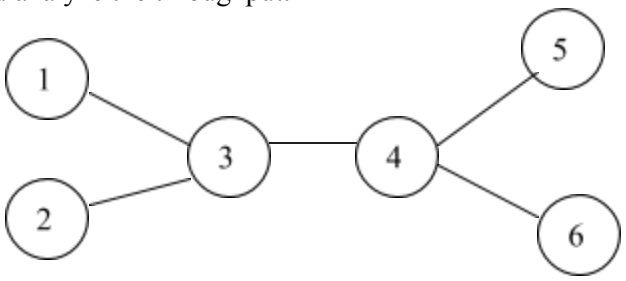
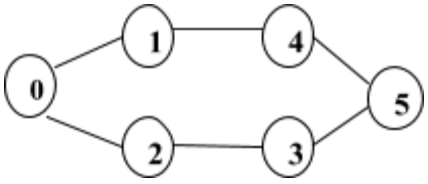
Part A (Wired Program)

1. Simulate a four node point-to-point network with duplex link as follows: n0-n2, n1-n2 and n2-n3. Apply TCP agent between n0-n3 and UDP agent between n1-n3. Apply relevant applications over TCP and UDP agents. Set the queue size to 5 and vary the bandwidth to find the number of packet dropped and received by TCP/UDP using awk script and grep command.

2. Set up the network topology as shown in fig 1. Simulate the different types of Internet traffic such as FTP between nodes n1-n6 and telnet between the nodes n2-n5. Plot congestion window for ftp and telnet and analyze the throughput.



3. Design networks that demonstrate the working of Distance vector routing protocol. The link between 1 and 4 breaks at 1.0ms and comes up at 3.0ms. Assume that the source node 0 transmits packets to node 4. Plot the congestion window when TCP sends packets via other nodes. Assume your own parameters for bandwidth and delay.



4. Consider a client and a server. The server is running a FTP application over TCP. The client sends a request to download a file of size 10 MB from the server. Write a TCL script to simulate this scenario. Let node n0 be the server and node n1 be the client. TCP packet size is 1500 Bytes.

5. Demonstrate the working of multicast routing protocol. Plot the congestion window for the source node and write your observation on protocol performance. Assume your own parameters for bandwidth and delay

STEPS TO INSTALL NS2 ON LINUX:

1. in root - create a directory ns2
2. copy .tar file into this folder[ns-allinone-2.31.tar.gz]
3. type at the prompt [root@lacalhost~]#cd ns2 … that is get into ns2 folder
4. #tar –zxvf ns-allinone-2.31.tar.gz …. This is to unzip the files
5. now get into ns-allinone-2.31 i.e # cd ns-allinone-2.31
6. # ./install … this command is used to install ns2. takes time : 10-15 mins
7. # cd .. [go to root]
8. #ls –a [to see hidden files]
9. vi .bash_profile
10. copy pathset file to this to set the path , in this :wq to save and quit
11. for first time installation type # source .bash_profile(otherwise OS does it)
12. type on the prompt ns. - #ns if we get % symbol, then ns is installed properly, press ctrl+C

a. type the tcl code in gedit, and save your filename with extension .tcl
b. to view example, you can use,

ns-allinone-2.31/ns-2.31/sample
or
ns-allinone-2.31/ns-2.31/tcl/ex

a. to run the animator type : nam namefile.nam
b. to view the trace file : gedit out.tr

PROCEDURE TO SET UP WIRED NETWORK IN NETWORK SIMULATIOR-2

Step 1: Create Simulator Class' Object
Step 2: Store results in File
Step 3: Create Nodes
Step 4: Connect Nodes
Step 5: Create Agent (TCP or UDP)
Step 6: Connect Agents on Both Nodes
Step 7: Setup Application over Agent
Step 8: Attach Application with Agent
Step 9: Create finish procedure Flush Buffer and Start NAM
Step 10: Schedule events
Step 11: Start Simulation
Step 12: Save the Simulation program as <filename>.tcl
Step 13: Run the Simulation using ns command
Ex: $ns filename.tcl
Step 14: Check for trace file
Ex: gedit filename.tr
Step 15: Measure the required performance using suitable filters.

