VISVESVARAYA TECHNOLOGICAL UNIVERSITY

"Jnana Sangama", Belgaum – 590 018

COMPUTER NETWORK LABORATORY
16MCA36
LAB MANUAL

Submitted by

Name:
USN:

Department of Master of Computer Applications

Maharaja Institute of Technology, Mysore

Belawadi, Srirangapatna Taluk,Mandya- 571438

August- November 2017

Department of Master of Computer Applications

LAB MANUAL

COMPUTER NETWORK LABORATORY

16MCA36

NAME: .....

USN: .....

Prepared by
Mahalakshmi M
Department of MCA

Maharaja Institute of Technology, Mysore

Belawadi, Srirangapatna Taluk,Mandya- 571438

August- November 2017

Maharaja Institute of Technology Mysore

Belawadi, SrirangapatnaTaluk,Mandya- 571438

Department of Master of Computer Applications

CERTIFICATE

This is to certify that the Computer Networks Laboratory (16MCA36)has been satisfactorily completed by ……bearing the USN ……..as prescribed by the ***Visvesvaraya Technological University,*** *during the III semester of the academic year* ***2017-18***

Prof. Manjunath B
Head of the Department

Faculty In-charge

Vision & Mission of the College
Vision To gift the nation with highly disciplined, responsible, and skilled engineering graduates.
Mission ♦ To advance the well-being of society through excellence in teaching, research and service that exploits the rapidly changing technical diversity via a collaborative environment that stimulates faculty, staff and students to reach their highest potential through continuous learning. ♦Instill the foundation of professionalism and provide the tools required to advance the technological world. ♦Inculcate knowledge and ethics, and nurture/foster innovation and team man ship among the graduates/alumnae. ♦Endow eminent education to serve the society with graduates/alumnae possessing ace skills and potential. ♦Sustain highest reception of the institute's alumnae among the stakeholders.

MAHARAJA INSTITUTE OF TECHNOLOGY MYSORE

DEPARTMENT OF MCA

Vision & Mission of the Department
Vision To become a center par-excellence in order to create hard-core professionals enriched with deep knowledge and high-level expertise in modern technology, out of young talents.
Mission Our Mission is to Facilitate Quality Engineering Education to Equip and Enrich Young Men and Women to Meet Global challenges in Development, Innovation and Application of Technology in the service of Humanity.

MAHARAJA INSTITUTE OF TECHNOLOGY MYSORE

DEPARTMENT OF MCA

PROGRAMME EDUCATIONAL OBJECTIVES (PEOS):
1. To gain knowledge and proficiency for analysis, design and problem solving, to have a successful career in industry and for higher studies. 2. To promote application of technical knowledge coupled with project management abilities. 3. To imbibe leadership qualities with professional ethics and communication skills. 4. To provide positive attitude for lifelong learning.

MAHARAJA INSTITUTE OF TECHNOLOGY MYSORE

DEPARTMENT OF MCA

Program Outcomes (POs):

1. An ability to apply knowledge in computer applications to become successful professionals.
2. An ability to develop logic and understand the essential mathematics related to Information Technology.
3. An ability to Design, implement, and evaluates a software product.
4. An ability to apply skills for solving technical problems in software development.
5. An ability to familiarize with emerging & advanced software tools.
6. An ability to experience the industrial environment for understanding the impact of computational solutions in a global & societal context.
7. An ability to analyze the knowledge of contemporary issues.
8. An ability to apply professional ethics.
9. An ability to get readiness to collaborate in a multi-disciplinary team.
10. An ability to communicate effectively.
11. An ability to participate in life-long learning.
12. An ability to handle the projects through appropriate project management techniques.

MAHARAJA INSTITUTE OF TECHNOLOGY MYSORE

DEPARTMENT OF MCA

PROGRAMME SPECIFIC OUTCOMES (PSOS):
PSO1: Problem-Solving Skills: The ability to understand, analyze and develop computer programs in the areas related to algorithms, system software, simulation, software design, web design, big data analytics, and networking for efficient design of computer-based systems of varying complexity. **PSO2: Professional Skills:** The ability to apply standard practices and strategies in software project development using open-ended programming environments to deliver a quality applications for business success. **PSO3: Successful Career and Entrepreneurship:** The ability to employ modern computer languages, technologies, environments, and platforms in creating innovative career paths to be an entrepreneur & zeal for higher studies & research.

Content sheet

Note 1: In the practical Examination student has to execute one program from PART-A and one from PARTB.

