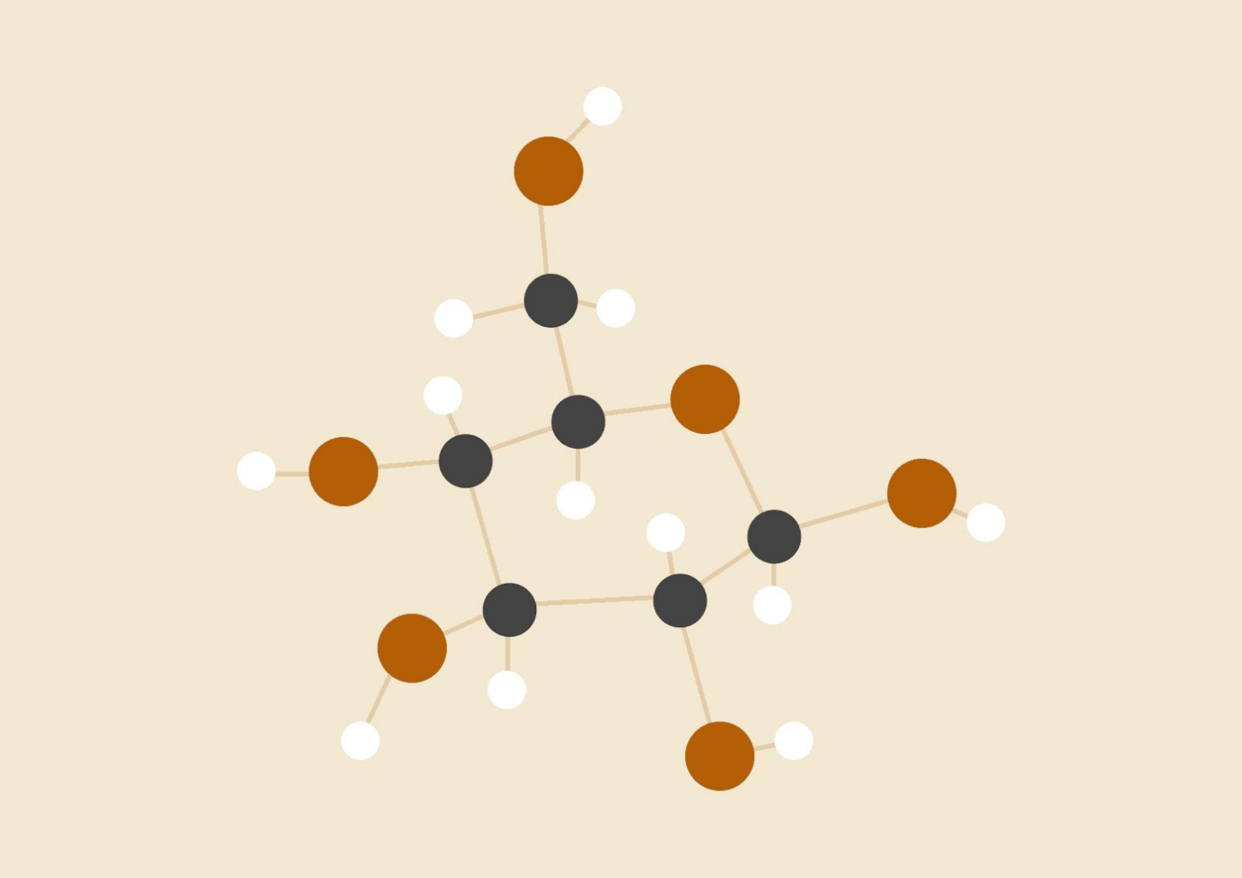
Lab Assignment 3



Shashank Kashyap

17114070
CSE III

Problem Statement 1

Write a socket program in C to determine class, Network and Host ID of an IPv4 address.

Solution:

- **For determining the class:** The idea is to check first octet of IP address. As we know, for class **A** first octet will range from **1 – 126**, for class **B** first octet will range from **128 – 191**, for class **C** first octet will range from **192- 223**, for class **D** first octet will range from **224 – 239**, for class **E** first octet will range from **240 – 255**.

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

char findClassOfIP(char str[]) {
    // storing first octet in arr[] variable
    char arr[4];
    int i = 0;
    while (str[i] != '.') {
       arr[i] = str[i];
       i++;
    }

    i--;
    int ip = 0, j = 1;
    while (i >= 0) {
       // converting str[] variable into number for comparison
       ip = ip + (str[i] - '0') * j;
       j = j * 10;
       i--;
    }

    if (ip >=1 && ip <= 126) {
       return 'A';
    }else if (ip >= 128 && ip <= 191) {
       return 'B';
    }else if (ip >= 192 && ip <= 223) {
       return 'C';
    }else if (ip >= 224 && ip <= 239) {
        return 'D';
    }else {
        return 'E';
    }
}
```

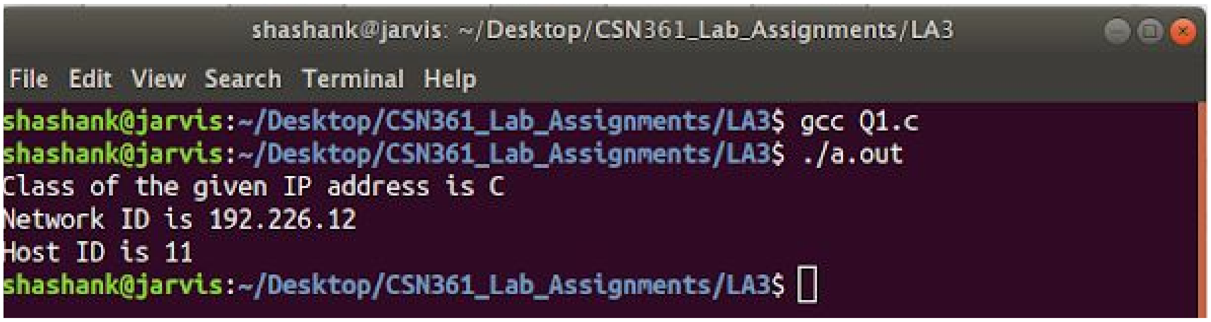
- **For determining the Network and Host ID:** We know that Subnet Mask for Class **A** is **8**, for Class **B** is **16** and for Class **C** is **24** whereas Class **D** and **E** is not divided into Network and Host ID.

