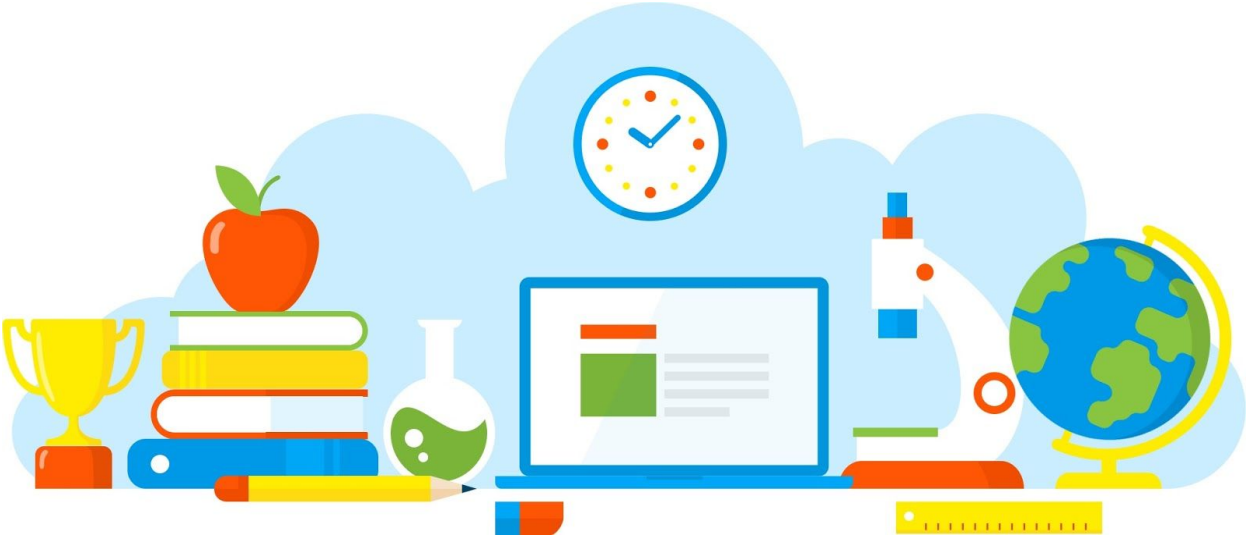


Assignment Report

This report contains work from Assignment 3 of course CSN 361.

Name- Kaustubh Trivedi
Enrollment Number - 17114044
Class - CSE B.Tech. 3rd Year

Submission Files - Repository Link

QUESTION 1: Write a socket program in C to determine class, Network and Host ID of an IPv4 address.

SOLUTION:

With an IPv4 IP address, there are five classes of available IP ranges: Class A, Class B, Class C and Class D.

Class A: 1 - 127 : 1 octet

Class B: 128 - 191 : 2 octets

Class C: 191 - 223: 3 octets

Class D: 224 - 254

CODE:

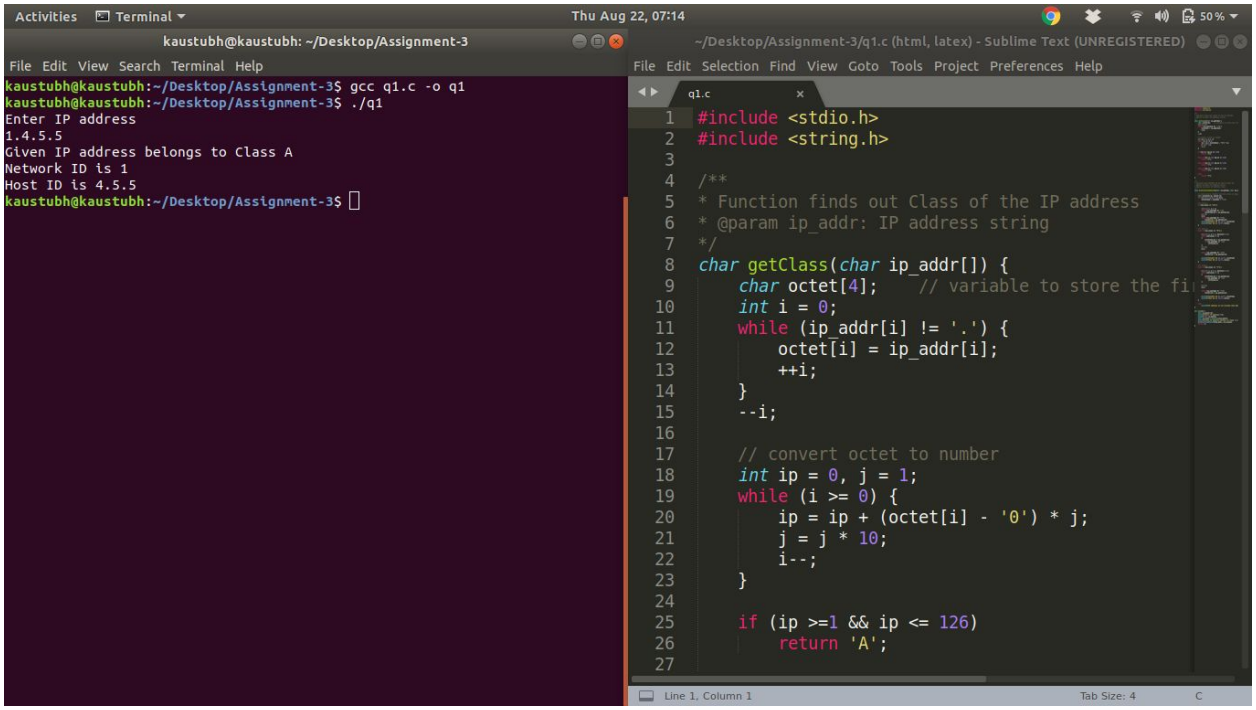
```c
#include <stdio.h>
#include <string.h>

/**
* Function finds out Class of the IP address
* @param ip_addr: IP address string
*/
char getClass(char ip_addr[]) {
    char octet[4];    // variable to store the first octet
    int i = 0;
    while (ip_addr[i] != '.') {
        octet[i] = ip_addr[i];
        ++i;
    }
    --i;

    // convert octet to number
    int ip = 0, j = 1;
    while (i >= 0) {
        ip = ip + (octet[i] - '0') * j;
        j = j * 10;
        i--;
    }

    if (ip >=1 && ip <= 126)
```

```c
        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';
}

/**
* Function gets Network ID as well as Host ID
* @param ip_addr: IP address string
* @param ip_class: IP address class
*/
void getNetworkAndHostID(char ip_addr[], char ip_class) {

    // Initializing network and host array to NULL
    char network[12], host[12];
    for (int k = 0; k < 12; k++)
        network[k] = host[k] = '\0';

    /// Class A
    if (ip_class == 'A') {

        int i = 0, j = 0;
        while (ip_addr[j] != '.')
            network[i++] = ip_addr[j++];
        i = 0;
        j++;
        while (ip_addr[j] != '\0')
            host[i++] = ip_addr[j++];
        printf("Network ID is %s\n", network);
        printf("Host ID is %s\n", host);
    }

    /// Class B
    else if (ip_class == 'B') {
```

```
        int i = 0, j = 0, dotCount = 0;
        while (dotCount < 2)
        {
            network[i++] = ip_addr[j++];
            if (ip_addr[j] == '.')
                dotCount++;
        }
        i = 0;
        j++;

        while (ip_addr[j] != '\0')
            host[i++] = ip_addr[j++];

        printf("Network ID is %s\n", network);
        printf("Host ID is %s\n", host);
    }

    /// Class C
    else if (ip_class == 'C') {

        int i = 0, j = 0, dotCount = 0;
        while (dotCount < 3)
        {
            network[i++] = ip_addr[j++];
            if (ip_addr[j] == '.')
                dotCount++;
        }

        i = 0;
        j++;

        while (ip_addr[j] != '\0')
            host[i++] = ip_addr[j++];

        printf("Network ID is %s\n", network);
        printf("Host ID is %s\n", host);
    }

    else
        printf("IP address is not divided into Network and Host ID in this
class\n");
}
```

```c
int main() {
    char ip_addr[14];
    printf("Enter IP address\n");
    scanf("%s", ip_addr);
    char ip_class = getClass(ip_addr);
    printf("Given IP address belongs to Class %c\n", ip_class);
    getNetworkAndHostID(ip_addr, ip_class);
    return 0;
}
```

SCREENSHOTS:



QUESTION 2: Write a C program to demonstrate File Transfer using UDP.