Note 2: Change of program is not permitted in the Practical Examination

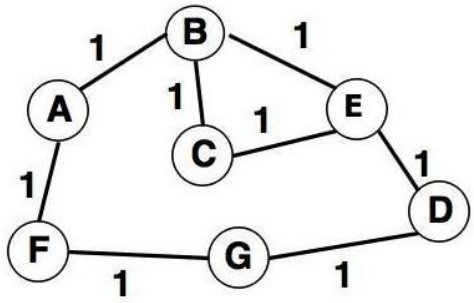
PART – A

Note: Implement the following Computer Networks concepts using C/C++

Program No. 1
Write a program for distance vector algorithm to find suitable path for transmission

Theory:

- Distance vector routing is the dynamic routing algorithm and also known as Bellman-Ford routing algorithm and Ford- Fulkerson algorithm.
- It was designed for small network topologies.
- In this algorithm, node router constructs a table containing the distance (total cost of path) to all other nodes and distributes that vector to its immediate neighbors.
- For distance vector routing, it is assumed that each node knows the cost of the link to each of its directly connected neighbors.
- A link, which is 'down' (which is not working) is assigned as an infinite cost.



- The shortest path can be computed as:

Information at Node	Cost to Reach Node						
	A	B	C	D	E	F	G
A	**0**	**1**	**2**	**3**	**2**	**1**	**2**
B	**1**	**0**	**1**	**2**	**1**	**2**	**3**
C	**2**	**1**	**0**	**2**	**1**	**3**	**3**
D	**3**	**2**	**2**	**0**	**1**	**2**	**2**
E	**2**	**1**	**1**	**1**	**0**	**3**	**2**
F	**1**	**2**	**3**	**2**	**3**	**0**	**1**
G	**2**	**3**	**3**	**1**	**2**	**1**	**0**

Every node sends a message to its directly connected neighbors For example: A sends its information to B and F.
After communicating to each directly connected node the shortest path can be easy to compute (as shown in above table).

Distance Vector Algorithm –

- A router transmits its distance vector to each of its neighbors in a routing packet.
- Each router receives and saves the most recently received distance vector from each of its neighbors.
- A router recalculates its distance vector when:
- It receives a distance vector from a neighbor containing different information than before.
- It discovers that a link to a neighbor has gone down.

Step-by-Step

c(x,v) = cost for direct link from x to v
Node x maintains costs of direct links c(x,v)
Dx(y) = estimate of least cost from x to y
Node x maintains distance vector Dx = [Dx(y): y ϵ N]
Node x maintains its neighbors' distance vectors
For each neighbor v, x maintains Dv = [Dv(y): y ϵ N]
Each node v periodically sends Dv to its neighbors
And neighbors update their own distance vectors
Dx(y) ← minv{c(x,v) + Dv(y)} for each node y ϵ N

Advantages of Distance Vector routing –

It is simpler to configure and maintain than link state routing.

Disadvantages of Distance Vector routing –

- It is slower to converge than link state.
- It is at risk from the count-to-infinity problem.
- It creates more traffic than link state since a hop count change must be propagated to all routers and processed on each router. Hop count updates take place on a periodic basis, even if there are no changes in the network topology, so bandwidth-wasting broadcasts still occur.
- For larger networks, distance vector routing results in larger routing tables than link state since each router must know about all other routers. This can also lead to congestion on WAN links.

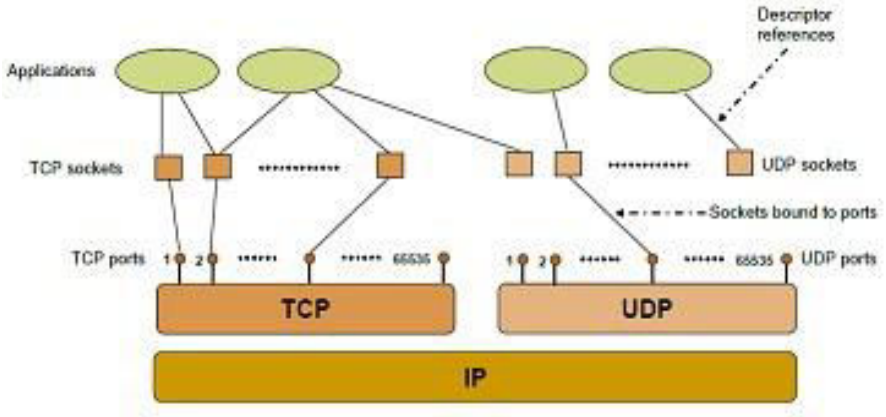
Program No. 2
Using TCP/IP sockets, write a client-server program to make the client send the file name and to make the server send back the contents of the requested file if present.

Theory
Berkley Sockets

- Universally known as Sockets
- It is an abstraction through which an application may send and receive data
- Provide generic access to inter-process communication services
  e.g. IPX/SPX, Appletalk, TCP/IP

Two types of (TCP/IP) sockets

- Streamsockets (e.g. uses TCP)
  provide reliable byte-stream service
- Datagramsockets (e.g. uses UDP)
  provide best-effort datagram service
  messages up to 65.500 bytes



Client-Server communication

- Server
  Passively waits for and responds to clients
  Passive socket
- Client
  Initiates the communication
  Must know the address and the port of the server
  Active socket

Sockets -Procedures

Primitive	Meaning
Socket	Create a new communication endpoint
Bind	Attach a local address to a socket
Listen	Announce willingness to accept connections
Accept	Block caller until a connection request arrives
Connect	Actively attempt to establish a connection
Send	Send some data over the connection
Receive	Receive some data over the connection
Close	Release the connection

Algorithm:

Server Side

- Start.
- Create a socket using socket() system call.
- Bind the socket to an address using bind() system call.
- Listen to the connection using listen() system call.
- accept connection using accept()
- Receive filename and transfer contents of file with client.
- Stop.

