PRG 1:-write a c program distance vector algorithm to find suitable path for transmission

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

struct Node {
   int cost[10];
   int nextHop[10];
};

void dist(int n, struct Node nodes[]) {
   int updated;

   do {
      updated = 0;

      for (int i = 0; i < n; i++) {
         for (int j = 0; j < n; j++) {
            if (i != j) {
               for (int k = 0; k < n; k++) {
                  if (k != i && nodes[i].cost[k] != INT_MAX && nodes[k].cost[j] != INT_MAX) {
                     int newCost = nodes[i].cost[k] + nodes[k].cost[j];
                     if (newCost < nodes[i].cost[j]) {
                        nodes[i].cost[j] = newCost;
                        nodes[i].nextHop[j] = k;
                        updated = 1;
                     }
                  }
               }
            }
         }
      }
   } while (updated);
}

int main() {
   int n;
   printf("Enter the number of nodes: ");
   scanf("%d", &n);

   struct Node nodes[n];

   printf("Enter the cost matrix for each node:\n");
```

```
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            scanf("%d", &nodes[i].cost[j]);
            if (i != j && nodes[i].cost[j] == 0) {
                nodes[i].cost[j] = INT_MAX;
            }
            nodes[i].nextHop[j] = j;
        }
    }

    dist(n, nodes);

    for (int i = 0; i < n; i++) {
        printf("\nRouting table for node %c:\n", i + 65);
        printf("----------------------------------------\n");
        printf("Dest\tCost\tNext Hop\n");
        printf("----------------------------------------\n");
        for (int j = 0; j < n; j++) {
            if (i != j) {
                printf("%c\t", j + 65);
                if (nodes[i].cost[j] == INT_MAX) {
                    printf("INF\t");
                } else {
                    printf("%d\t", nodes[i].cost[j]);
                }
                printf("%c\n", nodes[i].nextHop[j] + 65);
            }
        }
    }

    return 0;
}
```

PRG 1 - OUTPUT:-

```
student@fedora:~/cn$ ./a.out
Enter the number of nodes: 3
Enter the cost matrix for each node:
0 2 3
2 0 7
3 7 0

Routing table for node A:
----------------------------------------
Dest    Cost    Next Hop
----------------------------------------
B       2       B
C       3       C

Routing table for node B:
----------------------------------------
Dest    Cost    Next Hop
----------------------------------------
A       2       A
C       5       A

Routing table for node C:
----------------------------------------
Dest    Cost    Next Hop
----------------------------------------
A       3       A
B       5       A
student@fedora:~/cn$
```

PRG 3:-Write a program for Hamming code generation for error detection and correction.

```
#include <stdio.h>
#include <math.h>
void calculate_parity_bits(int data[], int hamming_code[]) {
    hamming_code[0] = data[0] ^ data[1] ^ data[3];
    hamming_code[1] = data[0] ^ data[2] ^ data[3];
    hamming_code[3] = data[1] ^ data[2] ^ data[3];
}
void generate_hamming_code(int data[]) {
    int hamming_code[7] = {0};
    hamming_code[2] = data[0]; // D1 -> position 3 (index 2)
    hamming_code[4] = data[1]; // D2 -> position 5 (index 4)
    hamming_code[5] = data[2]; // D3 -> position 6 (index 5)
    hamming_code[6] = data[3]; // D4 -> position 7 (index 6)
    calculate_parity_bits(data, hamming_code);
    printf("Generated Hamming Code : ");
    for (int i = 0; i < 7; i++) {
        printf("%d", hamming_code[i]);
    }
    printf("\n");
}
void check_and_correct_hamming_code(int received[]) {
    int parity_check[3] = {0};
    parity_check[0] = received[0] ^ received[2] ^ received[4] ^ received[6]; // P1 check
    parity_check[1] = received[1] ^ received[2] ^ received[5] ^ received[6]; // P2 check
    parity_check[2] = received[3] ^ received[4] ^ received[5] ^ received[6]; // P3 check
    int error_position = parity_check[0] * 1 + parity_check[1] * 2 + parity_check[2] * 4;
    if (error_position == 0) {
        printf("No error detected.\n");
    } else {
        printf("Error detected at position: %d\n", error_position);
        printf("Corrected Hamming Code: ");
        received[error_position - 1] = !received[error_position - 1]; // Flip the incorrect bit
        for (int i = 0; i < 7; i++) {
            printf("%d", received[i]);
        }
        printf("\n");
    }
}
int main() {
    int data[4];
    // Input 4 data bits
    printf("Enter 4 data bits : ");
```

