CSN361_L3

22.10.2019

Divyanshu Salve
CSE (III)
17114027

Q1: Write a socket program in C to determine class, Network and Host ID of an IPv4 address.

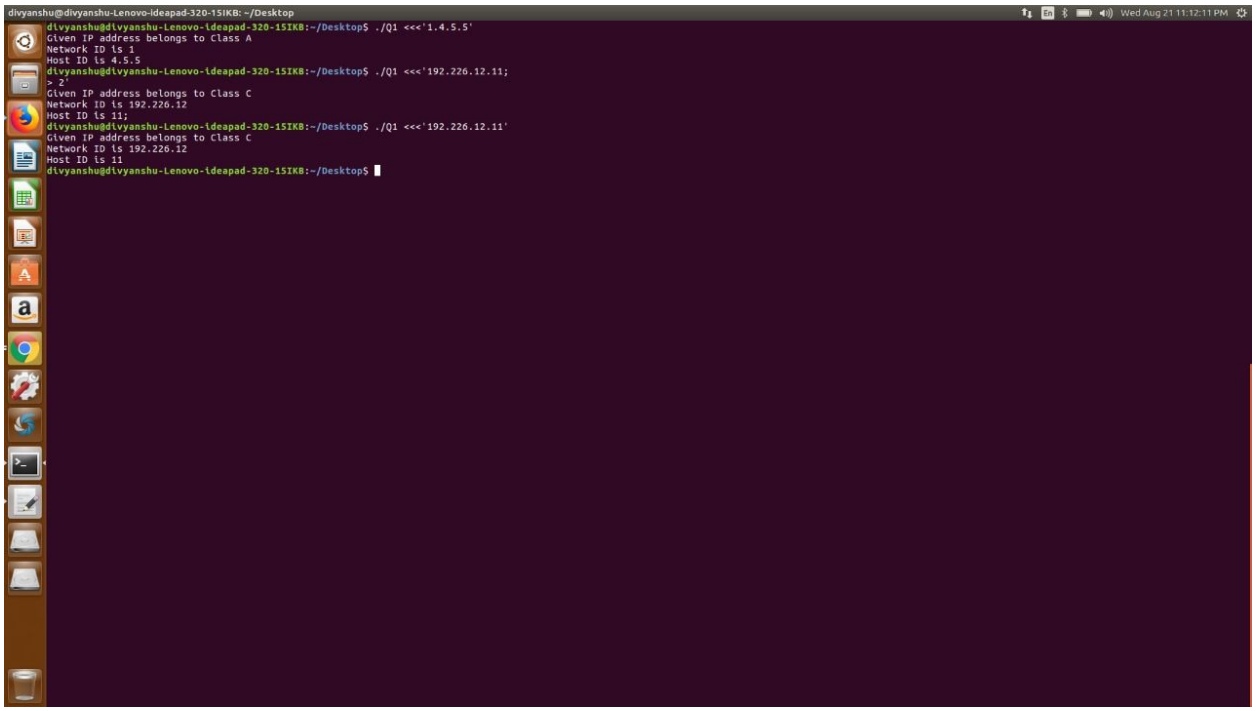
Sol:

```
char findClass(char str[])
{
    char arr[4];
    int i = 0;
    while (str[i] != '.')
    {
        arr[i] = str[i];
        i++;
    }
    i--;
    int ip = 0, j = 1;
    while (i >= 0)
    {
        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';
}
void separate(char str[], char ipClass)
{
    char network[12], host[12];
    for (int k = 0; k < 12; k++)
        network[k] = host[k] = '\0';
    if (ipClass == 'A')
    {
        int i = 0, j = 0;
        while (str[j] != '.')
            network[i++] = str[j++];
        i = 0;
        j++;
        while (str[j] != '\0')
            host[i++] = str[j++];
        printf("Network ID is %s\n", network);
        printf("Host ID is %s\n", host);
    }
    else if (ipClass == 'B')
    {
        int i = 0, j = 0, dotCount = 0;
        while (dotCount < 2)
        {
            network[i++] = str[j++];
            if (str[j] == '.')
```

```
            dotCount++;
        }
        i = 0;
        j++;
        while (str[j] != '\0')
            host[i++] = str[j++];

        printf("Network ID is %s\n", network);
        printf("Host ID is %s\n", host);
    }
    else if (ipClass == 'C')
    {
        int i = 0, j = 0, dotCount = 0;

        while (dotCount < 3)
        {
            network[i++] = str[j++];
            if (str[j] == '.')
                dotCount++;
        }
        i = 0;
        j++;
        while (str[j] != '\0')
            host[i++] = str[j++];
        printf("Network ID is %s\n", network);
        printf("Host ID is %s\n", host);
    }
    else
        printf("In this Class, IP address is not"
```

```
        " divided into Network and Host ID\n");
}
int main()
{
   char str[] = "192.226.12.11";
   gets(str);
   char ipClass = findClass(str);
   printf("Given IP address belongs to Class %c\n",
                                ipClass);
   separate(str, ipClass);
   return 0;
}
```