Client Side

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

Program No. 3

Implement the above program using message queues or FIFOs as IPC channels

Theory:

The program is implemented using a Server-Client model where the server-side is being run on one terminal window and the client-side is run on another terminal window. Initially, server listens to the requests made by the client. Once the client sends a request, the request is processed by the server. Now the server is in the read mode, i.e. it can read the read the request.

The client-side ideally consists of a FIFO structure set-up using the "mkfifo" command. The requests have to be followed by responses from the server-side. Hence, another FIFO structure is used to implement the response that gets generated at the server-side. Thus we have two FIFO systems in place, namely *request* in read mode and *response* in write mode.

The request consists of the name of the file. Once a request is placed by the client, on the server side, the respective file is opened and contents read. The contents are then passed to the FIFO structure of the client side and thus the contents are displayed.

Algorithm:

Server Side

- Create two string variables for file name and buffer.
- Create three integer variables for Request,Response and file operations.
- Create two FIFO files for response and request using the function mkfifo and pass the respective parameters.
- Pass the request from the client side as any file to the FIFO.
- Using file operation variable open the file in read mode
- Check the file for any characters contained in it. If not present print File not found. else print the contents of the file by passing the contents of the file to the buffer variable using a while loop.
- Print "request sent".
- Unlink the response and request file descriptors.

Client Side

- Create two string variables for filename and buffer.
- Create two integer variables for Request file descriptor and Response descriptors.
- Using the request file descriptors open the fifo file which has the request queue and using the response pointer open the fifo file which has he response queue in it.
- Pass the name of the file having contents to the filename variable.
- Output the contents obtained from the server side.
- Close the request and response descriptors using closed function.

Program No. 4

Write a program for Hamming code generation for error detection and correction.

Theory:

Hamming code is a set of error-correction codes that can be used to detect and correct the errors that can occur when the data is moved or stored from the sender to the receiver. It is technique developed by R.W. Hamming for error correction.

Redundant bits –Redundant bits are extra binary bits that are generated and added to the information-carrying bits of data transfer to ensure that no bits were lost during the data transfer.

The number of redundant bits can be calculated using the following formula

$2^r > m + r + 1$

where, r = redundant bit, m = data bit

Suppose the number of data bits is 7, then the number of redundant bits can be calculated using=2^4>7+4+1

Thus, the number of redundant bits= 4

Parity bits –

A parity bit is a bit appended to a data of binary bits to ensure that the total number of 1's in the data is even or odd. Parity bits are used for error detection. There are two types of parity bits:

Even parity bit: In the case of even parity, for a given set of bits, the number of 1's is counted. If that count is odd, the parity bit value is set to 1, making the total count of occurrences of 1's an even number. If the total number of 1's in a given set of bits is already even, the parity bit's value is 0.

Odd Parity bit: In the case of even parity, for a given set of bits, the number of 1's are counted. If that count is even, the parity bit value is set to 1, making the total count of occurrences of 1's an odd number. If the total number of 1's in a given set of bits is already odd, the parity bit's value is 0.

Algorithm of Hamming code –

The Hamming Code is simply the use of extra parity bits to allow the identification of an error.

- Write the bit positions starting from 1 in binary form (1, 10, 11, 100, etc).
- All the bit positions that are a power of 2 are marked as parity bits (1, 2, 4, 8, etc).
- All the other bit positions are marked as data bits.
- Each data bit is included in a unique set of parity bits, as determined its bit position in binary form.
  Parity bit 1 covers all the bits positions whose binary representation includes a 1 in the least significant

position (1, 3, 5, 7, 9, 11, etc).
Parity bit 2 covers all the bits positions whose binary representation includes a 1 in the second position from
the least significant bit (2, 3, 6, 7, 10, 11, etc).
Parity bit 4 covers all the bits positions whose binary representation includes a 1 in the third position from
the least significant bit (4–7, 12–15, 20–23, etc).
Parity bit 8 covers all the bits positions whose binary representation includes a 1 in the fourth position from
the least significant bit bits (8–15, 24–31, 40–47, etc).
In general each parity bit covers all bits where the bitwise AND of the parity position and the bit position is
non-zero.