```
// Function to separate Network ID as well as Host ID and print them
void separateID(char str[], char ipClass) {
    // Initializing netID and host array to NULL
    char netID[12], hostID[12];
    for (int k = 0; k < 12; k++) {
        netID[k] = hostID[k] = '\0';
    }

    // for class A, only first octet is Network ID
    // and rest are Host ID
    if (ipClass == 'A') {
        int i = 0, j = 0;
        while (str[j] != '.') {
            netID[i++] = str[j++];
        }
        i = 0;
        j++;
        while (str[j] != '\0') {
            hostID[i++] = str[j++];
        }
        printf("Network ID is %s\n", netID);
        printf("Host ID is %s\n", hostID);
    }

    // for class B, first two octet are Network ID
    // and rest are Host ID
    else if (ipClass == 'B') {
        int i = 0, j = 0, dotCount = 0;

        // storing in netID[] up to 2nd dot
        // dotCount keeps track of number of dots or octets passed
        while (dotCount < 2) {
            netID[i++] = str[j++];
```

```
             if (str[j] == '.') {
                   dotCount++;
             }
       }
       i = 0;
       j++;

       while (str[j] != '\0') {
             hostID[i++] = str[j++];
       }

       printf("Network ID is %s\n", netID);
       printf("Host ID is %s\n", hostID);
    }

    // for class C, first three octet are Network ID
    // and rest are Host ID
    else if (ipClass == 'C') {
       int i = 0, j = 0, dotCount = 0;

       // storing in netID[] up to 3rd dot
       // dotCount keeps track of number of
       // dots or octets passed
       while (dotCount < 3) {
             netID[i++] = str[j++];
             if (str[j] == '.') {
                   dotCount++;
             }
       }

       i = 0;
       j++;

       while (str[j] != '\0') {
             hostID[i++] = str[j++];
       }

       printf("Network ID is %s\n", netID);
       printf("Host ID is %s\n", hostID);
    }else {
       // Class D and E are not divided in Network
       // and Host ID
       printf("In this Class, IP address is not"
       " divided into Network and Host ID\n");
    }
```

```
}
```

Execution:


shashank@jarvis: ~/Desktop/CSN361_Lab_Assignments/LA3
File Edit View Search Terminal Help

```
shashank@jarvis:~/Desktop/CSN361_Lab_Assignments/LA3$ gcc Q1.c
shashank@jarvis:~/Desktop/CSN361_Lab_Assignments/LA3$ ./a.out
Class of the given IP address is C
Network ID is 192.226.12
Host ID is 11
shashank@jarvis:~/Desktop/CSN361_Lab_Assignments/LA3$
```

Problem Statement 2

Write a C program to demonstrate File Transfer using UDP.

Solution:

- The server starts and waits for filename.

```
############ Server ##############
int main() {
    int sockfd, nBytes;
    struct sockaddr_in addr_con;
    int addrlen = sizeof(addr_con);
    addr_con.sin_family = AF_INET;
    addr_con.sin_port = htons(PORT_NO);
    addr_con.sin_addr.s_addr = INADDR_ANY;
    char net_buf[NET_BUF_SIZE];
    FILE* fp;

    // socket()
    sockfd = socket(AF_INET, SOCK_DGRAM, IP_PROTOCOL);

    if (sockfd < 0) {
        printf("\nfile descriptor not received!!\n");
    }else {
        printf("\nfile descriptor %d received\n", sockfd);
    }

    // bind()
    if (bind(sockfd, (struct sockaddr*)&addr_con, sizeof(addr_con)) == 0) {
        printf("\nSuccessfully binded!\n");
    }else {
        printf("\nBinding Failed!\n");
    }
```

- The client sends a filename.

```
############# client ##############
char Decrypt(char ch) {
    return ch ^ cipherKey;
}

// function to receive file
int receiveFile(char* buf, int s) {
    int i;
    char ch;
    for (i = 0; i < s; i++) {
        ch = buf[i];
        ch = Decrypt(ch);
        if (ch == EOF) {
              return 1;
        }else {
              printf("%c", ch);
        }
    }
    return 0;
}

// driver code
int main() {
    int sockfd, nBytes;
    struct sockaddr_in addr_con;
    int addrlen = sizeof(addr_con);
    addr_con.sin_family = AF_INET;
    addr_con.sin_port = htons(PORT_NO);
    addr_con.sin_addr.s_addr = inet_addr(IP_ADDRESS);
    char net_buf[NET_BUF_SIZE];
    FILE* fp;

    // socket()
    sockfd = socket(AF_INET, SOCK_DGRAM,
                     IP_PROTOCOL);

    if (sockfd < 0) {
        printf("\nfile descriptor not received!!\n");
    }else {
        printf("\nfile descriptor %d received\n", sockfd);
    }

    while (1) {
        printf("\nPlease enter file name to receive:\n");
        scanf("%s", net_buf);
        sendto(sockfd, net_buf, NET_BUF_SIZE,
            sendrecvflag, (struct sockaddr*)&addr_con,
            addrlen);

        printf("\n---------Data Received---------\n");

        while (1) {
            // receive
            clearBuffer(net_buf);
            nBytes = recvfrom(sockfd, net_buf, NET_BUF_SIZE,
                                    sendrecvflag, (struct sockaddr*)&addr_con,
                                    &addrlen);

            // process
            if (receiveFile(net_buf, NET_BUF_SIZE)) {
                break;
            }
        }
        printf("\n-------------------------------\n");
    }
    return 0;
}
```

- The server receives filename.

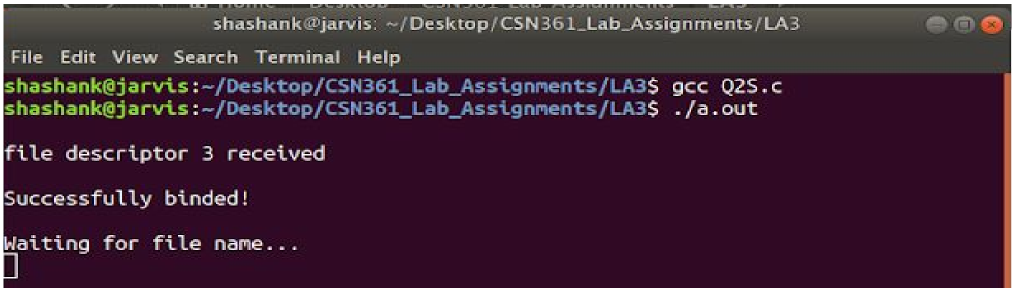
```
########## server ##########
while (1) {
       printf("\nWaiting for file name...\n");

       // receive file name
       clearBuffer(net_buf);

       nBytes = recvfrom(sockfd, net_buf,
                               NET_BUF_SIZE, sendrecvflag,
                               (struct sockaddr*)&addr_con, &addrlen);

       fp = fopen(net_buf, "r");
       printf("\nFile Name Received: %s\n", net_buf);
       if (fp == NULL) {
             printf("\nFile open failed!\n");
       }else {
             printf("\nFile Successfully opened!\n");
       }

       while (1) {

             // process
             if (sendFile(fp, net_buf, NET_BUF_SIZE)) {
                   sendto(sockfd, net_buf, NET_BUF_SIZE,
                         sendrecvflag,
                         (struct sockaddr*)&addr_con, addrlen);
                   break;
             }

             // send
             sendto(sockfd, net_buf, NET_BUF_SIZE,
                   sendrecvflag,
                   (struct sockaddr*)&addr_con, addrlen);
             clearBuffer(net_buf);
       }
       if (fp != NULL) {
             fclose(fp);
       }
    }
    return 0;
}
```

- If file is present, server starts reading file and continues to send a buffer filled with file contents encrypted until file-end is reached. Security: Handled by encryption. Protocol : UDP. Encryption: XOR encryption.

