Program – 1

Implement three nodes point – to – point network with duplex links between them. Set the queue size, vary the bandwidth and find the number of packets dropped.

Using TCP Technique (Connection-oriented Protocol)

TCL Code: (P1tcp.tcl)

```
# Simulation parameters setup
set val(stop) 6.0; #stopping time of the simulation

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

# Open the NS trace file
set tracefile [open 1.tr w]
$ns trace-all $tracefile

# Open the NAM trace file
set namfile [open 1.nam w]
$ns namtrace-all $namfile

# Nodes Definition
# Create 3 nodes
set n1 [$ns node]
set n2 [$ns node]
set n3 [$ns node]

# Links Definition
# Createlinks between nodes
$ns duplex-link $n1 $n2 1000Kb 60ms DropTail
$ns queue-limit $n1 $n2 14
# Change the values below for the table
$ns duplex-link $n2 $n3 500Kb 60ms DropTail
$ns queue-limit $n2 $n3 4
$ns duplex-link-op $n1 $n2 queuePos 0.5
$ns duplex-link-op $n2 $n3 queuePos 0.2
```

```
# Agents Definition
# Setup a TCP connection
set tcp0 [new Agent/TCP]
$ns attach-agent $n1 $tcp0
set sink1 [new Agent/TCPSink]
$ns attach-agent $n3 $sink1
$ns connect $tcp0 $sink1
$tcp0 set packetSize_ 1500

# Applications Definition
# Setup a FTP Application over TCP connection
set ftp0 [new Application/FTP]
$ftp0 attach-agent $tcp0
$ns at 0.2 "$ftp0 start"
$ns at 5.0 "$ftp0 stop"

# Termination
# Define a 'finish' procedure
proc finish {} {
global ns tracefile namfile
$ns flush-trace
close $tracefile
close $namfile
exec nam 1.nam &
exit 0
}

$ns at $val(stop) "$ns nam-end-wireless $val(stop)"
$ns at $val(stop) "finish"
$ns at $val(stop) "puts \"done\" ; $ns halt"
$ns run
```

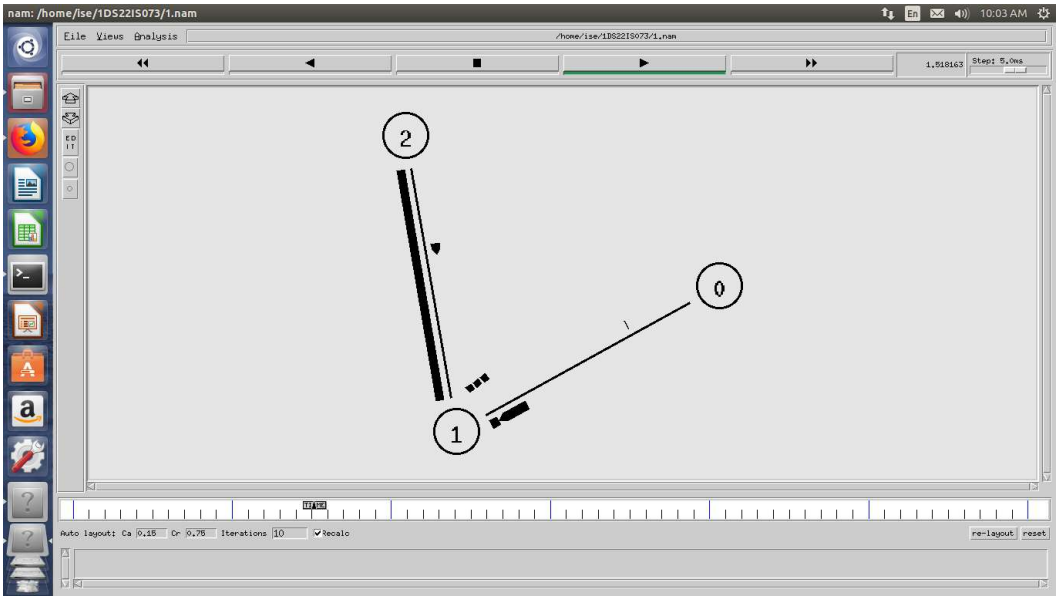
AWK Code: (P1tcp.awk)

```
BEGIN {
count=0;
total=0;
}
{
event=$1;
```

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

Output:

No of packets dropped: 5



By changing the bandwidth at destination node,

Bandwidth	Queue Size	No. of Packets dropped
500 Kb	4	5
1 Mb	4	0
100 Kb	4	10
800 Kb	4	1
300 Kb	4	6

By changing the Queue size at destination node,

Bandwidth	Queue Size	No. of Packets dropped
100 Kb	4	10
100 Kb	6	11
100 Kb	10	7
100 Kb	14	3
100 Kb	16	1

Using UDP Technique (Connectionless Protocol)

TCL Code: (P1udp.tcl)

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

# Open the NS trace file
set tf [ open lab1.tr w]
$ns trace-all $tf

# Open the NAM trace file
set nf [ open lab1.nam w]
$ns namtrace-all $nf

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

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

# Providing the link
$ns duplex-link $n0 $n2 10kb 100ms DropTail
$ns duplex-link $n1 $n2 10kb 10ms DropTail # Change

# Set the queue size b/w the nodes
$ns set queue-limit $n0 $n2 5
$ns set queue-limit $n1 $n2 5 # Change

# Agents Definition
# Setup a UDP connection
set udp0 [new Agent/UDP]
set udp1 [new Agent/UDP]
```

```
$ns attach-agent $n0 $udp0
$ns attach-agent $n1 $udp1

# Applications Definition
# Setup a CBR Application over UDP connection
set cbr0 [new Application/Traffic/CBR]
set cbr1 [new Application/Traffic/CBR]
$cbr0 attach-agent $udp0
$cbr1 attach-agent $udp1