- Since we check for even parity set a parity bit to 1 if the total number of ones in the positions it checks is odd.
- Set a parity bit to 0 if the total number of ones in the positions it checks is even

Program No. 5

Write a program for congestion control using leaky bucket algorithm

Theory:

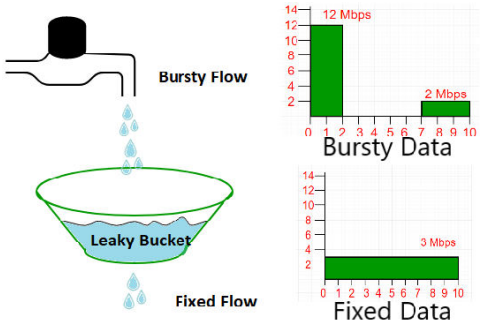
In the network layer, before the network can make Quality of service guarantees, it must know what traffic is being guaranteed. One of the main causes of congestion is that traffic is often bursty.

To understand this concept first we have to know little about traffic shaping. **Traffic Shaping** is a mechanism to control the amount and the rate of the traffic sent to the network. Approach of congestion management is called Traffic shaping. Traffic shaping helps to regulate rate of data transmission and reduces congestion.
There are 2 types of traffic shaping algorithms:

1. Leaky Bucket
2. Token Bucket

Suppose we have a bucket in which we are pouring water in a random order but we have to get water in a fixed rate, for this we will make a hole at the bottom of the bucket. It will ensure that water coming out is in a some fixed rate, and also if bucket will full we will stop pouring in it.

The input rate can vary, but the output rate remains constant. Similarly, in networking, a technique called leaky bucket can smooth out bursty traffic. Bursty chunks are stored in the bucket and sent out at an average rate



In the figure, we assume that the network has committed a bandwidth of 3 Mbps for a host. The use of the leaky bucket shapes the input traffic to make it conform to this commitment. In Figure the host sends a burst of data at a rate of 12 Mbps for 2 s, for a total of 24 Mbits of data. The host is silent for 5 s and then sends data at a rate of 2 Mbps for 3 s, for a total of 6 Mbits of data. In all, the host has sent 30 Mbits of data in 10 s. The leaky bucket smooth's the traffic by sending out data at a rate of 3 Mbps during the same 10 s.

Without the leaky bucket, the beginning burst may have hurt the network by consuming more bandwidth than is set aside for this host. We can also see that the leaky bucket may prevent congestion.

A simple leaky bucket algorithm can be implemented using FIFO queue. A FIFO queue holds the packets. If the traffic consists of fixed-size packets (e.g., cells in ATM networks), the process removes a fixed number of packets from the queue at each tick of the clock. If the traffic consists of variable-length packets, the fixed output rate must be based on the number of bytes or bits.

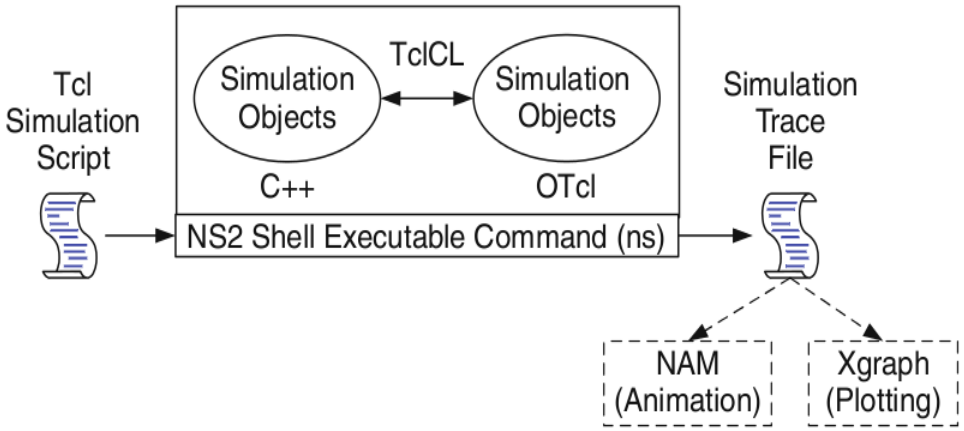
Algorithm:

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

Introduction to NS-2:

- Widely known as NS2, is simply an event driven simulation tool.
- Useful in studying the dynamic nature of communication networks.
- Simulation of wired as well as wireless network functions and protocols (e.g., routing algorithms, TCP, UDP) can be done using NS2.
- In general, NS2 provides users with a way of specifying such network protocols and simulating their corresponding behaviors.

Basic Architecture of NS2



Tcl scripting

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

Basics of TCL

Syntax: command arg1 arg2 arg3

- **Hello World!**

  puts stdout{Hello, World!}

  Hello, World!

- **Variables** Command Substitution

  set a 5 set len [string length foobar]

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

- **Simple Arithmetic**

  expr 7.2 / 4

- **Procedures**

  proc Diag {a b} {

  set c [expr sqrt($a * $a + $b * $b)]

  return $c }

  puts "Diagonal of a 3, 4 right triangle is [Diag 3 4]"

  Output: Diagonal of a 3, 4 right triangle is 5.0

- **Loops**

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

  . . . . . .

  } }

Wired TCL Script Components

Create the event scheduler

Open new files & turn on the tracing

Create the nodes

Setup the links

Configure the traffic type (e.g., TCP, UDP, etc)

Set the time of traffic generation (e.g., CBR, FTP)

Terminate the simulation

NS Simulator Preliminaries.

