duplex-link $n(0) $n(2) 2Mb 10ms DropTail
$ns duplex-link $n(1) $n(2) 2Mb 10ms DropTail
```

```
$ns simplex-link $n(2) $n(3) 0.3Mb 100ms DropTail
$ns simplex-link $n(3) $n(2) 0.3Mb 100ms DropTail

#Node n(3), n(4) and n(5) are considered in a LAN
set lan [$ns newLan "$n(3) $n(4) $n(5)" 0.5Mb 40ms LL Queue/DropTail
MAC/802_3 Channel]

#Orientation to the nodes
$ns duplex-link-op $n(0) $n(2) orient right-down
$ns duplex-link-op $n(1) $n(2) orient right-up
$ns simplex-link-op $n(2) $n(3) orient right

#Setup queue between n(2) and n(3) and monitor the queue
$ns queue-limit $n(2) $n(3) 20
$ns simplex-link-op $n(2) $n(3) queuePos 0.5

#Set error model on link n(2) and n(3) and insert the error model
set loss_module [new ErrorModel]
$loss_module ranvar [new RandomVariable/Uniform]
$loss_module drop-target [new Agent/Null]
$ns lossmodel $loss_module $n(2) $n(3)

#Setup TCP Connection between n(0) and n(4)
set tcp0 [new Agent/TCP/Newreno]
$tcp0 set fid_ 1
$tcp0 set window_ 8000
$tcp0 set packetSize_ 552
$ns attach-agent $n(0) $tcp0
set sink0 [new Agent/TCPSink/DelAck]
$ns attach-agent $n(4) $sink0
$ns connect $tcp0 $sink0

#Apply FTP Application over TCP
set ftp0 [new Application/FTP]
$ftp0 set type_ FTP
$ftp0 attach-agent $tcp0

#Setup UDP Connection between n(1) and n(5)
set udp0 [new Agent/UDP]
$udp0 set fid_ 2
$ns attach-agent $n(1) $udp0
set null0 [new Agent/Null]
$ns attach-agent $n(5) $null0
$ns connect $udp0 $null0

#Apply CBR Traffic over UDP
set cbr0 [new Application/Traffic/CBR]
$cbr0 set type_ CBR
```

```
$cbr0 set packetSize_ 1000
$cbr0 set rate_ 0.1Mb
$cbr0 set random_ false
$cbr0 attach-agent $udp0

#Schedule events
$ns at 0.1 "$cbr0 start"
$ns at 1.0 "$ftp0 start"
$ns at 124.0 "$ftp0 stop"
$ns at 124.5 "$cbr0 stop"
$ns at 125.0 "Finish"

#Run Simulation
$ns run
```

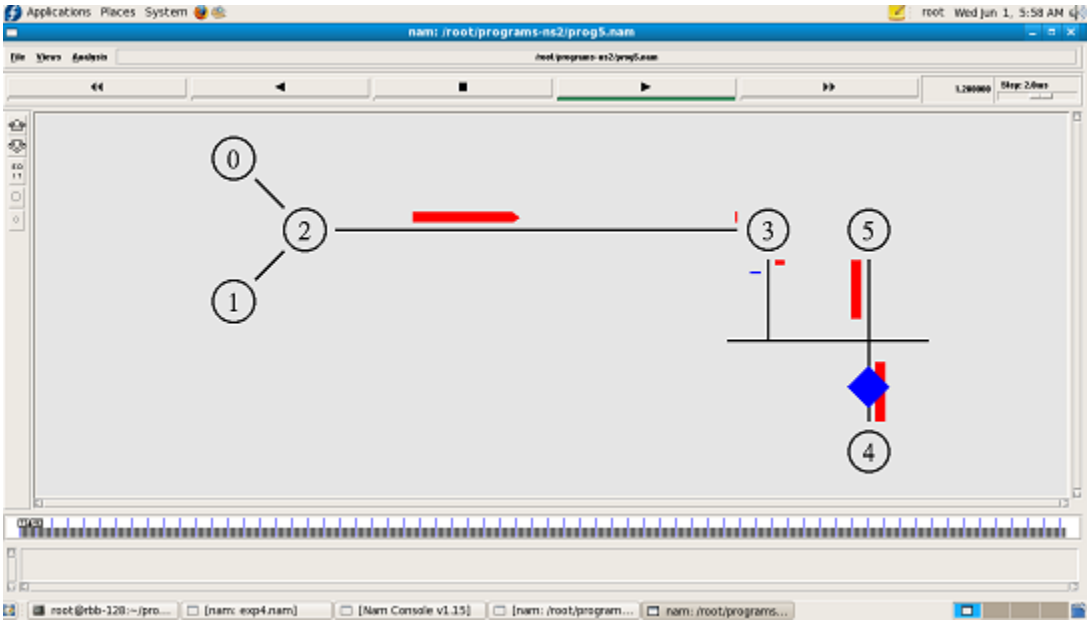
3.3	**Expected Results**

The throughput of FTP is

73196.227642276426 bytes per second

The throughput of CBR is

37347.26688102894 bytes per second



PROGRAM 6: Wireless LAN

1	**Problem Statement**
For a wireless network consisting of three mobile nodes (n0-n2), Nodes are configured with the specific parameters of a wireless node. Initial location of the node is fixed. Nodes are given mobility with fixed speed and fixed destination location. TCP agent is attached to node0 and TCP sink agent is attached to node1. Both the agents are connected and FTP application is attached to TCP agent. Write a Tcl script and make an ad-hoc simulation to analyze the output in the trace file. Use the routing protocol as Adhoc on demand distance vector (AODV).	
2	**Student Learning Outcomes**
After successful completion of this lab, the students will be able to • Deploy nodes in a wireless network. • Implement AODV in a wireless network.	
3	**Design of the Program**
3.1	**Theory** A **wireless local area network** (**WLAN**) is a wireless computer network that links two or more devices using a wireless distribution method (often spread spectrum or OFDM radio) within a limited area such as a home, school, computer laboratory, or office building. This gives users the ability to move around within a local coverage area and still be connected to the network, and can provide a connection to the wider Internet. Most modern WLANs are based on IEEE 802.11 standards, marketed under the Wi-Fi brand name. All components that can connect into a wireless medium in a network are referred to as stations (STA). All stations are equipped with wireless network interface controllers (WNICs). Wireless stations fall into one of two categories: wireless access points, and clients. Access points (APs), normally wireless routers, are base stations for the wireless network. They transmit and receive radio frequencies for wireless enabled devices to communicate with. Wireless clients can be mobile devices such as laptops, personal digital assistants, IP phones and other smart phones, or fixed devices such as desktops and workstations that are equipped with a wireless network interface.

3.2	**TCL SCRIPT:**

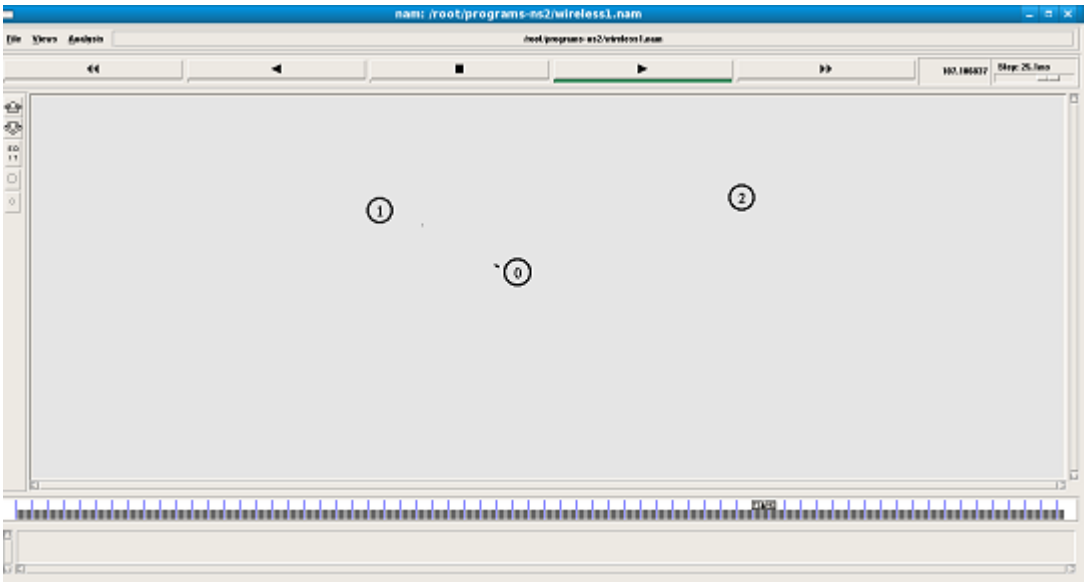
```
# setting different parameters
1.  set val(chan) Channel/WirelessChannel
2.  set val(prop) Propagation/TwoRayGround
3.  set val(netif) Phy/WirelessPhy
4.  set val(mac) Mac/802_11
5.  set val(ifq) Queue/DropTail/PriQueue
6.  set val(ll) LL
7.  set val(ant) Antenna/OmniAntenna
8.  set val(ifqlen) 50
9.  set val(nn) 3
10. set val(rp) AODV
11. set val(x) 500
12. set val(y) 400
13. set val(stop) 150