set null0 [new Agent/Null]
$ns attach-agent $n2 $null0

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

# The below code is used to connect the agents.
$ns connect $udp0 $null0
$ns connect $udp1 $null0

# The below code is used to set the packet size to 500
$cbr0 set packetSize_ 500Mb
$cbr1 set packetSize_ 500Mb

# The below code is used to set the interval of the packets,
$cbr0 set interval_ 0.01
$cbr1 set interval_ 0.01

# Termination
# Define a 'finish' procedure
proc finish {} {
global ns nf  tf
$ns flush-trace
exec nam lab1.nam &
close $tf
close $nf
exit 0
}
```

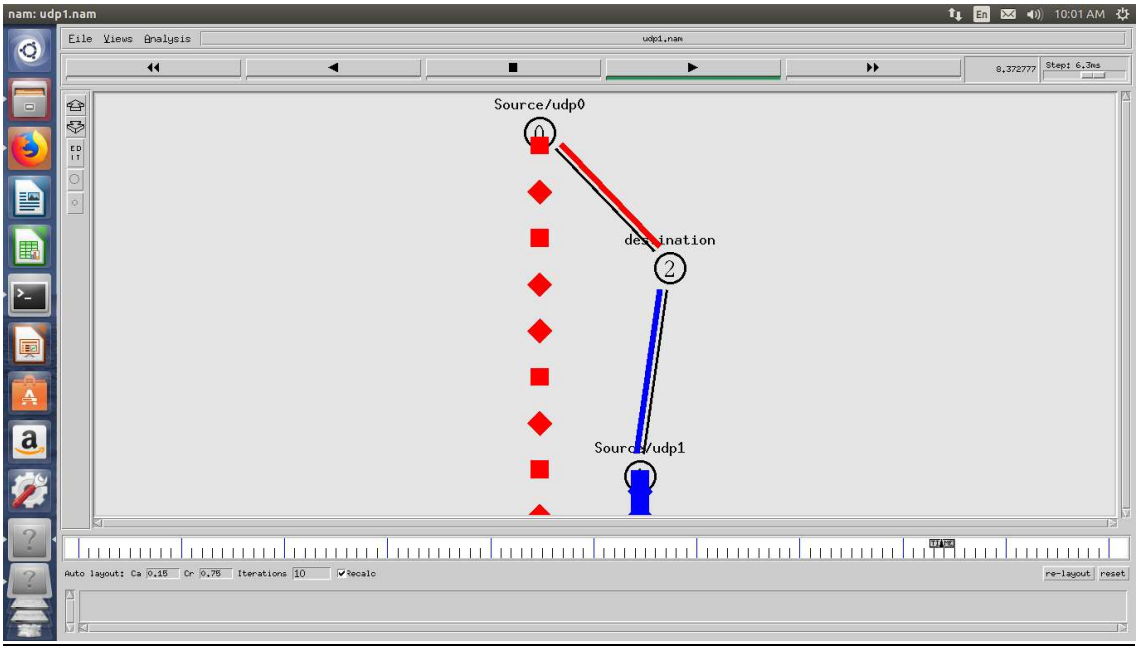
```
$ns at 0.1 "$cbr0 start"
$ns at 0.1 "$cbr1 start"
$ns at 9.5 "$cbr0 stop"
$ns at 10.0 "$cbr1 stop"
$ns at 10.0 "finish"
$ns run
```

AWK Code: (P1udp.awk)

```
BEGIN{
count=0;
}
{
if($1=="d")
count++ ;
}
END{
printf("The Total no of Packets Dropped due to Congestion : %d\n", count)
}
```

Output:

The Total no of Packets Dropped due to Congestion: 1785



By changing the bandwidth at destination node,

Bandwidth	Queue Size	No. of Packets dropped
10 Kb	5	1785
100 Kb	5	1562
200 Kb	5	1315
300 Kb	5	1067
500 Kb	5	868

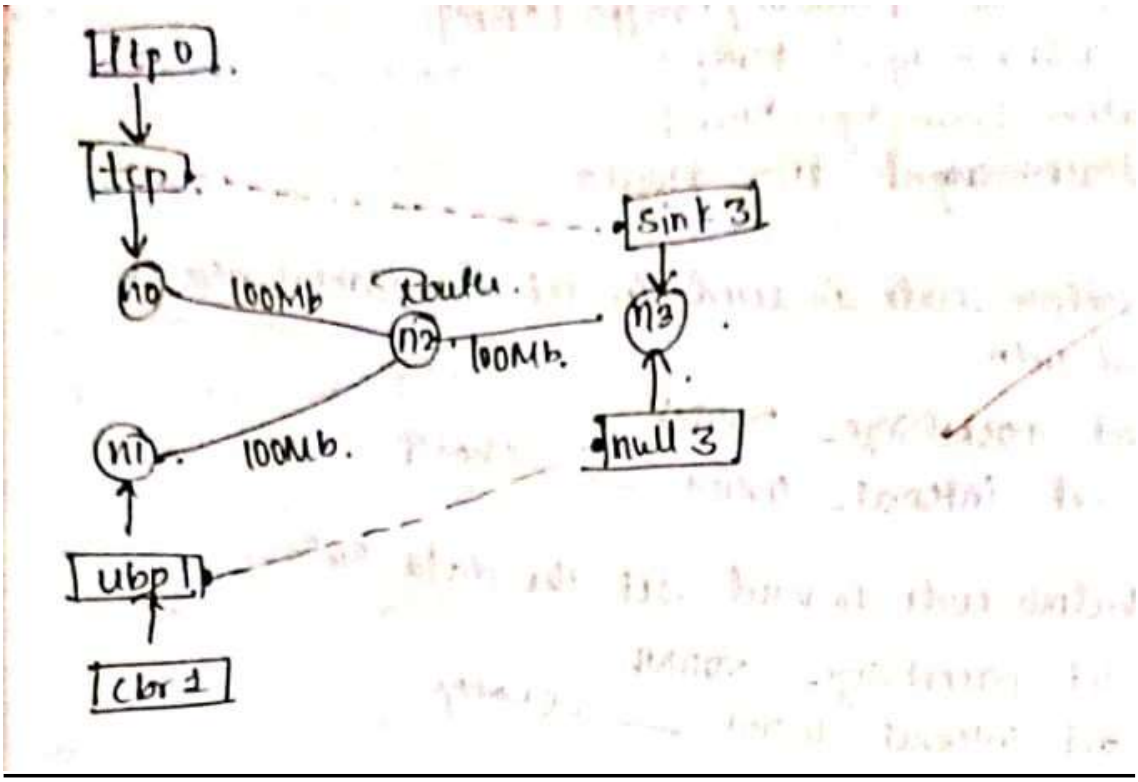
By changing the Queue size at destination node,

Bandwidth	Queue Size	No. of Packets dropped
10 Kb	3	1574
10 Kb	5	1574
10 Kb	7	1574
10 Kb	9	1574
10 Kb	11	1574

Program – 2

Simulate a four-node point-to-point network with the links connected as follows: n0 – n2, n1 – n2 and n2 – n3. Apply TCP agent between n0-n3 and UDP between n1 – n3. Apply relevant applications over TCP and UDP agents changing the parameter and determine the number of packets sent by TCP / UDP.

Topology:



TCL Code: (P2.tcl)

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

# Open the NS trace file
set tf [open p2.tr w]
$ns trace-all $tf

# Open the NAM trace file
set nf [open p2.nam w]
$ns namtrace-all $nf
```

```
# The below code is used to create the nodes.
set n0 [$ns node]
set n1 [$ns node]
set n2 [$ns node]
set n3 [$ns node]

# The below code is used to give color and label the nodes.
$ns color 1 "red"
$ns color 2 "blue"
$n0 label "Source/TCP"
$n1 label "Source/UDP"
$n2 label "Router"
$n3 label "Destination"

# Providing the link
$ns duplex-link $n0 $n2 100Mb 1ms DropTail #Change
$ns duplex-link $n1 $n2 100Mb 1ms DropTail #Change
$ns duplex-link $n2 $n3 100Mb 1ms DropTail

# Agent & Application Definition of TCP Connection
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

# Agent & Application Definition of UDP Connection
set udp1 [new Agent/UDP]
$ns attach-agent $n1 $udp1
set cbr1 [new Application/Traffic/CBR]
$cbr1 attach-agent $udp1
set null3 [new Agent/Null]
$ns attach-agent $n3 $null3

# The below code is used to set the packet size and interval size of packets
$ftp0 set packetSize_ 500Mb
$ftp0 set interval_ 0.01 #Change
$cbr1 set packetSize_ 500Mb
$cbr1 set interval_ 0.01 #Change
```

```
# To set color to packets
$tcp0 set class_ 1
$udp1 set class_ 2

# The below code is used to connect the agents
$ns connect $tcp0 $sink3
$ns connect $udp1 $null3

# Termination defining a ‘finish’ procedure
proc finish {} {
global ns nf tf
$ns flush-trace
exec nam p2.nam &
close $nf
close $tf
exit 0
}

$ns at 0.1 "$cbr1 start"
$ns at 0.2 "$ftp0 start"
$ns at 9.5 "$cbr1 stop"
$ns at 10.0 "finish"
$ns run
```