```
char Encrypt(char ch) {
    return ch ^ cipherKey;
}

// function sending file
int sendFile(FILE* fp, char* buf, int s) {
    int i, len;
    if (fp == NULL) {
       strcpy(buf, nofile);
       len = strlen(nofile);
       buf[len] = EOF;
       for (i = 0; i <= len; i++) {
             buf[i] = Encrypt(buf[i]);
       }
       return 1;
    }

    char ch, ch2;
    for (i = 0; i < s; i++) {
       ch = fgetc(fp);
       ch2 = Encrypt(ch);
       buf[i] = ch2;
       if (ch == EOF) {
             return 1;
       }
    }
    return 0;
}
```

Execution:

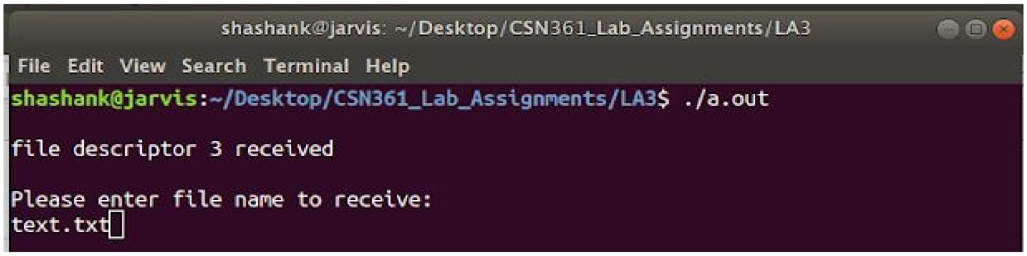
- Start the server and wait for the filename.



shashank@jarvis: ~/Desktop/CSN361_Lab_Assignments/LA3
File Edit View Search Terminal Help

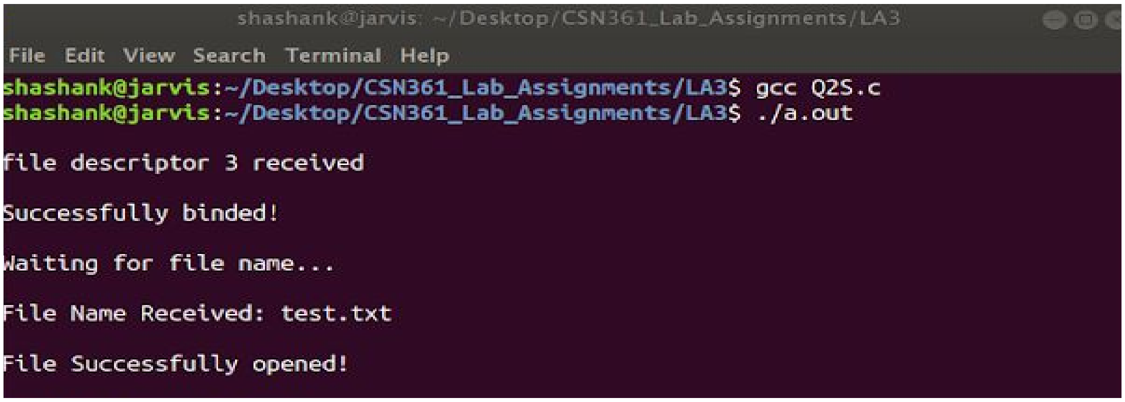
```
shashank@jarvis:~/Desktop/CSN361_Lab_Assignments/LA3$ gcc Q2S.c
shashank@jarvis:~/Desktop/CSN361_Lab_Assignments/LA3$ ./a.out

file descriptor 3 received

Successfully binded!

Waiting for file name...
```

- Start the client and enter the name of the file to be requested from the server.

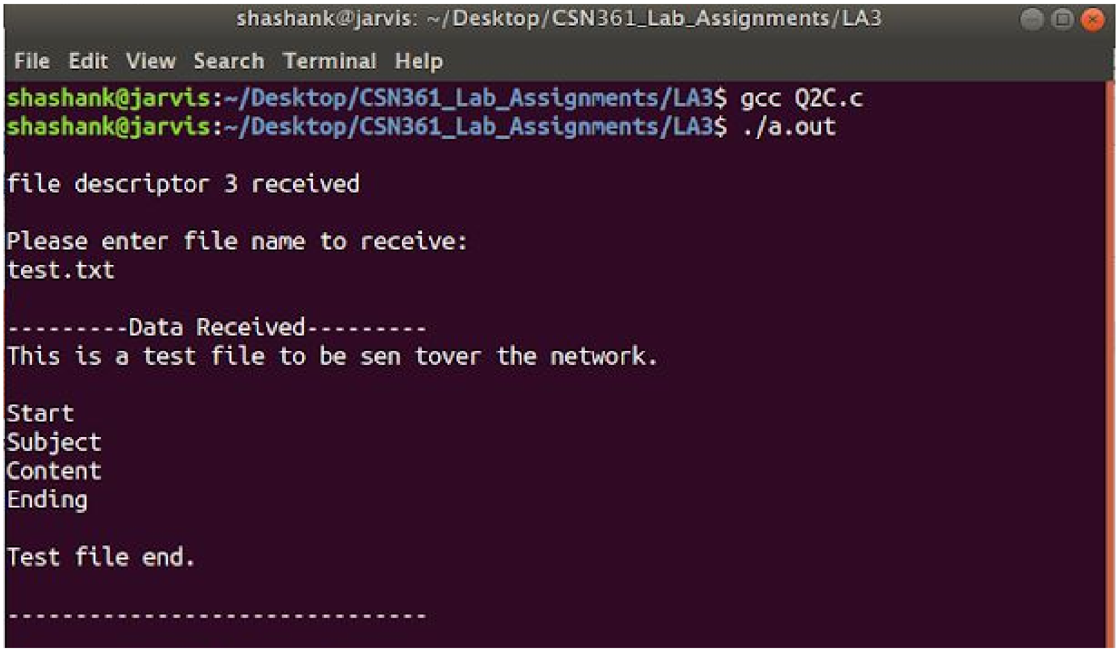

shashank@jarvis: ~/Desktop/CSN361_Lab_Assignments/LA3
File Edit View Search Terminal Help

```
shashank@jarvis:~/Desktop/CSN361_Lab_Assignments/LA3$ ./a.out

file descriptor 3 received

Please enter file name to receive:
text.txt
```

- Let server send the file and receive it accordingly.


shashank@jarvis: ~/Desktop/CSN361_Lab_Assignments/LA3
File Edit View Search Terminal Help

```
shashank@jarvis:~/Desktop/CSN361_Lab_Assignments/LA3$ gcc Q2S.c
shashank@jarvis:~/Desktop/CSN361_Lab_Assignments/LA3$ ./a.out

file descriptor 3 received

Successfully binded!