#scheduler object creation--------#

14. set ns              [new Simulator]

#creating trace file and nam file
15. set tracefd       [open wireless1.tr w]
16. set windowVsTime2 [open win.tr w]
17. set namtrace      [open wireless1.nam w]

18. $ns trace-all $tracefd
19. $ns namtrace-all-wireless $namtrace $val(x) $val(y)

# set up topography object
20. set topo       [new Topography]

21. $topo load_flatgrid $val(x) $val(y)

22. create-god $val(nn)

# configure the nodes
23. $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 OFF \
    -movementTrace ON

for {set i 0} {$i < $val(nn) } { incr i } {
set node_($i) [$ns node]
}

    # Provide initial location of mobile nodes
$node_(0) set X_ 5.0
$node_(0) set Y_ 5.0
$node_(0) set Z_ 0.0

$node_(1) set X_ 490.0
$node_(1) set Y_ 285.0
$node_(1) set Z_ 0.0

$node_(2) set X_ 150.0
$node_(2) set Y_ 240.0
$node_(2) set Z_ 0.0

    # Generation of movements
$ns at 10.0 "$node_(0) setdest 250.0 250.0 3.0"
$ns at 15.0 "$node_(1) setdest 45.0 285.0 5.0"
$ns at 19.0 "$node_(2) setdest 480.0 300.0 5.0"


    # Set a TCP connection between node_(0) and node_(1)
set tcp [new Agent/TCP/Newreno]
$tcp set class_ 2
set sink [new Agent/TCPSink]
$ns attach-agent $node_(0) $tcp
$ns attach-agent $node_(1) $sink
$ns connect $tcp $sink
set ftp [new Application/FTP]
$ftp attach-agent $tcp
$ns at 10.0 "$ftp start"

set tcp [new Agent/TCP/Newreno]
$tcp set class_ 2
set sink [new Agent/TCPSink]
$ns attach-agent $node_(1) $tcp
$ns attach-agent $node_(2) $sink
$ns connect $tcp $sink
set ftp [new Application/FTP]
$ftp attach-agent $tcp
```

```
$ns at 10.0 "$ftp start"


# Printing the window size
proc plotWindow {tcpSource file} {
global ns
set time 0.01
set now [$ns now]
set cwnd [$tcpSource set cwnd_]
puts $file "$now $cwnd"
$ns at [expr $now+$time] "plotWindow $tcpSource $file" }
$ns at 10.0 "plotWindow $tcp $windowVsTime2"


# Define node initial position in nam
for {set i 0} {$i < $val(nn)} { incr i } {
#30 defines the node size for nam
$ns initial_node_pos $node_($i) 30
}


# Telling nodes when the simulation ends
for {set i 0} {$i < $val(nn) } { incr i } {
$ns at $val(stop) "$node_($i) reset";
}


# ending nam and the simulation
$ns at $val(stop) "$ns nam-end-wireless $val(stop)"
$ns at $val(stop) "stop"
$ns at 150.01 "puts \"end simulation\" ; $ns halt"
proc stop {} {
global ns tracefd namtrace
$ns flush-trace
close $tracefd
close $namtrace
exec nam wireless1.nam &
}

$ns run
```

3.3	**Expected Results**

num_nodes is set 3
warning: Please use -channel as shown in tcl/ex/wireless-mitf.tcl
INITIALIZE THE LIST xListHead
channel.cc:sendUp - Calc highestAntennaZ_ and distCST_
highestAntennaZ_ = 1.5, distCST_ = 550.0
SORTING LISTS ...DONE!
end simulation



Simulation Assignment Questions

1. Simulate an Ethernet LAN using N nodes (6-10), change error rate and data rate and compare throughput.

2. Simulate the transmission of ping messages over a network topology consisting of 12 nodes and find the number of packets dropped due to congestion.
3. Simulate the transmission of ping messages over a network topology consisting of 24 nodes and find the number of packets dropped due to congestion.
4. Simulate an Ethernet LAN using n nodes(6<n<10) and set multiple traffic nodes and determine collision across different nodes.
5. Simulate an Ethernet LAN using n nodes(6<n<10) and plot Congestion window for different source and destination
6. Simulate a simple ESS and with transmitting nodes in Wireless LAN determine the performance with respect to transmission of packets.

PART-B

Program 7: CRC

1	Problem Statement

Assume a message represented by the polynomial $M(x) = x^n + ... x2 + x + 1$, choosing 'n' in the range of 4 to 16. Message is sent from the sender with a checksum. When it arrives at the receiver's end, checksum is recalculated for detecting the error. Hence, write a program for error detecting code using CRC-CCITT (16-bits), considering 10001000000100001 as the standard divisor polynomial.

2	Student Learning Outcomes

After successful completion of this lab, the student will able to

- Assess the need for error detection
- Learn the concept of Cyclic Redundancy Check
- Apply CRC method for error detection

3	Design of the Program
3.1	Theory

A **cyclic redundancy check** (CRC) is an error-detecting code commonly used in digital networks and storage devices to detect accidental changes to raw data. Blocks of data entering these systems get a short check value attached, based on the remainder of a polynomial division of their contents; on retrieval the calculation is repeated, and corrective action can be taken against presumed data corruption if the check values do not match

It does error checking via polynomial division. In general, a bit string

bn-1bn-2bn-3…b2b1b0

As

bn-1Xn-1 + bn-2 Xn-2 + bn-3 Xn-3 + …b2 X2 + b1 X1 + b0

Ex: - 10010101110

As

X10 + X7 + X5 + X3 + X2 + X1

All computations are done in modulo 2

To compute an n-bit binary CRC, line the bits representing the input in a row, and position the (n+1)-bit pattern representing the CRC's divisor (called a "polynomial") underneath the left-hand end of the row.

Start with the message to be encoded:

11010011101100

This is first padded with zeroes corresponding to the bit length n of the CRC. Here is the first calculation for computing a 3-bit CRC:

11010011101100 000 <--- input right padded by 3 bits

1011 <--- divisor (4 bits) = x^3+x+1

01100011101100 000 <--- result

If the input bit above the leftmost divisor bit is 0, do nothing. If the input bit above the leftmost divisor bit is 1, the divisor is XORed into the input (in other words, the input bit above each 1-bit in the divisor is toggled). The divisor is then shifted one bit to the right, and the process is repeated until the divisor reaches the right-hand end of the input row. Here is the entire calculation:

11010011101100 000 <--- input right padded by 3 bits

1011 <--- divisor

01100011101100 000 <--- result

1011 <--- divisor ...

00111011101100 000

1011

00010111101100 000

1011

00000001101100 000

1011

00000000110100 000

1011

00000000011000 000

1011

00000000001110 000

1011

00000000000101 000

101 1

00000000000000 100 <---remainder (3 bits)

Since the leftmost divisor bit zeroed every input bit it touched, when this process ends the only bits in the input row that can be nonzero are the n bits at the right-hand end of the row. These n bits are the remainder of the division step, and will also be the value of the CRC function (unless the chosen CRC specification calls for some postprocessing).

The validity of a received message can easily be verified by performing the above calculation again, this time with the check value added instead of zeroes. The remainder should equal zero if there are no detectable errors.

11010011101100 100 <--- input with check value

1011 <--- divisor

01100011101100 100 <--- result

1011 <--- divisor ...

00111011101100 100

.....

00000000001110 100

1011

00000000000101 100

101 1

---

0 <--- remainder

3.2	**Algorithm.**

1. Given a bit string, append 0S to the end of it (the number of 0s 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).
5. 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.

3.3 Coding using C Language