AWK Code: (P2.awk)

```
BEGIN{
tcp=0;
udp=0;
}
{
if($1=="r"&&$3=="2"&&$4=="3"&& $5=="tcp")
tcp++;
if($1=="r"&&$3=="2"&&$4=="3"&&$5=="cbr")
udp++;
}
END{
printf("\n Total number of packets sent by TCP : %d\n",tcp);
printf("\n Total number of packets sent by UDP : %d\n",udp);
}
```

Output:

Total number of packets sent by TCP: 46891

Total number of packets sent by UDP: 941

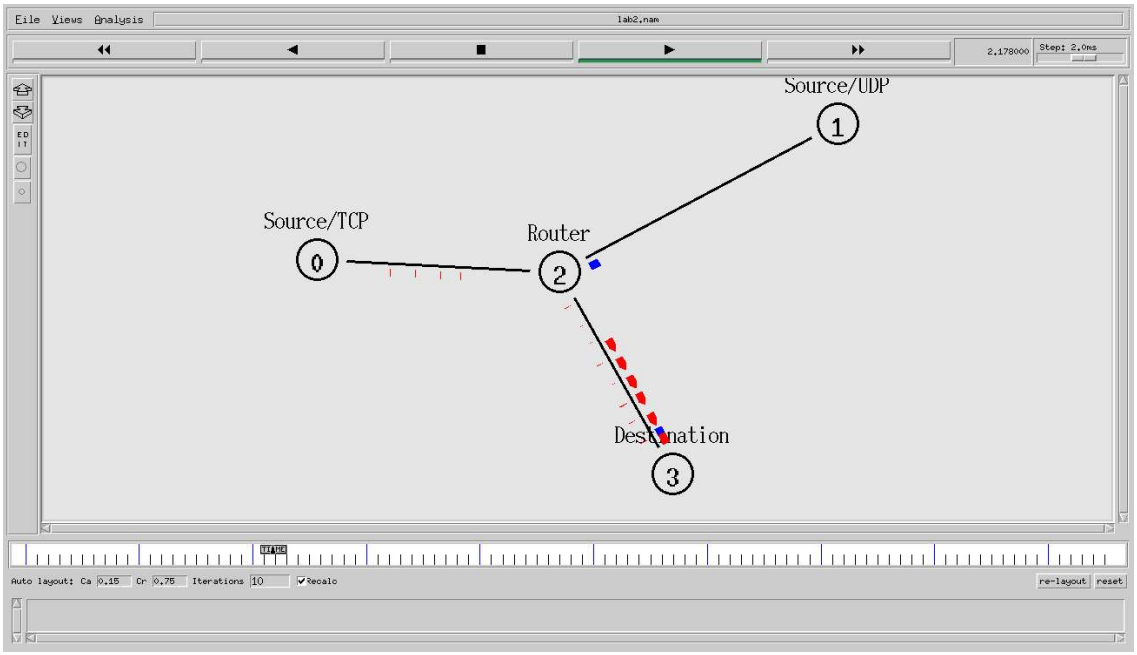


Table:

By changing the Bandwidth & Interval Size at source node,

Bandwidth	Packet Interval	TCP Packets Sent	UDP Packets Sent
100 Mb	0.01	46891	941
100 Mb	0.001	46887	9400
200 Mb	0.01	47380	941
200 Mb	0.001	47377	9400

Program – 3

Implement transmission of ping messages / trace route over a network topology consisting of 6 nodes and find the number of packets dropped due to congestion.