```
    for (int i = 0; i < 4; i++) {
        scanf("%d", &data[i]);
    }
    generate_hamming_code(data);
    int received[7];
    printf("Enter the received 7-bit Hamming code : ");
    for (int i = 0; i < 7; i++) {
        scanf("%d", &received[i]);
    }
    check_and_correct_hamming_code(received);
    return 0;
}
```

OUTPUT:-

```
student@fedora:~/c$ gcc prg2.c
student@fedora:~/c$ ./a.out
Enter 4 data bits : 1 0 1 1
Generated Hamming Code : 0110011
Enter the received 7-bit Hamming code : 0 1 1 0 0 0 1
Error detected at position: 6
Corrected Hamming Code: 0110011
student@fedora:~/c$ ./a.out
Enter 4 data bits : 1 0 1 1
Generated Hamming Code : 0110011
Enter the received 7-bit Hamming code : 0 1 1 0 0 1 1
No error detected.
student@fedora:~/c$
```

PRG4:-Write a program for congestion control using leaky bucket algorithm.

```
#include <stdio.h>
int min(int x, int y)
{
   return (x < y) ? x : y;
}
int main()
 {

   int drop = 0, mini, nsec, cap, count = 0;
   int i, inp[25], process;
   printf("Enter the Bucket Size:\n");
   scanf("%d", &cap);
   printf("Enter the Processing Rate:\n");
   scanf("%d", &process);
   printf("Enter the number of seconds for the simulation:\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 Received | Packet Sent | Packet Left | Packet Dropped\n");
   printf("-------------------------------------------------------------------\n");
   for (i = 0; i < nsec || count > 0; i++) {
      int received = (i < nsec) ? inp[i] : 0;
      // Packets received at this second
      count += received; // Add packets to the bucket
      // Check for overflow
      if (count > cap) {
         drop = count - cap; // Calculate dropped packets
         count = cap;       // Limit bucket to its capacity
      }
      // Determine packets sent
      mini = min(count, process);
      count -= mini; // Reduce the bucket count
      // Print results for the current second
      printf("%-7d| %-16d| %-12d| %-12d| %-14d\n", i + 1, received, mini, count, drop);
      drop = 0; // Reset dropped packets
   }
   return 0;
}
```

OUTPUT:-

```
student@fedora:~$ cd c
student@fedora:~/c$ gcc prg3.c
student@fedora:~/c$ ./a.out
Enter the Bucket Size:
5
Enter the Processing Rate:
2
Enter the number of seconds for the simulation:
3
Enter the Size of the Packet Entering at 1 sec:
4
Enter the Size of the Packet Entering at 2 sec:
5
Enter the Size of the Packet Entering at 3 sec:
6

