

Computer Networks Laboratory CSN361

Lab Assignment 3

Ashutosh Chaubey (17114015)

Problem Statement 1

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

Algorithm:

- Extract the first octet
- Decide the class based on the value of the first octet
- Decide the number of octets in Network and Host ID based on the class

Data Structures:

Char array to store the ip address

Code:

```
problem1.c x problem2_server.c x problem2_client.c x problem3.tcl x
1  /** @file problem1.c
2   * @brief Problem Statement 1 : program in C to determine class, Network and Host ID of an IPv4 address.
3   * @author Ashutosh Chaubey
4   */
5
6  #include<stdio.h>
7  #include<string.h>
8
9  /** @brief Function to separate Network ID as well as Host ID and print them
10   */
11  void separate(char str[], char ipClass)
12  {
13      // Initializing network and host array to NULL
14      char network[12], host[12];
15      for (int k = 0; k < 12; k++)
16          network[k] = host[k] = '\0';
17
18      // for class A, only first octet is Network ID
19      // and rest are Host ID
20      if (ipClass == 'A')
21      {
22          int i = 0, j = 0;
23          while (str[j] != '.')
24              network[i++] = str[j++];
25          i = 0;
26          j++;
27          while (str[j] != '\0')
28              host[i++] = str[j++];
29          printf("Network ID of given IPv4 %s\n", network);
30          printf("Host ID of given IPv4 %s\n", host);
31      }
32
33      // for class B, first two octet are Network ID
34      // and rest are Host ID
35      else if (ipClass == 'B')
36      {
37          int i = 0, j = 0, dotCount = 0;
38
39          // storing in network[] up to 2nd dot
40          // dotCount keeps track of number of
41          // dots or octets passed
42          while (dotCount < 2)
43          {
44              network[i++] = str[j++];
45              if (str[j] == '.')
46                  dotCount++;
47          }
48          i = 0;
49          j++;
50
51          while (str[j] != '\0')
52              host[i++] = str[j++];
53
54          printf("Network ID of given IPv4 %s\n", network);
55          printf("Host ID of given IPv4 %s\n", host);
56      }
57
58      // for class C, first three octet are Network ID
59      // and rest are Host ID
60      else if (ipClass == 'C')
61      {
62          int i = 0, j = 0, dotCount = 0;
63
64          // storing in network[] up to 3rd dot
65          // dotCount keeps track of number of
66          // dots or octets passed
67          while (dotCount < 3)
```

```

67     while (dotCount < 3)
68     {
69         network[i++] = str[j++];
70         if (str[j] == '.')
71             dotCount++;
72     }
73
74     i = 0;
75     j++;
76
77     while (str[j] != '\0')
78         host[i++] = str[j++];
79
80     printf("Network ID of given IPv4 %s\n", network);
81     printf("Host ID of given IPv4 %s\n", host);
82 }
83
84 // Class D and E are not divided in Network
85 // and Host ID
86 else
87     printf("In this Class, IP address is not"
88           " divided into Network and Host ID\n");
89 }
90
91 /** @brief Function to find out the class
92  */
93 char findClass(char str[])
94 {
95     // storing first octet in arr[] variable
96     char arr[4];
97     int i = 0;
98     while (str[i] != '.')
99     {
100         arr[i] = str[i];
101         i++;
102     }
103     i--;
104
105     // converting str[] variable into number for
106     // comparison
107     int ip = 0, j = 1;
108     while (i >= 0)
109     {
110         ip = ip + (str[i] - '0') * j;
111         j = j * 10;
112         i--;
113     }
114
115     // Class A
116     if (ip >= 1 && ip <= 126)
117         return 'A';
118
119     // Class B
120     else if (ip >= 128 && ip <= 191)
121         return 'B';
122
123     // Class C
124     else if (ip >= 192 && ip <= 223)
125         return 'C';
126
127     // Class D
128     else if (ip >= 224 && ip <= 239)
129         return 'D';
130

```

```

131     // Class E
132     else
133     {
134         return 'E';
135     }
136
137     /** @brief Problem Statement 1 entrypoint.
138     */
139     int main()
140     {
141         char str[] = "200.226.12.20";
142         char ipClass = findClass(str);
143         printf("Given IPv4 address belongs to Class %c\n",
144               ipClass);
145         separate(str, ipClass);
146         return 0;
147     }

```

Output

```

(base) djikstra@helios:~/Academic/CSN361/L3$ ./problem1
Input IP : 200.226.12.20
Given IPv4 address belongs to Class C
Newtork ID of given IPv4 200.226.12
Host ID of given IPv4 20
(base) djikstra@helios:~/Academic/CSN361/L3$

```

Problem Statement 2

Write a C program to demonstrate File Transfer using UDP.