TCL Code: (P3Ping.tcl)

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

# Open the NS trace file & NAM trace file
set tf [open p3ping.tr w]
$ns trace-all $tf
set nf [open p3ping.nam w]
$ns namtrace-all $nf

# The below code is used to create the nodes.
set n0 [$ns node]
set n1 [$ns node]
set n2 [$ns node]
set n3 [$ns node]
set n4 [$ns node]
set n5 [$ns node]
set n6 [$ns node]

# The below code is used to give color and label the nodes.
$n0 label "Ping0"
$n2 label "Router"
$n4 label "Ping4"
$n5 label "Ping5"
$n6 label "Ping6"
$ns color 1 "red"
$ns color 2 "blue"

# Providing the link
$ns duplex-link $n0 $n2 100Mb 300ms DropTail
$ns duplex-link $n2 $n6 1Mb 300ms DropTail
$ns duplex-link $n5 $n2 100Mb 300ms DropTail
$ns duplex-link $n2 $n4 1Mb 300ms DropTail
$ns duplex-link $n3 $n2 1Mb 300ms DropTail
$ns duplex-link $n1 $n2 1Mb 300ms DropTail
```

```
# Set the queue size b/w the nodes
$ns queue-limit $n0 $n2 5
$ns queue-limit $n2 $n6 2
$ns queue-limit $n2 $n4 3
$ns queue-limit $n5 $n2 5

# Agent & Pings Creation
set ping0 [new Agent/Ping]
$ns attach-agent $n0 $ping0

set ping4 [new Agent/Ping]
$ns attach-agent $n4 $ping4

set ping5 [new Agent/Ping]
$ns attach-agent $n5 $ping5

set ping6 [new Agent/Ping]
$ns attach-agent $n6 $ping6

$ping0 set packetSize_ 50000
$ping0 set interval_ 0.0001

$ping5 set packetSize_ 50000
$ping5 set interval_ 0.0001

# To set color to packets
$ping0 set class_ 1
$ping5 set class_ 2

# The below code is used to connect the agents
$ns connect $ping0 $ping4
$ns connect $ping5 $ping6

# Creation of Ping Message & RTT
Agent/Ping instproc recv {from rtt} {
      $self instvar node_
puts "The node [$node_ id] received  a reply from $from with the round trip
time of $rtt"
}
```

```
# Communication
$ns at 0.1 "$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"

$ns at 1.1 "$ping0 send"
$ns at 1.2 "$ping0 send"
$ns at 1.3 "$ping0 send"
$ns at 1.4 "$ping0 send"
$ns at 1.5 "$ping0 send"
$ns at 1.6 "$ping0 send"
$ns at 1.7 "$ping0 send"
$ns at 1.8 "$ping0 send"
$ns at 1.9 "$ping0 send"

$ns at 0.1 "$ping5 send"
$ns at 0.2 "$ping5 send"
$ns at 0.3 "$ping5 send"
$ns at 0.4 "$ping5 send"
$ns at 0.5 "$ping5 send"
$ns at 0.6 "$ping5 send"
$ns at 0.7 "$ping5 send"
$ns at 0.8 "$ping5 send"
$ns at 0.9 "$ping5 send"

$ns at 1.1 "$ping5 send"
$ns at 1.2 "$ping5 send"
$ns at 1.3 "$ping5 send"
$ns at 1.4 "$ping5 send"
$ns at 1.5 "$ping5 send"
$ns at 1.6 "$ping5 send"
$ns at 1.7 "$ping5 send"
$ns at 1.8 "$ping5 send"
$ns at 1.9 "$ping5 send"
```

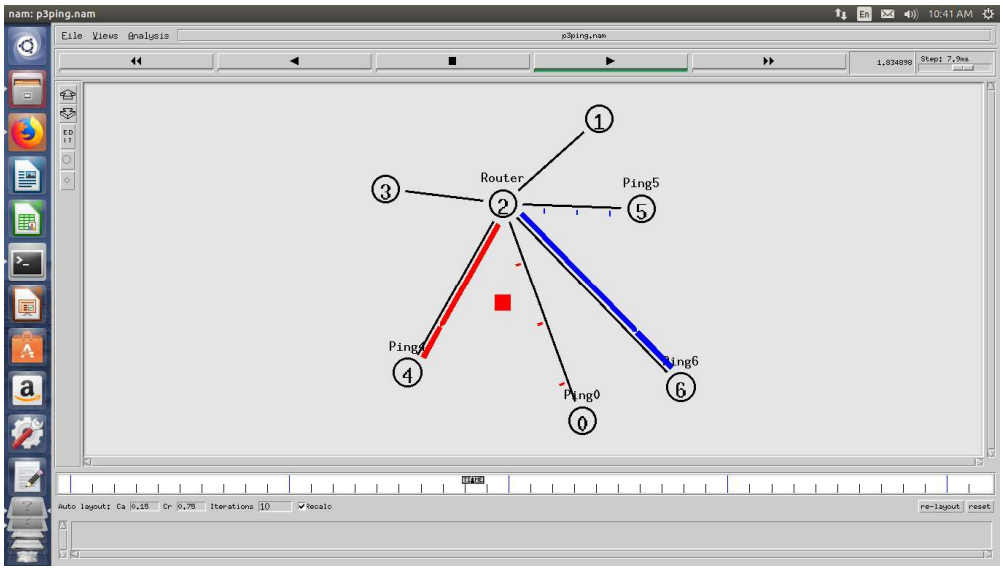
```
# Termination defining a ‘finish’ procedure
proc finish {} {
global ns nf tf
exec nam p3ping.nam &
$ns flush-trace
close $tf
close $nf
exit 0
}
$ns at 5.0 "finish"
$ns run
```

AWK Code: (P3ping.awk)