Part A (Wired Program)

1. **Simulate a four node point-to-point network with duplex link as follows: n0-n2, n1-n2 and n2-n3. Apply TCP agent between n0-n3 and UDP agent between n1-n3. Apply relevant applications over TCP and UDP agents. Set the queue size and vary the bandwidth.Find the number of packet dropped and received by TCP/UDP using awk script and grep command**

Solution:

Aim: To understand and design the point-to-point network structure. Also here we understand the working of TCP and UDP transport protocol.

Theory:

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

Save the following program as ex1.tcl

```
set ns [new Simulator]

set tf [open ex1.tr w]
$ns trace-all $tf

set nf [open ex1.nam w]
$ns namtrace-all $nf
```

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

$ns duplex-link $n0 $n2 2Mb 2ms DropTail
$ns duplex-link $n1 $n2 2Mb 2ms DropTail
$ns duplex-link $n2 $n3 0.4Mb 10ms DropTail
$ns queue-limit $n0 $n2    5

set udp1 [new Agent/UDP]
$ns attach-agent $n0 $udp1
set null1 [new Agent/Null]
$ns attach-agent $n3 $null1
$ns connect $udp1 $null1
set cbr1 [new Application/Traffic/CBR]
$cbr1 attach-agent $udp1
$ns at 1.1 "$cbr1 start"

set tcp [new Agent/TCP]
$ns attach-agent $n3 $tcp
set sink [new Agent/TCPSink]
$ns attach-agent $n1 $sink
$ns connect $tcp $sink
set ftp [new Application/FTP]
$ftp attach-agent $tcp

$ns at 0.1 "$ftp start"

$ns at 10.0 "finish"

proc finish {} {
        global ns tf nf
        $ns flush-trace
        close $tf
        close $nf
        puts "running nam..."
        exec nam ex1.nam &
```

```
        exit 0
}
```

$ns run

Expected output: Animated 4 node structure is displayed. We need to see the trace file to understand what has happened to the data flow.

Grep

grep "^r" ex1.tr **#packets received**

To calculate number of packet dropped by TCP and udp we need to write awk script.

Save the below program as ex1.awk

```
BEGIN {
        tcp_count=0;
        udp_count=0;
}
{
        if ( $1 == "d" && $5 == "tcp")
                tcp_count ++;

     if ( $1 == "d" && $5 == "cbr")
                udp_count ++;
}
END {
printf("Number of packet dropped in TCP      %d\n", tcp_count);
printf("Number of packet dropped in UDP     %d\n", udp_count);
}
```

To run the awk script you need to execute the command shown bellow on the terminal.

awk –f ex1.awk ex1.tr

2. **Simulate the different types of Internet traffic such as FTP and Telnet over a network, Plot congestion window and analyze the throughput.**

Solution :

Aim: To understand and design TCP Applications like FTP and Telnet. Here we see that the throughput of both the application vary depending on the data size

Theory:

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

Save the following program as ex2.tcl

```
set ns [new Simulator]

set tf [open ex2.tr w]
```

```
$ns trace-all $tf

set nf [open ex2.nam w]
$ns namtrace-all $nf

set cwind [open win2.tr w]

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

$ns duplex-link $n0 $n2 5Mb 2ms DropTail
$ns duplex-link $n1 $n2 5Mb 2ms DropTail
$ns duplex-link $n2 $n3 1.5Mb 10ms DropTail
set tcp0 [new Agent/TCP]
$ns attach-agent $n0 $tcp0
set sink0 [new Agent/TCPSink]
$ns attach-agent $n3 $sink0
$ns connect $tcp0 $sink0
set ftp [new Application/FTP]
$ftp attach-agent $tcp0

$ns at 1.2 "$ftp start"

set tcp1 [new Agent/TCP]
$ns attach-agent $n1 $tcp1
set sink1 [new Agent/TCPSink]
$ns attach-agent $n0 $sink1
$ns connect $tcp1 $sink1
set telnet [new Application/Telnet]
$telnet attach-agent $tcp1

$ns at 1.5 "$telnet start"

$ns at 10.0 "finish"

