CSN - 361

Lab Assignment - 3

Bhavye Jain
17114020

Problem Statement 1

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

SOLUTION:

The program first reads the first octet of the IP address. This is used to indicate the class of the address as follows:

1 - 126	:	A
128 - 191	:	B
192 - 223	:	C
224 - 239	:	D
240 - 255	:	E

The Network ID is indicated in terms of octets as follows:

Class A	:	first 1 octet
Class B	:	first 2 octets
Class C	:	first 3 octets

The remaining part gives the Host ID.

Classes D and E are not divided into Network and Host IDs.

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

// Function to find out the Class
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)
```

```
        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 to get Network ID as well as Host ID
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");
}

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;
}
```

```
bhavye@ultron:~/CSN361$ ./q1
Enter IP address
1.4.5.5
Given IP address belongs to Class A
Network ID is 1
Host ID is 4.5.5
```

```
bhavye@ultron:~/CSN361$ ./q1
Enter IP address
192.226.12.11
Given IP address belongs to Class C
Network ID is 192.226.12
Host ID is 11
bhavye@ultron:~/CSN361$
```

Problem Statement 2

Write a C program to demonstrate File Transfer using UDP.

SOLUTION:

Server Program:

```c
#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 to clear buffer
void clearBuffer(char* b) {
    int i;
    for (i = 0; i < NET_BUF_SIZE; i++)
        b[i] = '\0';
}

// function to encrypt using xor
char encrypt(char ch) {
    return ch ^ cipherKey;
}

// function to send 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;
}

// 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 = 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;
}
```

Client Program:

```
#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

// function to clear buffer
void clearBuf(char* b) {
    int i;
    for (i = 0; i < NET_BUF_SIZE; i++)
        b[i] = '\0';
}
```

```
// function for decryption
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;
    // 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;
}
```

Run server:

```
bhavye@ultron:~/CSN361$ ./q2_server

file descriptor 3 received

Binding successfull!

Listening for file request...
```

Run client:

```
bhavye@ultron:~/CSN361$ ./q2_client

file descriptor 3 received

Enter file name to receive:
```

Specify the file to fetch:

```
bhavye@ultron:~/CSN361$ ./q2_client

file descriptor 3 received

Enter file name to receive:
text.txt
```

Server responds:

```
bhavye@ultron:~/CSN361$ ./q2_server

file descriptor 3 received

Binding successfull!

Listening for file request...

Request Received For File: text.txt

File Opened Successfully!

Data Sent!

Listening for file request...
```

Client receives file:

```
bhavye@ultron:~/CSN361$ ./q2_client

file descriptor 3 received

Enter file name to receive:
text.txt

File Requested!

--------------Data Received--------------
Hello world! This is a test text for assignment 2
of CSN_361 Computer Networks Laboratory!
-----------------------------------------