```
BEGIN{
count=0;
}
{
if($1=="d")
count++;
}
END{
printf("The total no of packets dropped due to congestion %d\n",count);
}
```

Output:

The total no of packets dropped due to congestion 25



Program – 4

Implement an Ethernet LAN using n nodes, set multiple traffic nodes, and determine collision across different nodes.

TCL Code: (P4.tcl)

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

# Open the NS trace file & NAM trace file
set tf [open p4.tr w]
$ns trace-all $tf
set nf [open p4.nam w]
$ns namtrace-all $nf

# The below code is used to create the nodes.
set n0 [$ns node]
set n1 [$ns node]
set n2 [$ns node]
set n3 [$ns node]
set n4 [$ns node]

# Create LAN
# Change the values below for the table
$ns make-lan -trace on "$n0 $n1 $n2 $n3 $n4" 100Mb 10ms LL
Queue/DropTail Mac/802_3

# The below code is used to give color the nodes.
$ns color 1 "red"
$ns color 2 "blue"
$ns color 3 "green"

# Agent & Application Definition of TCP Connection
set tcp0 [new Agent/TCP]
$ns attach-agent $n0 $tcp0
set ftp0 [new Application/FTP]
$ftp0 attach-agent $tcp0
set sink2 [new Agent/TCPSink]
$ns attach-agent $n2 $sink2
$ns connect $tcp0 $sink2 #Connecting the agents
```

```
# Agent & Application Definition of UDP Connection
set udp2 [new Agent/UDP]
$ns attach-agent $n2 $udp2
set cbr2 [new Application/Traffic/CBR]
$cbr2 attach-agent $udp2
set null1 [new Agent/Null]
$ns attach-agent $n1 $null1
$ns connect $udp2 $null1 #Connecting the agents

# To set color to packets
$udp2 set class_ 1
$tcp0 set class_ 2

# Agent & Application Definition of TCP Connection
set tcp1 [new Agent/TCP]
$ns attach-agent $n1 $tcp1
set ftp1 [new Application/FTP]
$ftp1 attach-agent $tcp1
set sink3 [new Agent/TCPSink]
$ns attach-agent $n3 $sink3
$ns connect $tcp1 $sink3 #Connecting the agents

# To set color to packets
$tcp1 set class_ 3

# The below code is used to set the interval size of packets
$ftp0 set interval_ 0.001
$cbr2 set interval_ 0.001
$ftp1 set interval_ 0.01

# Termination defining a ‘finish’ procedure
proc finish {} {
global ns nf tf
$ns flush-trace
exec nam p4.nam &
close $tf
close $nf
exit 0
}
```

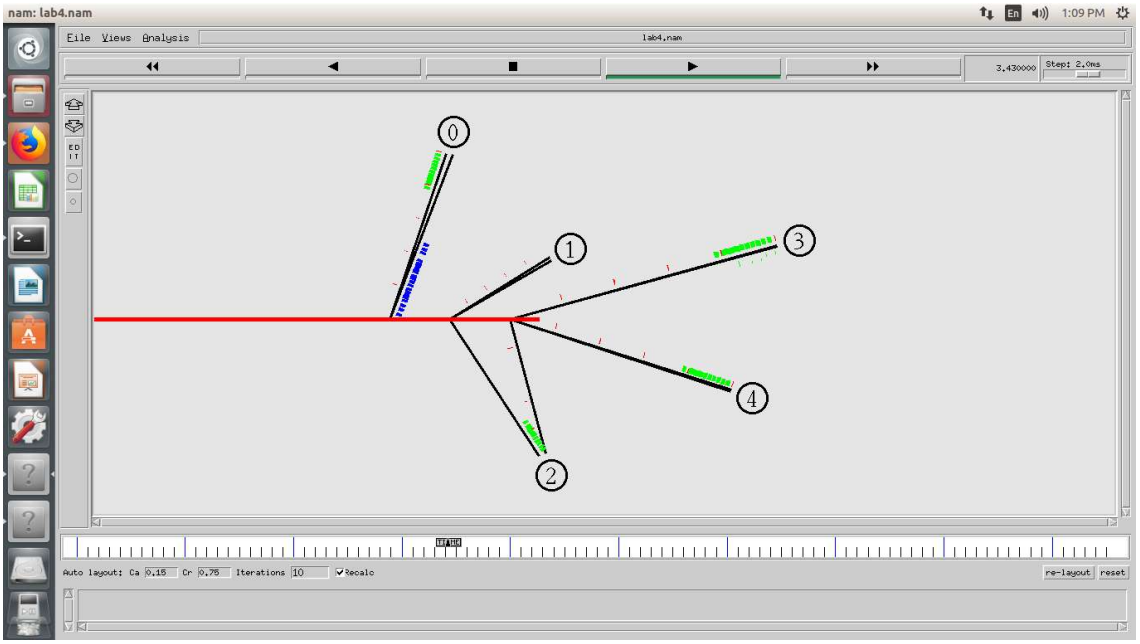
```
$ns at 0.1 "$cbr2 start"
$ns at 1.2 "$ftp1 start"
$ns at 1.3 "$ftp0 start"
$ns at 9.5 "$cbr2 stop"
$ns at 9.5 "$ftp1 stop"
$ns at 10.0 "finish"
$ns run
```

AWK Code: (P4.awk)

```
BEGIN{
count=0;
}
{
if($1=="c")
count++;
}
END{
printf("\n The total Packet Collision %d\n",count);
}
```

Output:

The total Packet Collision 1671



By changing the Bandwidth at LAN Creation,

Bandwidth	Link Delay	No. of Collisions
100 Mb	10 ms	1671
200 Mb	10 ms	4
300 Mb	10 ms	4

By changing the Link delay at LAN Creation,

Bandwidth	Link Delay	No. of Collisions
100 Mb	10 ms	1671
100 Mb	20 ms	947
100 Mb	30 ms	639

Program – 5

Implement simple ESS and with transmitting nodes in wireless LAN by simulation and determine the performance with respect to transmission of packets.

TCL Code: (P5.tcl)

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

# Open the NS trace file
set tf [open p5.tr w]
$ns trace-all $tf

# Setup topography object
set topo [new Topography]
$topo load_flatgrid 700 700

# Open the NAM trace file
set nf [open p5.nam w]
$ns namtrace-all-wireless $nf 700 700

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

# Create God object to manage movement patterns of mobile nodes
create-god 4
```

```
# The below code is used to create the nodes.
set n0 [$ns node]
set n1 [$ns node]
set n2 [$ns node]
set n3 [$ns node]

# The below code is used to label the nodes.
$n0 label "tcp0"
$n1 label "sink1"
$n2 label "tcp1"
$n3 label "sink2"

# Set initial node positions
$n0 set X_ 50
$n0 set Y_ 50
$n0 set Z_ 0
$n1 set X_ 200
$n1 set Y_ 200
$n1 set Z_ 0
$n2 set X_ 400
$n2 set Y_ 400
$n2 set Z_ 0
$n3 set X_ 600
$n3 set Y_ 600
$n3 set Z_ 0