SOLUTION CODE CLIENT:

```
#include <arpa/inet.h>
#include <netinet/in.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/socket.h>
#include <sys/types.h>
#include <unistd.h>

#define IP_PROTOCOL 0
#define IP_ADDRESS "127.0.0.1" // localhost
#define PORT_NO 15050
#define NET_BUF_SIZE 32
#define cipherKey 'S'
#define sendrecvflag 0

/**
* Fuction clears buffer
* @param b: Buffer
*/
void clearBuf(char* b) {
    int i;
    for (i = 0; i < NET_BUF_SIZE; i++)
        b[i] = '\0';
}

/**
* Function encrypts using xor
* @param ch: The character to be decrypted
*/
char decrypt(char ch) {
    return ch ^ cipherKey;
}

/**
* Function receives file
* @param buf: The buffer
* @param s: The buffer size
```

```
*/
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;
}

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;

    /// create 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("\nEnter file name to receive:\n");
        scanf("%s", net_buf);
        printf("\nFile Requested!\n");
        sendto(sockfd, net_buf, NET_BUF_SIZE, sendrecvflag, (struct
sockaddr*)&addr_con, addrlen);

        printf("\n__Data Received__\n");
```

```
        while (1) {

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

            if (receiveFile(net_buf, NET_BUF_SIZE)) {
                break;
            }
        }

printf("\n-----------------------------------------------------------------
------\n");
    }
    return 0;
}
```

SOLUTION CODE SERVER:

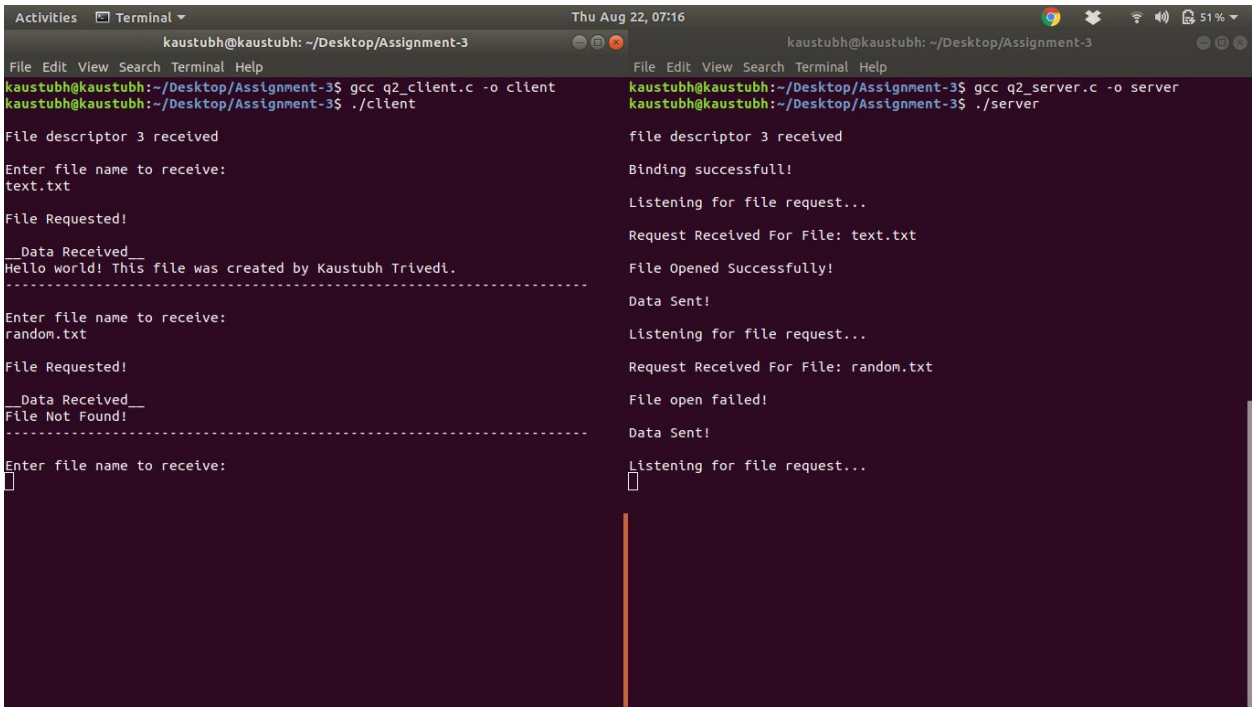
```
#include <arpa/inet.h>
#include <netinet/in.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/socket.h>
#include <sys/types.h>
#include <unistd.h>

#define IP_PROTOCOL 0
#define PORT_NO 15050
#define NET_BUF_SIZE 32
#define cipherKey 'S'
#define sendrecvflag 0
#define nofile "File Not Found!"

/**
* Function clears buffer
* @param b: Buffer
*/
void clearBuffer(char* b) {
```

```
    int i;
    for (i = 0; i < NET_BUF_SIZE; i++)
        b[i] = '\0';
}

/**
* Function encrypts using xor
* @param ch: Character to be encrypted
*/
char encrypt(char ch) {
    return ch ^ cipherKey;
}

/**
* Function sends file
* @param fp: FILE pointer
* @param buf: Buffer
* @param s: Buffer size
*/
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;
}

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;

    /// create 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);

    if (bind(sockfd, (struct sockaddr*)&addr_con, sizeof(addr_con)) == 0)
        printf("\nBinding successfull!\n");
    else
        printf("\nBinding Failed!\n");

    while (1) {
        printf("\nListening for file request...\n");

        /// receive file name
        clearBuffer(net_buf);

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

        printf("\nRequest Received For File: %s\n", net_buf);

        fp = fopen(net_buf, "r");

        if (fp == NULL)
            printf("\nFile open failed!\n");
        else
            printf("\nFile Opened Successfully!\n");

        while (1) {
            if (sendFile(fp, net_buf, NET_BUF_SIZE)) {
                sendto(sockfd, net_buf, NET_BUF_SIZE, sendrecvflag, (struct
```

```
sockaddr*)&addr_con, addrlen);
                break;
            }

            sendto(sockfd, net_buf, NET_BUF_SIZE, sendrecvflag, (struct
sockaddr*)&addr_con, addrlen);
            clearBuffer(net_buf);
        }

        printf("\nData Sent!\n");

        if (fp != NULL)
            fclose(fp);
    }
    return 0;
}
```