```
#include<stdio.h>
#include<string.h>

char data[100],concatdata[117],src_crc[17],dest_crc[17],frame[120],divident[18];
char divisor[18]="10001000000100001",res[17]="0000000000000000";

void crc_cal(int node)       // function to calculate Cyclic Redundancy Check code
{
        int i,j;
        for(j=17;j<=strlen(concatdata);j++)
{
if(divident[0]=='1')                // performing an ex-or operation
                {
for(i=1;i<=16;i++)
if(divident[i]!=divisor[i])
divident[i-1]='1';
else
divident[i-1]='0';
                 }

              else
                {
                 for(i=1;i<=16;i++)
                 divident[i-1]=divident[i];
                }
if(node==0)
divident[i-1]=concatdata[j];
               else
                divident[i-1]=frame[j];
}
divident[i]='\0';
```

```
29. printf("\ncrc is %s\n",divident);
30. if(node==0)
31. {
32.       strcpy(src_crc,divident);
33. }
34. else
35.          strcpy(dest_crc,divident);
36.       }
37.       int main()
38.       {
39. int i;
40. printf("\n At src node :\n Enter the msg to be sent :");
41. gets(data);
42. strcpy(concatdata,data);
43. strcat(concatdata,"0000000000000000");   //   Message appended with 16  zeros
44. for(i=0;i<=16;i++)
45. divident[i]=concatdata[i];
46. divident[i]='\0';
47. crc_cal(0);
48. printf("\ndata is:\t");
49. puts(data);
50. printf("\n The frame transmitted is :\t");
51. printf("\n%s%s",data,src_crc);
52. printf("\n\t\tSOURCE NODE TRANSMITTED THE FRAME---->");
53. printf("\n\n\n\n\t\t\tAT DESTINATION NODE\nenter the recived frame:\t");
54. gets(frame);
55. for(i=0;i<=16;i++)
56. divident[i]=frame[i];
57. divident[i]='\0';
58. crc_cal(1);
59. if((strcmp(dest_crc,res))==0)
60. printf("\nRecived frame is error free .\n ");
61. else
62. printf("\nRecived frame containes one or more error ");
63. return 1;
64.       }
```

3.4	**Expected Results**

RUN 1:

```
$ cc prog1.c
$ ./a.out

                AT SOURCE NODE
                enter the data to be send :110011


                crc is 0000011000110000
                data is :       110011
                the frame transmitted is :  1100110000011000110000
                SOURCE NODE TRANSMITTED THE FRAME


AT DESTINATION NODE
enter the received frame:  1100110000011000110000
crc is 0000000000000000
received frame is error free
```

RUN 2:

```
AT SOURCE NODE
enter the data to be send :110011
crc is 0000011000110000
data is :       110011


the frame transmitted is : 1100110000011000110000


 SOURCE NODE TRANSMITTED THE FRAME

 AT DESTINATION NODE
enter the received frame:       1100110000011000110001
crc is 0000000000000001

received frame has one or more error
```

3.5	**Implementation Phase:** Compile the Program and note the syntax errors/warning if occurred during each compilation. Remove those syntax errors/warnings for generation of executable file.		
3.6	**Simulation of Errors/Warnings:**		
3.6.1	**Syntax Errors/Warnings:**		
	S. No.	**Errors/Warnings**	**Reflection of the Errors/Warnings in**
3.6.2	**Logical Errors:**		
	S. No.	**Error Simulated**	**Reflection of the error in compilation/output**

Program 8: Distance Vector

1	**Problem Statement**

A network topology consists of n nodes (varying from 4-10). Links connecting the nodes have some costs/weights associated with them. Links for unreachable nodes are indicated by infinity (say, 999).The sample cost adjacency matrix is given below for a network of 4 nodes:

0	2	99	99
2	0	1	99
99	1	0	4
99	99	4	0

Write a program for distance vector algorithm to find suitable path for transmission by using the routing table for each node.

2	**Student Learning Outcomes**

After successful completion of this lab, the student will be able to

- Explore the need for routing algorithm
- Recapitulate the concept of distance vector algorithm
- Implement the distance vector algorithm

3	**Design of the Program**
3.1	**Theory**

Routing algorithm is a part of network layer software which is responsible for deciding which output line an incoming packet should be transmitted on. If the subnet uses datagram internally, this decision must be made anew for every arriving data packet since the best route may have changed since last time. If the subnet uses virtual circuits internally, routing decisions are made only when a new established route is being set up. The latter case is sometimes called session routing, because a rout remains in force for an entire user session (e.g., login session at a terminal or a file).

Routing algorithms can be grouped into two major classes: adaptive and non adaptive. Non adaptive algorithms do not base their routing decisions on measurement or estimates of current traffic and topology. Instead, the choice of route to use to get from I to J (for all I and J) is compute in advance, offline, and downloaded to the routers when the network ids booted. This procedure is sometime called static routing.

Adaptive algorithms, in contrast, change their routing decisions to reflect changes in the topology, and usually the traffic as well. Adaptive algorithms differ in where they get information (e.g., locally, from adjacent routers, or from all routers), when they change the routes (e.g., every ΔT sec, when the load changes, or when the topology changes), and what metric is used for optimization (e.g., distance, number of hops, or estimated transit time).

Two algorithms in particular, distance vector routing and link state routing are the most popular. Distance vector routing algorithms operate by having each router maintain a table (i.e., vector) giving the best known distance to each destination and which line to get there. These tables are updated by exchanging information with the neighbors.

The distance vector routing algorithm is sometimes called by other names, including the distributed Bellman-Ford routing algorithm and the Ford-Fulkerson algorithm, after the researchers who developed it (Bellman, 1957; and Ford and Fulkerson, 1962). It was the original ARPANET routing algorithm and was also used in the Internet under the RIP and in early versions of DECnet and Novell's IPX. AppleTalk and Cisco routers use improved distance vector protocols.

In distance vector routing, each router maintains a routing table indexed by, and containing one entry for, each router in subnet. This entry contains two parts: the preferred out going line to use for that destination, and an estimate of the time or distance to that destination. The metric used might be number of hops, time delay in milliseconds, total number of packets queued along the path, or something similar.

The router is assumed to know the "distance" to each of its neighbor. If the metric is hops, the distance is just one hop. If the metric is queue length, the router simply examines each queue. If the metric is delay, the router can measure it directly with special ECHO packets hat the receiver just time stamps and sends back as fast as possible.

The Count to Infinity Problem.

Distance vector routing algorithm reacts rapidly to good news, but leisurely to bad news. Consider a router whose best route to destination X is large. If on the next exchange neighbor A suddenly reports a short delay to X, the router just switches over to using the line to A to send traffic to X. In one vector exchange, the good news is processed.

To see how fast good news propagates, consider the five node (linear) subnet of following figure, where the delay metric is the number of hops. Suppose A is down initially and all the other routers know this. In other words, they have all recorded the delay to A as infinity.

A	B	C	D	E		A	B	C	D	E	
∞	∞	∞	∞		Initially	1	2	3	4		Initially
1	∞	∞	∞		After 1 exchange	3	2	3	4		After 1 exchange
1	2	∞	∞		After 2 exchange	3	3	3	4		After 2 exchange

	1	2	3	∞	After 3 exchange	5	3	5	4	After 3 exchange
	1	2	3	4	After 4 exchange	5	6	5	6	After 4 exchange
						7	6	7	6	After 5 exchange
						7	8	7	8	After 6 exchange
							:			
						∞	∞	∞	∞	

Many ad hoc solutions to the count to infinity problem have been proposed in the literature, each one more complicated and less useful than the one before it. The split horizon algorithm works the same way as distance vector routing, except that the distance to X is not reported on line that packets for X are sent on (actually, it is reported as infinity). In the initial state of right figure, for example, C tells D the truth about distance to A but C tells B that its distance to A is infinite. Similarly, D tells the truth to E but lies to C.

3.2 **Algorithm**

1. Start
2. By convention, the distance of the node to itself is assigned to zero and when a node is unreachable the distance is accepted as 999.
3. Accept the input distance matrix from the user (dmin]) that represents the distance between each node in the network.
4. Store the distance between nodes in a suitable variable.
5. Calculate the minimum distance between two nodes by iterating.

   If the distance between two nodes is larger than the calculated alternate available path, replace the existing distance with the calculated distance.

6. Print the shortest path calculated.
7. Stop.

3.3	**Coding using C Language**