Enter file name to receive:
```

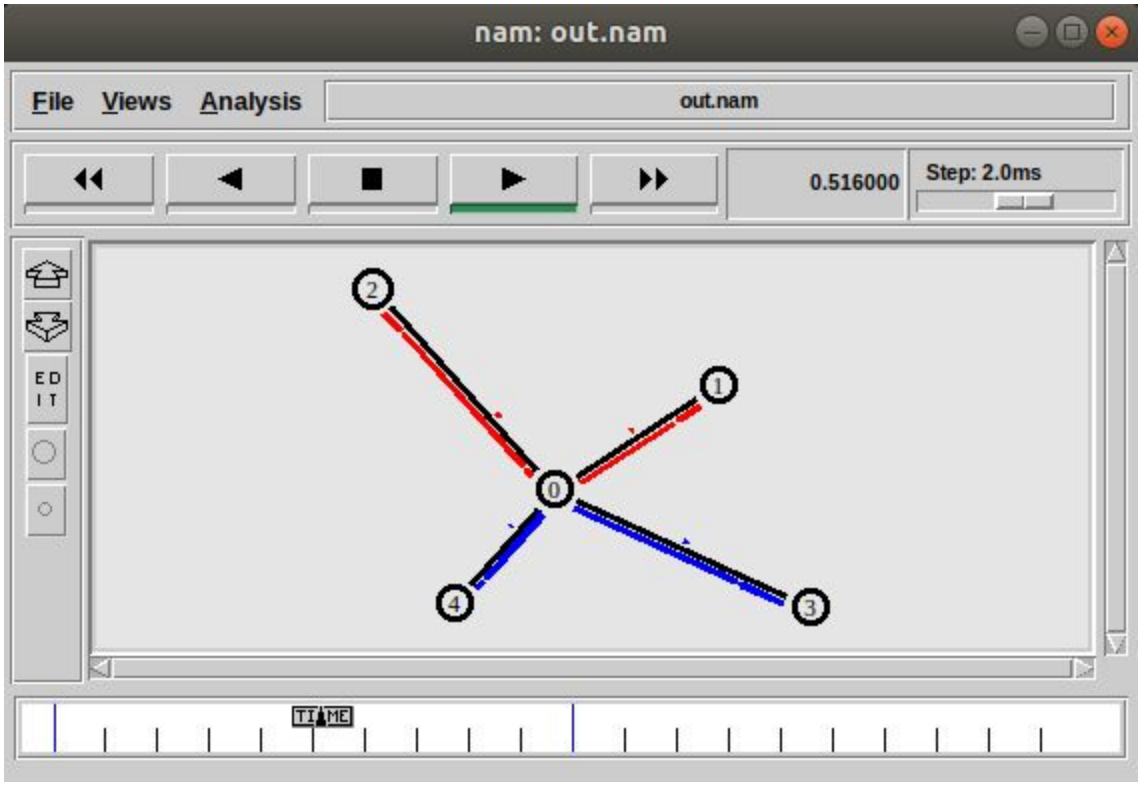
Problem Statement 3

*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:

```
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
```


nam: out.nam
File
Views
Analysis
out.nam
0.516000
Step: 2.0ms
EDIT
TIME
0
1
2
3
4

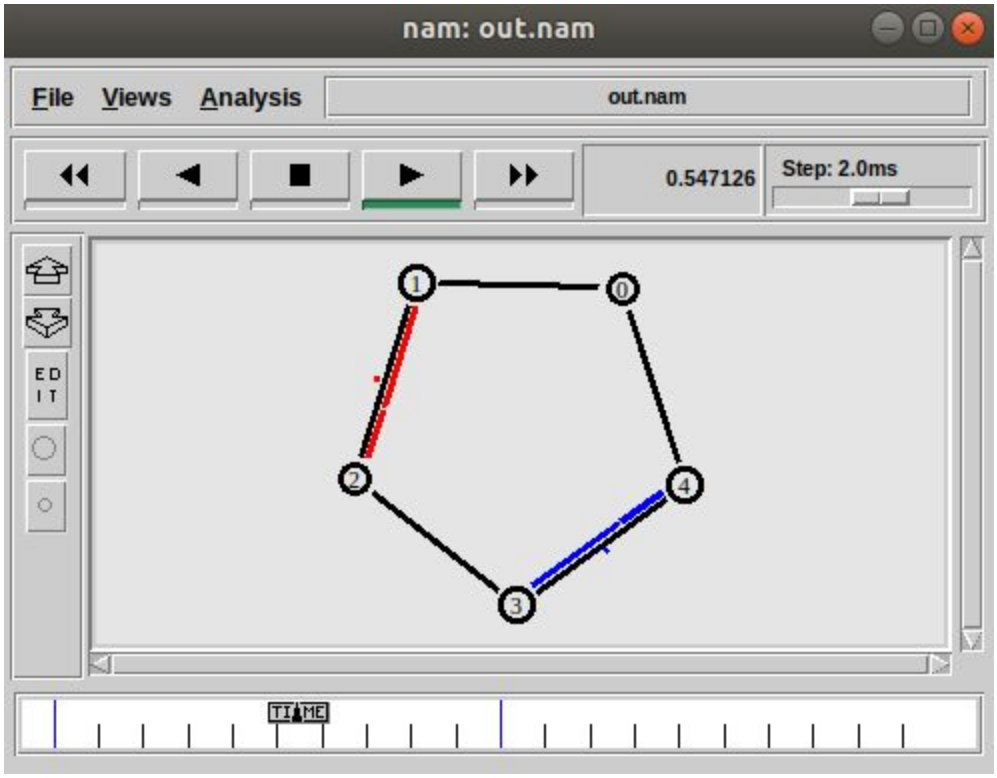
Problem Statement 4

*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:

```
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
```



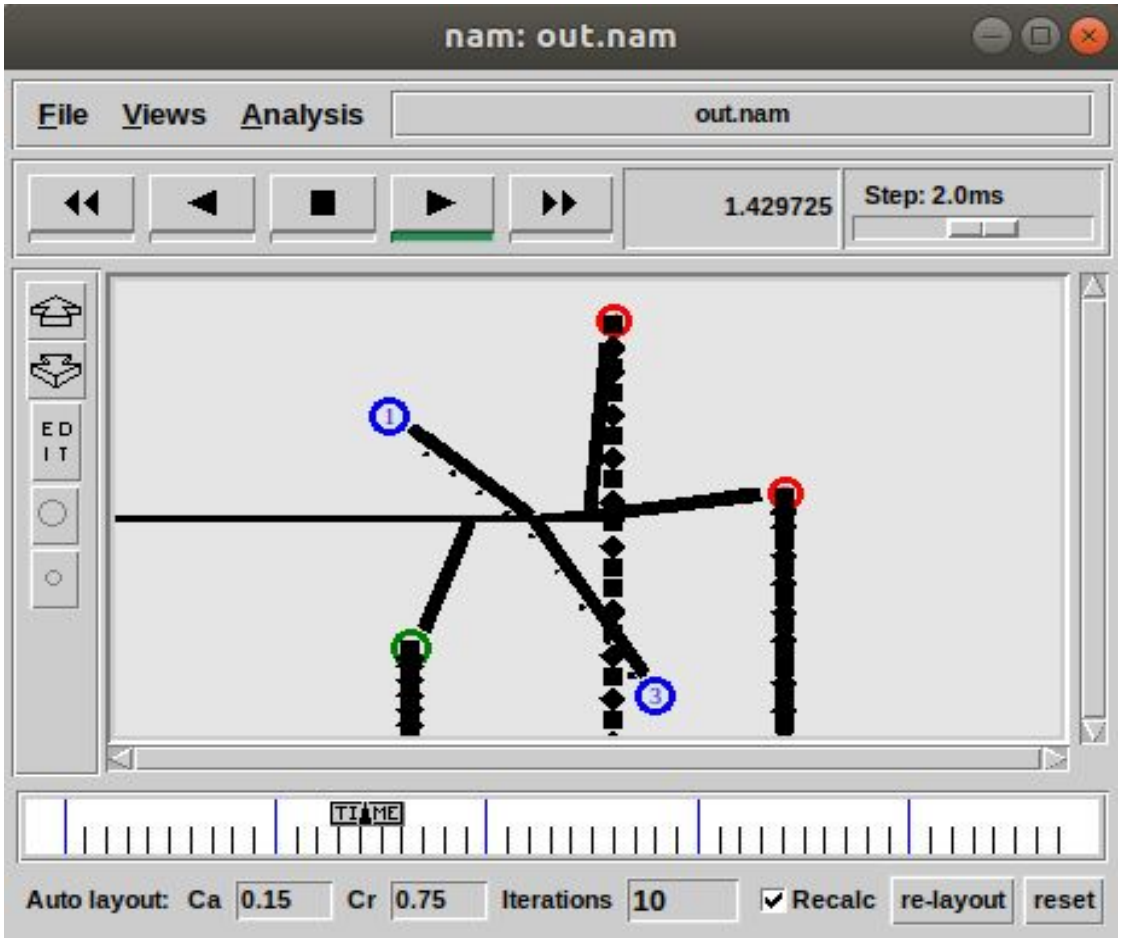
Problem Statement 5

*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
set ns [new Simulator]

#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 &
    exit 0
}

#Create four nodes
set n0 [$ns node]
set n1 [$ns node]
set n2 [$ns node]
set n3 [$ns node]
set n4 [$ns node]

$n0 color green
$n1 color blue
$n2 color red
$n3 color blue
$n4 color red
```

```
#CreateLanbetween the nodes
set lan0 [$ns newLan "$n0 $n1 $n2 $n3 $n4" 0.5Mb 40ms LL Queue/DropTail MAC/Csma/Cd
Channel]

#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"
$ns at 4.5 "$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
File
Views
Analysis
out.nam
1.429725
Step: 2.0ms
ED
IT
1
3
TIME
Auto layout:
Ca
0.15
Cr
0.75
Iterations
10
Recalc
re-layout
reset