Algorithm:

- Create a socket for the server and the client
- Make the socket ready to receive file name to be read
- Send file name from client

Data Structures:

- Int fd: File descriptor
- Sockaddr_in : TO store the info about address

Code:

Server -

```
problem1.c x problem2_server.c x problem2_client.c x
1  /** @file problem2_server.c
2   * @brief Problem Statement 2 : C program to demonstrate File Transfer using UDP.
3   * @author Ashutosh Chaubey
4   */
5
6  #include <arpa/inet.h>
7  #include <netinet/in.h>
8  #include <stdio.h>
9  #include <stdlib.h>
10 #include <string.h>
11 #include <sys/socket.h>
12 #include <sys/types.h>
13 #include <unistd.h>
14
15 #define IP_PROTOCOL 0
16 #define PORT_NO 15050
17 #define NET_BUF_SIZE 32
18 #define cipherKey 'S'
19 #define sendrecvflag 0
20 #define nofile "File Not Found!"
21
22
23 /** @brief function to encrypt
24  */
25 char Cipher(char ch)
26 {
27     return ch ^ cipherKey;
28 }
29
30 /** @brief function sending file
31  */
32 int sendFile(FILE* fp, char* buf, int s)
33 {
34     int i, len;
35     if (fp == NULL) {
36         strcpy(buf, nofile);
37         len = strlen(nofile);
38         buf[len] = EOF;
39         for (i = 0; i <= len; i++)
40             buf[i] = Cipher(buf[i]);
41         return 1;
42     }
43
44     char ch, ch2;
45     for (i = 0; i < s; i++) {
46         ch = fgetc(fp);
47         ch2 = Cipher(ch);
48         buf[i] = ch2;
49         if (ch == EOF)
50             return 1;
51     }
52     return 0;
53 }
54
55 /** @brief function to clear buffer
56  */
57 void clearBuf(char* b)
58 {
59     int i;
60     for (i = 0; i < NET_BUF_SIZE; i++)
61         b[i] = '\0';
62 }
63
```

```

55  /** @brief function to clear buffer
56  */
57  void clearBuf(char* b)
58  {
59      int i;
60      for (i = 0; i < NET_BUF_SIZE; i++)
61          b[i] = '\0';
62  }
63
64  /** @brief Problem Statement 2 entrypoint.
65  */
66  int main()
67  {
68      int sockfd, nBytes;
69      struct sockaddr_in addr_con;
70      int addrlen = sizeof(addr_con);
71      addr_con.sin_family = AF_INET;
72      addr_con.sin_port = htons(PORT_NO);
73      addr_con.sin_addr.s_addr = INADDR_ANY;
74      char net_buf[NET_BUF_SIZE];
75      FILE* fp;
76
77      // socket()
78      sockfd = socket(AF_INET, SOCK_DGRAM, IP_PROTOCOL);
79
80      if (sockfd < 0)
81          printf("\nfile descriptor not received!!\n");
82      else
83          printf("\nfile descriptor %d received\n", sockfd);
84
85      // bind()
86      if (bind(sockfd, (struct sockaddr*)&addr_con, sizeof(addr_con)) == 0)
87          printf("\nSuccessfully binded!\n");
88      else
89          printf("\nBinding Failed!\n");
90
91      while (1) {
92          printf("\nWaiting for file name...\n");
93
94          // receive file name
95          clearBuf(net_buf);
96
97          nBytes = recvfrom(sockfd, net_buf,
98                          NET_BUF_SIZE, sendrecvflag,
99                          (struct sockaddr*)&addr_con, &addrlen);
100
101          fp = fopen(net_buf, "r");
102          printf("\nFile Name Received: %s\n", net_buf);
103          if (fp == NULL)
104              printf("\nFile open failed!\n");
105          else
106              printf("\nFile Successfully opened!\n");
107
108          while (1) {
109              // process
110              if (sendFile(fp, net_buf, NET_BUF_SIZE)) {
111                  sendto(sockfd, net_buf, NET_BUF_SIZE,
112                          sendrecvflag,
113                          (struct sockaddr*)&addr_con, addrlen);
114                  break;
115              }
116
117              // send
118              sendto(sockfd, net_buf, NET_BUF_SIZE,
119                      sendrecvflag,
120                      (struct sockaddr*)&addr_con, addrlen);
121              clearBuf(net_buf);