Waiting for file name...

File Name Received: test.txt

File Successfully opened!
```


shashank@jarvis: ~/Desktop/CSN361_Lab_Assignments/LA3
File Edit View Search Terminal Help

```
shashank@jarvis:~/Desktop/CSN361_Lab_Assignments/LA3$ gcc Q2C.c
shashank@jarvis:~/Desktop/CSN361_Lab_Assignments/LA3$ ./a.out

file descriptor 3 received

Please enter file name to receive:
test.txt

---------Data Received---------
This is a test file to be sen tover the network.

Start
Subject
Content
Ending

Test file end.

------------------------------
```

Problem Statement 3

Write a TCL code for network simulator NS2 to demonstrate the star topology among a set of computer nodes. Given N nodes, one node will be assigned as the central node and the other nodes will be connected to it to form the star. You have to set up a TCP connection between k pairs of nodes and demonstrate the packet transfer between them using Network Animator (NAM). Use File Transfer protocol (FTP) for the same. Each link should have different color of packets to differentiate the packets transferred between each pair of nodes. The program should take the number of nodes (N) as input followed by k pairs of nodes.

Solution:

- Create a simulator object, open the nam trace file, define a ‘finish’ procedure.

```
set netSim [new Simulator]

$netSim color 0 Red
$netSim color 1 Blue
$netSim color 2 Azure
$netSim color 3 Coral
$netSim color 4 Cyan

set f [open 3.nam w]
$netSim namtrace-all $f

proc finish {} {
      global netSim f
      $netSim flush-trace
      close $f

      exec nam 3.nam &
      exit 0
}
```

- Create the given number of nodes

```
puts "Enter no. of Nodes: "
gets stdin N
set n(0) [$netSim node]
for {set i 1} {$i < $N} {incr i} {
      set n($i) [$netSim node]
      $netSim duplex-link $n($i) $n(0) 1Mb 10ms DropTail
}
```

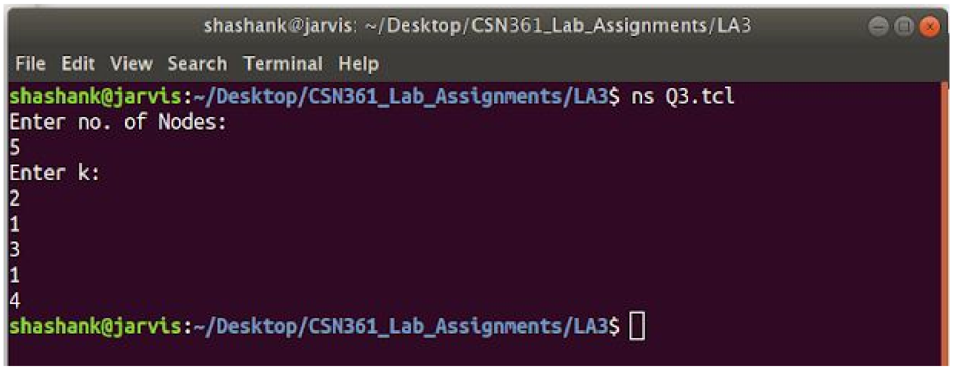
- Create links between nodes, create TCP agent and attach it to node, create FTP and attach it to TCP agent.

```
puts "Enter k: "
gets stdin k
for {set i 0} {$i < $k} {incr i} {
      gets stdin i1
      gets stdin i2
      set TCP [new Agent/TCP]
      $TCP set class_ [expr $i%5]
      $netSim attach-agent $n($i1) $TCP

      set sink [new Agent/TCPSink]
      $netSim attach-agent $n($i2) $sink
      $netSim connect $TCP $sink
      $TCP set fid_ $i

      set ftp($i) [new Application/FTP]
      $ftp($i) attach-agent $TCP
      $ftp($i) set type_ FTP
}
# $netSim duplex-link $n0 $n1 1Mb 10ms DropTail
```

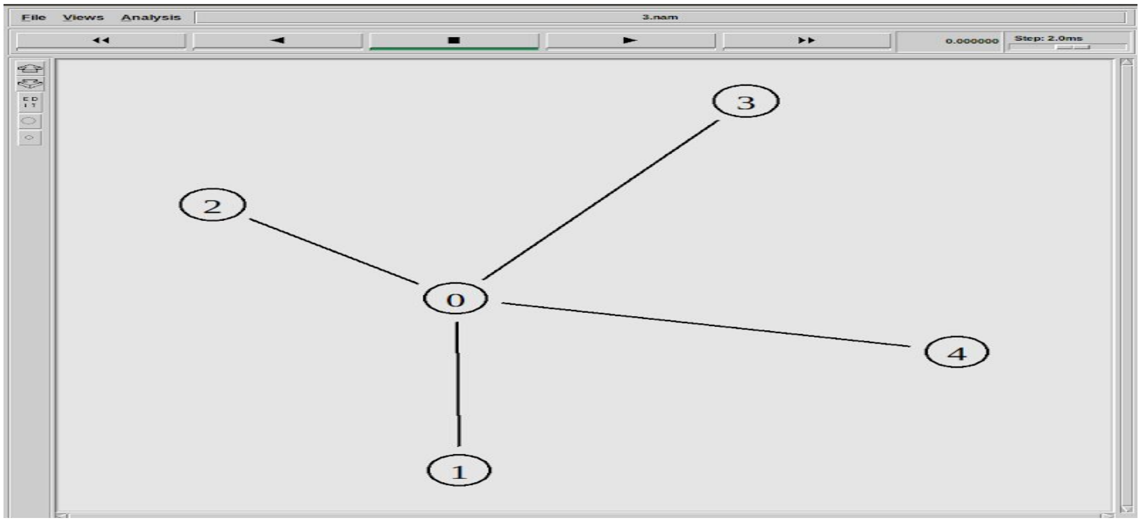
- Schedule events for agents, run the simulation

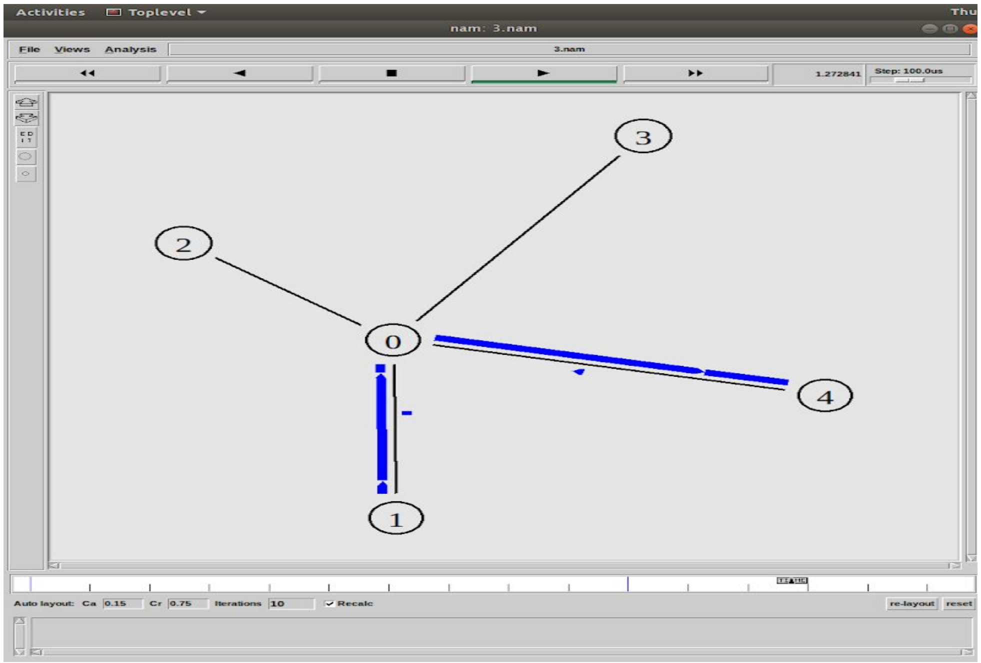
```
for {set i 0} {$i < $k} {incr i} {
      $netSim at [expr ($i/10)+0.1] "$ftp($i) start"
      $netSim at [expr ($i/10)+1.5] "$ftp($i) stop"
}
$netSim at [expr ($k/10)+1.5] "finish"