```
1. #include<stdio.h>
2.  struct rtable                                  // creating a  structure for RoutingTable
3.  {
4.                int dist[20],nextnode[20];
5.   }table[20];
6.  int cost[10][10],n;
7.  void distvector()                    // function for Distance vector calculation
8.  {
9.               int i,j,k,count=0;
10.              for(i=0;i<n;i++)
11.              {
12.                        for(j=0;j<n;j++)
13.                         {
14.                              table[i].dist[j]=cost[i][j];
15.                              table[i].nextnode[j]=j;
16.                         }
17.              }
18.       do
19.          {
20.               count=0;
21.                   for(i=0;i<n;i++)
22.                   {
23.                       for(j=0;j<n;j++)
24.                              {
25.                              for(k=0;k<n;k++)
26.                                        {
27.                                                  if(table[i].dist[j]>cost[i][k]+table[k].dist[j])
28.                                                  {
29.                                                    table[i].dist[j]=table[i].dist[k]+table[k].dist[j];
30.                                                    table[i].nextnode[j]=k;
31.                                                    count++;
32.                                                  }
33.                                        }
34.                               }
35.                   }
36.        } while(count!=0);
37. }
38.      int main()
39.       {
40.             int i,j;
41.               printf("\nenter the no of vertices:\t");
42.                scanf("%d",&n);
```

```
43.             printf("\nenter the cost matrix\n");
44.             for(i=0;i<n;i++)
45.             for(j=0;j<n;j++)
46.             scanf("%d",&cost[i][j]);
47.             distvector();
48.             for(i=0;i<n;i++)
49.             {
50.                 printf("\nstate value for router %c \n",i+65);
51.                 printf("\ndestnode\tnextnode\tdistance\n");
52.                 for(j=0;j<n;j++)
53.                 {
54.                         if(table[i].dist[j]==99)
                        // the path for the node does not exist

55.                 printf("%c\t\t-\t\t infinite\n",j+65);
56.                         else
57.                 printf("%c\t\t%c\t\t%d\n",j+65,table[i].nextnode[j]+65,table[i].dist[j]);
58.                     }

59.     }
60.  return 0;

61.  }
```

3.4 Expected Results

enter the no of vertices: 4

enter the cost matrix
0 2 99 99
2 0 1 99
99 1 0 4
99 99 4 0

state value for router A

destnode	nextnode	distance
A	A	0
B	B	2
C	B	3
D	B	7

state value for router B

destnode	nextnode	distance
A	A	2
B	B	0

C	C	1
D	C	5

state value for router C

destnode	nextnode	distance
A	B	3
B	B	1
C	C	0
D	D	4

state value for router D

destnode	nextnode	distance
A	C	7
B	C	5
C	C	4
D	D	0

3.5	**Implementation Phase:** Compile the Program and note the syntax errors/warning if occurred during each compilation. Remove those syntax errors/warnings for generation of executable file.		
3.6	**Simulation of Errors/Warnings:**		
3.6.1	**Syntax Errors/Warnings:**		
	S. No.	**Errors/Warnings**	**Reflection of the Errors/Warnings in**
3.6.2	**Logical Errors:**		
	S. No.	**Error Simulated**	**Reflection of the error in compilation/output**

Program 9: Client-Server using TCP/IP

1	Problem Statement

Using TCP/IP sockets, write a client-server program to make client send the file name and the server to send back the contents of the requested file file name "sample.txt" with the following contents:"HI I AM AT REVA UNIVERSITY. SEE YOU SOON". Steps for establishing a TCP socket on the client side are the following:

- Create a socket using the socket() function;
- Connect the socket to the address of the server using the connect() function;
- Send and receive data by means of the read() and write() functions.

The steps involved in establishing a TCP socket on the ***server side*** are as follows:

- Create a socket with the socket() function;
- Bind the socket to an address using the bind() function;
- Listen for connections with the listen() function;
- Accept a connection with the accept() function system call.
  This call typically blocks until a client connects with the server.
- Send and receive data by means of send() and receive().

Display suitable error message in case the the file is not present in the server.

2	Student Learning Outcomes

After successful completion of this lab, the student will able to

- Learn TCP/IP network services.
- Apply the concepts learned to write programs to make communication possible between multiple nodes in a network.

3	Design of the Program
3.1	**Theory**

Several protocols for different problems) Protocol Suites or Protocol Families: TCP/IP. TCP/IP provides end-to-end connectivity specifying how data should be

- formatted, addressed, transmitted, routed, and received at the destination
- can be used in the internet and in stand-alone private networks
- it is organized into layers

Internet Protocol (IP)

	• provides a datagram service • packets are handled and delivered independently • best-effort protocol • may loose, reorder or duplicate packets A *socket* is the mechanism that most popular operating systems provide to give programs access to the network. It allows messages to be sent and received between applications (unrelated processes) on different networked machines. The sockets mechanism has been created to be independent of any specific type of network. IP, however, is by far the most dominant network and the most popular use of sockets.
3.2	**Algorithm/Steps**

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.

3.3	**Coding using C Language**

Client Programming

```
1.  #include<stdio.h>
2.  #include<sys/types.h>
3.  #include<sys/socket.h>
```

```
4.          #include<netinet/in.h>
5.          #include<sys/fcntl.h>
6.          #include<stdlib.h>
7.          #include<string.h>
8.          #include<arpa/inet.h>

9.          int main(int argc,char *argv[])
10.         {
11.                 int sockfd,portno,n;
12.                 struct sockaddr_in seradd;
13.                 char buffer[4096],*serip;

14.                     if(argc<4)
15.                       {
16.                         fprintf(stderr,"usage %s serverip filename port",argv[0]);
17.                         exit(0);
18.                       }

19.               serip=argv[1];
20.               portno=atoi(argv[3]);
    /*---- Create the socket. The three arguments are: ----*/
     /* 1) Internet domain 2) Stream socket 3) Default protocol (TCP in this case) */
21.               sockfd=socket(AF_INET,SOCK_STREAM,0);