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

Initialization and Termination of TCL Script in NS-2

An ns simulation starts with the command

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

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

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

#Open the Trace file

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

#Open the NAM trace file

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

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

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

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

#Define a 'finish' procedure

```
Proc finish { } {
global ns tracefile1 namfile
$ns flush-trace
Close $tracefile1
Close $namfile
Exec nam out.nam &
```

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

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

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

will be used to call "**finish**" at time 125sec. Indeed, the **at** method of the simulator allows us to schedule events explicitly.

The simulation can then begin using the command

```
$ns run
```

Definition of a network of links and nodes

The way to define a node is

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

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

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

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

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

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

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

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

Agents and Applications

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

FTP over TCP

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

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

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

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

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

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

#Setup a UDP connection

```
set udp [new Agent/UDP]
$ns attach-agent $n1 $udp
set null [new Agent/Null]
$ns attach-agent $n5 $null
$ns connect $udp $null
$udp set fid_2
```

#setup a CBR over UDP connection

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

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

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

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

CBR over UDP

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

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

```
$cbr set interval_ 0.005
```

The packet size can be set to some value using

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

Scheduling Events

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

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

The scheduler is started when running ns that is through the command $ns run.
The beginning and end of the FTP and CBR application can be done through the following command

Structure of Trace Files

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

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

1. The first field is the event type. It is given by one of four possible symbols r, +, -, d which correspond respectively to receive (at the output of the link), enqueued, dequeued and dropped.
2. The second field gives the time at which the event occurs.
3. Gives the input node of the link at which the event occurs.
4. Gives the output node of the link at which the event occurs.
5. Gives the packet type (eg CBR or TCP)
6. Gives the packet size
7. Some flags
8. This is the flow id (fid) of IPv6 that a user can set for each flow at the input OTcl script one can further use this field for analysis purposes; it is also used when specifying stream color for the NAM display.

9. This is the source address given in the form of "node.port".
10. This is the destination address, given in the same form.
11. This is the network layer protocol's packet sequence number. Even though UDP implementations in a real network do not use sequence number, ns keeps track of UDP packet sequence number for analysis purposes
12. The last field shows the Unique id of the packet.

XGRAPH

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

Syntax:

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

Options are listed here

/-bd <color> (Border)

This specifies the border color of the xgraph window.

/-bg <color> (Background)

This specifies the background color of the xgraph window.

/-fg<color> (Foreground)

This specifies the foreground color of the xgraph window.

/-lf <fontname> (LabelFont)

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

/-t<string> (Title Text)

This string is centered at the top of the graph.

/-x <unit name> (XunitText)

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

/-y <unit name> (YunitText)

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

Awk- An Advanced

awk is a programmable, pattern-matching, and processing tool available in UNIX. It works equally well with text and numbers.

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

Syntax:

```
awk option 'selection_criteria {action}' file(s)
```

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

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

Variables

Awk allows the user to use variables of there choice. You can now print a serial number, using the variable kount, and apply it those directors drawing a salary exceeding 6700:

```
$ awk –F"|" '$3 == "director" && $6 > 6700 {
kount =kount+1
printf " %3f %20s %-12s %d\n", kount,$2,$3,$6 }' empn.lst
```

THE –f OPTION: STORING awk PROGRAMS IN A FILE

You should holds large awk programs in separate file and provide them with the awk extension for easier identification. Let's first store the previous program in the file empawk.awk:

$ cat empawk.awk

Observe that this time we haven't used quotes to enclose the awk program. You can now use awk with the –f *filename* option to obtain the same output:

```
Awk –F"|" –f empawk.awk empn.lst
```

THE BEGIN AND END SECTIONS

Awk statements are usually applied to all lines selected by the address, and if there are no addresses, then they are applied to every line of input. But, if you have to print something before processing the first line, for example, a heading, then the BEGIN section can be used gainfully. Similarly, the end section useful in printing some totals after processing is over.
The BEGIN and END sections are optional and take the form

BEGIN {action}

END {action}

These two sections, when present, are delimited by the body of the awk program. You can use them to print a suitable heading at the beginning and the average salary at the end.

BUILT-IN VARIABLES

Awk has several built-in variables. They are all assigned automatically, though it is also possible for a user to reassign some of them. You have already used NR, which signifies the record number of the current line. We'll now have a brief look at some of the other variable.

The FS Variable*:* as stated elsewhere, awk uses a contiguous string of spaces as the default field delimiter. FS redefines this field separator, which in the sample database happens to be the |. When used at all, it must occur in the BEGIN section so that the body of the program knows its value before it starts processing:

BEGIN {FS="|"}

This is an alternative to the –F option which does the same thing.

The OFS Variable: when you used the print statement with comma-separated arguments, each argument was separated from the other by a space. This is awk's default output field separator, and can reassigned using the variable OFS in the BEGIN section:

BEGIN { OFS="~" }

When you reassign this variable with a ~ (tilde), awk will use this character for delimiting the print arguments. This is a useful variable for creating lines with delimited fields.