$netSim run
```

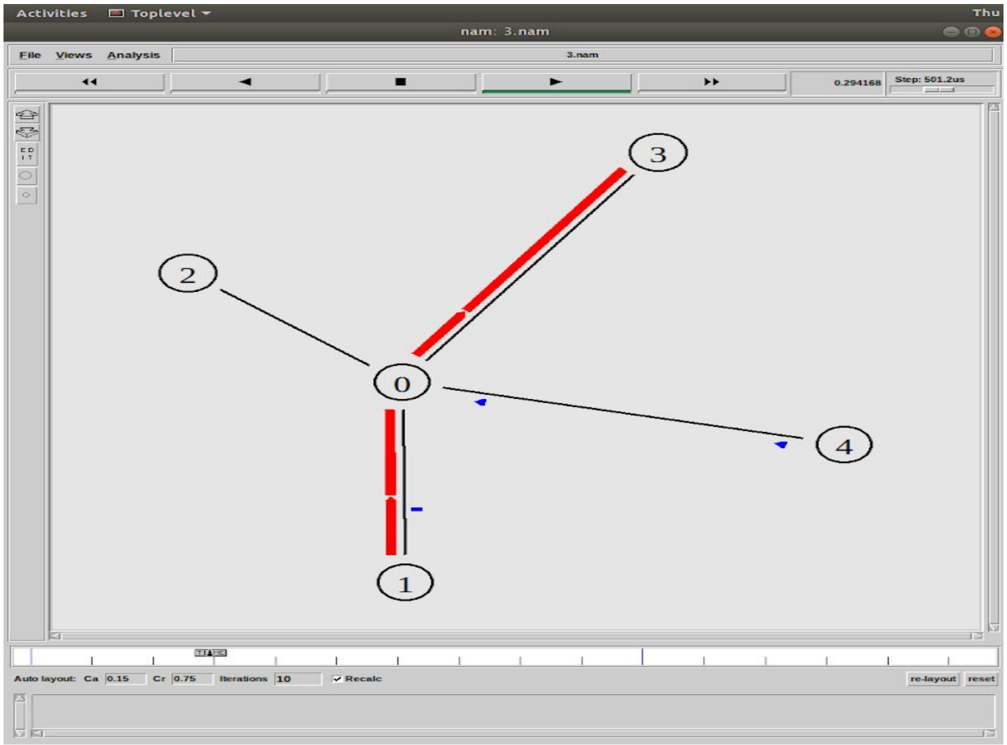
Execution:


shashank@jarvis: ~/Desktop/CSN361_Lab_Assignments/LA3
File Edit View Search Terminal Help

```
shashank@jarvis:~/Desktop/CSN361_Lab_Assignments/LA3$ ns Q3.tcl
Enter no. of Nodes:
5
Enter k:
2
1
3
1
4
shashank@jarvis:~/Desktop/CSN361_Lab_Assignments/LA3$ 
```







Problem Statement 4

Write a TCL code for network simulator NS2 to demonstrate the ring topology among a set of computer nodes. Given N nodes, each node will be connected to two other nodes in the form of a ring. You have to set up a TCP connection between k pairs of nodes and demonstrate packet transfer between them using Network Animator (NAM). Use File Transfer protocol (FTP) for the same. Each link should have different color of packets to differentiate the packets transferred between each pair of nodes. The program should take the number of nodes (N) as input followed by k pairs of nodes.

Solution:

- Create a simulator object, open the nam trace file, define a ‘finish’ procedure.

```
set netSim [new Simulator]

$netSim color 0 Red
$netSim color 1 Blue
$netSim color 2 Azure
$netSim color 3 Coral
$netSim color 4 Cyan

set f [open 4.nam w]
$netSim namtrace-all $f

proc finish {} {
      global netSim f
      $netSim flush-trace
      close $f

      exec nam 4.nam &
      exit 0
}
```

- Create the given number of nodes

```
puts "Enter no. of Nodes: "
gets stdin N
set n(0) [$netSim node]
set y 0
for {set i 1} {$i < $N} {incr i} {
      set n($i) [$netSim node]
      $netSim duplex-link $n($y) $n($i) 1Mb 10ms DropTail
      set y $i
}
$netSim duplex-link $n($y) $n(0) 1Mb 10ms DropTail
```

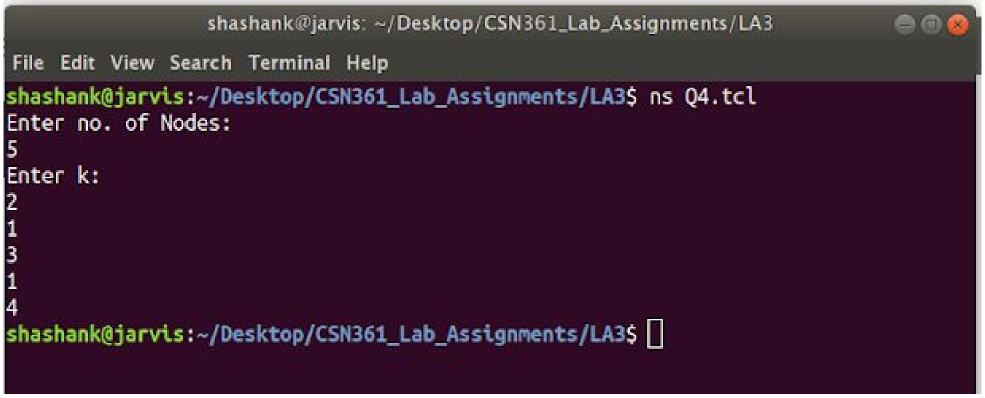
- Create links between nodes, create TCP agent and attach it to node, create FTP and attach it to TCP agent.