# Set node destinations
$ns at 0.1 "$n0 setdest 50 50 15"
$ns at 0.1 "$n1 setdest 200 200 25"
$ns at 0.1 "$n2 setdest 400 400 25"
$ns at 0.1 "$n3 setdest 600 600 25"

# TCP connections
set tcp0 [new Agent/TCP]
$ns attach-agent $n0 $tcp0
set ftp0 [new Application/FTP]
$ftp0 attach-agent $tcp0
set sink1 [new Agent/TCPSink]
$ns attach-agent $n1 $sink1
$ns connect $tcp0 $sink1 #Connecting the agents
```

```
set tcp1 [new Agent/TCP]
$ns attach-agent $n2 $tcp1
set ftp1 [new Application/FTP]
$ftp1 attach-agent $tcp1
set sink2 [new Agent/TCPSink]
$ns attach-agent $n3 $sink2
$ns connect $tcp1 $sink2 #Connecting the agents

# Start FTP traffic
$ns at 5 "$ftp0 start"
$ns at 10 "$ftp1 start"

# Node movement
$ns at 100 "$n2 setdest 500 500 25"

# Finish procedure to close files and open NAM
proc finish {} {
   global nf ns tf
   $ns flush-trace
   close $tf
   close $nf
   exec nam p5.nam &
   exit 0
}

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

AWK Code: (P5.awk)

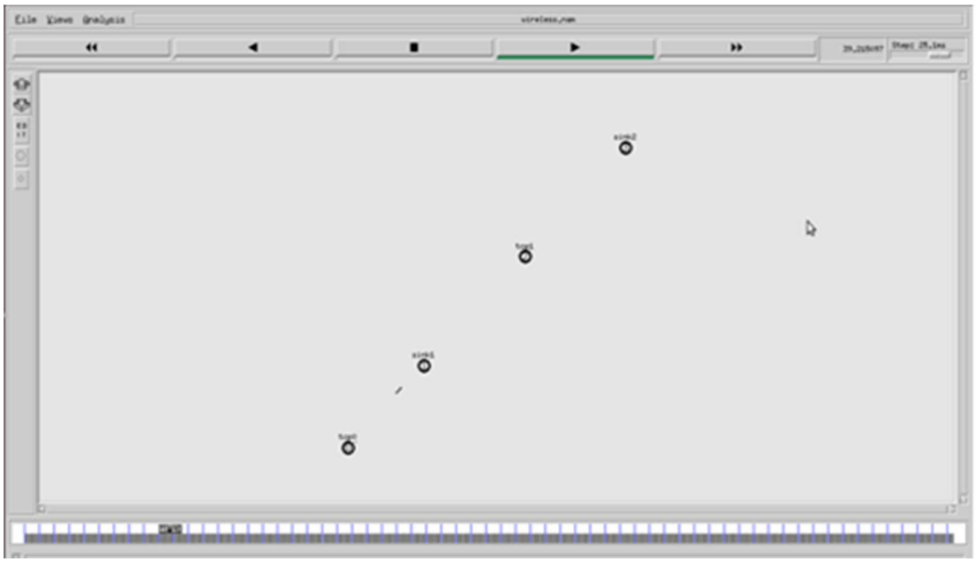
```
BEGIN{
count1=0
count2=0
pack1=0
pack2=0
time1=0
time2=0
}
{
if($1=="r" && $3=="_1_" && $4=="AGT")
```

```
{
count1++
pack1=pack1+$8
time1=$2
}
if($1=="r" && $3=="_3_" && $4=="AGT")
{
count2++
pack2=pack2+$8
time2=$2
}
}
END{
printf(" The Throughput  from n0 to n1:
%fMbps\n",((count1*pack1*8)/(time1*1000000)))
printf(" The Throughput  from n2 to n3:
%fMbps\n",((count2*pack2*8)/(time2*1000000)))
}
```

Output:

The Throughput from n0 to n1: 3452.697785 Mbps

The Throughput from n2 to n3: 4397.096913 Mbps



Program – 6

Design, develop & execute a program to implement error detection using the Cyclic Redundancy Check – Consultative Committee for International Telegraphy and Telephony (CRC-CCITT) – (16-bit) algorithm.

C Code: (P6.c)

```
#include<stdio.h>
#include<string.h>
#define N strlen(g)

char t[128],cs[128],g[]="10001000000100001";
int a,e,c;
void xor()
{
   for(c=1;c<N;c++)
      cs[c]=((cs[c]==g[c])?'0':'1');
}
void crc()
{
   for(e=0;e<N;e++)
      cs[e]=t[e];
   do
   {
      if(cs[0]=='1')
         xor();
      for(c=0;c<N-1;c++)
         cs[c]=cs[c+1];
      cs[c]=t[e++];
   }while(e<=a+N-1);
}
int main()
{
   printf("\nEnter poly : ");
   scanf("%s",t);
   printf("\nGenerating Polynomial is : %s",g);
   a=strlen(t);
   for(e=a;e<a+N-1;e++) t[e]='0';
      printf("\nModified t[u] is :  %s",t);
   crc();
```

```
    printf("\nChecksum is : %s",cs);
    for (e=a;e<a+N-1;e++) t[e]=cs[e-a];
       printf("\nFinalCodeword is : %s",t);
    printf("\nTest Error detection 0(yes) 1(no) ? : ");
    scanf("%d",&e);
    if (e==0)
    {
       printf("Enter position where error is to inserted : ");
       scanf("%d",&e);
       t[e]=(t[e]=='0')?'1':'0';
       printf("Errorneous data: %s\n",t);
    }
    crc();
    for(e=0;(e<N-1)&&(cs[e]!='1');e++);
       if(e<N-1)
          printf("Error detected.");
       else
          printf("No Error Detected.");
    return 0;
}
```

Output:

```
Enter poly: 1011101
Generating Polynomial is: 10001000000100001
Modified t[u] is:  10111010000000000000000
Checksum is: 1000101101011000
FinalCodeword is: 10111011000101101011000
Test Error detection 0(yes) 1(no) ? : 0
Enter position where error is to inserted: 3
Errorneous data: 10101011000101101011000
Error detected.


Enter poly: 1011101
Generating Polynomial is: 10001000000100001
Modified t[u] is:  10111010000000000000000
Checksum is: 1000101101011000
FinalCodeword is: 10111011000101101011000
Test Error detection 0(yes) 1(no) ? : 1
No Error Detected.
```

Program – 7

Write a program for distance vector algorithm to find suitable path for transmission.