Second | Packet Received | Packet Sent | Packet Left | Packet Dropped
-------------------------------------------------------------------
1      | 4               | 2           | 2           | 0
2      | 5               | 2           | 3           | 2
3      | 6               | 2           | 3           | 4
4      | 0               | 2           | 1           | 0
5      | 0               | 1           | 0           | 0
```

PART-B

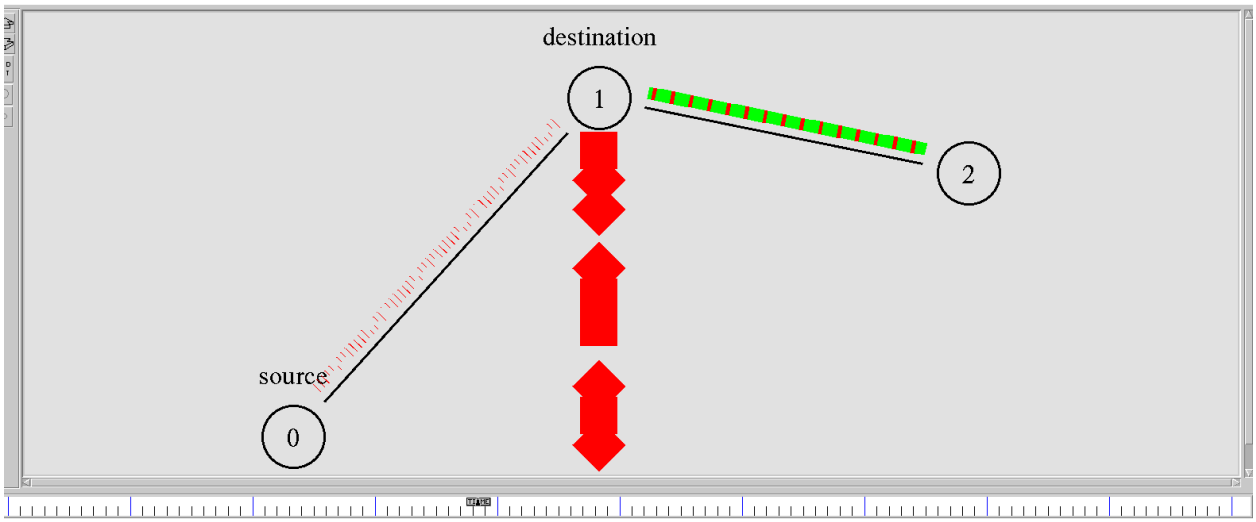
1. Simulate a three nodes point to point network with duplex links between them. Set the queue size and vary the bandwidth and find the number of packets dropped.

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

# Open the trace file for writing
set tf [open prgb1.tr w]

# Set trace-all to the opened file
$ns trace-all $tf

# Open a NAM trace file for visualizations
set nm [open prgb1.nam w]
$ns namtrace-all $nm

# Create nodes
set n0 [$ns node]
set n1 [$ns node]
set n2 [$ns node]

# Color the nodes
$ns color 1 "red"
$ns color 2 "green"

# Label the nodes
$n0 label "source"
$n1 label "destination"

# Create links and set bandwidth to vary and check for packet drops
$ns duplex-link $n0 $n1 10Mb 300ms DropTail
$ns duplex-link $n1 $n2 1Mb 300ms DropTail

# Limiting the queue size to 10 packets
$ns set queue-limit $n0 $n1 10
$ns set queue-limit $n1 $n2 5

# Setup a UDP connection
set udp0 [new Agent/UDP]
$ns attach-agent $n0 $udp0
```

```
# Create a CBR traffic source and attach it to udp0
set cbr1 [new Application/Traffic/CBR]
$cbr1 set packetSize_ 500
$cbr1 set interval_ 0.005
$cbr1 attach-agent $udp0

# Create a UDP agent and attach it to node n1
set udp1 [new Agent/UDP]
$ns attach-agent $n1 $udp1

# Create a CBR traffic source and attach it to udp1
set cbr2 [new Application/Traffic/CBR]
$cbr2 set packetSize_ 500
$cbr2 set interval_ 0.005
$cbr2 attach-agent $udp1

# Create a null agent (traffic sink) and attach it to node n2
set null1 [new Agent/Null]
$ns attach-agent $n2 $null1

# Set UDP packets to red and blue color
$udp0 set class_ 1
$udp1 set class_ 2

# Connect traffic sources with the traffic sink
$ns connect $udp0 $null1
$ns connect $udp1 $null1

# Define a 'finish' procedure
proc finish {} {
   global ns tf nm
   exec nam prgb1.nam &
   $ns flush-trace
   close $tf
   close $nm
   exit 0
}

# Schedule events for the CBR agents
$ns at 0.1 "$cbr1 start"
$ns at 0.1 "$cbr2 start"

# Call the finish procedure after 10 seconds
$ns at 10.0 "finish"
```

 # Run the simulation
$ns run

OUTPUT:-



//Dup.awk

```
BEGIN {
count=0;
}
{
if($1=="d")
count++;
}
END {
printf("the total number of packets dropped due to congestion:%d\n\n",count);
}
```

OUTPUT:-

```
student@fedora:~/c$ awk -f Dup.awk prgb1.tr
the total number of packets dropped due to congestion:1392
```

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.

```
set ns [new Simulator]

set nf [open star.nam w]

$ns namtrace-all $nf


set tf [open star.tr w]

$ns trace-all $tf


set n0 [$ns node]

set n1 [$ns node]

set n2 [$ns node]

set n3 [$ns node]

set n4 [$ns node]


$n0 label "TCP source"

$n3 label "TCP Destination"

$n1 label "UDP Source"

$n2 label "UDP Destination"


$n0 shape square

$n3 shape square

$n4 shape circle
```

```
$n1 shape hexagon

$n2 shape hexagon


$n0 color "green"

$n3 color "red"

$n1 color "green"

$n2 color "red"

$n4 color "black"

$ns color 1 "red"

$ns color 2 "blue"


$ns duplex-link $n0 $n4 1Mb 10ms DropTail

$ns duplex-link $n1 $n4 1Mb 10ms DropTail

$ns duplex-link $n2 $n4 1Mb 10ms DropTail

$ns duplex-link $n3 $n4 1Mb 10ms DropTail


set tcp0 [new Agent/TCP]

$ns attach-agent $n0 $tcp0

set sink0 [new Agent/TCPSink]

$ns attach-agent $n3 $sink0


set udp0 [new Agent/UDP]

$ns attach-agent $n1 $udp0


set null0 [new Agent/Null]
```