SCREENSHOTS:



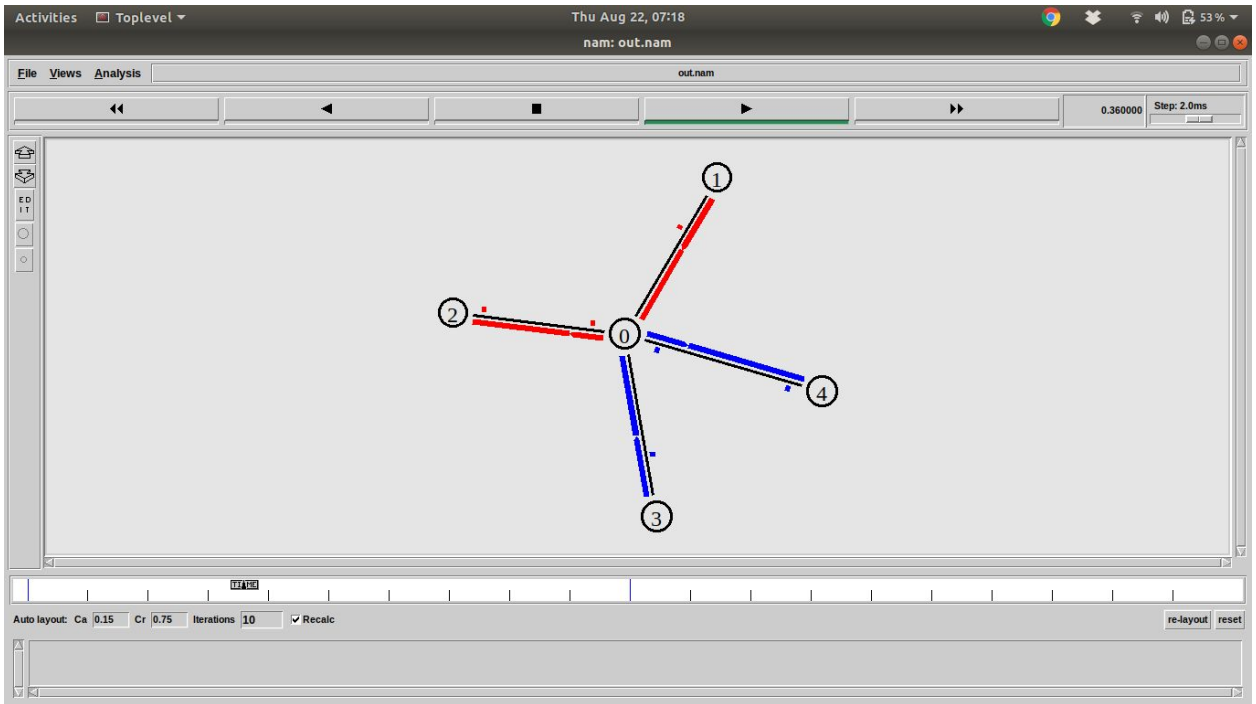
QUESTION 3: Write a TCL code for network simulator NS2 to demonstrate the star topology among a set of computer nodes. Given N nodes, one node will be assigned as the central node and the other nodes will be connected to it to form the star. You have to set up a TCP connection between k pairs of nodes 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 CODE:

```
set data [gets stdin]
scan $data "%d %d" N k

set ns  [new Simulator]

$ns color 0 Red
$ns color 1 Blue
$ns color 2 Yellow
$ns color 3 Green
$ns color 4 Orange
$ns color 5 Black

$ns rtproto DV

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

proc finish {} {
    global ns nf
    $ns flush-trace
    close $nf
    exec nam out.nam
    exit 0
}

for {set i 0} {$i < $N} {incr i} {
      set n($i) [$ns node]
}

for {set i 1} {$i < $N} {incr i} {
      $ns duplex-link $n($i) $n(0) 1Mb 10ms DropTail
```

```
}

for {set i 0} {$i < $k} {incr i} {
      set input [gets stdin]
      scan $input "%d %d" u v
      set tcp [new Agent/TCP]
      $ns attach-agent $n($u) $tcp
      $tcp set class_ $i
      $tcp set fid_ $i
      set sink [new Agent/TCPSink]
      $ns attach-agent $n($v) $sink
      $ns connect $tcp $sink
      set ftp0 [new Application/FTP]
      $ftp0 attach-agent $tcp
      $ns at 0.1 "$ftp0 start"
      $ns at 1.5 "$ftp0 stop"
}

$ns at 2.0 "finish"
$ns run
```