C Code: (P7.c)

```
#include <stdio.h>
#include <stdlib.h>
void rout_table();
int d[10][10], via[10][10];
int i, j, k, l, m, n, g[10][10], temp[10][10], ch, cost;
int main()
{
   system("clear");
   printf("Enter the value of number of nodes: ");
   scanf("%d", &n);
   rout_table();
   for (i = 0; i < n; i++)
      for (j = 0; j < n; j++)
         temp[i][j] = g[i][j];
   for (i = 0; i < n; i++)
      for (j = 0; j < n; j++)
         via[i][j] = i;
   while (1)
   {
      for (i = 0; i < n; i++)
      {
         for (j = 0; j < n; j++)
         {
            if (d[i][j])
               for (k = 0; k < n; k++)
               {
                  if (g[i][j] + g[j][k] < g[i][k])
                  {
                     g[i][k] = g[i][j] + g[j][k];
                     via[i][k] = j;
                  }
               }
         }
      }
```

```
        for (i = 0; i < n; i++)
        {
            printf("Table for router %c:\n", i + 97);
            for (j = 0; j < n; j++)
                printf("%c :: %d via %c\n",j + 97, g[i][j], via[i][j] + 97);
        }
        break;
    }
}
void rout_table()
{
    printf("\nEnter the routing table:\n\t");
    for (i = 1; i <= n; i++)
        printf("%c\t", i + 96);
    printf("\n");
    for (i = 0; i < n; i++)
    {
        printf("%c ", i + 97);
        for (j = 0; j < n; j++)
        {
            scanf("%d", &g[i][j]);
            if (g[i][j] != 999)
                d[i][j] = 1;
        }
    }
}
```

Output:

```
Enter the value of number of nodes: 4
Enter the routing table:
        a       b       c       d
a 0 6 7 999
b 6 0 999 3
c 7 999 0 5
d 999 3 5 0
Table for router a:
a :: 0 via a
b :: 6 via a
c :: 7 via a
d :: 9 via b
```

Table for router b:
a :: 6 via b
b :: 0 via b
c :: 8 via d
d :: 3 via b
Table for router c:
a :: 7 via c
b :: 8 via d
c :: 0 via c
d :: 5 via c
Table for router d:
a :: 9 via b
b :: 3 via d
c :: 5 via d
d :: 0 via d

Program – 8

From a given vertex in a weighted connected graph, find shortest paths to other vertices using Link state algorithm.

C Code: (P8.c)

```
#include<stdio.h>
void dij(int n,int cost[10][10],int source,int dest,int d[],int p[])
{
   int i,j,u,v,min,s[10];
   for(i=0;i<n;i++)
   {
      d[i]=cost[source][i];
      s[i]=0;
      p[i]=source;
   }
   s[source]=1;
   for(i=1;i<n;i++)
   {
      min=999;
      u=-1;
      for(j=0;j<n;j++)
      {
         if(s[j]==0)
         {
            if(d[j]<min)
            {
               min=d[j];
               u=j;
            }
         }
      }
      if(u==-1)
         return;
      s[u]=1;
      if(u==dest)
         return;
      for(v=0;v<n;v++)
      {
         if(s[v]==0)
```

```
            {
               if(d[u]+cost[u][v]<d[v])
               {
                  d[v]=d[u]+cost[u][v];
                  p[v]=u;
               }
            }
         }
      }
}
void print_path(int source,int destination,int d[],int p[])
{
   int i;
   i=destination;
   while(i!=source)
   {
      printf("%d<-",i);
      i=p[i];
   }
   printf("%d=%d\n",i,d[destination]);
}
int main()
{
   int cost[10][10],n,d[10],p[10],i,j;
   printf("Enter the number of nodes in the network\n");
   scanf("%d",&n);
   printf("Enter the cost n between every nodes\n");
   for(i=0;i<n;i++)
      for(j=0;j<n;j++)
         scanf("%d",&cost[i][j]);
   printf("enter the source node\n");
   scanf("%d",&i);
   for(j=0;j<n;j++)
   {
      dij(n,cost,i,j,d,p);
      if(d[j]==999)
         printf("%d is not reachable from %d\n",j,i);
      else if(i!=j)
         print_path(i,j,d,p);
   }
```

```
    return 0;
}
```

Output:

```
Enter the number of nodes in the network
4
Enter the cost n between every nodes
0 6 7 999
6 0 999 3
7 999 0 5
999 3 5 0
enter the source node
1
0<-1=6
2<-3<-1=8
3<-1=3
```

Program – 9

Write a program for congestion control using leaky bucket algorithm.

C Code: (P9.c)

```
#include<stdio.h>
#include<strings.h>
int min(int x,int y)
{
   if(x<y)
      return x;
   else
      return y;
}
int main()
{
   int drop=0,mini,nsec,cap,count=0,i,inp[25],process;
   system("clear");
   printf("Enter The Bucket Size\n");
   scanf("%d",&cap);
   printf("Enter The Operation Rate\n");
   scanf("%d",&process);
   printf("Enter The No. Of Seconds You Want To Stimulate\n");
   scanf("%d",&nsec);
   for(i=0;i<nsec;i++)
   {
      printf("Enter The Size Of The Packet Entering At %d sec\n",i+1);
      scanf("%d",&inp[i]);
   }
   printf("\nSecond|Packet Recieved|Packet Sent|Packet Left|Packet
Dropped|\n");
   printf("-------------------------------------------------------------------------\n");
   for(i=0;i<nsec;i++)
   {
      count+=inp[i];
      if(count>cap)
      {
         drop=count-cap;
         count=cap;
      }
```

```
        printf("%d",i+1);
        printf("\t%d",inp[i]);
        mini=min(count,process);
        printf("\t\t%d",mini);
        count=count-mini;
        printf("\t\t%d",count);
        printf("\t\t%d\n",drop);
        drop=0;
    }
    for(;count!=0;i++)
    {
        if(count>cap)
        {
            drop=count-cap;
            count=cap;
        }
        printf("%d",i+1);
        printf("\t0");
        mini=min(count,process);
        printf("\t\t%d",mini);
        count=count-mini;
        printf("\t\t%d",count);
        printf("\t\t%d\n",drop);
    }
}
```

Output:

```
Enter The Bucket Size
5
Enter The Operation Rate
2
Enter The No. Of Seconds You Want To Stimulate
3
Enter The Size Of The Packet Entering At 1 sec
5
Enter The Size Of The Packet Entering At 2 sec
4
Enter The Size Of The Packet Entering At 3 sec
3
```

Second|Packet Recieved|Packet Sent|Packet Left|Packet Dropped|

---

1 5 2 3 0

2 4 2 3 2

3 3 2 3 1

4 0 2 1 0

5 0 1 0 0

Program – 10

Using TCP/IP sockets, write a client-server program to make the client send the file name and to make the sever send back the contents of the requested file if present.