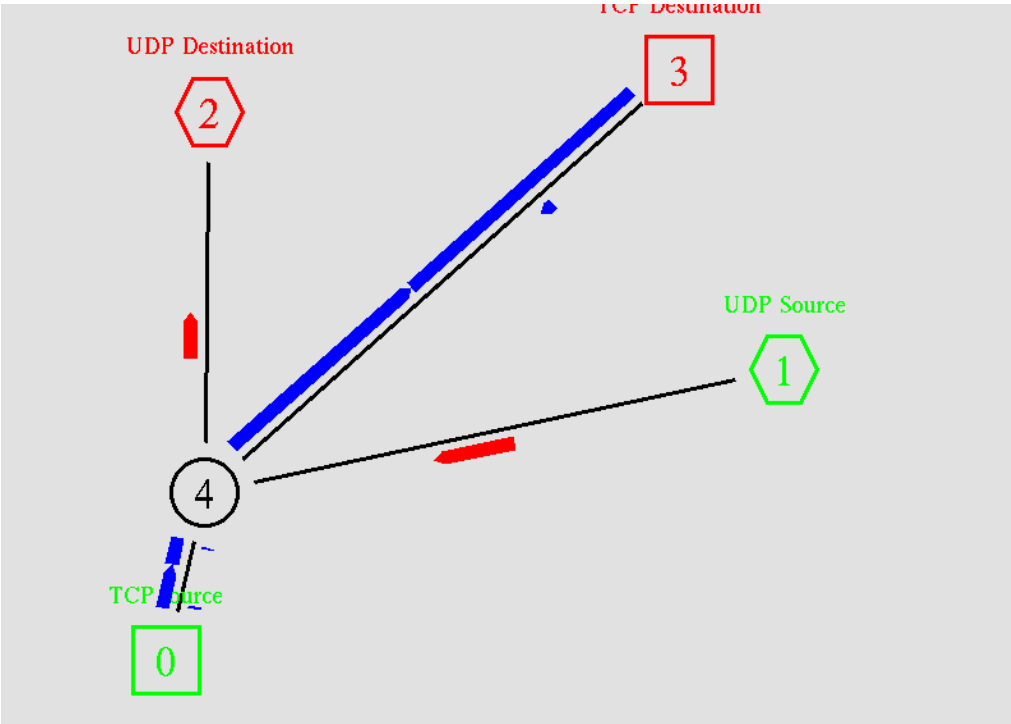
```
$ns attach-agent $n2 $null0

$udp0 set class_ 1

$tcp0 set class_ 2

$ns connect $tcp0 $sink0

$ns connect $udp0 $null0

set cbr0 [new Application/Traffic/CBR]

$cbr0 attach-agent $udp0

$cbr0 set PacketSize_ 500

$cbr0 set interval_ 0.01

set ftp0 [new Application/FTP]

$ftp0 attach-agent $tcp0

$ns at 0.5 "$ftp0 start"

$ns at 9.5 "$ftp0 stop"

$ns at 1.0 "$cbr0 start"

$ns at 9.9 "$cbr0 stop"

proc finish {} {

global ns nf tf

close $tf

close $nf
```

exec nam star.nam & }

$ns at 10.0 "finish"