SCREENSHOTS:



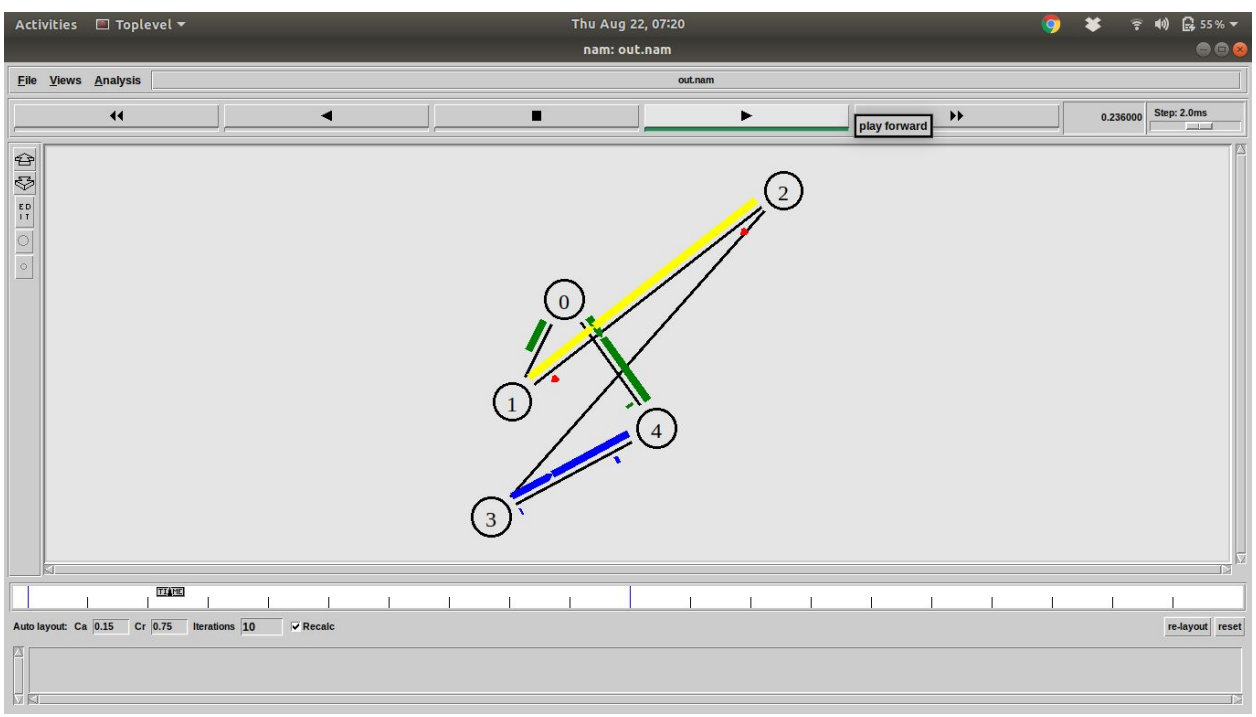
QUESTION 4: Write a TCL code for network simulator NS2 to demonstrate the ring topology among a set of computer nodes. Given N nodes, each node will be connected to two other nodes in the form of a ring. You have to set up a TCP connection between k pairs of nodes and demonstrate packet transfer between 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 CODE:

```
set data [gets stdin]
scan $data "%d %d" N k

set ns  [new Simulator]

$ns color 0 Red
$ns color 1 Blue
$ns color 2 Yellow
$ns color 3 Green
$ns color 4 Orange
$ns color 5 Black

$ns rtproto DV

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

proc finish {} {
    global ns nf
    $ns flush-trace
    close $nf
    exec nam out.nam
    exit 0
}

for {set i 0} {$i < $N} {incr i} {
      set n($i) [$ns node]
}

for {set i 0} {$i < $N} {incr i} {
      $ns duplex-link $n($i) $n([expr ($i + 1) % $N]) 1Mb 10ms DropTail
}
```

```
for {set i 0} {$i < $k} {incr i} {
      set input [gets stdin]
      scan $input "%d %d" u v
      set tcp [new Agent/TCP]
      $ns attach-agent $n($u) $tcp
      $tcp set class_ $i
      $tcp set fid_ $i
      set sink [new Agent/TCPSink]
      $ns attach-agent $n($v) $sink
      $ns connect $tcp $sink
      set ftp0 [new Application/FTP]
      $ftp0 attach-agent $tcp
      $ns at 0.1 "$ftp0 start"
      $ns at 1.5 "$ftp0 stop"
}