```



```

123     }
124     if (fp != NULL)
125         fclose(fp);
126 }
127 return 0;
128 }

```

Client -

```

problem1.c  x  problem2_server.c  x  problem2_client.c  x
1  /** @file problem2_client.c
2  *  @brief Problem Statement 2 : C program to demonstrate File Transfer using UDP.
3  *  @author Ashutosh Chaubey
4  */
5
6  #include <arpa/inet.h>
7  #include <netinet/in.h>
8  #include <stdio.h>
9  #include <stdlib.h>
10 #include <string.h>
11 #include <sys/socket.h>
12 #include <sys/types.h>
13 #include <unistd.h>
14
15 #define IP_PROTOCOL 0
16 #define IP_ADDRESS "127.0.0.1" // localhost
17 #define PORT_NO 15050
18 #define NET_BUF_SIZE 32
19 #define cipherKey 'S'
20 #define sendrecvflag 0
21
22 /** @brief function to clear buffer
23 */
24 void clearBuf(char* b)
25 {
26     int i;
27     for (i = 0; i < NET_BUF_SIZE; i++) {
28         b[i] = '\0';
29     }
30 }
31
32 /** @brief function for decryption
33 */
34 char Cipher(char ch)
35 {
36     return ch ^ cipherKey;
37 }
38
39 /** @brief function to receive file
40 */
41 int recvFile(char* buf, int s)
42 {
43     int i;
44     char ch;
45     for (i = 0; i < s; i++) {
46         ch = buf[i];
47         ch = Cipher(ch);
48         if (ch == EOF)
49             return 1;
50         else
51             printf("%c", ch);
52     }
53     return 0;
54 }
55
56
57 /** @brief Problem Statement 2 entrypoint.
58 */
59 int main()
60 {
61     int sockfd, nBytes;
62     struct sockaddr_in addr_con;
63     int addrlen = sizeof(addr_con);
64     addr_con.sin_family = AF_INET;
65     addr_con.sin_port = htons(PORT_NO);
66     addr_con.sin_addr.s_addr = inet_addr(IP_ADDRESS);
67     char net_buf[NET_BUF_SIZE];
68     FILE* fp;

```

```

69
70 // socket()
71 sockfd = socket(AF_INET, SOCK_DGRAM,
72                IP_PROTOCOL);
73
74 if (sockfd < 0)
75     printf("\nfile descriptor not received!!\n");
76 else
77     printf("\nfile descriptor %d received\n", sockfd);
78
79 while (1) {
80     printf("\nPlease enter file name to receive:\n");
81     scanf("%s", net_buf);
82     sendto(sockfd, net_buf, NET_BUF_SIZE,
83            sendrecvflag, (struct sockaddr*)&addr_con,
84            addrlen);
85
86     printf("\n-----Data Received-----\n");
87
88     while (1) {
89         // receive
90         clearBuf(net_buf);
91         nBytes = recvfrom(sockfd, net_buf, NET_BUF_SIZE,
92                           sendrecvflag, (struct sockaddr*)&addr_con,
93                           &addrlen);
94
95         // process
96         if (recvFile(net_buf, NET_BUF_SIZE)) {
97             break;
98         }
99     }
100     printf("\n-----\n");
101 }
102 return 0;
103 }

```

Output

```

(base) djikstra@helios:~/Academic/CSN361/L3$ gcc problem2_server.c -o problem2_server
(base) djikstra@helios:~/Academic/CSN361/L3$ ./problem2_server

```

```

file descriptor 3 received

```

```

Successfully binded!

```

```

Waiting for file name...

```

```
(base) djikstra@helios:~/Academic/CSN361/L3$ ./problem2_client  
file descriptor 3 received  
Please enter file name to receive:  
test.txt  
  
-----Data Received-----  
HELLO WORLD  
-----  
  
Please enter file name to receive:  
█
```

```
(base) djikstra@helios:~/Academic/CSN361/L3$ gcc problem2_server.c -o problem2_server  
(base) djikstra@helios:~/Academic/CSN361/L3$ ./problem2_server  
file descriptor 3 received  
Successfully binded!  
Waiting for file name...  
File Name Received: test.txt  
File Successfully opened!  
Waiting for file name...  
█
```

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.