Q2. Write a C program to demonstrate File Transfer using UDP.

Sol:

Server:

```
// server code for UDP socket programming
#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!"

void clearBuf(char* b)
{
	int i;
	for (i = 0; i < NET_BUF_SIZE; i++)
		b[i] = '\0';
```

```
}

char Cipher(char ch)
{
	return ch ^ cipherKey;
}
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] = Cipher(buf[i]);
		return 1;
	}
	char ch, ch2;
	for (i = 0; i < s; i++) {
		ch = fgetc(fp);
		ch2 = Cipher(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;
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("\nSuccessfully binded!\n");
else
        printf("\nBinding Failed!\n");
while (1) {
        printf("\nWaiting for file name...\n");
        clearBuf(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) {
            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);
            clearBuf(net_buf);
        }
        if (fp != NULL)
            fclose(fp);
    }
    return 0;
}
```

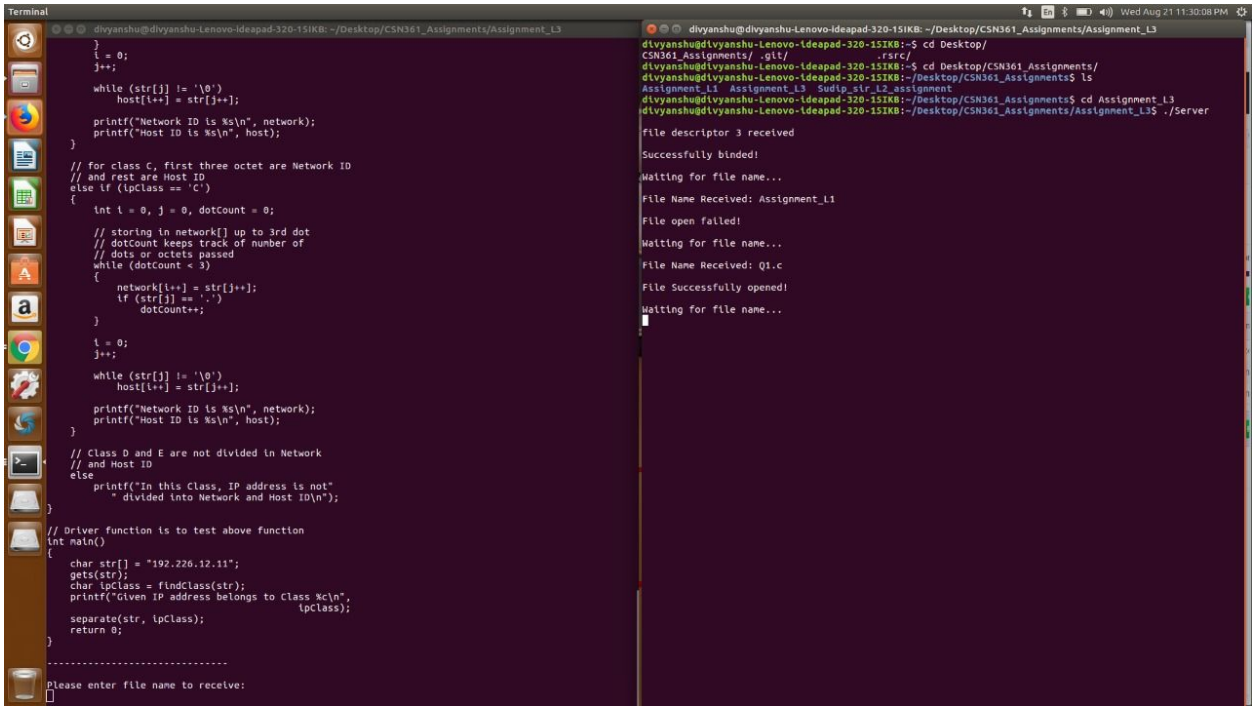
Client:

```
    // client code for UDP socket programming
#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

void clearBuf(char* b)
{
	int i;
	for (i = 0; i < NET_BUF_SIZE; i++)
		b[i] = '\0';
}
char Cipher(char ch)
{
	return ch ^ cipherKey;
}
int recvFile(char* buf, int s)
{
	int i;
	char ch;
	for (i = 0; i < s; i++) {
		ch = buf[i];
		ch = Cipher(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;
        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) {
```

```
                        clearBuf(net_buf);
                        nBytes = recvfrom(sockfd, net_buf, NET_BUF_SIZE,
                                                    sendrecvflag, (struct
sockaddr*)&addr_con,
                                                    &addrlen);
                        if (recvFile(net_buf, NET_BUF_SIZE)) {
                                break;
                        }
                }
                printf("\n------------------------------\n");
        }
        return 0;
}
```