22.                     if(sockfd<0)
23.                      {
24.                        perror("\n Error in creating socket.\n");
25.                        perror("\n Client on line.");
26.                      }

27.               bzero((char *)&seradd,sizeof(seradd));


       / *---- Configure settings of the server address struct ----*/
                 /* Address family = Internet */
28.               seradd.sin_family=AF_INET;
           /* Set IP address */
29.               seradd.sin_addr.s_addr=inet_addr(serip);
    /* Set port number, using htons function to use proper byte order */
30.               seradd.sin_port=htons(portno);


    /*---- Connect the socket to the server using the address struct ----*/
31.                     if(connect(sockfd,(struct sockaddr *)&seradd,sizeof(seradd))<0)
32.                       {
33.                          perror("\n Error in connection setup \n");
34.                         write(sockfd,argv[2],strlen(argv[2])+1);
35.                         bzero(buffer,4096);
36.                       }

37.                     n=read(sockfd,buffer,4096);
```

```
38.                     if(n<=0)
39.                       {
40.                             perror("\n File not found");
41.                             exit(0);
42.                       }

43.                     write (1,buffer,n);
44. }
```

Server Programming

```
1.          #include<stdio.h>
2.          #include<sys/types.h>
3.          #include<sys/socket.h>
4.          #include<netinet/in.h>
5.          #include<sys/fcntl.h>
6.          #include<stdlib.h>
7.          int main(int argc,char *argv[])
8.           {
9.                      int fd,sockfd,newsockfd,clilen,portno,n;
10.                     struct sockaddr_in seradd,cliadd;
11.                     char buffer[4096];

12.                     if(argc<2)
13.                     {
14.                         fprintf(stderr,"\n\n No port\n");
15.                         exit(1);
16.                     }

17.                    portno=atoi(argv[1]);
    /*---- Create the socket. The three arguments are: ----*/
     /* 1) Internet domain 2) Stream socket 3) Default protocol (TCP in this case) */
18.
19.                     sockfd=socket(AF_INET,SOCK_STREAM,0);

20.                     if(sockfd<0)
21.                      {
22.                         error("\n error opening socket.\n");
23.                         bzero((char *)&seradd,sizeof(seradd));
          /*---- Configure settings of the server address struct ----*/
                /* Address family = Internet */
24.                         seradd.sin_family=AF_INET;
25.                         seradd.sin_addr.s_addr=(htonl)INADDR_ANY;
    /* Set port number, using htons function to use proper byte order */
26.                         seradd.sin_port=htons(portno);
     /*---- Bind the address struct to the socket ----*/
27.                           if(bind(sockfd,(struct sockaddr *)&seradd,sizeof(seradd))<0)
28.                              {
29.                                   perror("\n IP addr cannt bind");
    /*---- Listen on the socket, with 5 max connection requests queued ----*/
30.                                   listen(sockfd,5);
31.                                   clilen=sizeof(cliadd);
32.                                  printf("\n Server waiting for clint request");
```

```
33.                                     while(1)
34.                                     {
    /*---- Accept call creates a new socket for the incoming connection ----*/
35.                                 newsockfd=accept(sockfd,(struct sockaddr *)&cliadd,&clilen);

36.                                      if(newsockfd<0)
37.                                      perror("\n Server cannot accept connection request ");
38.                                      bzero(buffer,4096);
39.                                      read(newsockfd,buffer,4096);
40.                                      fd=open(buffer,O_RDONLY);

41.                                            if(fd<0)
42.                                           {
43.                                            perror("\n File  doesnot exist");
44.                                            while(1)
45.                                             {
46.                                                n=read(fd,buffer,4096);
47.                                                if(n<=0)
48.                                                exit(0);
49.                                                write(newsockfd,buffer,n);
50.                                                printf("\n File transfer completet\n");
51.                                             }
52.                                             close(fd);
53.                                             close(newsockfd);
54.                                           }
55.                                         return 0;
```

3.4	**Expected Results**

AT TERMINAL 1 : (SERVER PROGRAM)

```
$ cc prog4s.c

$ ./a.out 7000



SERVER:Waiting for client....

SERVER:hsr.c

SERVER:hsr.c found!

 Transfering the contents ...

File content #include<stdio.h>

int main()
```

```
{
        printf("\nascii of a %d",'h');
       return 1;
}
```

AT TERMINAL 2 : (CLIENT PROGRAM)

```
$ cc prog4c.c
$ ./a.out 127.0.0.1 7000
client online
 server online
client: enter path with filename:
hsr.c

client: displaying from socket

int main()
{
 printf("\nascii of a %d",'h');
return 1;
}
```

3.5	**Simulation of Errors/Warnings:**		
3.5.1	**Syntax Errors/Warnings:**		
	S. No.	**Errors/Warnings Simulated**	**Reflection of the Errors/Warnings in compilation/output**

3.5.3	**Logical Errors:**		
	S. No.	**Error Simulated**	**Reflection of the error in compilation/output**

Program 10: Message queues or FIFOs

1	**Problem Statement**

There is a single Server process which runs continuously in background, even though if there is no client to interact with it. Client processes runs in foreground and interacts with the server process. Both the client and server processes run on the same machine.

The Client process accepts a command (a shell command) from the user and send's it to the Server via a FIFO which is a public channel between Client and Server for processing. Assume this FIFO name as **PUBLIC fifo** since its existence is known to all clients and the server. Once the command is received, the Server executes it using the **popen–pclose** sequence (which generates an unnamed pipe in the Server process). After execution Server process returns the output of the command executed to the client over a FIFO which is a private channel between the client and server. Let us name this FIFO as PRIVATE fifo. The Client, upon receipt, displays the output on the screen.

Implement this scenario using message queues or FIFOs as interprocess communication channels by adopting the following procedure:

Functionality steps of a Client process:

- Create a unique name for the PRIVATE fifo and invoke it.
- Open the PUBLIC fifo in write mode.
- Prompt for command from user.
- Write command to PUBLIC fifo for Server to process.
- Open PRIVATE fifo in read mode to read the contents from Server.

Steps of a Server process:

- Generate a PUBLIC fifo and open it in both read and write mode. Wait for the message from a Client process.
- Read the message from PUBLIC fifo.
- Open the Client's PRIVATE fifo in write mode.
- Execute the command from Client process using popen.
- Write the output into Client's PRIVATE fifo.

NOTE: Each and every Client process should have its own unique PRIVATE fifo to receive information from Server.

2	**Student Learning Outcomes**

After successful completion of this lab, the student will able to

- Learn message queues or FIFOs as IPC channels.
- Apply the concepts learned to write programs to make communication possible between multiple nodes in a network.

3	**Design of the Program**
3.1	**Theory**

Interprocess Communication plays a fundamental role in the transformation of QNX Neutrino from an embedded real-time kernel into a full-scale POSIX operating system. As various service-providing processes are added to the microkernel, IPC is the “glue” that connects those components into a cohesive whole.

Although message passing is the primary form of IPC in QNX Neutrino, several other forms are available as well. Unless otherwise noted, those other forms of IPC are built over our native message passing. The strategy is to create a simple, robust IPC service that can be tuned for performance through a simplified code path in the microkernel; more “feature cluttered” IPC services can then be implemented from these.

3.2	**Algorithm/Steps**

Algorithm (Client Side)

1. Start.

2. Open well known server FIFO in write mode.
3. Write the pathname of the file in this FIFO and send the request.
4. Open the client specified FIFO in read mode and wait for reply.
5. When the contents of the file are available in FIFO, display it on the terminal
6. Stop.

Algorithm (Server Side)

1. Start.
2. Create a well known FIFO using mkfifo()
3. Open FIFO in read only mode to accept request from clients.
4. When client opens the other end of FIFO in write only mode, then read the contents and store it in buffer.
5. Create another FIFO in write mode to send replies.
6. Open the requested file by the client and write the contents into the client specified FIFO and terminate the connection.
7. Stop.

3.3	**Coding using C Language**

Client Programming