The NF variable: NF comes in quite handy for cleaning up a database of lines that don't contain the right number of fields. By using it on a file, say emp.lst, you can locate those lines not having 6 fields, and which have crept in due to faulty data entry:

$awk 'BEGIN {FS = "|"}

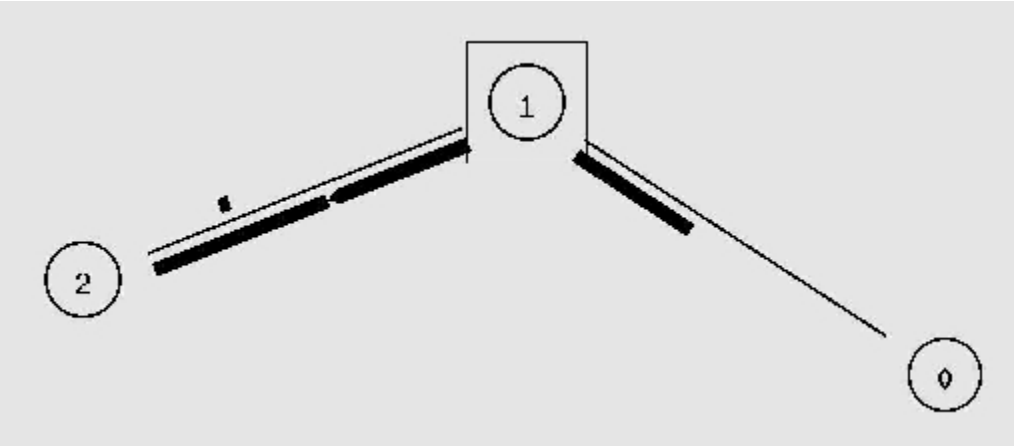
NF! =6 { Print "Record No ", NR, "has", "fields"}' empx.lst

Program 1

Simulate a three nodes point — to — point network with duplex links between them. Set the queue size and vary the bandwidth and find the number of packets dropped.

Algorithm

1. Start Simulator
2. Create three Nodes
3. Assign duplex link between the three nodes
4. Use the TCP/IP Application
5. Vary the bandwidth in the link.
6. Run Simulator , AWK script
7. Stop Simulator

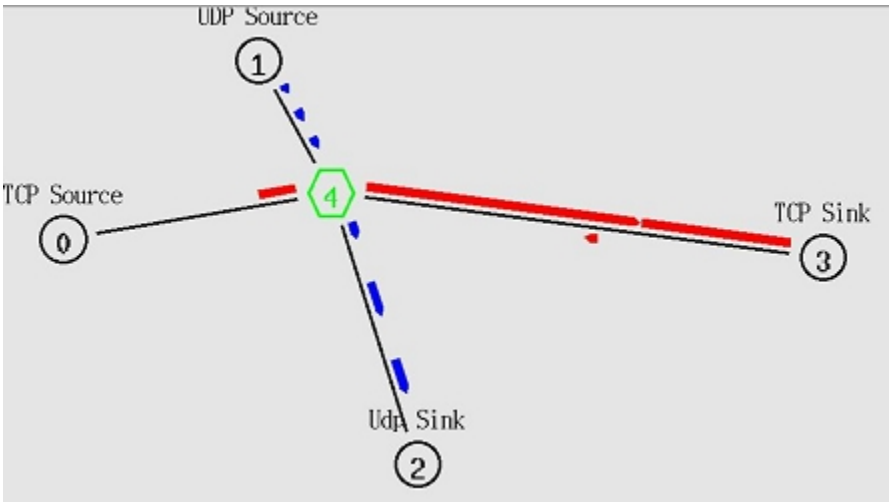


Program 2

Simulate the network with five nodes n0, n1, n2, n3, n4, forming a star topology. The node n4 is at the center. Node n0 is a TCP source, which transmits packets to node n3 (a TCP sink) through the node n4. Node n1 is another traffic source, and sends UDP packets to node n2 through n4. The duration of the simulation time is 10 seconds.

Algorithm

1. Start Simulator
2. Create a star topology using five nodes
3. Assign duplex link between the nodes
4. Use the TCP and UDP Application
5. Run Simulator
6. Stop Simulator