```
puts "Enter k: "
gets stdin k
for {set i 0} {$i < $k} {incr i} {
      gets stdin i1
      gets stdin i2
      set TCP [new Agent/TCP]
      $TCP set class_ [expr $i%5]
      $netSim attach-agent $n($i1) $TCP

      set sink [new Agent/TCPSink]
      $netSim attach-agent $n($i2) $sink
      $netSim connect $TCP $sink
      $TCP set fid_ $i

      set ftp($i) [new Application/FTP]
      $ftp($i) attach-agent $TCP
      $ftp($i) set type_ FTP
}
```

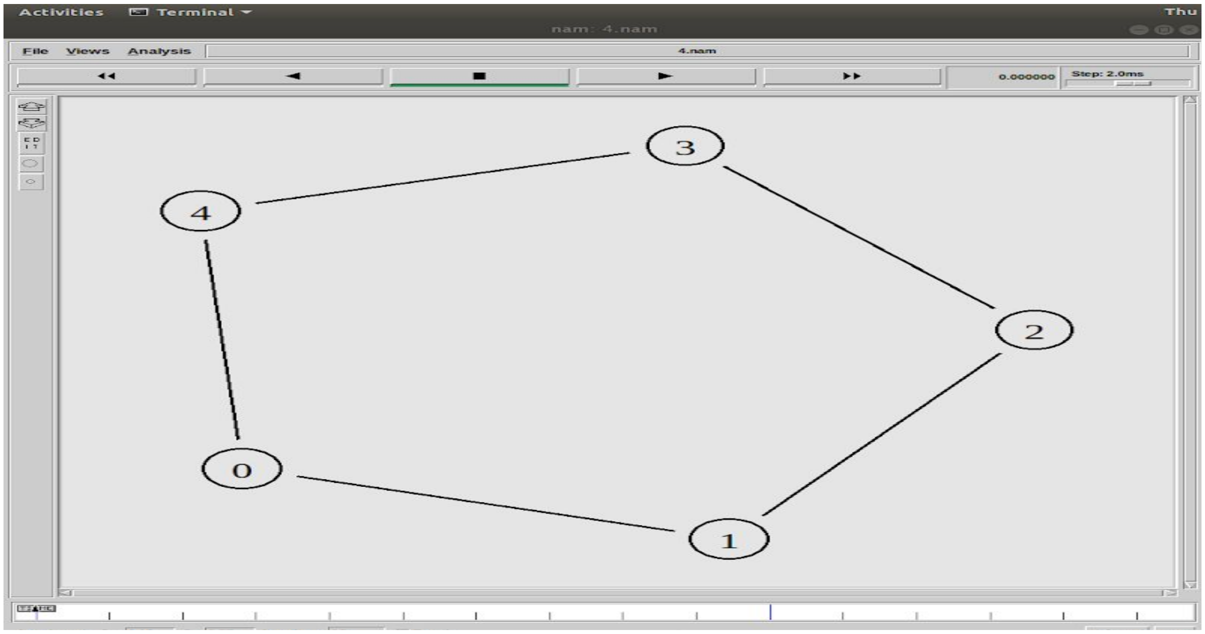
- Schedule events for agents, run the simulation

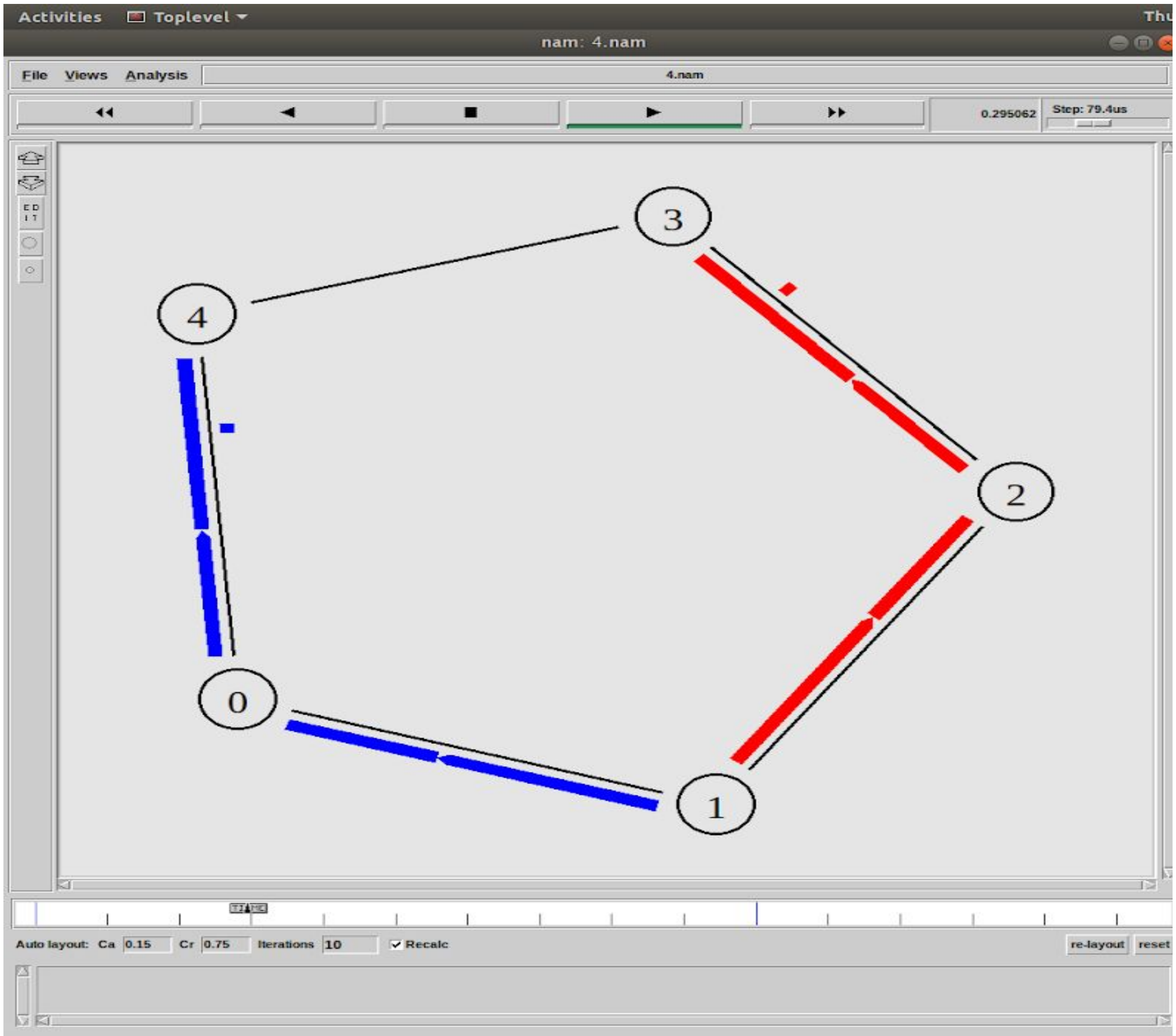
```
for {set i 0} {$i < $k} {incr i} {
      $netSim at [expr ($i/10)+0.1] "$ftp($i) start"
      $netSim at [expr ($i/10)+1.5] "$ftp($i) stop"
}
$netSim at [expr ($k/10)+1.5] "finish"