Algorithm:

- Create an empty nam file
- Bind the nam file to current tcl code's nam-trace
- Input n and k pairs
- Create n nodes and connect every created node to first node
- Create k FTP connections among specified nodes
- Specify time to start and stop transaction

Data Structures:

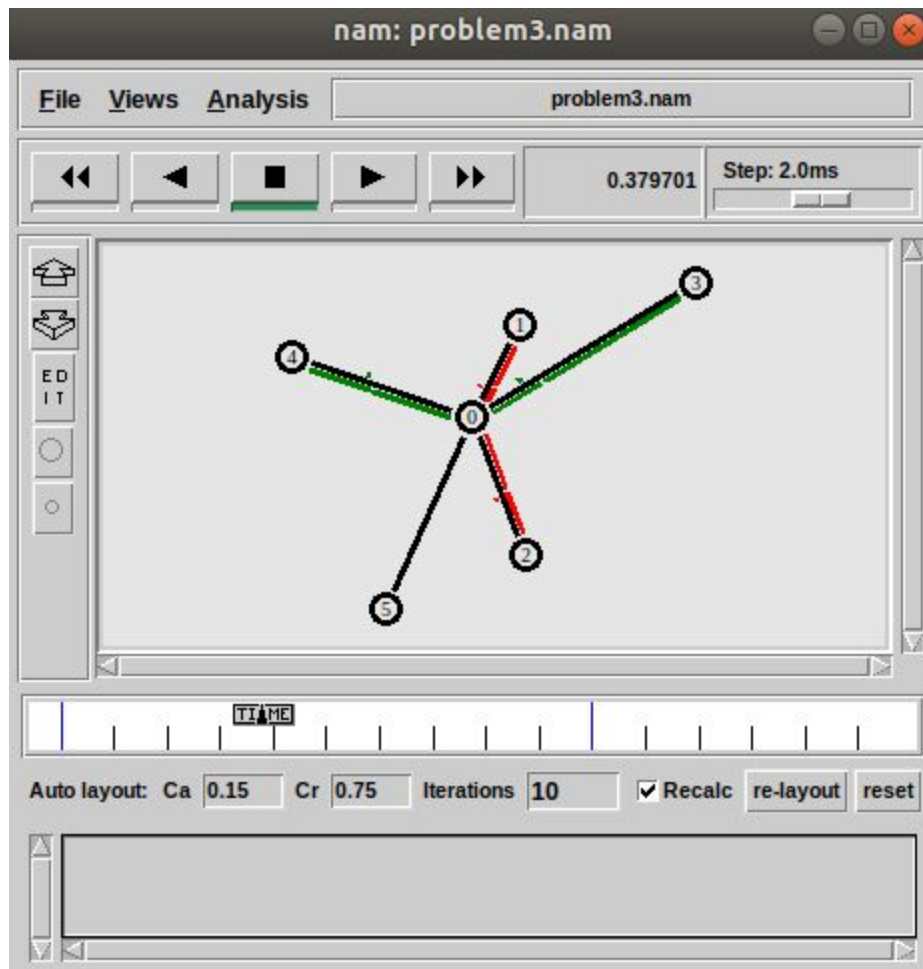
- Ns node to represent nodes

Code:

```
problem1.c x problem2_server.c x problem2_client.c x problem3.tcl x
1 1 # \file problem3.tcl
2 2 # Problem Statement 3 : Demonstrating Star Topology
3 3 #\verbatim
4 4
5 5 set ns [new Simulator]
6 6
7 7 $ns color 0 Red
8 8 $ns color 1 Green
9 9 $ns color 2 Coral
10 10 $ns color 3 Blue
11 11 $ns color 4 Azure
12 12
13 13 set f [open problem3.nam w]
14 14 $ns namtrace-all $f
15 15
16 16 proc finish {} {
17 17     global ns f
18 18     $ns flush-trace
19 19     close $f
20 20
21 21     exec nam problem3.nam &
22 22     exit 0
23 23 }
24 24 puts "Enter no. of Nodes: "
25 25 gets stdin N
26 26 set n(0) [$ns node]
27 27 for {set i 1} {$i < $N} {incr i} {
28 28     set n($i) [$ns node]
29 29     $ns duplex-link $n($i) $n(0) 1Mb 10ms DropTail
30 30 }
31 31 puts "Enter k: "
32 32 gets stdin k
33 33 for {set i 0} {$i < $k} {incr i} {
34 34     gets stdin i1
35 35     gets stdin i2
36 36     set tcp [new Agent/TCP]
37 37     $tcp set class_ [expr ($i+1)%5]
38 38     $ns attach-agent $n($i1) $tcp
39 39
40 40     set sink [new Agent/TCPSink]
41 41     $ns attach-agent $n($i2) $sink
42 42     $ns connect $tcp $sink
43 43     $tcp set fid_ $i
44 44
45 45     set ftp($i) [new Application/FTP]
46 46     $ftp($i) attach-agent $tcp
47 47     $ftp($i) set type_ FTP
48 48 }
49 49
50 50 for {set i 0} {$i < $k} {incr i} {
51 51     $ns at [expr ($i/10)+0.1] "$ftp($i) start"
52 52     $ns at [expr ($i/10)+1.5] "$ftp($i) stop"
53 53 }
54 54 $ns at [expr ($k/10)+1.5] "finish"
55 55
56 56 $ns run
```