```
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
#include<fcntl.h>
#include<sys/types.h>
#include<sys/stat.h>
#include<unistd.h>
 #define FIFO1 "fifo1"
 #define FIFO2 "fifo2"
 int main()
   {
        char p[100],c[5000];
        int num,fd,fd2,f1;
        mknod(FIFO1,S_IFIFO|0666,0);      // creating a named pipe
        mknod(FIFO2,S_IFIFO|0666,0);
        printf("\n Client online...\n");
/*
 * open them - one for reading and one
 * for writing.
 */

        fd=open(FIFO1,O_WRONLY);
```

```
18.                     fd2=open(FIFO2,O_RDONLY);
19.                     printf("Client : Enter the filename . \n\n");
20.                     scanf("%s",p);
21.                     num=write(fd,p,strlen(p));
22.                         if(num==-1)
23.                         {
24.                             perror("\nWrite Error.\n");
25.                             return 1;
26.                         }
27.                          else
28.                             {
29.                                 printf("\n Waiting for reply\n");
30.                                 if((num=read(fd2,c,5000))==-1)

31.                                perror("\nError while reading from fifo \n");
32.                                    Else
                /*
                 * Read the data from the file and write to
                 * the IPC descriptor.
                 */

33.                                     {
34.                                        c[num]=0;
35.                                        printf("%s",c);
36.                                     }
37.                                 }
38.                     return 1;
39.                 }
```

Server Programming

```
1.                  #include<stdio.h>
2.                  #include<stdlib.h>
3.                  #include<string.h>
4.                  #include<fcntl.h>
5.                 #include<sys/types.h>
6.                 #include<sys/stat.h>
7.                 #include<unistd.h>
8.                 #define FIFO1 "fifo1"
9.                 #define FIFO2 "fifo2"
10.
11.                int main()
12.                {
13.                   char p[100],c[5000];
```

```
14.                     int num,fd,fd2,f1;
15.                     mknod(FIFO1,S_IFIFO|0666,0);   // creating two  pipes
16.                     mknod(FIFO2,S_IFIFO|0666,0);
17.                     printf("\n Server online...\n");
        /*
        * open them - one for reading and one
        * for writing.
        */

18.                     fd=open(FIFO1,O_RDONLY);
19.                     fd2=open(FIFO2,O_WRONLY);
20.                     printf("Server online\n waiting for client \n\n");
21.                     if((num=read(fd,p,100))==-1)
22.                     perror("\n Read Error ");
23.                     else
24.                       {
25.                         p[num]='\0';
26.                         printf("\n File is %s .\n",p);

                    /* Error.  Format an error message and send it back
                    * to the client.
                    */

27.                         if((f1=open(p,O_RDONLY))<0)
28.                         {
29.                           write(fd2,"File not found",15);
30.                           return 1;
31.                         }
32.                     else
33.                         {
34.                           stdin=fdopen(f1,"r");
35.                           num=0;
36.                           while((ch=fgetc(stdin))!=EOF)
37.                           c[num++]=ch;
38.                           c[num]=0;
39.                           printf(" Server: Transfering the contents of :%s ",p);
40.                           if(num=write(fd2,c,strlen(c))==-1)
41.                           printf("\n Error in writting to FIFO\n");
42.                           Else
43.                           printf("\n File transfer completed \n");
44.                     }
45.                     }
46.                     }
47.                     }
```

3.4	**Expected Results**

OUTPUT: **AT TERMINAL 1:**

```
$ cc prog5s.c
$ ./a.out


SERVER ONLINE client online
waiting for request


server:hsr.c found
tranfering the contents
file contents #include<stdio.h>
int main()
{
printf("\nascii of a %d",'h');
 return 1;
}


server :tranfer completed
```

AT TERMINAL 2:

```
$ cc prog5c.c
$ ./a.out
waiting for server...
SERVER ONLINE !
CLIENT:Enter the path
hsr.c
Waiting for reply....
```

File recieved! displaying the contents:

```
#include<stdio.h>
int main()
{
printf("\nascii of a %d",'h');
 return 1;
}
```

3.5	**Simulation of Errors/Warnings:**		
3.5.1	**Syntax Errors/Warnings:**		
	S. No.	**Errors/Warnings Simulated**	**Reflection of the Errors/Warnings in compilation/output**
3.5.2	**Logical Errors:**		
	S. No.	**Error Simulated**	**Reflection of the error in compilation/output**

Program 11: RSA

1	**Problem Statement**
Choose the two prime numbers, p=17 and q=11. Write a program for public key encryption system using RSA algorithm to encrypt and decrypt the message. For a message M=”he12ll34o“, show the encryption and decryption.	
2	**Student Learning Outcomes**
After successful completion of this lab, the student will be able to • Explore the importance of data security. • Perform RSA encryption/decryption on given data.	
3	**Design of the Program**
3.1	**Theory**

Cryptography has a long and colorful history. The message to be encrypted, known as the plaintext, are transformed by a function that is parameterized by a key. The output of the encryption process, known as the cipher text, is then transmitted, often by messenger or radio. The enemy, or intruder, hears and accurately copies down the complete cipher text. However, unlike the intended recipient, he does not know the decryption key and so cannot decrypt the cipher text easily. The art of breaking ciphers is called cryptanalysis the art of devising ciphers (cryptography) and breaking them (cryptanalysis) is collectively known as cryptology.

There are several ways of classifying cryptographic algorithms. They are generally categorized based on the number of keys that are employed for encryption and decryption, and further defined by their application and use. The three types of algorithms are as follows:

1. Secret Key Cryptography (SKC): Uses a single key for both encryption and decryption. It is also known as symmetric cryptography.
2. Public Key Cryptography (PKC): Uses one key for encryption and another for decryption. It is also known as asymmetric cryptography.
3. Hash Functions: Uses a mathematical transformation to irreversibly "encrypt" information

Public-key cryptography has been said to be the most significant new development in cryptography. Generic PKC employs two keys that are mathematically related although knowledge of one key does not allow someone to easily determine the other key. One key is used to encrypt the plaintext and the other key is used to decrypt the cipher text. The important point here is that it does not matter which key is applied first, but that both keys are required for the process to work. Because pair of keys is required, this approach is also called asymmetric cryptography.

In PKC, one of the keys is designated the public key and may be advertised as widely as the owner wants. The other key is designated the private key and is never revealed to another party. It is straight forward to send messages under this scheme.

The RSA algorithm is named after Ron Rivest, Adi Shamir and Len Adleman, who invented it in 1977. 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.

3.2	**Algorithm** 1. Generate two large random primes, P and Q, of approximately equal size. 2. Compute $N = P \times Q$ 3. Compute $Z = (P-1) \times (Q-1)$. 4. Choose an integer E, $1 < E < Z$, such that GCD $(E, Z) = 1$ 5. Compute the secret exponent D, $1 < D < Z$, such that $E \times D \equiv 1 \pmod{Z}$ 6. The public key is (N, E) and the private key is (N, D).

Note: The values of *P, Q,* and *Z* should also be kept secret.

The message is encrypted using public key and decrypted using private key.

Examples of RSA encryption

Example:1

1. Select primes *P*=11, *Q*=3.

2. $N = P \times Q = 11 \times 3 = 33$
   $Z = (P-1) \times (Q-1) = 10 \times 2 = 20$

3. Lets 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*, *Z*) = GCD(3, 20) = 1

4. Compute *D* such that $(E \times D)-1 \pmod{Z}$
   compute $D = E^{n-1} \bmod Z = 3^{n-1} \bmod 20$
   find a value for *D* such that *Z* divides ((*E* x *D*)-1)
   find *D* such that 20 divides 3D-1.
   Simple testing (D = 1, 2, ...) gives D = 7
   Check: (*E* x *D*)-1 = 3.7 - 1 = 20, which is divisible by *Z*.

5. Public key = (*N*, *E*) = (33, 3)
   Private key = (*N*, *D*) = (33, 7).

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

Cipher code = $M^E \bmod N$

$= 7^3 \bmod 33$

$= 343 \bmod 33$

$= 13.$

Hence the cipher text c = 13.

To check decryption we compute Message' = $C^D \bmod N$

$= 13^7 \bmod 33$

$= 7.$

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

Example:2

1. Choose p = 3 and q = 11
2. Compute n = p * q = 3 * 11 = 33
3. Compute φ(n) = (p – 1) * (q – 1) = 2 * 10 = 20
4. Choose e such that 1 < e < φ(n) and e and n are coprime. Let e = 7
5. Compute a value for d such that (d * e) % φ(n) = 1. One solution is d = 3 [(3 * 7) % 20 = 1]
6. Public key is (e, n) => (7, 33)
7. Private key is (d, n) => (3, 33)
8. The encryption of m = 2 is c = 27 % 33 = 29
9. The decryption of c = 29 is m = 293 % 33 = 2

3.3 Coding using C Language