$netSim run
```

Execution:


shashank@jarvis: ~/Desktop/CSN361_Lab_Assignments/LA3
File Edit View Search Terminal Help

```
shashank@jarvis:~/Desktop/CSN361_Lab_Assignments/LA3$ ns Q4.tcl
Enter no. of Nodes:
5
Enter k:
2
1
3
1
4
shashank@jarvis:~/Desktop/CSN361_Lab_Assignments/LA3$ 
```





Auto layout: Ca 0.15 Cr 0.75 Iterations 10 ☑ Recalc

Problem Statement 5

Write a TCL code for network simulator NS2 to demonstrate the bus topology among a set of computer nodes. Given N nodes, each node will be connected to a common link. You have to set up a TCP connection between k pairs of nodes and demonstrate packet transfer between them using Network Animator (NAM). Use File Transfer protocol (FTP) for the same. Each link should have different color of packets to differentiate the packets transferred between each pair of nodes. The program should take the number of nodes (N) as input followed by k pairs of nodes.

Solution:

- Create a simulator object, open the nam trace file, define a 'finish' procedure.

```
set netSim [new Simulator]

set nf [open 5.nam w]
$netSim namtrace-all $nf

proc finish {} {
      global netSim nf
      $netSim flush-trace

      close $nf

      exec nam 5.nam &
      exit 0
}
```

- Create the given number of nodes.

```
set n0 [$netSim node]
set n1 [$netSim node]
set n2 [$netSim node]
set n3 [$netSim node]
set n4 [$netSim node]

$netSim make-lan "$n0 $n1 $n2 $n3 $n4" 0.5Mb 40ms LL Queue/DropTail
Mac/802_3
```

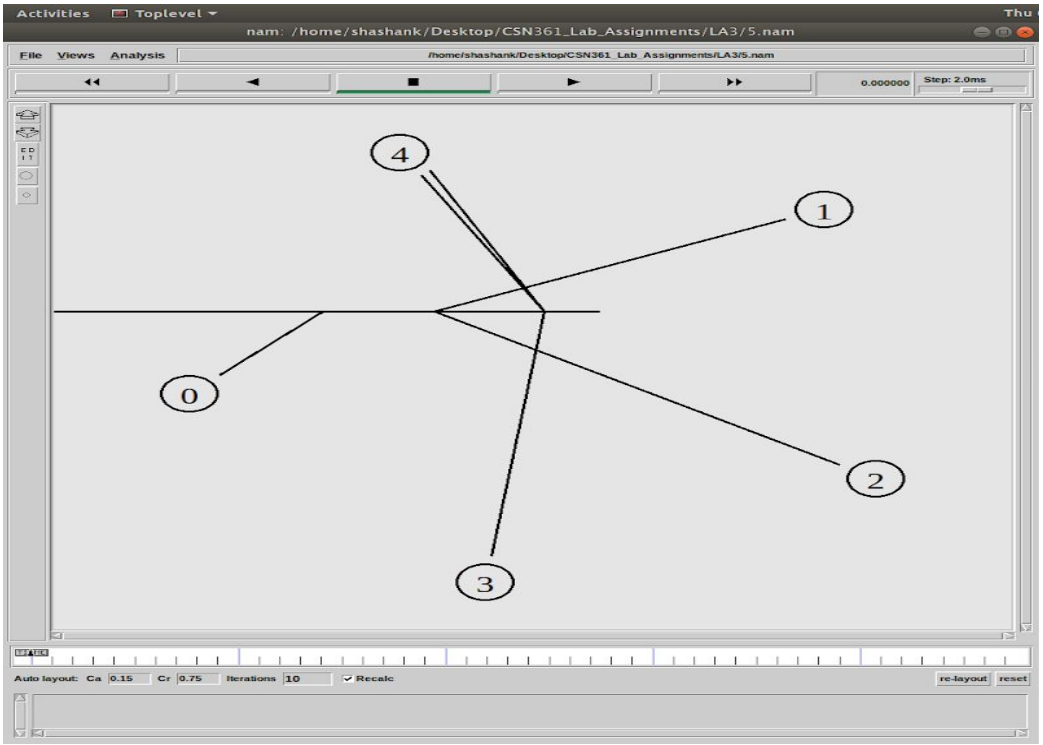
- Create links between nodes, create TCP agent and attach it to node, create FTP and attach it to TCP agent.

```
set tcp0 [new Agent/TCP]
$tcp0 set class_ 1
$netSim attach-agent $n1 $tcp0
set sink0 [new Agent/TCPSink]
$netSim attach-agent $n3 $sink0
$netSim connect $tcp0 $sink0

set cbr0 [new Application/Traffic/CBR]
$cbr0 set packetSize_ 500
$cbr0 set interval_ 0.01
$cbr0 attach-agent $tcp0
```

- Schedule events for agents, run the simulation

```
$netSim at 0.5 "$cbr0 start"
$netSim at 4.5 "$cbr0 stop"

$netSim at 5.0 "finish"