Output

```
(base) djikstra@helios:~/Academic/CSN361/L3$ ns problem3.tcl
Enter no. of Nodes:
6
Enter k:
2
1
2
3
4
(base) djikstra@helios:~/Academic/CSN361/L3$
```



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.

Algorithm:

- Create an empty nam file
- Bind the nam file to current tcl code's nam-trace
- Input n and k pairs
- Create n nodes and connect every created node to it's previous node
- Connect last node to the first node
- Create k FTP connections among specified nodes
- Specify time to start and stop transaction

Data Structures:

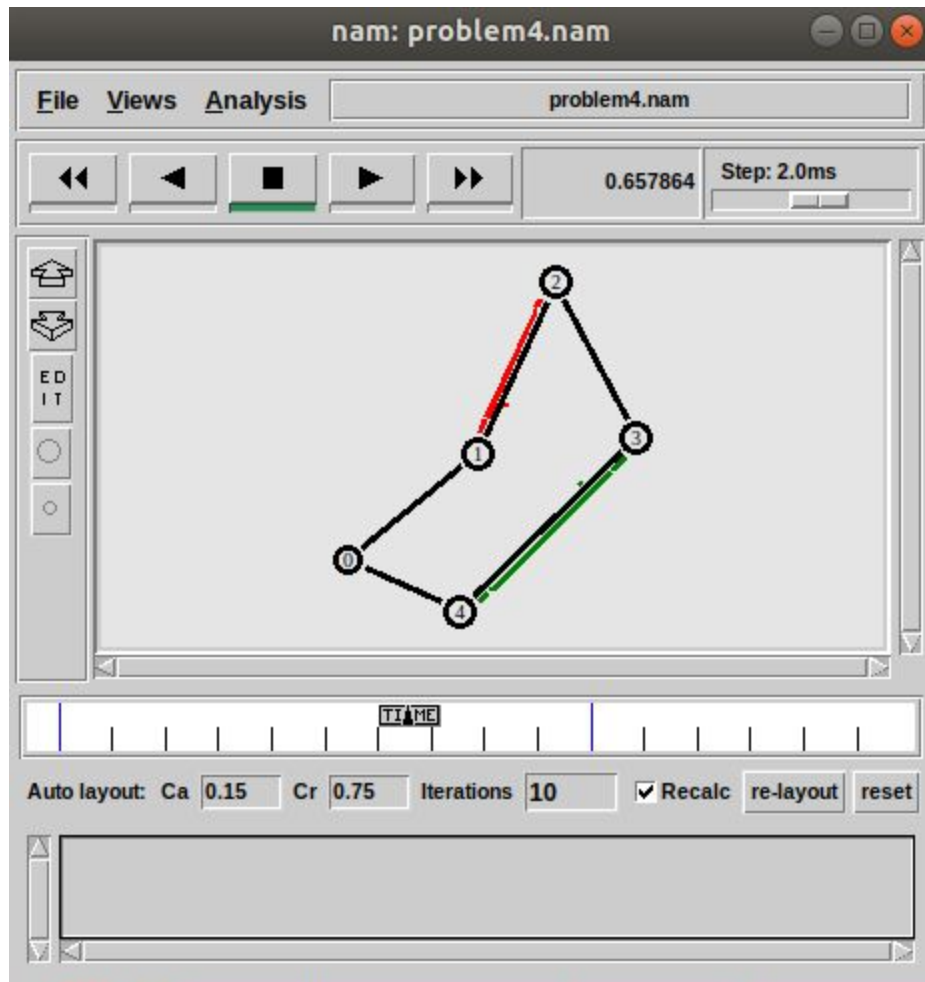
- Ns node to represent nodes

Code:

```
1  ## \file problem4.tcl
2  # Problem Statement 4 : Demonstrating Ring Topology
3  #\verbatim
4
5  set ns [new Simulator]
6
7  $ns color 0 Red
8  $ns color 1 Green
9  $ns color 2 Coral
10 $ns color 3 Blue
11 $ns color 4 Azure
12
13 set f [open problem4.nam w]
14 $ns namtrace-all $f
15
16 proc finish {} {
17     global ns f
18     $ns flush-trace
19     close $f
20
21     exec nam problem4.nam &
22     exit 0
23 }
24 puts "Enter no. of Nodes: "
25 gets stdin N
26 set n(0) [$ns node]
27 set y 0
28 for {set i 1} {$i < $N} {incr i} {
29     set n($i) [$ns node]
30     $ns duplex-link $n($y) $n($i) 1Mb 10ms DropTail
31     set y $i
32 }
33 $ns duplex-link $n($y) $n(0) 1Mb 10ms DropTail
34 puts "Enter k: "
35 gets stdin k
36 for {set i 0} {$i < $k} {incr i} {
37     gets stdin l1
38     gets stdin l2
39     set tcp [new Agent/TCP]
40     $tcp set class_ [expr $i%5]
41     $ns attach-agent $n($l1) $tcp
42
43     set sink [new Agent/TCPSink]
44     $ns attach-agent $n($l2) $sink
45     $ns connect $tcp $sink
46     $tcp set fid_ $i
47
48     set ftp($i) [new Application/FTP]
49     $ftp($i) attach-agent $tcp
50     $ftp($i) set type_ FTP
51 }
52 for {set i 0} {$i < $k} {incr i} {
53     $ns at [expr ($i/10)+0.1] "$ftp($i) start"
54     $ns at [expr ($i/10)+1.5] "$ftp($i) stop"
55 }
56 $ns at [expr ($k/10)+1.5] "finish"
57
58 $ns run
```

Output

```