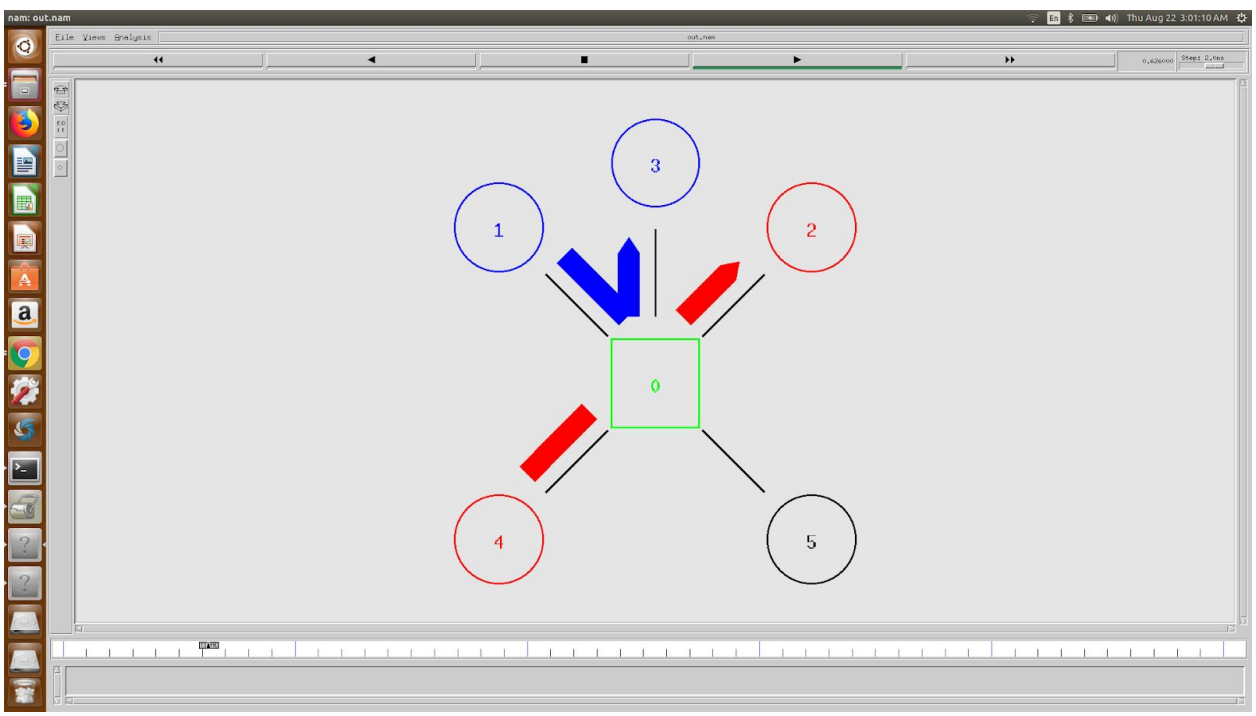
Q3: Star Topology Problem

```
#Divyanshu Salve
#17114027
#STAR TOPOLOGY



#Create a simulator object
set ns [new Simulator]

$ns color 1 Blue
$ns color 2 Red

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

#Define a 'finish' procedure
proc finish {} {
   global ns nf
   $ns flush-trace
   #Close the trace file
   close $nf
   #Executenam on the trace file
   exec nam out.nam &
   exit0
}

#Create four 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]

$n0 shape box
$n0 color green

$n4 color red
$n2 color red

$n1 color blue
$n3 color blue
#Create links between the nodes
$ns duplex-link $n0 $n1 1Mb 10ms DropTail

$ns duplex-link $n2 $n0 1Mb 10ms DropTail
$ns duplex-link $n3 $n0 1Mb 10ms DropTail
$ns duplex-link $n4 $n0 1Mb 10ms DropTail
$ns duplex-link $n5 $n0 1Mb 10ms DropTail

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