$netSim run
```

Execution:

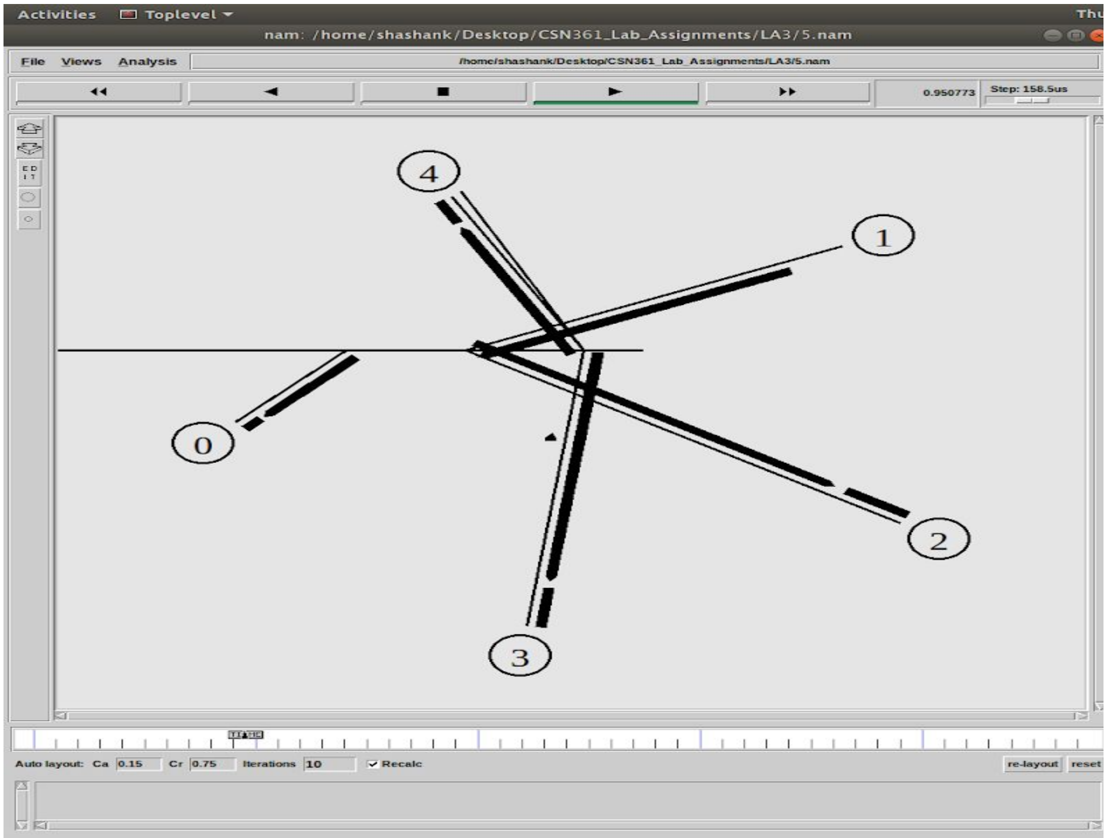


nam: /home/shashank/Desktop/CSN361_Lab_Assignments/LA3/5.nam

/home/shashank/Desktop/CSN361_Lab_Assignments/LA3/5.nam

0.000000 Step: 2.0ms

Auto layout: Ca 0.15 Cr 0.75 Iterations 10 ☑ Recalc

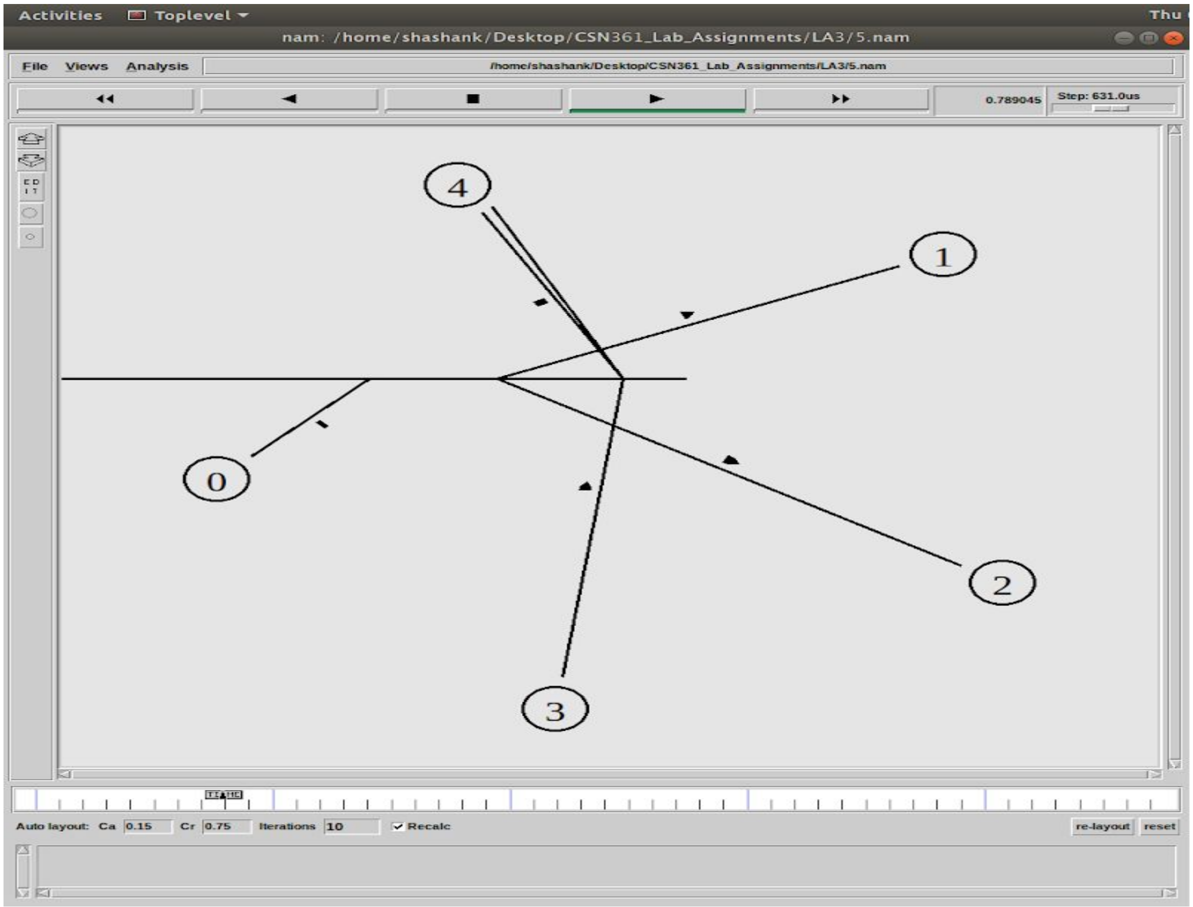


nam: /home/shashank/Desktop/CSN361_Lab_Assignments/LA3/5.nam

/home/shashank/Desktop/CSN361_Lab_Assignments/LA3/5.nam

0.950773 Step: 158.5us

Auto layout: Ca 0.15 Cr 0.75 Iterations 10 ☑ Recalc



nam: /home/shashank/Desktop/CSN361_Lab_Assignments/LA3/5.nam

/home/shashank/Desktop/CSN361_Lab_Assignments/LA3/5.nam

0.789045 Step: 631.0us

Auto layout: Ca 0.15 Cr 0.75 Iterations 10 ☑ Recalc