(base) djikstra@helios:~/Academic/CSN361/L3$ ns problem4.tcl
Enter no. of Nodes:
5
Enter k:
2
1
2
3
4
(base) djikstra@helios:~/Academic/CSN361/L3$
```

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.

Algorithm:

- Create an empty nam file
- Bind the nam file to current tcl code's nam-trace
- Input n and k pairs
- Create n nodes
- Make a LAN connection among the nodes
- Create k FTP connections among specified nodes
- Specify time to start and stop transaction

Data Structures:

- Ns node to represent nodes

Code:

```
problem1.c x problem2_server.c x problem2_client.c x problem3.tcl x problem4.tcl x problem5.tcl x
13 set f [open 5.nam w]
14 $ns namtrace-all $f
15
16 proc finish {} {
17     global ns f
18     $ns flush-trace
19     close $f
20
21     exec nam 5.nam &
22     exit 0
23 }
24 puts "Enter no. of Nodes: "
25 gets stdin N
26 set n(0) [$ns node]
27 set y "$n(0)"
28 for {set i 1} {$i < $N} {incr i} {
29     set n($i) [$ns node]
30     append y " "
31     append y "$n($i)"
32 }
33 puts $y
34 puts "$n(0) $n(1)"
35 $ns make-lan $y 0.5Mb 40ms LL Queue/DropTail Mac/802_3
36 puts "Enter k: "
37 gets stdin k
38 for {set i 0} {$i < $k} {incr i} {
39     gets stdin i1
40     gets stdin i2
41     set tcp [new Agent/TCP]
42     $tcp set class [expr $i%5]
43     $ns attach-agent $n($i1) $tcp
44
45     set sink [new Agent/TCPSink]
46     $ns attach-agent $n($i2) $sink
47     $ns connect $tcp $sink
48     $tcp set fid_ $i
49
50     set ftp($i) [new Application/FTP]
51     $ftp($i) attach-agent $tcp
52     $ftp($i) set type_ FTP
53 }
54 for {set i 0} {$i < $k} {incr i} {
55     $ns at [expr ($i/10)+0.1] "$ftp($i) start"
56     $ns at [expr ($i/10)+1.5] "$ftp($i) stop"
57 }
58 $ns at [expr ($k/10)+1.5] "finish"
59
60 $ns run
```

Output

```
(base) djkstra@helios:~/Academic/CSN361/L3$ ns problem5.tcl
Enter no. of Nodes:
3
_o10 _o13 _o16
_o10 _o13
warning: no class variable LanRouter::debug_
        see tcl-object.tcl in tclcl for info about this warning.
Enter k:
1
1
2
(base) djkstra@helios:~/Academic/CSN361/L3$
```