$ns run

OUTPUT:-



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

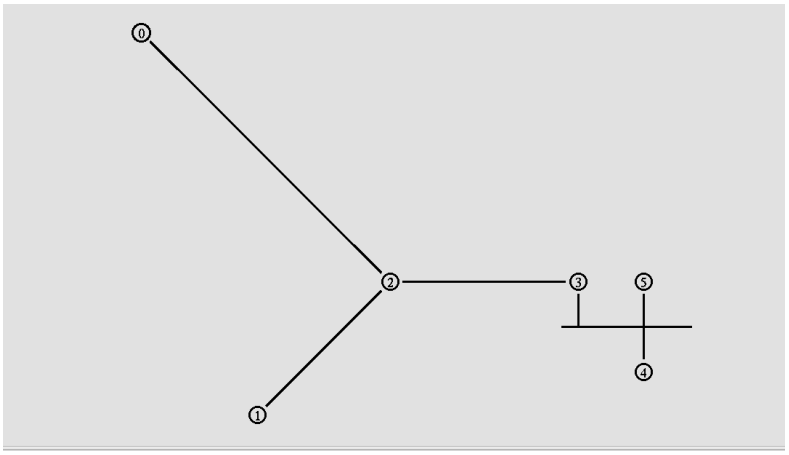
```
set ns [ new Simulator ]
set tr1 [ open b2.tr w ]
$ns trace-all $tr1
set nm1 [open b2.nam w ]
$ns namtrace-all $nm1

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 color 1 "blue"
$ns color 2 "red"
$ns duplex-link $n0 $n2 1Mb 200ms DropTail
$ns duplex-link $n1 $n2 1Mb 100ms DropTail
$ns duplex-link $n2 $n3 0.6Mb 100ms DropTail
$ns duplex-link-op $n0 $n2 orient right-down
$ns duplex-link-op $n1 $n2 orient right-up
$ns duplex-link-op $n2 $n3 orient right

set lan [$ns newLan "$n3 $n4 $n5" 0.6Mb 40ms LL Queue/DropTail Mac/802_3 Channel]
$ns set queue-limit $n2 $n3 10
$ns duplex-link-op $n2 $n3 queuePos 10.0

set tcp1 [new Agent/TCP]
$ns attach-agent $n0 $tcp1
set sink1 [new Agent/TCPSink]
$ns attach-agent $n4 $sink1

set ftp1 [new Application/FTP]
$ftp1 attach-agent $tcp1
$ftp1 set packetSize_ 500
$ftp1 set interval_ 0.005
$ns connect $tcp1 $sink1

set tcp2 [new Agent/TCP]
$ns attach-agent $n1 $tcp2
set sink2 [new Agent/TCPSink]
$ns attach-agent $n5 $sink2
```

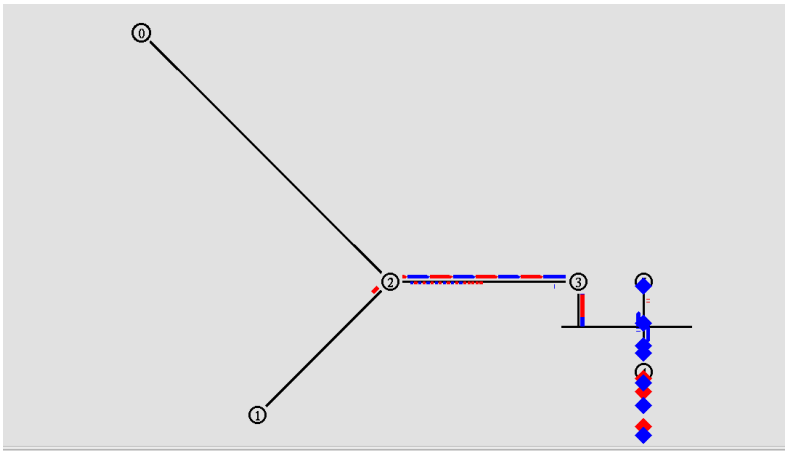
```
set ftp2 [new Application/FTP]
$ftp2 attach-agent $tcp2
$ftp2 set packetSize_ 500
$ftp2 set interval_ 0.001
$ns connect $tcp2 $sink2

$tcp1 set class_ 1
$tcp2 set class_ 2
$ns at 0.5 "$ftp1 start"
$ns at 4.9 "$ftp1 stop"
$ns at 1.0 "$ftp2 start"
$ns at 4.9 "$ftp2 stop"

proc finish {} {
global ns tr1 nm1
$ns flush-trace
exec nam b2.nam &
close $tr1
close $nm1
exit 0
}
$ns at 5.0 "finish"
$ns run
```

```
//dup.awk
BEGIN {
c=0;
c1=0
}
{
if($1=="d" && $9=="0.0" && $10 =="40")
c++;
if($1=="d" && $9=="1.0" && $10 n=="5.0")
c1++;
}
END{
printf("\n number of paclets dropped at the destination node4 %d",c);
printf("\n number of paclets dropped at the destination node5 %d",c1);
}
```