```
#include<stdio.h>
#include<stdlib.h>
#include<math.h>

int gcd(long m,long n)        // computing Greatest Common Divisor
  {
      if(n==0)return m;
      if(m<n)return gcd(n,m);
      return gcd(n,m%n);
  }
 int rsa(char *msg)
 {
       long p=0,q=0,n=0,e=0,d=0,phi=0;
       long nummes[100]={0};
       long encrypt[100]={0};
       long decrypt[100]={0};
      long i=0,j=0,nofelem=0;
       int prime[1000]={0};
       while(p==q)
       {
            p=11;
            q=17;
        }
      printf("\n Values of p and q are %ld and %ld",p,q);
      n=p*q;
```

```
25.         phi=(p-1)*(q-1);
26.         for(i=2;i<phi;i++)
27.         if(gcd(i,phi)==1)
28.         break;
29.         e=i;
30.         if((e*i-1)%phi==0)
31.         break;
32.        d=i;
33.        for(i=0;i<strlen(msg);i++)
34.        nummes[i]=msg[i]-96;
35.        nofelem=strlen(msg);
36.        for(i=0;i<nofelem;i++)
37.        {
38.                encrypt[i]=1;
39.                for(j=0;j<e;j++)
40.                encrypt[i]=(encrypt[i]*nummes[i])%n;
41.        }
42.        printf("\n Encrypted Message: \n");
43.        for(i=0;i<nofelem;i++)
44.        printf("%ld",encrypt[i]);
45.        for(i=0;i<nofelem;i++)
46.       {
47.               decrypt[i]=1;
48.               for(j=0;j<d;j++)
49.               decrypt[i]=(decrypt[i]*encrypt[i])%n;
50.       }
51.        printf("\n  Decrypted msg:\n");
52.        for(i=0;i<nofelem;i++)
53.        printf("%c",(char)(decrypt[i]+96));
54.        return 0;
55.     }

56.    int main()
57.   {
58.          char msg[100]={0};
59.          printf("\n This algorithm has been implemented for lower case letters only\n");
60.          printf("\n Enter msg :");
61.          scanf("%s",msg);
62.          rsa(msg);
63.          return 0;
64.    }
```

3.4	**Expected Results**

This algorithm has been implemented for lower case letters only

Enter msg :he12ll34o

Values of p and q are 11 and 17

Encrypted Message:

138125-38-964545-56-999

Decrypted msg:

he12ll34o

3.5	**Implementation Phase:** Compile the Program and note the syntax errors/warning if occurred during each compilation. Remove those syntax errors/warnings for generation of executable file.		
3.5	**Simulation of Errors/Warnings:**		
3.5.1	**Syntax Errors/Warnings:**		
	S. No.	**Errors/Warnings**	**Reflection of the Errors/Warnings in**
3.5.2	**Logical Errors:**		
	S. No.	**Error Simulated**	**Reflection of the error in compilation/output**

Program 12: Leaky Bucket Algorithm

1	Problem Statement

Examine node transmitting/receiving packets to/from other nodes. Using a random function; vary the packet size, considering the following cases.

Case i: Output rate of 5 Bytes and Bucket size of 10 Bytes.
Case ii: Output rate of 10 Bytes and Bucket size of 5 Bytes.

Write a program for congestion control implementing leaky bucket algorithm.

2	Student Learning Outcomes

After successful completion of this lab, the student will be able to

- Determine whether some sequence of discrete events conforms to defined limits on their average and peak rate or frequency.
- Monitor network traffic by varying the Output rate and Bucket size.

3	Design of the Program

3.1	Theory

The congesting control algorithms are basically divided into two groups: open loop and closed loop. Open loop solutions attempt to solve the problem by good design, in essence, to make sure it does not occur in the first place. Once the system is up and running, midcourse corrections are not made. Open loop algorithms are further divided into ones that act at source versus ones that act at the destination.

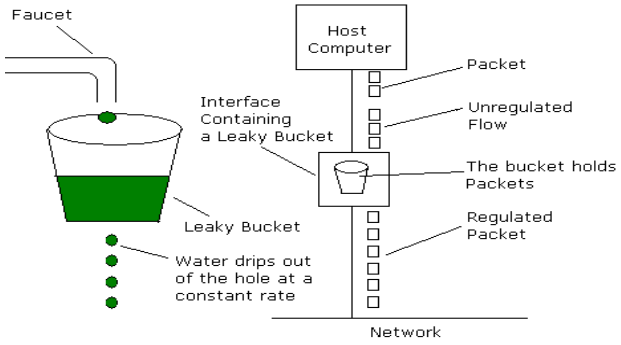
In contrast, closed loop solutions are based on the concept of a feedback loop if there is any congestion. Closed loop algorithms are also divided into two sub categories: explicit feedback and implicit feedback. In explicit feedback algorithms, packets are sent back from the point of congestion to warn the source. In implicit algorithm, the source deduces the existence of congestion by making local observation, such as the time needed for acknowledgment to come back.

The presence of congestion means that the load is (temporarily) greater than the resources (in part of the system) can handle. For subnets that use virtual circuits internally, these methods can be used at the network layer.

Another open loop method to help manage congestion is forcing the packet to be transmitted at a more predictable rate. This approach to congestion management is widely used in ATM networks and is called **traffic shaping**.

The other method is the leaky bucket algorithm. Each host is connected to the network by an interface containing a leaky bucket, that is, a finite internal queue. If a packet arrives at the queue when it is full, the packet is discarded. In other words, if one or more process are already queued, the new packet is unceremoniously discarded. This arrangement can be built into the hardware interface or simulate d by the host operating system. In fact it is nothing other than a single server queuing system with constant service time.

The host is allowed to put one packet per clock tick onto the network. This mechanism turns an uneven flow of packet from the user process inside the host into an even flow of packet onto the network, smoothing out bursts and greatly reducing the chances of congestion.



3.2	**Algorithm**

1. Start
2. Set the bucket size or the buffer size.
3. Set the output rate.
4. Transmit the packets such that there is no overflow.
5. Repeat the process of transmission until all packets are transmitted. (Reject packets where its size is greater than the bucket size)
6. Stop

3.3	**Coding using C Language**

```
1. #include<stdio.h>
2. #include<stdlib.h>

3. int trand(int a)
4. {
5.       int rn;
6.       rn=random()%10;
7.       rn=rn%a;
8.       if(rn==0)
9.       rn=1;
```