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 2.0 "plotWindow $tcp0 $cwind"
$ns at 5.5 "plotWindow $tcp1 $cwind"

proc finish {} {
        global ns tf nf cwind
        $ns flush-trace
        close $tf
        close $nf

        puts "running nam..."
        puts "FTP PACKETS.."
        puts "Telnet PACKETS.."
        exec nam ex2.nam &
      exec xgraph win2.tr &
        exit 0
}
$ns run
```

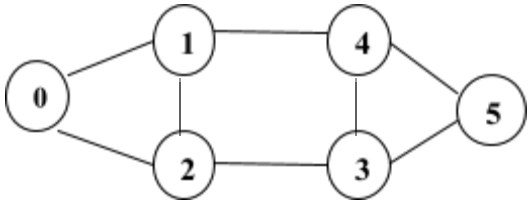
Expected output: Animated 4 node structure is displayed. We need to see the trace file to understand what has happened to the data flow depending on the application used.

Awk script to calculate throughput

```
BEGIN {
    last = 0
    tcp_sz = 0
    cbr_sz = 0
    total_sz = 0

}
{
  action = $1;
```

```
   time = $2;

   from = $3;

   to = $4;

   type = $5;

   pktsize = $6;

   flow_id = $8;

   src = $9;

   dst = $10;

   seq_no = $11;

   packet_id = $12;

         if (type == "tcp" && action == "r" && to == "3" )
         tcp_sz += pktsize

         if (type == "cbr" && action == "r" && to == "3" )
          cbr_sz += pktsize

 total_sz += pktsize
 }
END {
    print time, ( tcp_sz * 8 / 1000000)
   print time , (tcp_sz * 8 / 1000000 ), ( total_sz * 8 / 1000000)
 }
```

3. **Set up the topology with 6 nodes, and demonstrate the working of Distance vector routing protocol. The link between 1 and 4 breaks at 1.0ms and comes up at 3.0ms. Assume that the source node 0 transmits packets to node 5. Plot the congestion window when TCP sends packets via 4, 5, 6. Assume your own parameters for bandwidth and delay.**



Solution:

Aim: To understand the working o distance vector routing.

Theory:

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

nam.

Save the following program as ex4.tcl

```
set ns [new Simulator]

set tf [open ex4.tr w]
$ns trace-all $tf

set nf [open ex4.nam w]
$ns namtrace-all $nf
set cwind [open win4.tr w]

$ns color 1 Blue
$ns color 2 Red

$ns rtproto DV

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

$ns duplex-link $n0 $n1 0.3Mb 10ms DropTail
$ns duplex-link $n1 $n2 0.3Mb 10ms DropTail
$ns duplex-link $n2 $n3 0.3Mb 10ms DropTail
$ns duplex-link $n1 $n4 0.3Mb 10ms DropTail
$ns duplex-link $n3 $n5 0.5Mb 10ms DropTail
$ns duplex-link $n4 $n5 0.5Mb 10ms DropTail

$ns duplex-link-op $n0 $n1 orient right
$ns duplex-link-op $n1 $n2 orient right
$ns duplex-link-op $n2 $n3 orient up
$ns duplex-link-op $n1 $n4 orient up-left
$ns duplex-link-op $n3 $n5 orient up-left
$ns duplex-link-op $n4 $n5 orient right-up
```

```
set tcp [new Agent/TCP]
$ns attach-agent $n0 $tcp
set sink [new Agent/TCPSink]
$ns attach-agent $n5 $sink
$ns connect $tcp $sink
$tcp set fid_ 1

set ftp [new Application/FTP]
$ftp attach-agent $tcp

$ns rtmodel-at 1.0 down $n1 $n4
$ns rtmodel-at 3.0 up $n1 $n4

$ns at 0.1 "$ftp start"

$ns at 12.0 "finish"