OUTPUT:-

```
student@fedora:~$ ns prgb3.tcl
couldn't read file "prgb3.tcl": no such file or directory
student@fedora:~$ cd c
student@fedora:~/c$ ns prgb3.tcl
warning: no class variable LanRouter::debug_

        see tcl-object.tcl in tclcl for info about this warning.

student@fedora:~/c$ awk -f dup.awk b2.tr

 number of paclets dropped at the destination node4 0
 number of paclets dropped at the destination node5 57student@fedora:~/c$
```





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.

```
set ns [new Simulator]
set tf [open ftp.tr w]
$ns trace-all $tf
set nm [open ftp.nam w]
$ns namtrace-all $nm

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

$ns color 1 "red"
$ns color 2 "green"

$ns duplex-link $n0 $n2 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 $n2 $n4 0.3Mb 10ms DropTail

$ns set queue-limit $n0 $n2 5
$ns set queue-limit $n1 $n2 5
$ns set queue-limit $n2 $n3 5
$ns set queue-limit $n2 $n4 5

set tcp1 [new Agent/TCP]
$ns attach-agent $n0 $tcp1
set sink1 [new Agent/TCPSink]
$ns attach-agent $n3 $sink1

set ftp1 [new Application/FTP]
$ftp1 attach-agent $tcp1
$ftp1 set packetSize_ 500
$ftp1 set interval_ 0.005
$ns connect $tcp1 $sink1

set tcp2 [new Agent/TCP]
$ns attach-agent $n1 $tcp2
set sink2 [new Agent/TCPSink]
$ns attach-agent $n3 $sink2
```

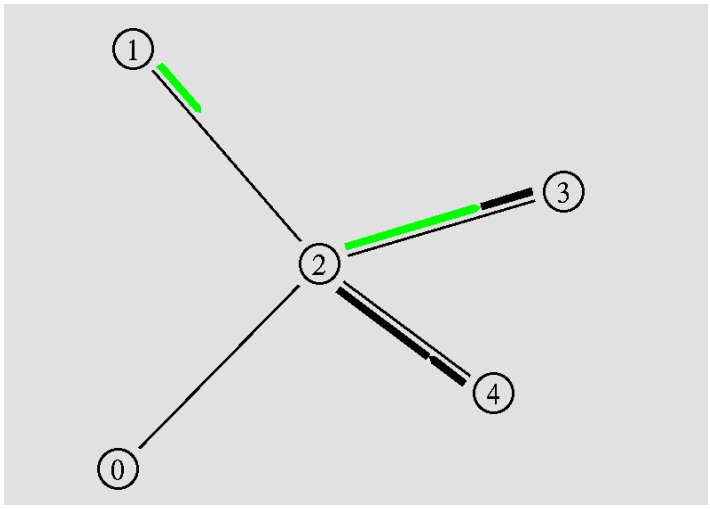
```
set telneto [new Application/Telnet]
$telneto attach-agent $tcp2
$telneto set packetSize_ 500
$telneto set interval_ 0.005
$ns connect $tcp2 $sink2

set tcp3 [new Agent/TCP]
$ns attach-agent $n4 $tcp3
set sink3 [new Agent/TCPSink]
$ns attach-agent $n3 $sink3

set ftp2 [new Application/FTP]
$ftp2 attach-agent $tcp3
$ftp2 set packetSize_ 500
$ftp2 set interval_ 0.005
$ns connect $tcp3 $sink3

$tcp1 set class_ 1
$tcp2 set class_ 2

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

$ns at 0.1 "$ftp1 start"
$ns at 0.1 "$ftp2 start"
$ns at 0.1 "$telneto start"
$ns at 5.0 "finish"

$ns run
```