```
#Create a TCP agent and attach it to node n0
set tcp0 [new Agent/TCP]
$ns attach-agent $n1 $tcp0
#Create a TCP Sink agent (a traffic sink) for TCP and attach it to node n3
set sink0 [new Agent/TCPSink]
$ns attach-agent $n3 $sink0
#Connect the traffic sources with the traffic sink
$ns connect $tcp0 $sink0

#Create a TCP agent and attach it to node n0
set tcp1 [new Agent/TCP]
$ns attach-agent $n4 $tcp1
#Create a TCP Sink agent (a traffic sink) for TCP and attach it to node n3
set sink1 [new Agent/TCPSink]
$ns attach-agent $n2 $sink1
#Connect the traffic sources with the traffic sink
$ns connect $tcp1 $sink1

$tcp0 set fid_ 1
$tcp1 set fid_ 2
# Create a FTP and attach it to tcp0
set ftp0 [new Application/FTP]
$ftp0 attach-agent $tcp1

# Create a CBR traffic source and attach it to tcp0
set cbr0 [new Application/Traffic/CBR]
$cbr0 attach-agent $tcp0
```

```
#Schedule events for the CBR agents
$ns at 0.5 "$cbr0 start"
$ns at 0.5 "$ftp0 start"

$ns at 3.5 "$ftp0 stop"
$ns at 4.5 "$cbr0 stop"


#Call the finish procedure after 5 seconds of simulation time
$ns at 5.0 "finish"

#Run the simulation
$ns run
```

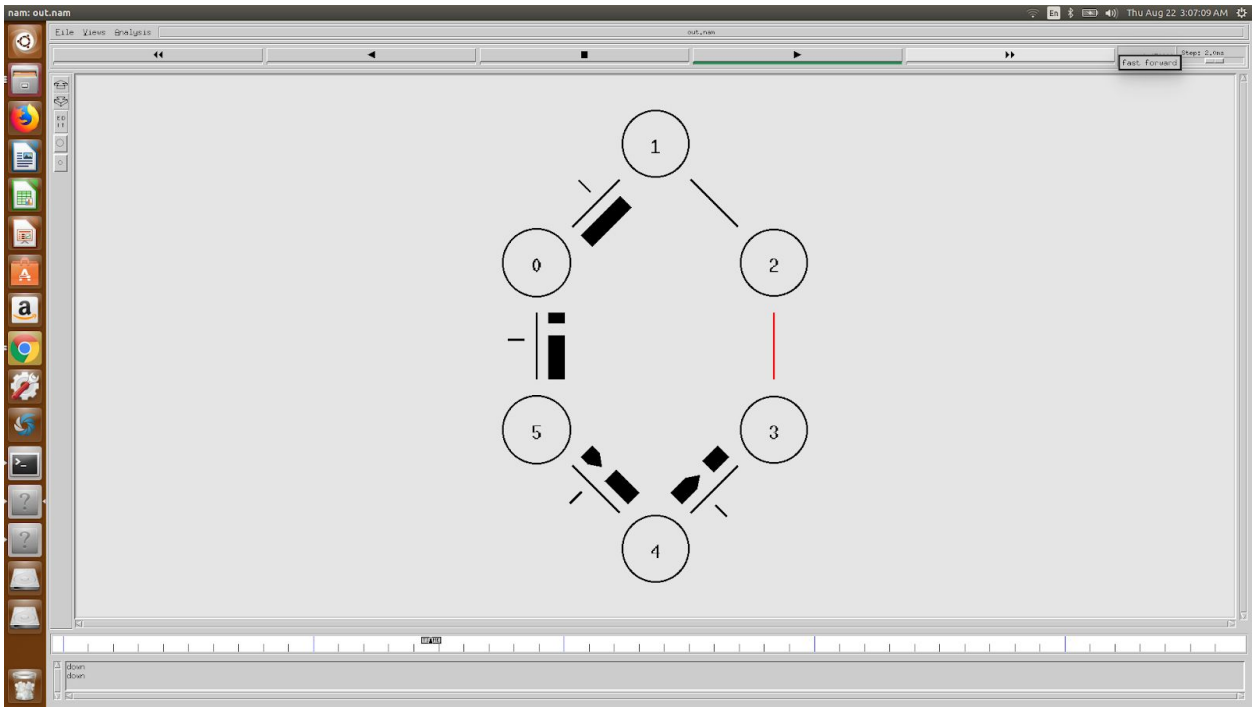


Q4: Ring Topology