C Code:

File-1 (server.c)

```
#include<stdio.h>
#include<stdlib.h>
#include<unistd.h>
#include<errno.h>
#include<string.h>
#include<sys/types.h>
#include<sys/stat.h>
#include<fcntl.h>
#include<netinet/in.h>
#include<sys/socket.h>
#include<arpa/inet.h>
#include<sys/wait.h>
#include<signal.h>
#define MYPORT 6490
#define BACKLOG 10

int main(void)
{
   int sockfd, fp, new_fd;
   struct sockaddr_in my_addr, their_addr;
   int sin_size;
   int yes = 1;
   char buf1[40], buf2[20000];
   // Create socket
   if ((sockfd = socket(AF_INET, SOCK_STREAM, 0)) == -1) {
      perror("socket");
      exit(1);
   }

   // Set socket options
```

```
    if (setsockopt(sockfd, SOL_SOCKET, SO_REUSEADDR, &yes, sizeof(int))
== -1) {
        perror("setsockopt");
        exit(1);
    }

    // Set up the address structure
    my_addr.sin_family = AF_INET;
    my_addr.sin_port = htons(MYPORT);
    my_addr.sin_addr.s_addr = INADDR_ANY;
    memset(&(my_addr.sin_zero), '\0', 8);

    // Bind the socket to the address
    if (bind(sockfd, (struct sockaddr *)&my_addr, sizeof(struct sockaddr)) == -1)
{
        perror("bind");
        exit(1);
    }

    // Listen for connections
    if (listen(sockfd, BACKLOG) == -1) {
        perror("listen");
        exit(1);
    }

    printf("\nSERVER is online!\nSERVER: Waiting for the client...\n");
    sin_size = sizeof(struct sockaddr_in);

    // Accept a connection
    if ((new_fd = accept(sockfd, (struct sockaddr *)&their_addr, &sin_size)) == -
1) {
        perror("accept");
        exit(1);
    }

    printf("\nSERVER: Got connection from %s\n",
inet_ntoa(their_addr.sin_addr));

    // Receive the file name from the client
    if (recv(new_fd, buf1, sizeof(buf1) - 1, 0) <= 0) {
```

```
        perror("recv");
        close(new_fd);
        close(sockfd);
        exit(1);
    }
    buf1[sizeof(buf1) - 1] = '\0'; // Ensure null-termination
    printf("SERVER: File requested: '%s'\n", buf1);

    // Open the file
    if ((fp = open(buf1, O_RDONLY)) < 0) {
        printf("File not found\n");
        strcpy(buf2, "File not found");
    } else {
        printf("SERVER: '%s' found and ready to transfer.\n", buf1);
        read(fp, buf2, sizeof(buf2));
        close(fp);
    }

    // Send file contents to the client
    send(new_fd, buf2, sizeof(buf2), 0);
    close(new_fd);
    close(sockfd);
    printf("Transfer successful\n");
    return 0;
}
```

File – 2 (client.c)

```
#include<stdio.h>
#include<stdlib.h>
#include<unistd.h>
#include<errno.h>
#include<string.h>
#include<netdb.h>
#include<sys/types.h>
#include<netinet/in.h>
#include<sys/socket.h>
#include<fcntl.h>

#define PORT 6490
```

```
int main()
{
    int sockfd;
    char buf1[40], buf2[20000];
    struct sockaddr_in their_addr;

    // Create socket
    if ((sockfd = socket(AF_INET, SOCK_STREAM, 0)) == -1) {
        perror("socket");
        exit(1);
    }

    // Set up the address structure
    their_addr.sin_family = AF_INET;
    their_addr.sin_port = htons(PORT);
    their_addr.sin_addr.s_addr = inet_addr("127.0.0.1");
    memset(&(their_addr.sin_zero), '\0', 8);

    // Connect to the server
    if (connect(sockfd, (struct sockaddr *)&their_addr, sizeof(struct sockaddr)) ==
-1) {
        perror("connect");
        exit(1);
    }

    printf("CLIENT is online!\n");
    printf("CLIENT: Enter the filename to be displayed: ");

    // Get file name from user input
    if (fgets(buf1, sizeof(buf1), stdin) == NULL) {
        perror("fgets");
        close(sockfd);
        exit(1);
    }
    buf1[strcspn(buf1, "\n")] = '\0'; // Remove newline character
    printf("CLIENT: Sending file name '%s'\n", buf1);

    // Send the file name to the server
    if (send(sockfd, buf1, strlen(buf1) + 1, 0) == -1) {
        perror("send");
```

```
        close(sockfd);
        exit(1);
    }

    // Receive file content from the server
    if (recv(sockfd, buf2, sizeof(buf2), 0) == -1) {
        perror("recv");
        close(sockfd);
        exit(1);
    }

    // Display the content of the file or error message
    printf("Displaying the contents of '%s':\n", buf1);
    printf("\n%s\n", buf2);

    // Close the socket
    close(sockfd);
    return 0;
}
```

Output:

Server-Side Output –

```
SERVER is online!
SERVER: Waiting for the client.....
SERVER: Got connection from
File requested
SERVER:  file found and ready to transfer
Transfer Success
```

Client-Side Output –

```
CLIENT is online!
CLIENT: Enter the filename to be displayed:
file
Displaying the contents of file

The contents of file are:-
This a demo of client server using Sockets
Just for trial.
Now End of file
```