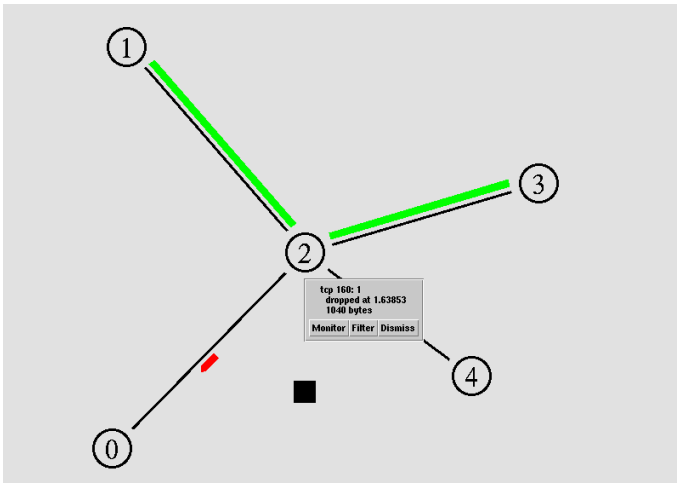
//ftp.awk

```
BEGIN{
drop=0;
sent=0;
rec=0;
ratio=0;
```

```
}
{
if ($1 == "d") drop++;
if ($1=="+")
sent++;
if($1=="r")
rec++;
ratio=(rec/sent)*100;
}
END{
printf" \nTotal number of packet sent from source:%d", sent;
printf" \nTotal number of packet received at destination :%d", rec;
printf" \nTotal number of packet dropped :%d", drop;
printf" \nPacket delivery ratio: %f\n",ratio;
}
```

OUTPUT:-





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

```
set ns [new Simulator -multicast on]
set tf [open mcast.tr w]
$ns trace-all $tf
set nm1 [open mcast.nam w]
$ns namtrace-all $nm1
set n0 [$ns node]
set n1 [$ns node]
set n2 [$ns node]
set n3 [$ns node]
set n4 [$ns node]
$ns color 1 "red"
$ns color 2 "blue"
$ns duplex-link $n0 $n1 1.5Mb 10ms DropTail
$ns duplex-link $n0 $n2 1.5Mb 10ms DropTail
$ns duplex-link $n2 $n3 1.5Mb 10ms DropTail
set mproto DM
set mrthandle [$ns mrtproto $mproto {}]
set group1 [Node allocaddr]
set group2 [Node allocaddr]
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
set udp1 [new Agent/UDP]
$ns attach-agent $n0 $udp1
$udp1 set dst_addr_ $group2
$udp1 set dst_port_ 0
set cbr2 [new Application/Traffic/CBR]
$cbr2 attach-agent $udp1
set rcvr1 [new Agent/Null]
$ns attach-agent $n1 $rcvr1
$ns at 1.0 "$n1 join-group $rcvr1 $group1"
set rcvr2 [new Agent/Null]
$ns attach-agent $n2 $rcvr2
$ns at 1.5 "$n2 join-group $rcvr2 $group2"
set rcvr3 [new Agent/Null]
$ns attach-agent $n3 $rcvr3
$ns at 2.0 "$n3 join-group $rcvr3 $group2"
$udp1 set class_ 1;
```

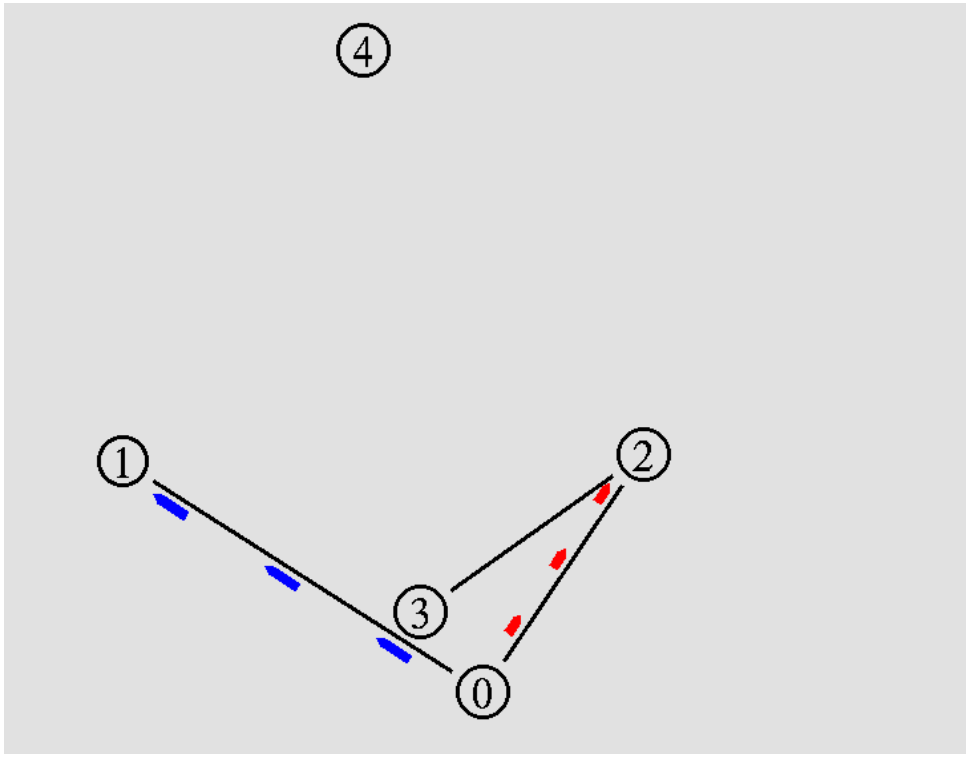
```
$udp0 set class_ 2;
$ns at 2.0 "$n3 join-group $rcvr2 $group2"
$ns at 0.5 "$cbr1 start"
$ns at 4.5 "$cbr1 stop"
$ns at 0.5 "$cbr2 start"
$ns at 4.5 "$cbr2 stop"
proc finish {} {
global ns tf nm1
$ns flush-trace
close $tf
close $nm1
exec nam mcast.nam &
exit 0
}
$ns at 5.0 "finish"
$ns run
```