```
10.       return(rn);
11.   }
12. int main()
13.  {
14.          int i,packet[5],psz,bsz,pszrn=0,clk,ptime,premain,orate,flag=0;
15.          for(i=0;i<5;i++)
16.          packet[i]=trand(6)*10;
17.         printf("\n enter the o/p rate");
18.         scanf("%d",&orate);
19.         printf("\n enter the bucket size");
20.         scanf("%d",&bsz);
21.         for(i=0;i<5;i++)
22.          {
23.           if((packet[i]+pszrn)>bsz)
24.           printf("\n incoming packet size %d is greater than bucket
    size_reject",packet[i]);
25.           else
26.            {
27.              for(;;)
28.                {
29.                    premain=4-i;
30.                    psz=packet[i];
31.                    pszrn+=psz;
32.                    printf("\nincoming packet size is %d",psz);
33.                    printf("\n no of packets waiting for transmission=%d",pszrn);
34.                    ptime=trand(4)*10;
35.                    printf("\n next packet will come at %d",ptime);
36.                    for(clk=0;clk<=ptime;clk++)
37.                    {
38.                        printf("\ntime left=%d",ptime-clk);
39.                        sleep(1);
40.                        if(pszrn)
41.                        {
42.                              if(pszrn==0)
43.                              printf("\n bucket is empty");
44.                              else
45.                               {
46.                                    if(pszrn>=orate)
47.                                     printf("%d bytes are transmitted",orate);
48.                                     else
49.                                     printf("\n %d bytes transmitted ",pszrn);
50.                                 }
51.                              if(pszrn<=orate)
52.                              pszrn=0;
53.                              else
54.                              pszrn-=orate;
55.                              printf("\n bytes remaining %d",pszrn);
56.                          }
```

```
                    else
                    printf("\n bytes remaining %d",pszrn);
                  }
              if(pszrn!=0)
              flag=1;
             break;
          }
        }
    }
    printf("\n\n");
    return 0;
}
```

3.4	**Expected Results**

RUN 1:

```
$ cc prog8.c
$ ./a.out
enter the o/p rate10
enter the bucket size5


incoming packet size 30 is greater than bucket size_reject
incoming packet size 10 is greater than bucket size_reject
incoming packet size 10 is greater than bucket size_reject
incoming packet size 50 is greater than bucket size_reject
incoming packet size 30 is greater than bucket size_reject
```

RUN 2:

```
enter the o/p rate5
enter the bucket size10


 incoming packet size 30 is greater than bucket size_reject
incoming packet size is 10
 no.of packets waiting for transmission= 10
next packet will come at 10
time left =105 bytes are transmitted
bytes remaining 5
time left =95 bytes are transmitted
bytes remaining 0
time left =8
bytes remaining 0
time left =7
```

```
 bytes remaining 0
time left =6
bytes remaining 0
time left =5
bytes remaining 0
time left =4
bytes remaining 0
time left =3
bytes remaining 0
time left =2
bytes remaining 0
time left =1
bytes remaining 0
time left =0
bytes remaining 0
incoming packet size is 10
 no.of packets waiting for transmission= 10
next packet will come at 20
time left =205 bytes are transmitted
bytes remaining 5
time left =195 bytes are transmitted
bytes remaining 0
time left =18
bytes remaining 0
time left =17
bytes remaining 0
time left =16
bytes remaining 0
time left =15
bytes remaining 0
time left =14
bytes remaining 0
time left =13
bytes remaining 0
time left =12
bytes remaining 0
time left =11
bytes remaining 0
time left =10
bytes remaining 0
time left =9
bytes remaining 0
time left =8
bytes remaining 0
time left =7
```

bytes remaining 0
time left =6
bytes remaining 0
time left =5
bytes remaining 0
time left =4
bytes remaining 0
time left =3
bytes remaining 0
time left =2
bytes remaining 0
time left =1
bytes remaining 0
time left =0
bytes remaining 0
incoming packet size 50 is greater than bucket size_reject
incoming packet size 30 is greater than bucket size_reject

3.5	**Implementation Phase:** Compile the Program and note the syntax errors/warning if occurred during each compilation. Remove those syntax errors/warnings for generation of executable file.		
3.5	**Simulation of Errors/Warnings:**		
3.5.1	**Syntax Errors/Warnings:**		
	S. No.	**Errors/Warnings**	**Reflection of the Errors/Warnings in**
3.5.2	**Logical Errors:**		
	S. No.	**Error Simulated**	**Reflection of the error in compilation/output**

PART-B Assignment Questions

1. Generate a 1-bit self correcting Hamming code for a given input data
2. Implement DIJKSTRAs algorithm for a maximum of ten nodes in a network topology.
3. Design TCP iterative Client and Server application to reverse the given input sentence
4. Design a TCP concurrent Server to convert a given text into upper case using multiplexing system call "select".
5. Design a TCP concurrent Server to echo given set of sentences using Poll functions.
6. Implement the following forms of IPC. a) Pipes b) FIFO

7. Implement file transfer using Message Queue form of IPC.
8. Write a Program to create an integer variable using Shared Memory concept and increment the variable simultaneously by two processes. Use Semaphores to avoid Race conditions
9. Implement the Diffie Hellman Key exchange algorithm.
10. Analyze the Leaky-Bucket program for different bucket size and packet length.
11. Implement TOKEN Bucket algorithm to check that data transmissions conform to defined limits on bandwidth and burstiness

Viva Voce Questions

12. What is the degree of polynomial used in CRC-CCITT?
13. List out the different error detection algorithms?
14. Expand CCITT?
15. Is CRC-CCITT is error detection and/or correction algorithm?
16. In which layers of OSI reference model CRC-CCITT is used?
17. What is the significance of redundancy term in CRC?
18. What is meant by count-to-infinity problem?
19. What is the difference between routing and forwarding?
20. Difference between adaptive and non-adaptive routing protocols?
21. How do you classify routing algorithms? Give examples for each.
22. State the drawbacks in distance vector algorithm?
23. How routers update distances to each of its neighbor?
24. How do you overcome count to infinity problem?

25. Differentiate between route, bridge, switch and hub.
26. Compare ping and telnet?
27. What is FTP?
28. What is meant by congestion window?
29. What is incoming throughput and outgoing throughput?
30. How do you generate multiple traffics across different sender-receiver pairs
31. Differentiate between logical and physical address.
32. Which address gets affected if a system moves from one place to another place?
33. Define Gateway?
34. What is cryptography?
35. How do you classify cryptographic algorithms?
36. What is public key?
37. What is private key?
38. What is encryption and decryption?
39. Define transposition.
40. What is secret key cryptography?
41. What do the terms key, cipher text and plaintext mean?
42. What is crypt analysis?
43. Define traffic shaping?
44. How do you classify congestion control algorithms?
45. Differentiate between Leaky bucket and Token bucket.
46. How do you implement Leaky bucket?
47. How do you generate busty traffic?
48. Differentiate between TCP and UDP.
49. Differentiate between Point-to-Point Connection and End-to-End connections.
50. What are protocols running in different layers?
51. What is Protocol Stack?
52. What is an IP address?
53. What is MAC address?
54. Why IP address is required when we have MAC address?
55. What is meant by port?
56. What are ephemerical port number and well known port numbers?
57. What is a socket?
58. What are the parameters of socket ()?
59. Describe bind (), listen (), accept (), connect (), send () and recv ().
60. What are system calls? Mention few of them.
61. What is IPC? Name three techniques.
62. Differentiate between socket and pipes? What is the advantage of FIFO over Pipe?
63. What is created by mkfifo?
64. Which system call is used to create Sys V message Queue?
65. Which one of the following is not system V IPC ?
66. What are System V IPC common attributes?