```
#Divyanshu Salve
#17114027
#RING TOPOLOGY

#Create a simulator object
set ns [new Simulator]

#Routing Protocol used is Distance Vector
$ns rtproto DV
#Open the nam trace file
set nf [open out.nam w]
$ns namtrace-all $nf

#Define a 'finish' procedure
proc finish {} {
   global ns nf
   $ns flush-trace
   #Close the trace file
   close $nf
   #Executenam on the trace file
   exec nam out.nam &
   exit0
}

#Create four 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]

#Create links between the nodes
$ns duplex-link $n0 $n1 1Mb 10ms DropTail
$ns duplex-link $n1 $n2 1Mb 10ms DropTail
$ns duplex-link $n2 $n3 1Mb 10ms DropTail
$ns duplex-link $n3 $n4 1Mb 10ms DropTail
$ns duplex-link $n4 $n5 1Mb 10ms DropTail
$ns duplex-link $n5 $n0 1Mb 10ms DropTail

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


#Create a TCP agent and attach it to node n0
set tcp0 [new Agent/TCP]
$tcp0 set class_ 1
$ns attach-agent $n1 $tcp0
#Create a TCP Sink agent (a traffic sink) for TCP and attach it to node n3
set sink0 [new Agent/TCPSink]
$ns attach-agent $n3 $sink0
#Connect the traffic sources with the traffic sink
```

```
$ns connect $tcp0 $sink0


# Create a CBR traffic source and attach it to tcp0
set cbr0 [new Application/Traffic/CBR]
$cbr0 set packetSize_ 500
$cbr0 set interval_ 0.01
$cbr0 attach-agent $tcp0

#Schedule events for the CBR agents
$ns at 0.5 "$cbr0 start"

# Break link n2---n3 at 1.3ms
$ns rtmodel-at 1.3 down $n2 $n3
$ns at 4.5 "$cbr0 stop"


#Call the finish procedure after 5 seconds of simulation time
$ns at 5.0 "finish"

#Run the simulation
$ns run
```


nam: out.nam
Thu Aug 22 3:07:09 AM
File Views Analysis
out.nam
Fast Forward
1
0
2
5
3
4

Q5: Bus Topology

```
## \file problem5.tcl
# Problem Statement 5 : Demonstrating Bus Topology
#\verbatim

set ns [new Simulator]

$ns color 0 Red
$ns color 1 Green
$ns color 2 Coral
$ns color 3 Blue
$ns color 4 Azure

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
}
puts "Enter no. of Nodes: "
gets stdin N
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 $y 0.5Mb 40ms LL Queue/DropTail Mac/802_3
puts "Enter k: "
gets stdin k
for {set i 0} {$i < $k} {incr i} {
    gets stdin i1
    gets stdin i2
    set tcp [new Agent/TCP]
    $tcp set class_ [expr $i%5]
    $ns attach-agent $n($i1) $tcp

    set sink [new Agent/TCPSink]
    $ns attach-agent $n($i2) $sink
```

```
    $ns connect $tcp $sink
    $tcp set fid_ $i

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