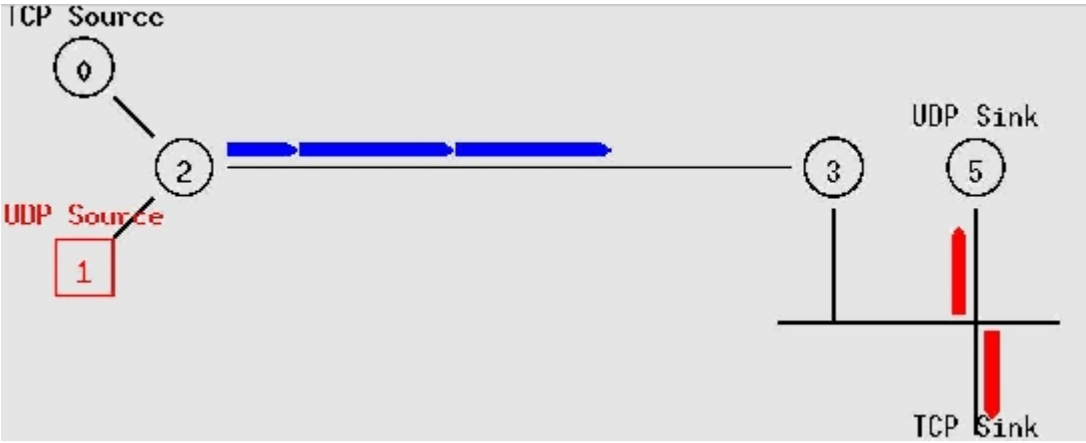


Program 3

Simulate to study transmission of packets over Ethernet LAN and determine the number of packets drop destination.

Algorithm

1. Start Simulator
2. Create a six or seven nodes
3. Create the Ethernet LAN
4. Use the TCP or UDP Application
5. Run Simulator
6. Stop Simulator

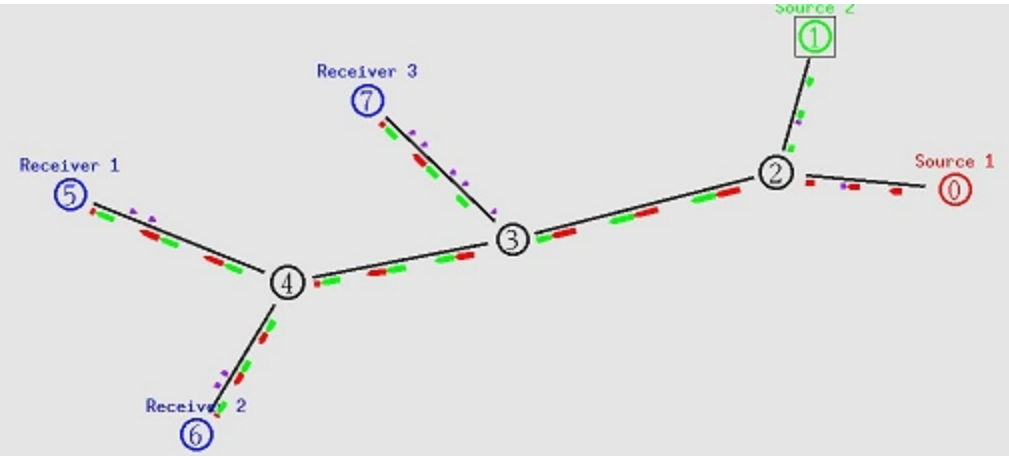


Program 4

Write a TCL Script to simulate working of multicasting routing protocol and analyze the throughput of the network.

Algorithm

1. Start Simulator
2. Create a eight nodes
3. Create the multicasting group of node
4. Use the TCP or UDP Application
5. Run Simulator
6. Stop Simulator

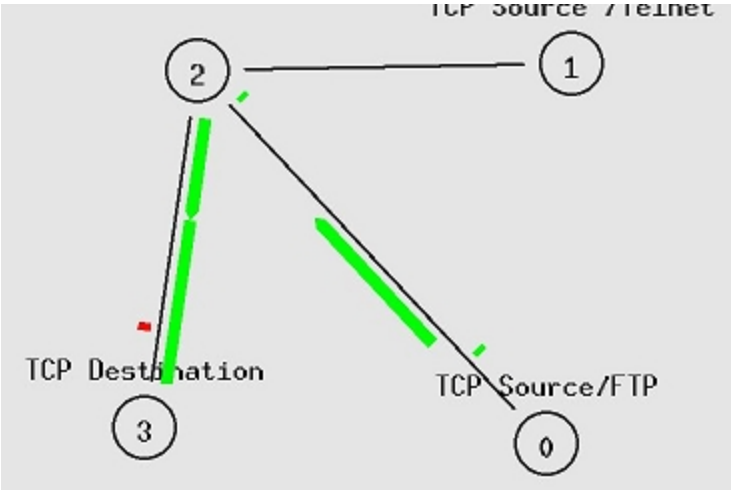


Program 5

Simulate the different types of internet traffic such as FTP and TELNET over a wired network and analyze the packet drop and packet delivery ratio in the network

Algorithm

1. Start Simulator
2. Create a four nodes
3. Use TCP application and FTP object
4. Use TCP application and TELNET object
5. Run Simulator , AWK script
6. Stop Simulator