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 1.0 "plotWindow $tcp $cwind"

proc finish {} {
        global ns tf nf cwind
        $ns flush-trace
        close $tf
        close $nf
        exec nam ex4.nam &
      exec xgraph win4.tr &
        exit 0
}
```

$ns run

Expected output: Animated 6 node structure is displayed. We need to see the nam file as well as the trace file to understand what has happened to the data flow.

4. **Consider a client and a server. The server is running a FTP application over TCP. The client sends a request to download a file of size 10 MB from the server. Write a TCL script to simulate this scenario. Let node n0 be the server and node n1 be the client. TCP packet size is 1500 Bytes.**

Solution:

Aim: To understand the working of a client and a server, when transmitting 10MB file from server to the client.

Theory:

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

Save the following program as ex6.tcl

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

#Open the NS trace file
set tracefile [open ex5.tr w]
$ns trace-all $tracefile

#Open the NAM trace file
set namfile [open ex5.nam w]
$ns namtrace-all $namfile

#Create 2 nodes
set s [$ns node]
set c [$ns node]

$ns color 1 Blue
```

```
#Create labels for nodes
$s label "Server"
$c label "Client"

#Create links between nodes
$ns duplex-link $s $c 10Mb 22ms DropTail

#Give node position (for NAM)
$ns duplex-link-op $s $c orient right


#Setup a TCP connection for node s(server)
set tcp0 [new Agent/TCP]
$ns attach-agent $s $tcp0
$tcp0 set packetSize_ 1500

#Setup a TCPSink connection for node c(client)
set sink0 [new Agent/TCPSink]
$ns attach-agent $c $sink0

$ns connect $tcp0 $sink0

#Setup a FTP Application over TCP connection
set ftp0 [new Application/FTP]
$ftp0 attach-agent $tcp0

$tcp0 set fid_ 1

proc finish { } {
        global ns tracefile namfile
        $ns flush-trace
        close $tracefile
        close $namfile
        exec nam ex5.nam &
        exec awk -f ex5transfer.awk ex5.tr &
        exec awk -f ex5convert.awk  ex5.tr > convert.tr &
```

```
        exec xgraph convert.tr -geometry 800*400 -t "bytes_received_at_client" -x
"time_in_secs" -y "bytes_in_bps"  &
                }

$ns at 0.01 "$ftp0 start"
$ns at 15.0 "$ftp0 stop"
$ns at 15.1 "finish"
$ns run
```

Expected output: Animated 2 node structure is displayed with the node labeled as client and server. We need to make use of the awk script to calculate the time required to transfer the 10 MB file from the server to client and duration for converting downloaded file into MB.

Save the following awk script as ex5transfer.awk

```
# AWK script to calulate the time required to transfer the 10 MB file from the server to
client
BEGIN {
        count=0;
        time=0;
      total_bytes_sent =0;
      total_bytes_received=0;

}
{
        if ( $1 == "r" &&  $4 == 1 && $5 == "tcp")
                                total_bytes_received += $6;

      if($1 == "+" &&  $3 == 0 && $5 == "tcp")
                                total_bytes_sent +=  $6;
}
END {
     system("clear");
     printf("\n Transmission time required to transfer the file is %f",$2);
```

```
    printf("\n Actual data sent from the server is %f Mbps",(total_bytes_sent)/1000000);
                    printf("\n Data Received by the client is %f Mbps\n",(total_bytes_received)/1000000);
}
```

Save the following awk script as ex5convert.awk

```
# AWK Script to convert the downloaded file into MB
BEGIN {
        count=0;
        time=0;
}
{
        if ( $1 == "r" &&  $4 == 1 && $5 == "tcp")
        {
                count += $6;
                time=$2;
                printf("\n%f\t%f",time,(count)/1000000);
        }
}
END {
}
```

5. **Set up topology and demonstrate the working of multicast routing protocol. Plot the congestion window for the source node and write your observation on protocol performance. Assume your own parameters for bandwidth and delay.**

Aim: To understand the working of multicast routing protocol and observe the protocol performance by plotting congestion window.