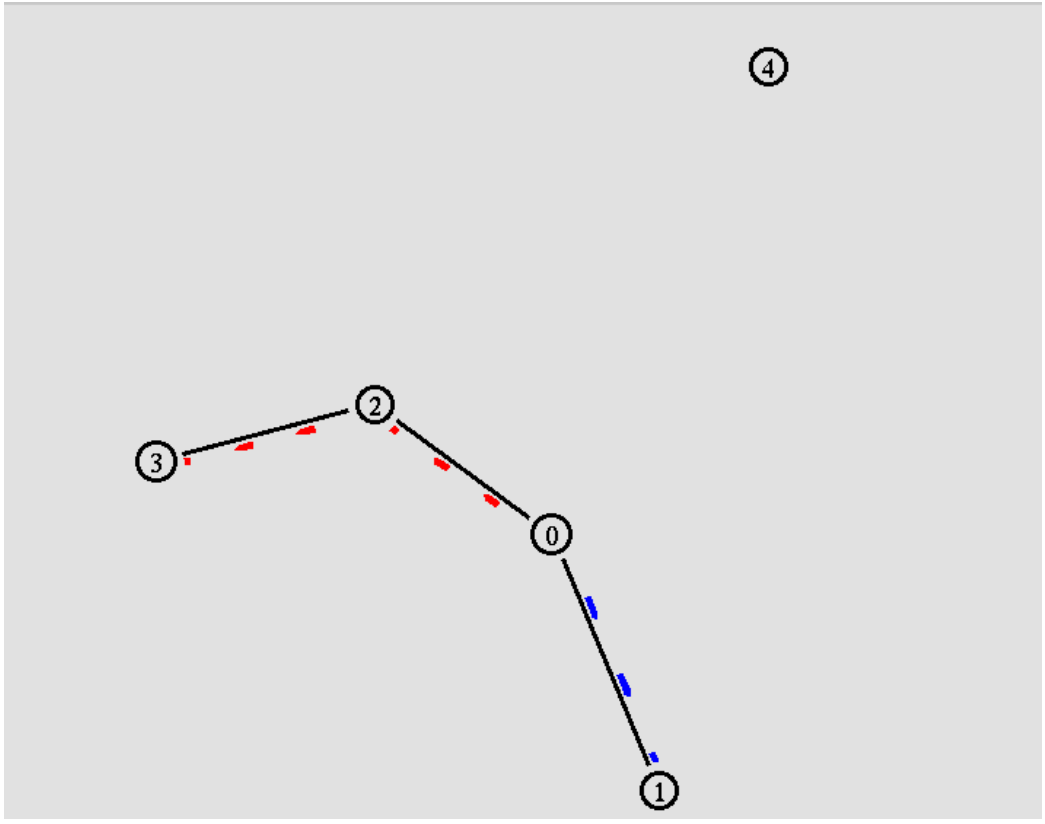
//mulrout.awk

```
BEGIN {count = 0;
time = 0;
}
{
if ($1 == "r") {
count += $6;
time = $2;
}
}
END {
system("clear");
printf("\n Total bytes transferred from sourec to destination is %d", count);
printf("\n Total simulation time %f",$2);
printf("\n Throughput: %f Mbps \n\n", (count/$2) * (8/1000000));
}
```

OUTPUT:-

```
Total bytes transferred from sourec to destination is 570320
Total simulation time 4.520860
Throughput: 1.009224 Mbps
```





0