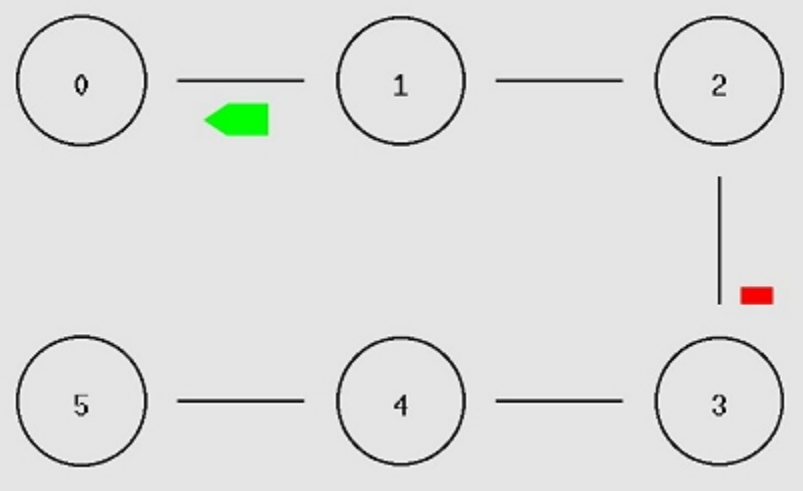


Program 6

Simulate the transmission of ping messages over a network topology consisting of 6 nodes and find the number of packets dropped due to congestion

Algorithm

1. Start Simulator
2. Create a six nodes
3. Define the Agent Ping
4. Use TCP application
5. Run Simulator , AWK script
6. Stop Simulator

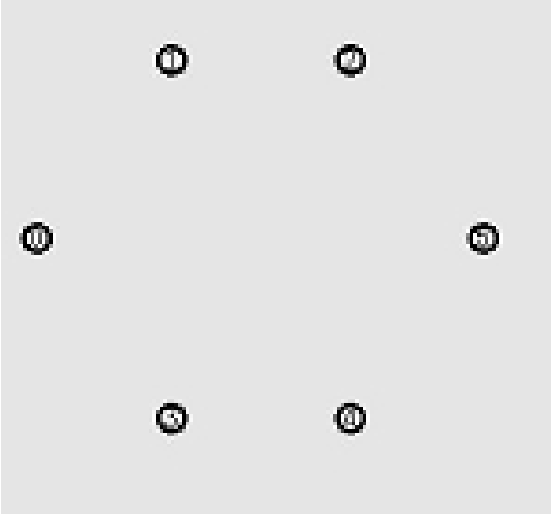


Program 7

Consider six nodes, moving within a flat topology of 700m x 700m. The initial positions of nodes are 0 (150,300), 1 (300,500), 2 (500,500),3 (300,100),4(500,100) and 5(650,300) respectively.

Algorithm

1. Start Simulator
2. Define the node configuration
3. Create a six nodes
4. Use TCP application
5. Run Simulator
6. Stop Simulator



Computer network viva questions

1. Define network?
2. What is a link?
3. What is a node?
4. What is gateway or Router?
5. What is point-to-point link?
6. What is Multiple Access?
7. What is protocol?
8. What are the key elements of protocols?
9. Define Bandwidth and Latency?
10. Define Routing?
11. What is peer to peer process?
12. What is multiplexing?
13. Name the categories of multiplexing?
14. What FDM?
15. What is TDM?
16. List the layers of OSI?
17. What are the concerns of the Physical Layer?
18. What are responsibilities of Data Link Layer?
19. What are responsibilities of Network Layer?
20. What are responsibilities of Transport Layer?
21. What are responsibilities of session Layer?
22. What are responsibilities of Presentation Layer?
23. What are responsibilities of Application Layer?
24. What are the categories of Transmission Media?
25. What are the types of error?
26. What is Error Detection? What are its Method?
27. What is CRC?
28. What is checksum?
29. Define retransmission?
30. Define Encoder?
31. Define Decoders?
32. What is Framing?
33. What is Bit Stuffing?
34. What is flow control?
35. What is Error Control?
36. What Automatic Repeat Request (ARQ)?
37. What is usage of sequence Number in Reliable Transmission?
38. What is pipelining?
39. What is Piggy Backing?
40. What are the two types of transmission technology available

41. What is subnet?
42. **What is cladding?**
43. **What is attenuation?**
44. **What is MAC address?**
45. **Difference between bit rate and baud rate**
46. **What is Bandwidth ?**
47. **What are the different type of networking /internetworking devices?**
48. **What is ICMP?**
49. **What is difference between ARP and RARP?**
50. **What is the range of addresses in the classes if internet addresses?**
51. **What are the important topologies for networks?**
52. **What is difference between baseband and broadband transmission?**
53. **What is virtual path?**
54. **What are the four files on the NS2 Simulator?**
55. **What is the use of a tr file?**
56. **What is the use of a nam file?**
57. **What does $1, $2 ….. indicate in the AWK file?**
58. **What is the difference between TCP and UDP?**
59. **Explain syntax of duplex link in ns2?**
60. **What is round trip time?**
61. **Explain fields of trace files.**
62. **Why NS2 is written in two languages?**
63. **List the primitives of the sockets?**
64. **List the different routing Techniques available in NS2**
65. **What is a flush-trace in ns2?**
66. **What is a trace-all ?**
67.