Theory:

- Create a simulator object.
- We open a file for writing that is going to be used for the “nam” trace data.
- We now attach the agent to the nodes.
- Now we attach the application to run on top of these nodes

- We now connect the agent and the application for its working
- Set the simulation time
- The next step is to add a 'finish' procedure.

```
#Create an event scheduler wit multicast turned on
set ns [new Simulator -multicast on]
#$ns multicast

#Turn on Tracing
set tf [open mcast.tr w]
$ns trace-all $tf

# Turn on nam Tracing
set fd [open mcast.nam w]
$ns namtrace-all $fd

# Create nodes
set n0 [$ns node]
set n1 [$ns node]
set n2 [$ns node]
set n3 [$ns node]
set n4 [$ns node]
set n5 [$ns node]
set n6 [$ns node]
set n7 [$ns node]

# Create links
$ns duplex-link $n0 $n2 1.5Mb 10ms DropTail
$ns duplex-link $n1 $n2 1.5Mb 10ms DropTail
$ns duplex-link $n2 $n3 1.5Mb 10ms DropTail
$ns duplex-link $n3 $n4 1.5Mb 10ms DropTail
$ns duplex-link $n3 $n7 1.5Mb 10ms DropTail
$ns duplex-link $n4 $n5 1.5Mb 10ms DropTail
$ns duplex-link $n4 $n6 1.5Mb 10ms DropTail

# Routing protocol: say distance vector
#Protocols: CtrMcast, DM, ST, BST
set mproto DM
set mrthandle [$ns mrtproto $mproto {}]
```

```
# Allocate group addresses
set group1 [Node allocaddr]
set group2 [Node allocaddr]

# UDP Transport agent for the traffic source
set udp0 [new Agent/UDP]
$ns attach-agent $n0 $udp0
$udp0 set dst_addr_ $group1
$udp0 set dst_port_ 0
set cbr1 [new Application/Traffic/CBR]
$cbr1 attach-agent $udp0

# Transport agent for the traffic source
set udp1 [new Agent/UDP]
$ns attach-agent $n1 $udp1
$udp1 set dst_addr_ $group2
$udp1 set dst_port_ 0
set cbr2 [new Application/Traffic/CBR]
$cbr2 attach-agent $udp1

# Create receiver
set rcvr1 [new Agent/Null]
$ns attach-agent $n5 $rcvr1
$ns at 1.0 "$n5 join-group $rcvr1 $group1"

set rcvr2 [new Agent/Null]
$ns attach-agent $n6 $rcvr2
$ns at 1.5 "$n6 join-group $rcvr2 $group1"

set rcvr3 [new Agent/Null]
$ns attach-agent $n7 $rcvr3
$ns at 2.0 "$n7 join-group $rcvr3 $group1"

set rcvr4 [new Agent/Null]
$ns attach-agent $n5 $rcvr1
$ns at 2.5 "$n5 join-group $rcvr4 $group2"
set rcvr5 [new Agent/Null]
```

```
$ns attach-agent $n6 $rcvr2
$ns at 3.0 "$n6 join-group $rcvr5 $group2"
set rcvr6 [new Agent/Null]
$ns attach-agent $n7 $rcvr3
$ns at 3.5 "$n7 join-group $rcvr6 $group2"

$ns at 4.0 "$n5 leave-group $rcvr1 $group1"
$ns at 4.5 "$n6 leave-group $rcvr2 $group1"
$ns at 5.0 "$n7 leave-group $rcvr3 $group1"

$ns at 5.5 "$n5 leave-group $rcvr4 $group2"
$ns at 6.0 "$n6 leave-group $rcvr5 $group2"
$ns at 6.5 "$n7 leave-group $rcvr6 $group2"

# Schedule events
$ns at 0.5 "$cbr1 start"
$ns at 9.5 "$cbr1 stop"

$ns at 0.5 "$cbr2 start"
$ns at 9.5 "$cbr2 stop"

$ns at 10.0 "finish"

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


# For nam

# Group 0 source
#$udp0 set fid_ 1
#$n0 color red
```

```
$n0 label "Source 1"

# Group 1 source
#$udp1 set fid_ 2
#$n1 color green
$n1 label "Source 2"

#Colors for packets from two mcast groups
$ns color 1 red
$ns color 2 green


$n5 label "Receiver 1"
$n5 color blue
$n6 label "Receiver 2"
$n6 color blue
$n7 label "Receiver 3"
$n7 color blue

$ns run
```

Expected output: Animated node structure is displayed with single sender multicasting data to specific receiver group.