$ns at 2.0 "finish"
$ns run
```

SCREENSHOTS:



QUESTION 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 CODE:

```
set ns [new Simulator]
set colors(0) Red
set colors(1) Blue
set colors(2) Black
set colors(3) Pink
set colors(4) Yellow
set colors(5) Green

set f [open problem5.nam w]
$ns namtrace-all $f

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

    exec nam problem5.nam &
    exit 0
}
set input [gets stdin]
scan $input "%d %d" N k
set n(0) [$ns node]
set y "$n(0)"
for {set i 1} {$i < $N} {incr i} {
    set n($i) [$ns node]
    append y " "
    append y "$n($i)"
}
puts $y
puts "$n(0) $n(1)"
$ns make-lan $y 0.5Mb 40ms LL Queue/DropTail Mac/802_3
for {set i 0} {$i < $k} {incr i} {
```

```
    set input [gets stdin]
      scan $input "%d %d" i1 i2
    set tcp [new Agent/TCP]
    $tcp set class_ [expr $i%5]
    $ns attach-agent $n($i1) $tcp

    set sink [new Agent/TCPSink]
    $ns attach-agent $n($i2) $sink
    $ns connect $tcp $sink
      $ns color $i $colors([expr ($i) % 6])
    $tcp set fid_ $i

    set ftp($i) [new Application/FTP]
    $ftp($i) attach-agent $tcp
    # $ftp($i) set type_ FTP

    $ns at 0.1 "$ftp($i) start"
      $ns at 1.5 "$ftp($i) stop"
}
# for {set i 0} {$i < $k} {incr i} {
#     $ns at [expr ($i/10)+0.1] "$ftp($i) start"
#     $ns at [expr ($i/10)+1.5] "$ftp($i) stop"
# }
# $ns at [expr ($k/10)+1.5] "finish"
$ns at 2.0 "finish"
$ns run
```

SCREENSHOTS: