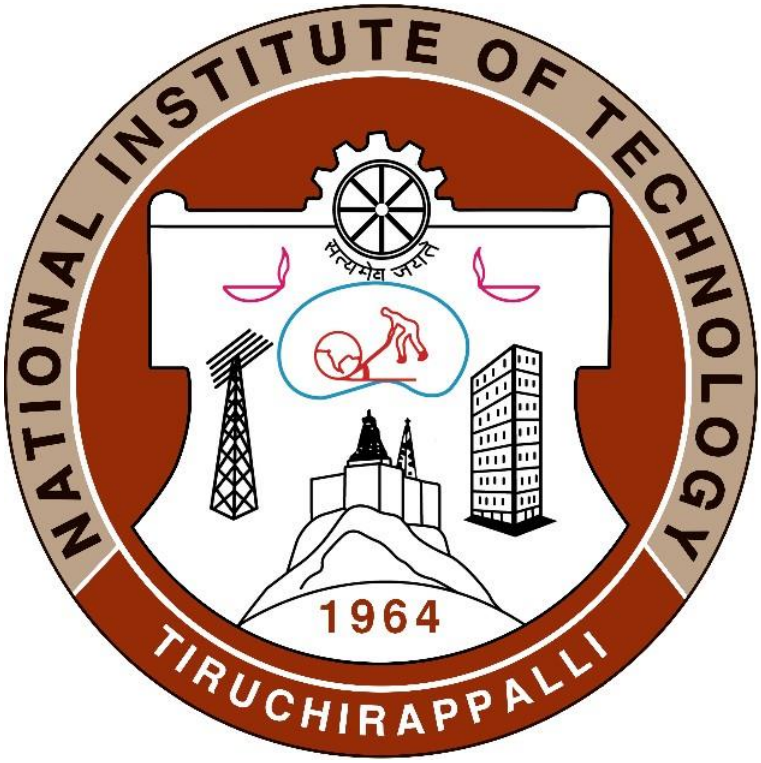
NATIONAL INSTITUTE OF TECHNOLOGY,TIRUCHIRAPALLI



COMPUTER NETWORKS

LAB – CSLR52

Name: Simmi Aggarwal

Roll Number: 106120119

Page of Contents

Calculator

.c File:

a) Client

```
#include<sys/types.h>
#include<sys/socket.h>
#include<stdio.h>
#include<netinet/in.h>
#include<unistd.h>
#include<string.h>
#include<strings.h>
#include<arpa/inet.h>
void main() {
        int b,sockfd,sin_size,con,n,len;
        char operator;
        int op1,op2,result;
        if((sockfd=socket(AF_INET,SOCK_STREAM,0))>0)
                printf("socket created sucessfully\n");

        struct sockaddr_in servaddr;

        servaddr.sin_family=AF_INET;
        servaddr.sin_addr.s_addr=inet_addr("127.0.0.1");
        servaddr.sin_port=6006;

        sin_size = sizeof(struct sockaddr_in);
        if((con=connect(sockfd,(struct sockaddr *) &servaddr,
sin_size))==0);//initiate a connection

        printf("connect sucessful\n");
        printf("Enter operation:\n +:Addition \n -: Subtraction \n /: Division
\n*:Multiplication \n");
        scanf("%c",&operator);
        printf("Enter operands:\n");
        scanf("%d %d", &op1, &op2);

        //write result onto the connection
        write(sockfd,&operator,10);
```

```
        write(sockfd,&op1,sizeof(op1));
        write(sockfd,&op2,sizeof(op2));
        read(sockfd,&result,sizeof(result));
        //print result from the server
        printf("Operation result from server=%d\n",result);
        close(sockfd);
}
```

b) Server

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

void main() {
        int b,sockfd,connfd,sin_size,l,n,len;
        char operator;
        int op1,op2,result;
        if((sockfd=socket(AF_INET,SOCK_STREAM,0))>0)
                printf("socket created sucessfully\n");//socket creation
                   //on success 0 otherwise -1

        struct sockaddr_in servaddr;
        struct sockaddr_in clientaddr;

        servaddr.sin_family=AF_INET;
        servaddr.sin_addr.s_addr=inet_addr("127.0.0.1");
        servaddr.sin_port=6006;

        if((bind(sockfd, (struct sockaddr *)&servaddr,sizeof(servaddr)))==0)
                printf("bind sucessful\n");//bind() assigns the
    // address specified by addr to the socket referred to by the file
     // descriptor sockfd. addrlen specifies the size, in bytes, of the
    //  address structure pointed to by addr. Traditionally, this operation is
     // called “assigning a name to a socket”.
```

```
        if((listen(sockfd,5))==0)//listen for connections on a socket
                printf("listen sucessful\n");
        sin_size = sizeof(struct sockaddr_in);
        if((connfd=accept(sockfd,(struct sockaddr *)&clientaddr,&sin_size))>0);
        printf("accept sucessful\n");

        read(connfd, &operator,10);
        read(connfd,&op1,sizeof(op1));
        read(connfd,&op2,sizeof(op2));
        switch(operator) {
              case '+': result=op1 + op2;
               printf("Result is: %d + %d = %d\n",op1, op2, result);
               break;
              case '-':result=op1 - op2;
                    printf("Result is: %d - %d = %d\n",op1, op2, result);
                    break;
              case '*':result=op1 * op2;
                    printf("Result is: %d * %d = %d\n",op1, op2, result);
                    break;
              case '/':result=op1 / op2;
                    printf("Result is: %d / %d = %d\n",op1, op2, result);
                    break;
              default:
                    printf("ERROR: Unsupported Operation");
        }

        write(connfd,&result,sizeof(result));
        close(sockfd);
}
```

c) URL

https://www.way2techin.com/2018/01/simple-calculator-using-tcp-socket.html

Chat Server

.c File:

a) Client

```
#include<netdb.h>
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
#include<sys/socket.h>
#define MAX 80
#define PORT 8080
#define SA struct sockaddr

void func(int sockfd) {
	char buff[MAX];
	int n;
	for (;;) {
		bzero(buff,sizeof(buff));
		printf("Enter the string : ");
		n=0;
		while((buff[n++]=getchar())!='\n')
			;
		write(sockfd,buff,sizeof(buff));
		bzero(buff,sizeof(buff));
		read(sockfd,buff,sizeof(buff));
		printf("From Server : %s",buff);
		if((strncmp(buff,"exit",4))==0) {
			printf("Client Exit...\n");
			break;
		}
	}
}

int main() {
	int sockfd,connfd;
	struct sockaddr_in servaddr,cli;
	// socket create and verification
	sockfd=socket(AF_INET,SOCK_STREAM,0);
```

```
        if(sockfd==-1) {
                printf("socket creation failed...\n");
                exit(0);
        }
        else
                printf("Socket successfully created..\n");
        bzero(&servaddr,sizeof(servaddr));

        // assign IP, PORT
        servaddr.sin_family=AF_INET;
        servaddr.sin_addr.s_addr=inet_addr("127.0.0.1");
        servaddr.sin_port=htons(PORT);

    // connect the client socket to server socket
        if(connect(sockfd,(SA*)&servaddr,sizeof(servaddr))!=0) {
                printf("connection with the server failed...\n");
                exit(0);
        }
        else
                printf("connected to the server..\n");

        // function for chat
        func(sockfd);
    // close the socket
        close(sockfd);
}
```

b) Server

```
#include<stdio.h>
#include<netdb.h>
#include<netinet/in.h>
#include<stdlib.h>
#include<string.h>
#include<sys/socket.h>
#include<sys/types.h>
#define MAX 80
#define PORT 8080
#define SA struct sockaddr
```

```
// Function designed for chat between client and server.
void func(int connfd) {
	char buff[MAX];
	int n;
	// infinite loop for chat
	for (;;) {
		bzero(buff,MAX);

		// read the message from client and copy it in buffer
		read(connfd,buff,sizeof(buff));

		// print buffer which contains the client contents
		printf("From client: %s\t To client : ",buff);
		bzero(buff,MAX);
		n=0;
		// copy server message in the buffer
		while((buff[n++]=getchar())!='\n')
			;

		// and send that buffer to client
		write(connfd,buff,sizeof(buff));

		// if msg contains "Exit" then server exit and chat ended.
		if(strncmp("exit",buff,4)==0) {
			printf("Server Exit...\n");
			break;
		}
	}
}

int main() {
	int sockfd,connfd,len;
	struct sockaddr_in servaddr,cli;

	// socket create and verification
	sockfd=socket(AF_INET,SOCK_STREAM,0);
```

```
if(sockfd==-1) {
        printf("socket creation failed...\n");
        exit(0);
}
else
        printf("Socket successfully created..\n");
bzero(&servaddr,sizeof(servaddr));

// assign IP, PORT
servaddr.sin_family=AF_INET;
servaddr.sin_addr.s_addr=htonl(INADDR_ANY);
servaddr.sin_port=htons(PORT);

// Binding newly created socket to given IP and verification
if((bind(sockfd,(SA*)&servaddr,sizeof(servaddr)))!=0) {
        printf("socket bind failed...\n");
        exit(0);
}
else
        printf("Socket successfully binded..\n");

if((listen(sockfd,5))!=0) {
        printf("Listen failed...\n");
        exit(0);
}
else
        printf("Server listening..\n");
len=sizeof(cli);

// Accept the data packet from client and verification
connfd=accept(sockfd,(SA*)&cli,&len);
if (connfd<0) {
        printf("server accept failed...\n");
        exit(0);
}
else
        printf("server accept the client...\n");
```

```
        func(connfd);
        // After chatting close the socket
        close(sockfd);
}
```

c) URL

https://www.geeksforgeeks.org/tcp-server-client-implementation-in-c/

Clients Chat Server

.c File:

a) Client 1

```
#include<stdio.h>
#include<sys/socket.h>
#include<netinet/in.h>
#include<string.h>

// function for chat
int compare_strings(char a[], char b[])
{
   int c = 0;
   while (a[c] == b[c])
   {
      if (a[c] == '\0' || b[c] == '\0')
      break;
      c++;
   }
   if (a[c] == '\0' && b[c] == '\0')
   return 0;
   else
   return -1;
}


int main() {
   int clientSocket;
   char buffer[1024];
   struct sockaddr_in serverAddr;
   socklen_t addr_size;
   int cmdEXIT = 0;
         // socket create and verification
   clientSocket = socket(PF_INET, SOCK_STREAM, 0);

   // assign IP, PORT
   serverAddr.sin_family = AF_INET;
   serverAddr.sin_port = htons(7891);
```

```
serverAddr.sin_addr.s_addr = inet_addr("127.0.0.1");

memset(serverAddr.sin_zero, '\0', sizeof serverAddr.sin_zero);
addr_size = sizeof serverAddr;

// connect the client socket to server socket
connect(clientSocket, (struct sockaddr *) &serverAddr, addr_size);
printf("Client 1 : ");
scanf(" %[^\n]s", buffer);
send(clientSocket,buffer,sizeof buffer - 1,0);
//transferring chats between clients through the server
while (cmdEXIT == 0)
{
   if (compare_strings(buffer, "exit")==-1)
   {
      memset(&buffer[0], 0, sizeof(buffer));
      int recvValue = recv(clientSocket, buffer, sizeof buffer - 1, 0);
      if (recvValue != 1)
      {
         if (compare_strings(buffer, "exit")==-1)
         {
            printf("Client 2 : ");
            printf("%s\n", buffer);
            memset(&buffer[0], 0, sizeof(buffer));
         }
         else cmdEXIT=1;
      }
      else
      {
         printf("Client 1 : ");
         scanf(" %[^\n]s", buffer);
         send(clientSocket,buffer,sizeof buffer - 1,0);
      }
   }
   else cmdEXIT=1;
}
```

```
    return 0;
}
```

b) Client 2

```
#include<stdio.h>
#include<sys/socket.h>
#include<netinet/in.h>
#include<string.h>

// function for chat
int compare_strings(char a[], char b[])
{
    int c = 0;
    while (a[c] == b[c])
    {
        if (a[c] == '\0' || b[c] == '\0')
        break;
        c++;
    }
    if (a[c] == '\0' && b[c] == '\0')
    return 0;
    else
    return -1;
}

int main() {
    int clientSocket;
    char buffer[1024];
    struct sockaddr_in serverAddr;
    socklen_t addr_size;
    int cmdEXIT = 0;
        // socket create and verification
    clientSocket = socket(PF_INET, SOCK_STREAM, 0);

    // assign IP, PORT
    serverAddr.sin_family = AF_INET;
    serverAddr.sin_port = htons(7891);
    serverAddr.sin_addr.s_addr = inet_addr("127.0.0.1");
```

```
    memset(serverAddr.sin_zero, '\0', sizeof serverAddr.sin_zero);
    addr_size = sizeof serverAddr;

    // connect the client socket to server socket
    connect(clientSocket, (struct sockaddr *) &serverAddr, addr_size);
    //transferring chats between clients through the server
    while (cmdEXIT == 0)
    {
       int recvValue = recv(clientSocket, buffer, sizeof buffer - 1, 0);
       if (recvValue != 1)
       {
          if (compare_strings(buffer, "exit")==-1)
          {
             printf("Client 1 : ");
             printf("%s\n", buffer);
             memset(&buffer[0], 0, sizeof(buffer));

          }
          else cmdEXIT = 1;
       }
       else
       {
          printf("Client 2 : ");
          scanf(" %[^\n]s", buffer);
          send(clientSocket,buffer,sizeof buffer - 1,0);
          if (compare_strings(buffer, "exit")==-1)
          {
             memset(&buffer[0], 0, sizeof(buffer));
          }
          else cmdEXIT = 1;
       }
    }
    return 0;
}
```

c) Server

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

```
#include<sys/socket.h>
#include<netinet/in.h>
#include<string.h>

// Function designed for chat between client and server.
int compare_strings(char a[],char b[]) {
   int c=0;
   while(a[c]==b[c]) {
      if(a[c]=='\0' || b[c]=='\0')
      break;
      c++;
   }
   if(a[c]=='\0' && b[c]=='\0')
         return 0;
   else
         return -1;
}

int main() {
   int welcomeSocket,Client1,Client2;
   struct sockaddr_in serverAddr;
   struct sockaddr_storage serverStorage;
   socklen_t addr_size;
   char buffer[1024];
   // socket create and verification
   welcomeSocket=socket(PF_INET,SOCK_STREAM,0);

   // assign IP, PORT
   serverAddr.sin_family=AF_INET;
   serverAddr.sin_port=htons(7891);
   serverAddr.sin_addr.s_addr=inet_addr("127.0.01");

   memset(serverAddr.sin_zero,'\0',sizeof serverAddr.sin_zero);
   // Binding newly created socket to given IP and verification
   bind(welcomeSocket,(struct sockaddr *) &serverAddr,sizeof(serverAddr));
   if (listen(welcomeSocket,5)==0)
      printf("Listening\n");
```

```
   else
      printf("Error\n");

   addr_size=sizeof serverStorage;
   // Accept the data packet from client 1 and verification
   Client1=accept(welcomeSocket,(struct sockaddr *)
&serverStorage,&addr_size);
   // Accept the data packet from client 2 and verification
   Client2=accept(welcomeSocket,(struct sockaddr *)
&serverStorage,&addr_size);

   int cmdEXIT=0;
   // Transferring chats between clients
   while(cmdEXIT==0) {
      recv(Client1,buffer,1024,0);
      printf("%s\nSend to Client_2\n",buffer);
      send(Client2,buffer,1024,0);
      if(compare_strings(buffer,"exit")==0) {
         cmdEXIT=1;
      }
      else {
         memset(&buffer[0],0,sizeof(buffer));
         recv(Client2,buffer,1024,0);
         printf("%s\nSend to Client_1\n",buffer);
         send(Client1,buffer,1024,0);
         if(compare_strings(buffer,"exit")==0)
         {
            //exiting chats
            cmdEXIT=1;
         }
      }
   }
   return 0;
}
```

d) URL

https://stackoverflow.com/questions/41077820/c-language-sockets-a-chat-between-two-clients-using-one-server-as-middle-ma

Array Sorting

.c File:

a) Client

```
#include<arpa/inet.h>
#include<stdio.h>
#include<string.h>
#include<sys/socket.h>
#include<unistd.h>

int main(int argc,char* argv[]) {
	int sock;
	struct sockaddr_in server;
	int server_reply[10];
	int number[10],i,temp;
	// Enter the array
	printf("Enter the array: ");
	for(int i=0;i<10;i++)
		scanf("%d",&number[i]);
	printf("\n");

	// Create socket
	sock=socket(AF_INET,SOCK_STREAM,0);
	if (sock==-1) {
		printf("Could not create socket");
	}
	puts("Socket created");

	// assign IP, PORT
	server.sin_addr.s_addr=inet_addr("127.0.0.1");
	server.sin_family=AF_INET;
	server.sin_port=htons(8880);

	// Connect to remote server
	if(connect(sock,(struct sockaddr*)&server,sizeof(server))<0) {
		perror("connect failed. Error");
		return 1;
	}
```

```
        puts("Connected\n");

        if(send(sock,&number,10*sizeof(int),0)<0) {
                puts("Send failed");
                return 1;
        }
        // Receive a reply from the server
        if(recv(sock,&server_reply,10*sizeof(int),0)<0) {
                puts("recv failed");
                return 0;
        }
        puts("Server reply :\n");
        for(i=0;i<10;i++) {
                printf("%d\n",server_reply[i]);
        }
   // close the socket
        close(sock);
        return 0;
}
```

b) Server

```
#include<arpa/inet.h>
#include<stdio.h>
#include<string.h>
#include<sys/socket.h>
#include<unistd.h>

void bubble_sort(int[],int);

// Driver code
int main(int argc,char* argv[]) {
        int socket_desc,client_sock,c,read_size;
        struct sockaddr_in server,client;
        int message[10],i;
        // Create socket
        socket_desc=socket(AF_INET,SOCK_STREAM,0);
        if(socket_desc==-1) {
                printf("Could not create socket");
```

```
        }
        puts("Socket created");

    // Prepare the sockaddr_in structure
        server.sin_family=AF_INET;
        server.sin_addr.s_addr=INADDR_ANY;
        server.sin_port=htons(8880);

        // Bind the socket
        if(bind(socket_desc,(struct sockaddr*)&server,sizeof(server))<0) {
                perror("bind failed. Error");
                return 1;
        }
        puts("bind done");
        // listen to the socket
        listen(socket_desc,3);

        puts("Waiting for incoming connections...");
        c=sizeof(struct sockaddr_in);

    // accept connection from an incoming client
        client_sock=accept(socket_desc,(struct
sockaddr*)&client,(socklen_t*)&c);

        if(client_sock<0) {
                perror("accept failed");
                return 1;
        }

        puts("Connection accepted");
        // Receive a message from client
        while((read_size=recv(client_sock,&message,10*sizeof(int),0))>0) {

                bubble_sort(message,10);

                write(client_sock,&message,10*sizeof(int));
        }
```

```
        if(read_size==0) {
                puts("Client disconnected");
        }
        else if(read_size==-1) {
                perror("recv failed");
        }
        return 0;
}

// Function to sort the array
void bubble_sort(int list[],int n)
{
        int c,d,t;
        for(c=0;c<(n-1);c++) {
                for(d=0;d<n-c-1;d++) {
                        if(list[d]>list[d+1]) {
                                t=list[d];
                                list[d]=list[d+1];
                                list[d+1]=t;
                        }
                }
        }
}
```

c) URL

https://www.geeksforgeeks.org/sort-array-using-socket-programming/

String Manipulation

.c File:

a) Client

```
#include<stdio.h>
#include<sys/socket.h>
#include<netinet/in.h>
#include<string.h>

int main() {
        int clientSocket,portNum,nBytes;
        char buffer[1024];
        struct sockaddr_in serverAddr;
        socklen_t addr_size;

        // Create socket
        clientSocket=socket(PF_INET, SOCK_STREAM, 0);

        portNum=7891;

        // assign IP, PORT
        serverAddr.sin_family=AF_INET;
        serverAddr.sin_port=htons(portNum);
        serverAddr.sin_addr.s_addr=inet_addr("127.0.0.1");

        memset(serverAddr.sin_zero,'\0',sizeof serverAddr.sin_zero);

        addr_size=sizeof serverAddr;
        connect(clientSocket,(struct sockaddr *) &serverAddr,addr_size);

        // Loop to send string to server
        while(1) {
        printf("Type a sentence to send to server:\n");
        fgets(buffer,1024,stdin);
        printf("You typed: %s",buffer);
        nBytes=strlen(buffer)+1;
        send(clientSocket,buffer,nBytes,0);
        recv(clientSocket,buffer,1024,0);
```

```
        printf("Received from server: %s\n\n",buffer);
        }

        return 0;
}
```

b) Server

```
#include<stdio.h>
#include<sys/socket.h>
#include<netinet/in.h>
#include<string.h>
#include<stdlib.h>

int main() {
        int welcomeSocket, newSocket, portNum, clientLen, nBytes;
        char buffer[1024];
        struct sockaddr_in serverAddr;
        struct sockaddr_storage serverStorage;
        socklen_t addr_size;
        int i;

        // Create socket
        welcomeSocket = socket(PF_INET, SOCK_STREAM, 0);

        portNum = 7891;

        // Prepare the sockaddr_in structure
        serverAddr.sin_family = AF_INET;
        serverAddr.sin_port = htons(portNum);
        serverAddr.sin_addr.s_addr = inet_addr("127.0.0.1");

        memset(serverAddr.sin_zero, '\0', sizeof serverAddr.sin_zero);

        bind(welcomeSocket, (struct sockaddr *) &serverAddr,
sizeof(serverAddr));

        if(listen(welcomeSocket,5)==0)
        printf("Listening\n");
```

```
        else
        printf("Error\n");

        addr_size = sizeof serverStorage;
        //loop to keep accepting new connections
        while(1){
        newSocket = accept(welcomeSocket, (struct sockaddr *) &serverStorage,
&addr_size);
                //fork a child process to handle the new connection
        if(!fork()){
                nBytes = 1;
                        //loop while connection is live
                while(nBytes!=0){
        nBytes = recv(newSocket,buffer,1024,0);

        for (i=0;i<nBytes-1;i++){
                buffer[i] = toupper(buffer[i]);
        }

        send(newSocket,buffer,nBytes,0);
                }
                close(newSocket);
                exit(0);
        }
                //if parent, close the socket and go back to listening new requests
        else{
                close(newSocket);
        }
        }

        return 0;
}
```

c) URL

https://www.programminglogic.com/sockets-programming-example-in-c-server-converts-strings-to-uppercase/

Point to Point

.tcl File:

```
#Create a simulator object
set ns [new Simulator]

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

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

#Define a 'finish' procedure
proc finish {} {
        global ns f nf
        $ns flush-trace
        close $f
        exec nam out.nam
        exit 0
}

#Creating nodes
set n0 [$ns node]
set n1 [$ns node]

#Create link between nodes
$ns duplex-link $n0 $n1 1Mb 10ms DropTail

#Setup a UDP connection
set udp0 [new Agent/UDP]
$ns attach-agent $n0 $udp0

# Create a CBR traffic source and attach it to udp0
set cbr0 [new Application/Traffic/CBR]
$cbr0 set packet_Size_ 500
$cbr0 set interval_ 0.005
```

```
$cbr0 attach-agent $udp0

#Create a Null agent (a traffic sink) and attach it to node n1
set null0 [new Agent/Null]
$ns attach-agent $n1 $null0

#Connect the traffic sources with the traffic sink
$ns connect $udp0 $null0

#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 File:

```
V -t * -v 1.0a5 -a 0
A -t * -n 1 -p 0 -o 0x7fffffff -c 30 -a 1
A -t * -h 1 -m 1073741823 -s 0
n -t * -a 0 -s 0 -S UP -v circle -c black -i black
n -t * -a 1 -s 1 -S UP -v circle -c black -i black
l -t * -s 0 -d 1 -S UP -r 1000000 -D 0.01 -c black
+ -t 0.5 -s 0 -d 1 -p cbr -e 210 -c 0 -i 0 -a 0 -x {0.0 1.0 0 --------null}
- -t 0.5 -s 0 -d 1 -p cbr -e 210 -c 0 -i 0 -a 0 -x {0.0 1.0 0-------- null}
h -t 0.5 -s 0 -d 1 -p cbr -e 210 -c 0 -i 0 -a 0 -x {0.0 1.0 -1 --------null}
+ -t 0.505 -s 0 -d 1 -p cbr -e 210 -c 0 -i 1 -a 0 -x {0.0 1.0 1--------null}
- -t 0.505 -s 0 -d 1 -p cbr -e 210 -c 0 -i 1 -a 0 -x {0.0 1.0 1 ------- null}
h -t 0.505 -s 0 -d 1 -p cbr -e 210 -c 0 -i 1 -a 0 -x {0.0 1.0 -1--------null}
```

.tr File:

```
+ 0.5 0 1 cbr 210 ------- 0 0.0 1.0 0 0
- 0.5 0 1 cbr 210--------0 0.0 1.0 0 0
+ 0.505 0 1 cbr 210--------0 0.0 1.0 1 1
- 0.505 0 1 cbr 210 ------- 0 0.0 1.0 1 1
+ 0.51 0 1 cbr 210 ------- 0 0.0 1.0 2 2
```

```
- 0.51 0 1 cbr 210 --------0 0.0 1.0 2 2
r 0.51168 0 1 cbr 210 --------0 0.0 1.0 0 0
+ 0.515 0 1 cbr 210 --------0 0.0 1.0 3 3
- 0.515 0 1 cbr 210 ------- 0 0.0 1.0 3 3
r 0.51668 0 1 cbr 210 --------0 0.0 1.0 1 1
+ 0.52 0 1 cbr 210 ------- 0 0.0 1.0 4 4
- 0.52 0 1 cbr 210 --------0 0.0 1.0 4 4
```

.awk File:

a) Throughput

```
#init variables
BEGIN {
recv=0;
gotime = 0;
time = 0;
packet_size = $6;
time_interval=0.01;
}
#body
{
 #refer variables to trace file format
     event = $1
          time = $2
          node_id = $4
          pktType = $5
             packet_size = $6
         #write throughput into .txt file
 if(time>gotime) {

  print gotime, (packet_size * recv * 8.0)/1000; #packet size * ... gives results in
kbps
  gotime+= time_interval;
  recv=0;
  }

#calculate throughput

  if (( event == "r") && ( pktType == "tcp" ))
```

```
 {
   recv++;
 }

}

END {
;
}
```

b) Packet Ratio

```
#init variables
BEGIN{
time=0;
time_interval=0.01;
packetsize=$6;
gotime=1;
a=0;
b=0;
}

{
        #refer variables to trace file format
        event=$1
        time=$2
        from=$3
        to=$4
        pktType=$5
        packet_size=$6
        #calculate packets sent and received
if(event=="+")
 {
   a++;
 }
else if(event=="-")
{
  b++;
}
```

```
#write packet delivery ratio into .txt file
 if(time>gotime) {
   print gotime,(a-b)/a;
   gotime+=time_interval;
 }

}

END{
;
}
```

.sh File:

a) Throughput Graph

```
set terminal png
#print output into
set output 'Result.png'
#set xrange [0.0:3.0]
set xlabel "Time(in seconds)"
set autoscale xfix
set autoscale
#set yrange [0:80]
set ylabel "Throughput(in Kbps)"
set grid
set style data linespoints
#print output into
set output "out.png"
#plot the graph
plot "out" using 1:2 title "TCP Throughput" lt rgb "blue"
```

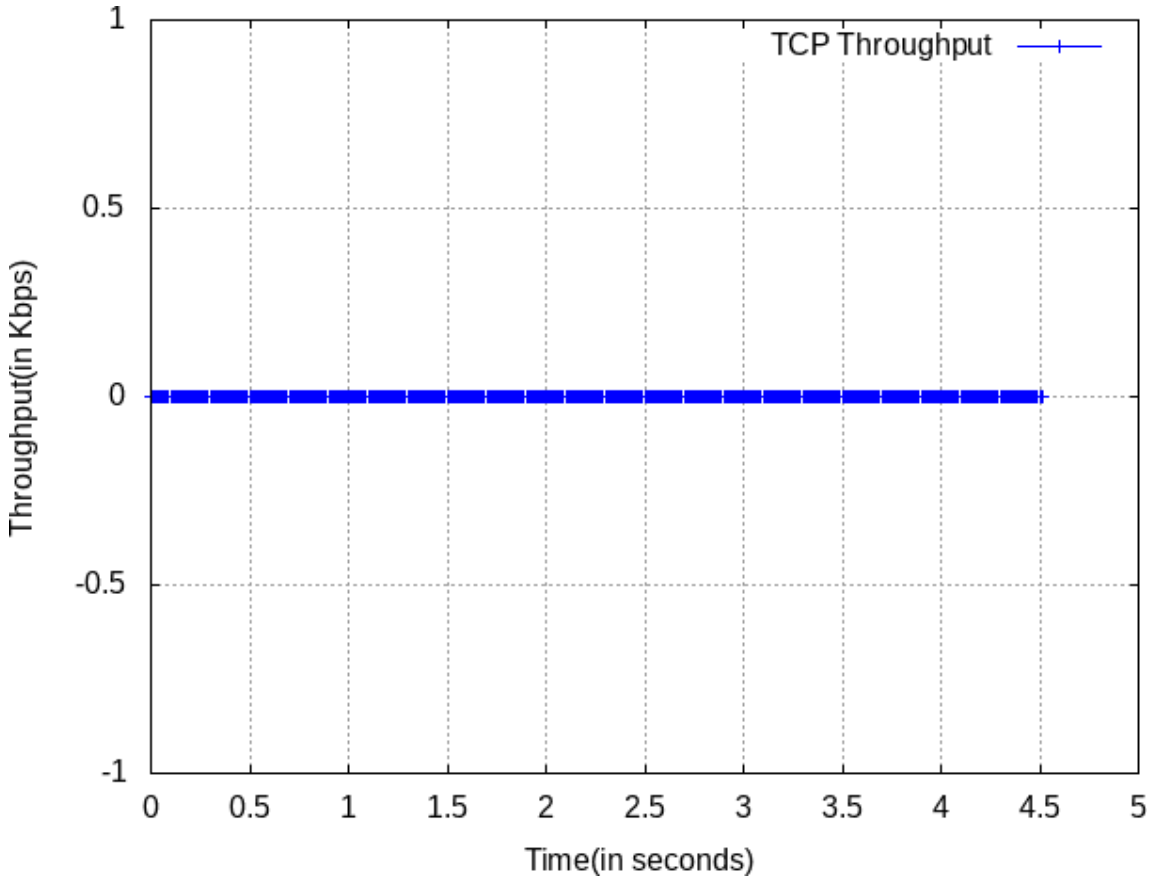
b) Packet Ratio Graph

```
set terminal png
#print output into
set output 'result1.png'
#set xrange
set xrange [0.00:2.00]
set xlabel "Time(in seconds)"
set autoscale
#set yrange
```

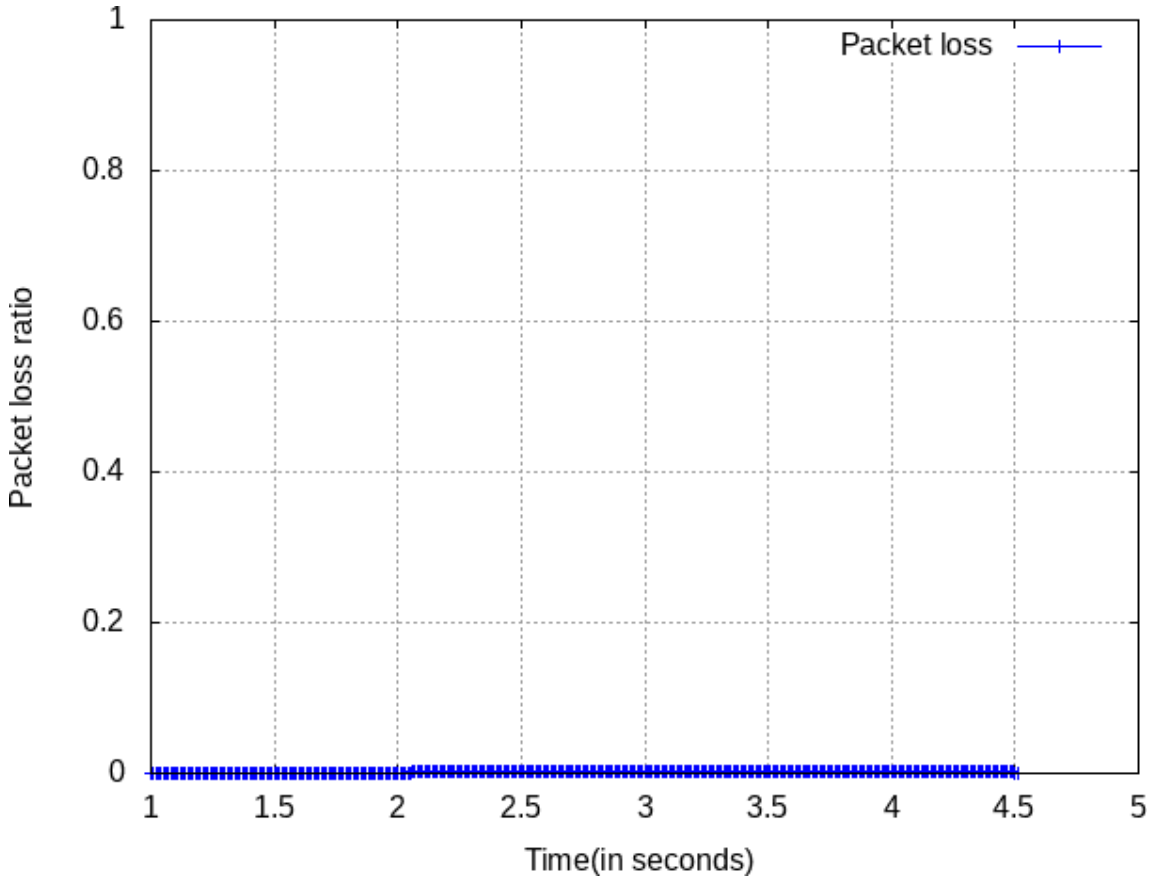
```
set yrange [0:1]
set ylabel "Packet loss ratio"
set grid
set style data linespoints
#plot the graph
plot "tp1" using 1:2 title "Packet loss" lt rgb "blue"
```

Graph:

a) Throughput



b) Packet Delivery Ratio



Three Sources Wired

.tcl File:

```
#Create a simulator object
set ns [new Simulator]

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

#Open the nam trace file
set f [open outwd.tr w]
$ns trace-all $f

#Creating nodes
set n0 [$ns node]
set n1 [$ns node]
set n2 [$ns node]
set n3 [$ns node]

#Create link between nodes
$ns duplex-link $n0 $n3 2Mb 4ms DropTail
$ns duplex-link $n1 $n3 2Mb 4ms DropTail
$ns duplex-link $n2 $n3 2Mb 4ms DropTail

#Setup TCP connection
set tcp0 [new Agent/TCP]
set tcp1 [new Agent/TCP]
set tcp2 [new Agent/TCP]

#Setup TCP sink connection
set sink0 [new Agent/TCPSink]
set sink1 [new Agent/TCPSink]
set sink2 [new Agent/TCPSink]

#Attach TCP agent to nodes n0, n1, n2
$ns attach-agent $n0 $tcp0
$ns attach-agent $n1 $tcp1
```

```
$ns attach-agent $n2 $tcp2

#Attach TCP sink agent to node n3
$ns attach-agent $n3 $sink0
$ns attach-agent $n3 $sink1
$ns attach-agent $n3 $sink2

#Connect TCP and TCP sink
$ns connect $tcp0 $sink0
$ns connect $tcp1 $sink1
$ns connect $tcp2 $sink2

#Setup FTP connection
set ftp0 [new Application/FTP]
set ftp1 [new Application/FTP]
set ftp2 [new Application/FTP]

#Attach FTP agent to tcp0, tcp1, tcp2
$ftp0 attach-agent $tcp0
$ftp1 attach-agent $tcp1
$ftp2 attach-agent $tcp2

#Define a 'finish' procedure
proc finish {} {
        global ns nf f
        $ns flush-trace
        close $nf
        close $f
        exec nam outwd.nam &
        exit
        }

#Schedule events for the FTP agents
$ns at 0.1 "$ftp0 start"
$ns at 0.2 "$ftp1 start"
$ns at 0.3 "$ftp2 start"
$ns at 1.0 "$ftp0 stop"
```

```
$ns at 1.1 "$ftp1 stop"
$ns at 1.2 "$ftp2 stop"

#Call the finish procedure after 1.3 seconds of simulation time
$ns at 1.3 "finish"

#Run the simulation
$ns run
```

.nam File:

```
V -t * -v 1.0a5 -a 0
A -t * -n 1 -p 0 -o 0x7fffffff -c 30 -a 1
A -t * -h 1 -m 1073741823 -s 0
n -t * -a 0 -s 0 -S UP -v circle -c black -i black
n -t * -a 1 -s 1 -S UP -v circle -c black -i black
n -t * -a 2 -s 2 -S UP -v circle -c black -i black
n -t * -a 3 -s 3 -S UP -v circle -c black -i black
l -t * -s 0 -d 3 -S UP -r 2000000 -D 0.0040000000000000001 -c black
l -t * -s 1 -d 3 -S UP -r 2000000 -D 0.0040000000000000001 -c black
l -t * -s 2 -d 3 -S UP -r 2000000 -D 0.0040000000000000001 -c black
+ -t 0.1 -s 0 -d 3 -p tcp -e 40 -c 0 -i 0 -a 0 -x {0.0 3.0 0-------- null}
- -t 0.1 -s 0 -d 3 -p tcp -e 40 -c 0 -i 0 -a 0 -x {0.0 3.0 0 ------- null}
```

.tr File:

```
+ 0.1 0 3 tcp 40--------0 0.0 3.0 0 0
- 0.1 0 3 tcp 40 ------- 0 0.0 3.0 0 0
r 0.10416 0 3 tcp 40--------0 0.0 3.0 0 0
+ 0.10416 3 0 ack 40 --------0 3.0 0.0 0 1
- 0.10416 3 0 ack 40--------0 3.0 0.0 0 1
r 0.10832 3 0 ack 40 ------- 0 3.0 0.0 0 1
+ 0.10832 0 3 tcp 1040 --------0 0.0 3.0 1 2
- 0.10832 0 3 tcp 1040--------0 0.0 3.0 1 2
+ 0.10832 0 3 tcp 1040 --------0 0.0 3.0 2 3
- 0.11248 0 3 tcp 1040--------0 0.0 3.0 2 3
r 0.11648 0 3 tcp 1040 ------- 0 0.0 3.0 1 2
+ 0.11648 3 0 ack 40 --------0 3.0 0.0 1 4
```

.awk File:

a) Throughput

```
#init variables
```

```
BEGIN {
recv=0;
gotime = 0;
time = 0;
packet_size = $6;
time_interval=0.01;
}
#body
{
 #refer variables to trace file format
     event = $1
          time = $2
          node_id = $4
          pktType = $5
             packet_size = $6
        #write throughput into .txt file
 if(time>gotime) {

 print gotime, (packet_size * recv * 8.0)/1000; #packet size * ... gives results in
kbps
 gotime+= time_interval;
 recv=0;
 }

#calculate throughput

 if (( event == "r") && ( pktType == "tcp" ))
 {
   recv++;
 }

} #body


END {
;
}
```

b) Packet Ratio

```
#init variables
BEGIN{
time=0;
time_interval=0.01;
packetsize=$6;
gotime=1;
a=0;
b=0;
}

{
        #refer variables to trace file format
        event=$1
        time=$2
        from=$3
        to=$4
        pktType=$5
        packet_size=$6
        #calculate packets sent and received
if(event=="+")
 {
   a++;
 }
else if(event=="-")
{
  b++;
}

#write packet delivery ratio into .txt file
 if(time>gotime) {
   print gotime,(a-b)/a;
   gotime+=time_interval;
 }

}
```

```
END{
;
}
```

.sh File:

a) Throughput Graph

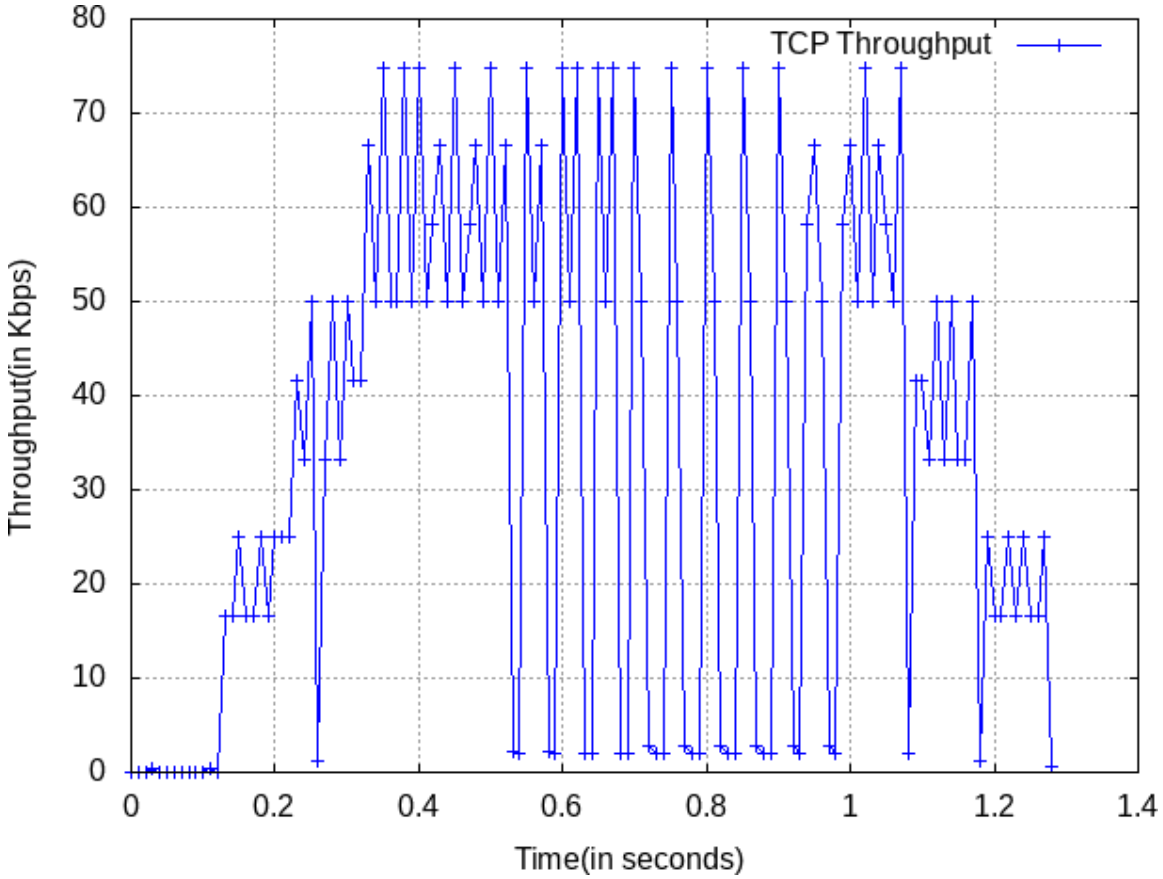
```
set terminal png
#print output into
set output 'Result.png'
#set xrange [0.0:3.0]
set xlabel "Time(in seconds)"
set autoscale xfix
set autoscale
#set yrange [0:80]
set ylabel "Throughput(in Kbps)"
set grid
set style data linespoints
#print output into
set output "outwd.png"
#plot the graph
plot "outwd" using 1:2 title "TCP Throughput" lt rgb "blue"
```

b) Packet Ratio Graph

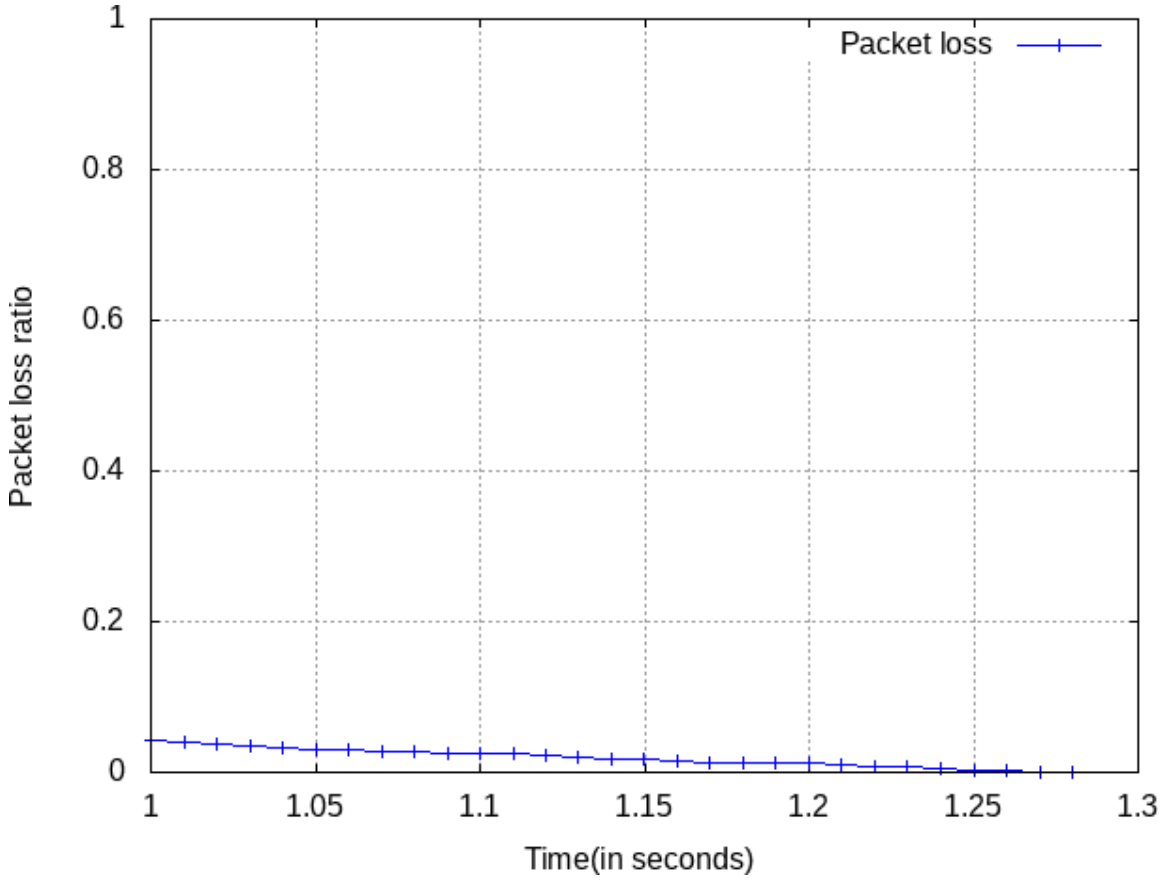
```
set terminal png
#print output into
set output 'result1.png'
#set xrange
set xrange [0.00:2.00]
set xlabel "Time(in seconds)"
set autoscale
#set yrange
set yrange [0:1]
set ylabel "Packet loss ratio"
set grid
set style data linespoints
#plot the graph
plot "tp1" using 1:2 title "Packet loss" lt rgb "blue"
```

Graph:

a) Throughput Graph



b) Packet Delivery Ratio Graph



Three Sources Wireless

.tcl File:

```
#Define options
set val(chan)         Channel/WirelessChannel    ;
set val(prop)         Propagation/TwoRayGround ;
set val(netif)        Phy/WirelessPhy          ;
set val(mac)          Mac/802_11               ;
set val(ifq)          Queue/DropTail/PriQueue    ;
set val(ll)           LL                       ;
set val(ant)          Antenna/OmniAntenna      ;
set val(ifqlen)       50                       ;
set val(nn)           4                        ;
set val(rp)           DSDV                     ;
set val(x)       500
set val(y)       500

#Create a simulator object/
set ns      [new Simulator]

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

#Open the nam file
set namf [open wrls.nam w]
$ns namtrace-all-wireless $namf $val(x) $val(y)
set topo  [new Topography]

#Set node API configuration
$topo load_flatgrid $val(x) $val(y)
create-god $val(nn)
set channel1 [new $val(chan)]
$ns node-config -adhocRouting $val(rp)\
         -llType $val(ll)\
         -macType $val(mac)\
         -ifqType $val(ifq)\
         -ifqLen $val(ifqlen)\
```

```
        -antType $val(ant)\
        -propType $val(prop)\
        -phyType $val(netif)\
        -topoInstance $topo\
        -energyModel "EnergyModel"\
        -initialEnergy 3.2\
        -txPower 0.3 \
        -rxPower 0.1 \
        -sleepPower 0.05 \
        -idlePower 0.1 \
        -agentTrace ON \
        -routerTrace ON \
        -macTrace ON \
        -movementTrace OFF \
        -channel $channel1

#Creating nodes
set n0 [$ns node]
set n1 [$ns node]
set n2 [$ns node]
set n3 [$ns node]

#Set random node motions
$n0 random-motion 0
$n1 random-motion 0
$n2 random-motion 0
$n3 random-motion 0

#Set initial node positions
$ns initial_node_pos $n1 20
$ns initial_node_pos $n0 20
$ns initial_node_pos $n2 20
$ns initial_node_pos $n3 70

#Set values on the XY plane
$n0 set X_ 5.0
$n0 set Y_ 2.0
```

```
$n0 set Z_ 0.0

$n1 set X_ 8.0
$n1 set Y_ 5.0
$n1 set Z_ 0.0

$n2 set X_ 18.0
$n2 set Y_ 15.0
$n2 set Z_ 0.0

$n3 set X_ 23.0
$n3 set Y_ 28.0
$n3 set Z_ 0.0

#Set final node co-ordinates
$ns at 3.0 "$n1 setdest 50.0 40.0 25.0"
$ns at 3.0 "$n0 setdest 48.0 38.0 5.0"
$ns at 4.0 "$n2 setdest 100.0 100.0 40.0"
$ns at 10.0 "$n3 setdest 490.0 480.0 30.0"

#Setup a TCP connection
set tcp0 [new Agent/TCP]
$ns attach-agent $n0 $tcp0
$tcp0 set class_ 2

#Setup a TCP sink connection
set sink0 [new Agent/TCPSink]
$ns attach-agent $n1 $sink0
$ns connect $tcp0 $sink0

#Setup a FTP connection
set ftp0 [new Application/FTP]
$ftp0 attach-agent $tcp0

Attach TCP agent to node n2
set tcp1 [new Agent/TCP]
$ns attach-agent $n2 $tcp1
```

```
$tcp1 set class_ 2

#Attach TCP sink agent to node n1
set sink1 [new Agent/TCPSink]
$ns attach-agent $n1 $sink1
$ns connect $tcp1 $sink1

#Attach FTP to TCP
set ftp1 [new Application/FTP]
$ftp1 attach-agent $tcp1

#Attach TCP agent to node n3
set tcp2 [new Agent/TCP]
$ns attach-agent $n3 $tcp2
$tcp2 set class_ 2

set sink2 [new Agent/TCPSink]
$ns attach-agent $n1 $sink2
$ns connect $tcp2 $sink2

set ftp2 [new Application/FTP]
$ftp2 attach-agent $tcp2

#Schedule events for the FTP agents
$ns at 0.5 "$ftp0 start";
$ns at 1.5 "$ftp1 start";
$ns at 2.0 "$ftp1 stop";
$ns at 2.5 "$ftp2 start";
$ns at 5.0 "finish"

#Define a 'finish' procedure
proc finish {} {
   global ns nf
   $ns flush-trace
   close $nf
   exec nam outwr.nam &
   exit 0
```

```
}

#Run the simulation
$ns run
```

.nam File:

```
n -t * -s 1 -x 0 -y 0 -Z 0 -z 20 -v circle -c green
n -t * -s 0 -x 0 -y 0 -Z 0 -z 20 -v circle -c green
n -t * -s 2 -x 0 -y 0 -Z 0 -z 20 -v circle -c green
n -t * -s 3 -x 0 -y 0 -Z 0 -z 70 -v circle -c green
V -t * -v 1.0a5 -a 0
W -t * -x 500 -y 500
A -t * -n 1 -p 0 -o 0x7fffffff -c 30 -a 1
A -t * -h 1 -m 1073741823 -s 0
+ -t 0.184410103 -s 0 -d -1 -p message -e 32 -c 2 -a 0 -i 0 -k RTR
- -t 0.184410103 -s 0 -d -1 -p message -e 32 -c 2 -a 0 -i 0 -k RTR
h -t 0.184410103 -s 0 -d -1 -p message -e 32 -c 2 -a 0 -i 0 -k RTR
+ -t 0.184845103 -s 0 -d -1 -p message -e 90 -c 2 -a 0 -i 0 -k MAC
```

.tr File:

```
s 0.184410103 _0_ RTR --- 0 message 32 [0 0 0 0] [energy 3.200000 ei 0.000 es
0.000 et 0.000 er 0.000] ------- [0:255 -1:255 32 0]
s 0.184845103 _0_ MAC  --- 0 message 90 [0 ffffffff 0 800] [energy 3.200000 ei
0.000 es 0.000 et 0.000 er 0.000]-------- [0:255 -1:255 32 0]
N -t 0.184845 -n 1 -e 3.181443
N -t 0.184845 -n 2 -e 3.181443
N -t 0.184845 -n 3 -e 3.181443
r 0.185565117 _1_ MAC  --- 0 message 32 [0 ffffffff 0 800] [energy 3.181443 ei
0.018 es 0.000 et 0.000 er 0.000]-------- [0:255 -1:255 32 0]
r 0.185565164 _2_ MAC  --- 0 message 32 [0 ffffffff 0 800] [energy 3.181443 ei
0.018 es 0.000 et 0.000 er 0.000]-------- [0:255 -1:255 32 0]
r 0.185565209 _3_ MAC  --- 0 message 32 [0 ffffffff 0 800] [energy 3.181443 ei
0.018 es 0.000 et 0.000 er 0.000]-------- [0:255 -1:255 32 0]
r 0.185590117 _1_ RTR  --- 0 message 32 [0 ffffffff 0 800] [energy 3.181443 ei
0.018 es 0.000 et 0.000 er 0.000]-------- [0:255 -1:255 32 0]
r 0.185590164 _2_ RTR --- 0 message 32 [0 ffffffff 0 800] [energy 3.181443 ei
0.018 es 0.000 et 0.000 er 0.000]-------- [0:255 -1:255 32 0]
r 0.185590209 _3_ RTR  --- 0 message 32 [0 ffffffff 0 800] [energy 3.181443 ei
0.018 es 0.000 et 0.000 er 0.000]-------- [0:255 -1:255 32 0]
```

s 0.500000000 _0_ AGT --- 1 tcp 40 [0 0 0 0] [energy 3.181299 ei 0.018 es 0.000 et 0.000 er 0.000] ------- [0:0 1:0 32 0] [0 0] 0 0

.awk File:

a) Throughput

```
#init variables
BEGIN {
      recvdSize = 0
      startTime = 2.0
      stopTime = 0
      sent=0
      received=0
      dropped=0
      forwarded=0
      gotime=0
      time_interval=0.01;
}
  {
    #refer variables to trace file format
    # Trace line format: new
          event = $1
       time = $2
       node_id = $3
       pkt_id = $6
       pkt_size = $8
          level = $4
  }
{
  # Store start time
  if(time>gotime) {

  print gotime, (recvdSize * received * 8.0)/1000; #packet size * ... gives results in kbps
  gotime+= time_interval;
  received=0;
  }
  if ((level == "AGT" && event == "s") && pkt_size >= 512) {
          sent++
```

```
        if (time < startTime) {
         startTime = time
         }
 }
 if (event == "D" && pkt_size >= 512) {
     dropped++
 }
 if (event == "f" && pkt_size >= 512) {
     forwarded++
 }

 # Update total received packets' size and store packets arrival time
 if (level == "AGT" && event == "r" && pkt_size >= 512) {
     if (time > stopTime) {
         stopTime = time
     }
     received++
     # Rip off the header
     hdr_size = pkt_size % 512
     pkt_size -= hdr_size
     # Store received packet's size
     recvdSize += pkt_size
 }
}
#Print throughput
 END {
     printf("Average Throughput[kbps] = %.2f\t\t
StartTime=%.2f\tStopTime=%.2f\n",(recvdSize/(stopTime-
startTime))*(8/1000),startTime,stopTime)
     print("Sent - ",sent)
     print("Received - ",received)
     print("Dropped - ",dropped)
     print("Forwarded",forwarded)
}
```

b) Packet Ratio

```
#init variables
BEGIN {
```

```
    sends=0;
    recvs=0;
    routing_packets=0;
    droppedPackets=0;
    highest_packet_id =0;
    sum=0;
    time_interval=0.01;
    recvnum=0;
    gotime=1;
   }
 {
   #refer variables to trace file format
 time = $2;
 packet_id = $6;
 event =$1;
 #write packet delivery ratio into .txt file
 if(time>gotime) {
  print gotime,(sends-recvs)/sends;
  gotime+=time_interval;
}
 # CALCULATE PACKET DELIVERY FRACTION
 if (( $1 == "s") &&  ( $7 == "cbr" ) && ( $4=="AGT" )) {  sends++; }
 if (( $1 == "r") && ( $7 == "cbr" ) && ( $4=="AGT" ))  {  recvs++; }
 # CALCULATE DELAY
 if ( start_time[packet_id] == 0 )  start_time[packet_id] = time;
 if (( $1 == "r") && ( $7 == "cbr" ) && ( $4=="AGT" )) { end_time[packet_id] =
time;  }
    else {  end_time[packet_id] = -1;  }
 # CALCULATE TOTAL AODV OVERHEAD
 if (($1 == "s" || $1 == "f" || $1="r") && $4 == "RTR" && ($7 =="AODV" ||$7
=="AOMDV")) routing_packets++;
 # DROPPED AODV PACKETS
 if (event == "D") droppedPackets++;
 }
 END {
 for ( i in end_time )
 {
```

```
 start = start_time[i];
 end = end_time[i];
 packet_duration = end - start;
 if ( packet_duration > 0 )
 {   sum += packet_duration;
     recvnum++;
 }
 }
 #Print packet delivery ratio
   delay=sum/recvnum;
   NRL = routing_packets/recvs; #normalized routing load
   PDF = (recvs/sends)*100; #packet delivery ratio[fraction]
   printf("Send Packets = %.2f\n",sends);
   printf("Received Packets = %.2f\n",recvs);
   printf("Routing Packets = %.2f\n",routing_packets++);
   printf("Packet Delivery Function = %.2f\n",PDF);
   printf("Normalised Routing Load = %.2f\n",NRL);
   printf("Average end to end delay(ms)= %.2f\n",delay*1000);
   print("No. of dropped packets = ",droppedPackets);

}
```

.sh File:

a) Throughput Graph

```
set terminal png
#print output into
set output 'Result.png'
#set xrange [0.0:3.0]
set xlabel "Time(in seconds)"
set autoscale xfix
set autoscale
#set yrange [0:80]
set ylabel "Throughput(in Kbps)"
set grid
set style data linespoints
#print output into
set output "outwr.png"
#plot the graph
```

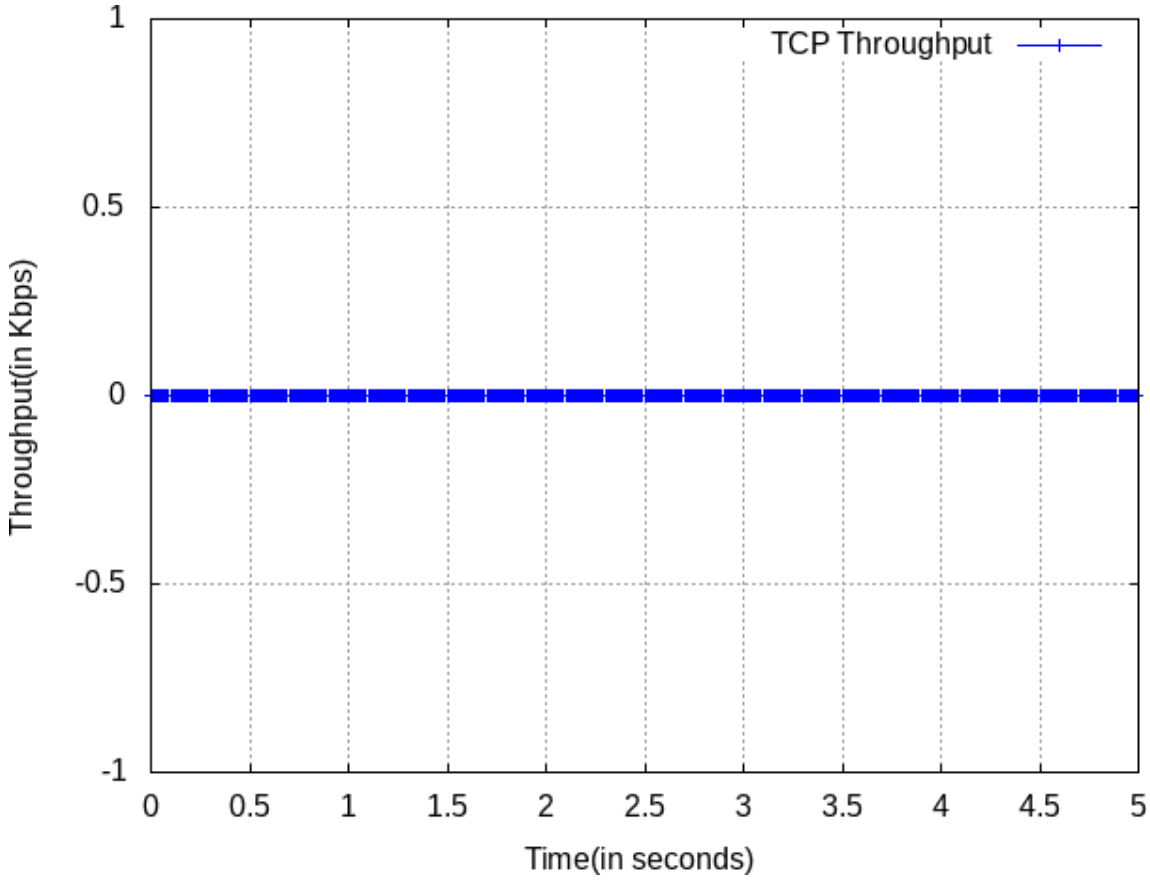
```
plot "outwr" using 1:2 title "TCP Throughput" lt rgb "blue"
```

b) Packet Ratio Graph

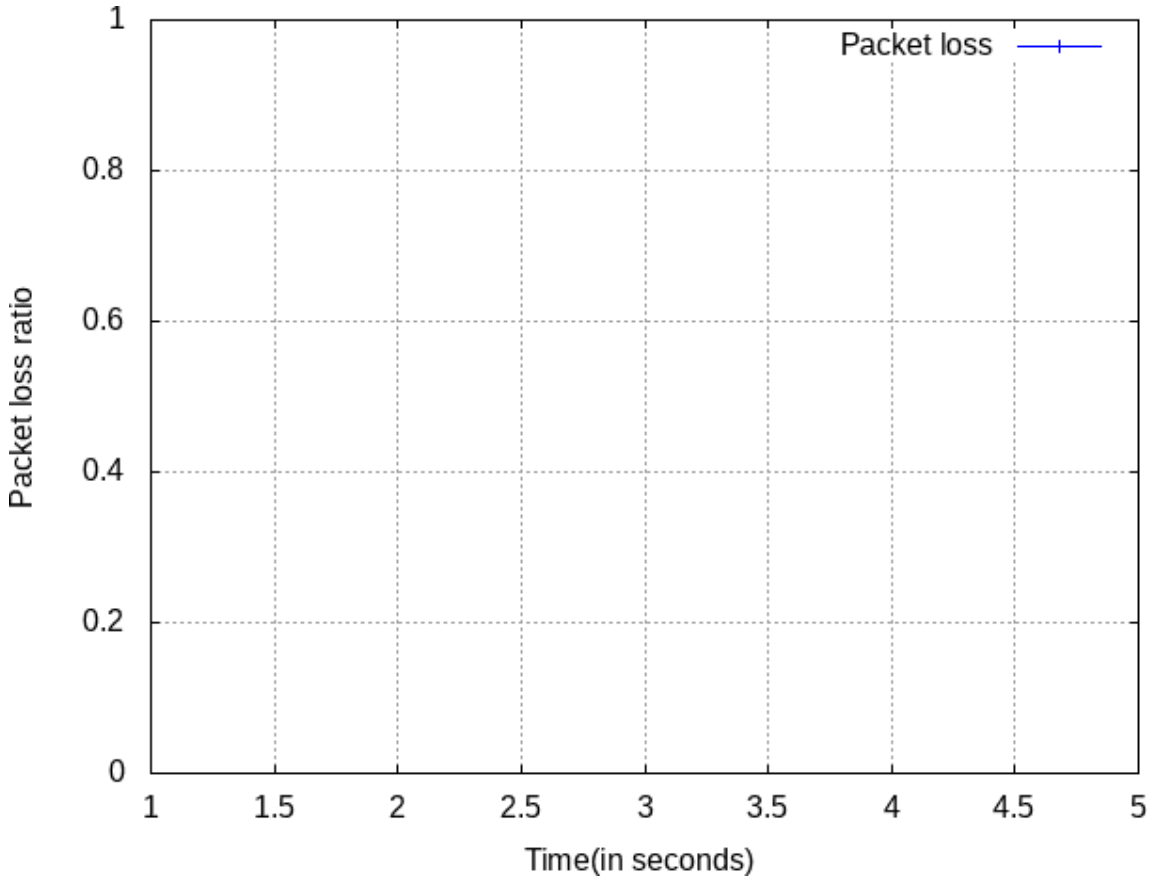
```
set terminal png
#print output into
set output 'result1.png'
#set xrange
set xrange [0.00:2.00]
set xlabel "Time(in seconds)"
set autoscale
#set yrange
set yrange [0:1]
set ylabel "Packet loss ratio"
set grid
set style data linespoints
#plot the graph
plot "tp1" using 1:2 title "Packet loss" lt rgb "blue"
```

Graph:

a) Throughput



b) Packet Delivery Ratio



Star Topology

.tcl File:

```
#Create a simulator object
set ns [new Simulator]

#Define different colors and labels for data flows
$ns color 1 blue
$ns color 2 red

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

#Open the nam trace file
set f [open star.tr w]
$ns trace-all $f

#Define a 'finish' procedure
proc finish {} {
    global ns nf f
    $ns flush-trace
    close $nf
    close $f
    exec nam star.nam
    exit 0
    }

#Creating nodes
for {set i 0} {$i<7} {incr i} {
set n($i) [$ns node]
}

#Create link between nodes
for {set i 1} {$i<7} {incr i} {
$ns duplex-link $n(0) $n($i) 512Kb 10ms SFQ
}
```

```
#Orienting nodes wrt topology
$ns duplex-link-op $n(0) $n(1) orient left-up
$ns duplex-link-op $n(0) $n(2) orient right-up
$ns duplex-link-op $n(0) $n(3) orient right
$ns duplex-link-op $n(0) $n(4) orient right-down
$ns duplex-link-op $n(0) $n(5) orient left-down
$ns duplex-link-op $n(0) $n(6) orient left

#Setup a TCP connection
set tcp0 [new Agent/TCP]
$tcp0 set class_ 1
$ns attach-agent $n(1) $tcp0

#Setup a TCP sink connection
set sink0 [new Agent/TCPSink]
$ns attach-agent $n(4) $sink0

#Connect the traffic sources with the traffic sink
$ns connect $tcp0 $sink0

#Setup a UDP connection
set udp0 [new Agent/UDP]
$udp0 set class_ 2
$ns attach-agent $n(2) $udp0

#Create a Null agent (a traffic sink) and attach it to node n(5)
set null0 [new Agent/Null]
$ns attach-agent $n(5) $null0

#Connect the traffic sources with the traffic sink
$ns connect $udp0 $null0

# Create a CBR traffic source and attach it to udp0
set cbr0 [new Application/Traffic/CBR]
$cbr0 set rate_ 256Kb
$cbr0 attach-agent $udp0
```

```
#Setup a FTP connection
set ftp0 [new Application/FTP]
$ftp0 attach-agent $tcp0

#Setup rtmodel for nodes
$ns rtmodel-at 0.5 down $n(0) $n(5)
$ns rtmodel-at 0.9 up $n(0) $n(5)

$ns rtmodel-at 0.7 down $n(0) $n(4)
$ns rtmodel-at 1.2 up $n(0) $n(4)

#Schedule events for the FTP agents
$ns at 0.1 "$ftp0 start"
$ns at 1.5 "$ftp0 stop"

#Schedule events for the CBR agents
$ns at 0.2 "$cbr0 start"
$ns at 1.3 "$cbr0 stop"

#Call the finish procedure after 2 seconds of simulation time
$ns at 2.0 "finish"
#Run the simulation
$ns run
```

.nam File:

```
V -t * -v 1.0a5 -a 0
A -t * -n 1 -p 0 -o 0x7fffffff -c 30 -a 1
A -t * -h 1 -m 1073741823 -s 0
c -t * -i 1 -n blue
c -t * -i 2 -n red
n -t * -a 4 -s 4 -S UP -v circle -c black -i black
n -t * -a 0 -s 0 -S UP -v circle -c black -i black
n -t * -a 5 -s 5 -S UP -v circle -c black -i black
n -t * -a 1 -s 1 -S UP -v circle -c black -i black
n -t * -a 6 -s 6 -S UP -v circle -c black -i black
n -t * -a 2 -s 2 -S UP -v circle -c black -i black
n -t * -a 3 -s 3 -S UP -v circle -c black -i black
```

.tr File:

```
+ 0.1 1 0 tcp 40--------1 1.0 4.0 0 0
- 0.1 1 0 tcp 40 ------- 1 1.0 4.0 0 0
r 0.110625 1 0 tcp 40 ------- 1 1.0 4.0 0 0
+ 0.110625 0 4 tcp 40--------1 1.0 4.0 0 0
- 0.110625 0 4 tcp 40 ------- 1 1.0 4.0 0 0
r 0.12125 0 4 tcp 40--------1 1.0 4.0 0 0
+ 0.12125 4 0 ack 40 --------1 4.0 1.0 0 1
- 0.12125 4 0 ack 40--------1 4.0 1.0 0 1
r 0.131875 4 0 ack 40--------1 4.0 1.0 0 1
+ 0.131875 0 1 ack 40 ------- 1 4.0 1.0 0 1
- 0.131875 0 1 ack 40--------1 4.0 1.0 0 1
r 0.1425 0 1 ack 40 ------- 1 4.0 1.0 0 1
```

.awk File:

a) Throughput

```
#init variables
BEGIN {
recv=0;
gotime = 0;
time = 0;
packet_size = $6;
time_interval=0.01;
}
#body
{
 #refer variables to trace file format
     event = $1
          time = $2
          node_id = $4
          pktType = $5
             packet_size = $6
         #write throughput into .txt file
 if(time>gotime) {

  print gotime, (packet_size * recv * 8.0)/1000; #packet size * ... gives results in
kbps
  gotime+= time_interval;
  recv=0;
```

```
 }

#calculate throughput

 if (( event == "r") && ( pktType == "tcp" ))
 {
   recv++;
 }

} #body


END {
;
}
```

b) Packet Ratio

```
#init variables
BEGIN{
time=0;
time_interval=0.01;
packetsize=$6;
gotime=1;
a=0;
b=0;
}

{
        #refer variables to trace file format
        event=$1
        time=$2
        from=$3
        to=$4
        pktType=$5
        packet_size=$6
        #calculate packets sent and received
if(event=="+")
 {
```

```
   a++;
 }
else if(event=="-")
{
  b++;
}
#write packet delivery ratio into .txt file
 if(time>gotime) {
   print gotime,(a-b)/a;
   gotime+=time_interval;
 }

}

END{
;
}
```

.sh File:

a) Throughput Graph

```
set terminal png
#print output into
set output 'Result.png'
#set xrange [0.0:3.0]
set xlabel "Time(in seconds)"
set autoscale xfix
set autoscale
#set yrange [0:80]
set ylabel "Throughput(in Kbps)"
set grid
set style data linespoints
#print output into
set output "star.png"
#plot the graph
plot "star" using 1:2 title "TCP Throughput" lt rgb "blue"
```

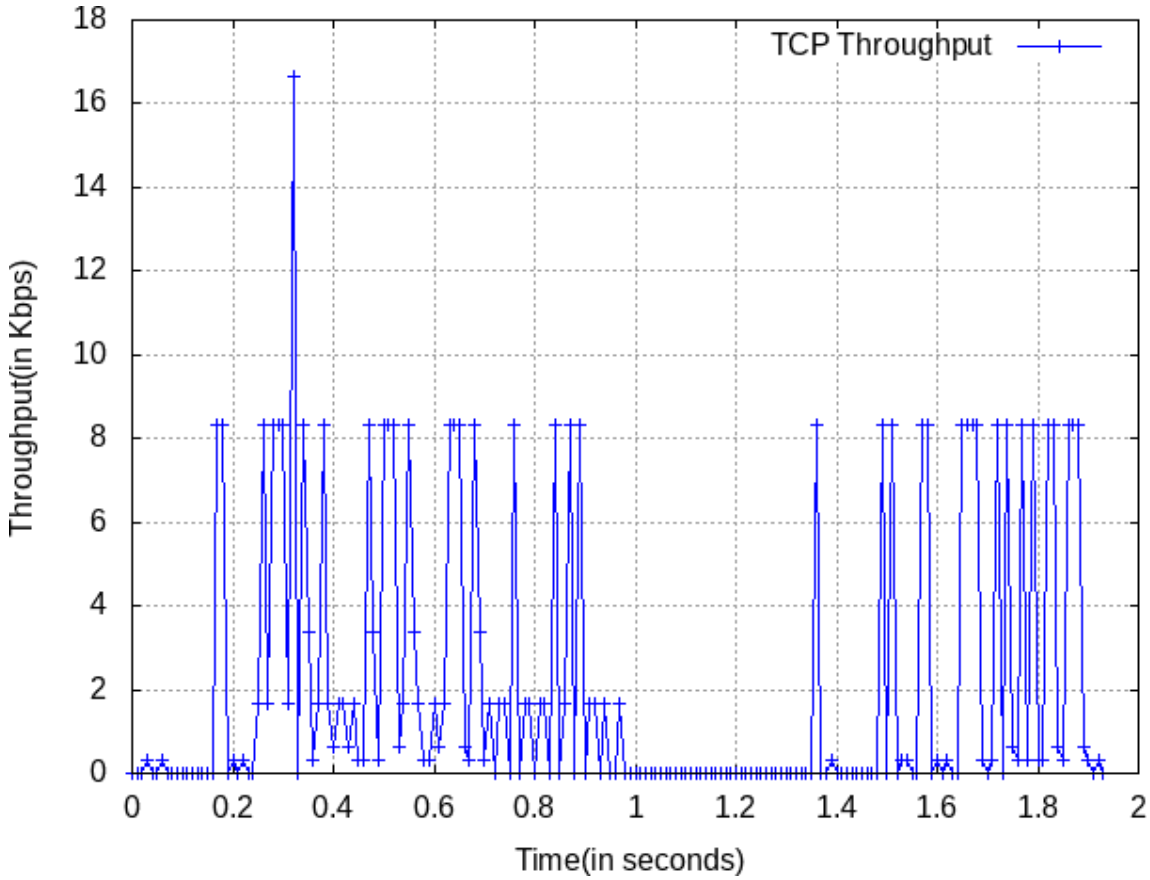
b) Packet Ratio Graph

```
set terminal png
#print output into
```

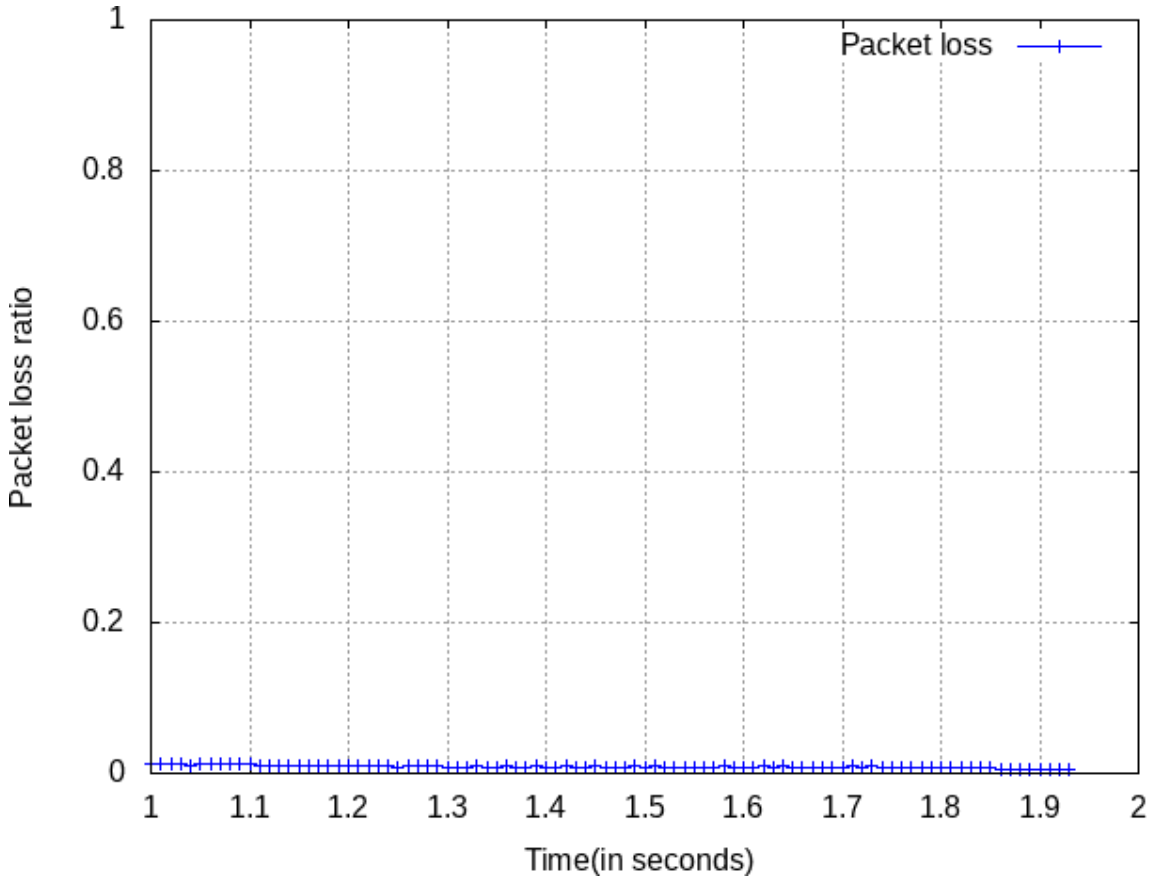
```
set output 'result1.png'
#set xrange
set xrange [0.00:2.00]
set xlabel "Time(in seconds)"
set autoscale
#set yrange
set yrange [0:1]
set ylabel "Packet loss ratio"
set grid
set style data linespoints
#plot the graph
plot "tp1" using 1:2 title "Packet loss" lt rgb "blue"
```

Graph:

a) Throughput



b) Packet Delivery Ratio



Bus Topology

.tcl File:

```
#LAN configuration
LanRouter set debug_ 0

#Create a simulator object
set ns [new Simulator]

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

#Open the nam trace file
set tf [open bus.tr w]
$ns trace-all $tf

#Define a 'finish' procedure
proc finish {} {
        global ns nf tf
        $ns flush-trace
        close $nf
        close $tf
        exec nam bus.nam &
        exit 0
}

#Creating nodes
for {set i 0} {$i < 10} {incr i} {
   set n($i) [$ns node]
}

#Set a dummy node
set dummy [$ns node]

#Create link between dummy node and n(0)
$ns duplex-link $dummy $n(0) 2Mb 10ms DropTail
$ns duplex-link-op $dummy $n(0) orient right
```

```
#Create link between nodes
set lan [ $ns newLan "$n(0) $n(1) $n(2) $n(3) $n(4) $n(5) $n(6) $n(7) $n(8)
$n(9)" 2Mb 10ms LL Queue/DropTail MAC/-802_3 Channel ]
#Set source and destination
set src0 [expr int(rand()*10)%10]
set dst0 [expr int(rand()*10)%10]
set src1 [expr int(rand()*10)%10]
set dst1 [expr int(rand()*10)%10]

#Setup a UDP connection udp0
set udp0 [new Agent/UDP]
$ns attach-agent $n($src0) $udp0

#Setup a UDP connection udp1
set udp1 [new Agent/UDP]
$ns attach-agent $n($src1) $udp1

#Setup a TCP sink connection sink0
set sink0 [new Agent/LossMonitor]
$ns attach-agent $n($dst0) $sink0

#Setup a TCP sink connection sink1
set sink1 [new Agent/LossMonitor]
$ns attach-agent $n($dst1) $sink1

#Connect the traffic sources with the traffic sink
$ns connect $udp0 $sink0
$ns connect $udp1 $sink1

# Create a CBR traffic source and attach it to udp0
set cbr0 [new Application/Traffic/CBR]
$cbr0 set packetSize_ 1000
$cbr0 set rate_ 4Mb
$cbr0 attach-agent $udp0

# Create a CBR traffic source and attach it to udp1
```

```
set cbr1 [new Application/Traffic/CBR]
$cbr1 set packetSize_ 1000
$cbr1 set rate_ 4Mb
$cbr1 attach-agent $udp1

#Schedule events for the CBR agents
$ns at 1.0 "$cbr0 start"
$ns at 1.0 "$cbr1 start"
$ns at 49.0 "$cbr0 stop"
$ns at 49.0 "$cbr1 stop"

#Call the finish procedure after 50 seconds of simulation time
$ns at 50.0 "finish"
#Run the simulation
$ns run
```

.nam File:

```
V -t * -v 1.0a5 -a 0
A -t * -n 1 -p 0 -o 0x7fffffff -c 30 -a 1
A -t * -h 1 -m 1073741823 -s 0
n -t * -a 0 -s 0 -S UP -v circle -c black -i black
n -t * -a 1 -s 1 -S UP -v circle -c black -i black
n -t * -a 2 -s 2 -S UP -v circle -c black -i black
n -t * -a 3 -s 3 -S UP -v circle -c black -i black
n -t * -a 4 -s 4 -S UP -v circle -c black -i black
n -t * -a 5 -s 5 -S UP -v circle -c black -i black
n -t * -a 6 -s 6 -S UP -v circle -c black -i black
n -t * -a 7 -s 7 -S UP -v circle -c black -i black
n -t * -a 8 -s 8 -S UP -v circle -c black -i black
```

.tr File:

```
h 1 3 11 cbr 1000 ------- 0 3.0 5.1 0 0
h 1 5 11 cbr 1000 ------- 0 5.0 6.0 0 1
h 1.002 3 11 cbr 1000 ------- 0 3.0 5.1 1 2
h 1.002 5 11 cbr 1000 ------- 0 5.0 6.0 1 3
h 1.004 3 11 cbr 1000 ------- 0 3.0 5.1 2 4
h 1.004 5 11 cbr 1000 ------- 0 5.0 6.0 2 5
h 1.006 3 11 cbr 1000 ------- 0 3.0 5.1 3 6
h 1.006 5 11 cbr 1000 ------- 0 5.0 6.0 3 7
```

```
h 1.008 3 11 cbr 1000 ------- 0 3.0 5.1 4 8
h 1.008 5 11 cbr 1000 ------- 0 5.0 6.0 4 9
+ 1.01 3 11 cbr 1000 ------- 0 3.0 5.1 0 0
- 1.01 3 11 cbr 1000--------0 3.0 5.1 0 0
```

.awk File:

a) Throughput

```
#init variables
BEGIN {
recv=0;
gotime = 0;
time = 0;
packet_size = $6;
time_interval=0.01;
}
#body
{
 #refer variables to trace file format
     event = $1
          time = $2
          node_id = $4
          pktType = $5
             packet_size = $6
         #write throughput into .txt file
 if(time>gotime) {

 print gotime, (packet_size * recv * 8.0)/1000; #packet size * ... gives results in
kbps
 gotime+= time_interval;
 recv=0;
 }

#calculate throughput

 if (( event == "r") && ( pktType == "tcp" ))
 {
   recv++;
 }
```

```
} #body


END {
;
}
```

b) Packet Ratio

```
#init variables
BEGIN{
time=0;
time_interval=0.01;
packetsize=$6;
gotime=1;
a=0;
b=0;
}
{
        #refer variables to trace file format
        event=$1
        time=$2
        from=$3
        to=$4
        pktType=$5
        packet_size=$6
        #calculate packets sent and received
if(event=="+")
 {
   a++;
 }
else if(event=="-")
{
  b++;
}
#write packet delivery ratio into .txt file
 if(time>gotime) {
```

```
   print gotime,(a-b)/a;
   gotime+=time_interval;
 }

}

END{
;
}
```

.sh File:

a) Throughput Graph

```
set terminal png
#print output into
set output 'Result.png'
#set xrange [0.0:3.0]
set xlabel "Time(in seconds)"
set autoscale xfix
set autoscale
#set yrange [0:80]
set ylabel "Throughput(in Kbps)"
set grid
set style data linespoints
#print output into
set output "bus.png"
#plot the graph
plot "bus" using 1:2 title "TCP Throughput" lt rgb "blue"
```

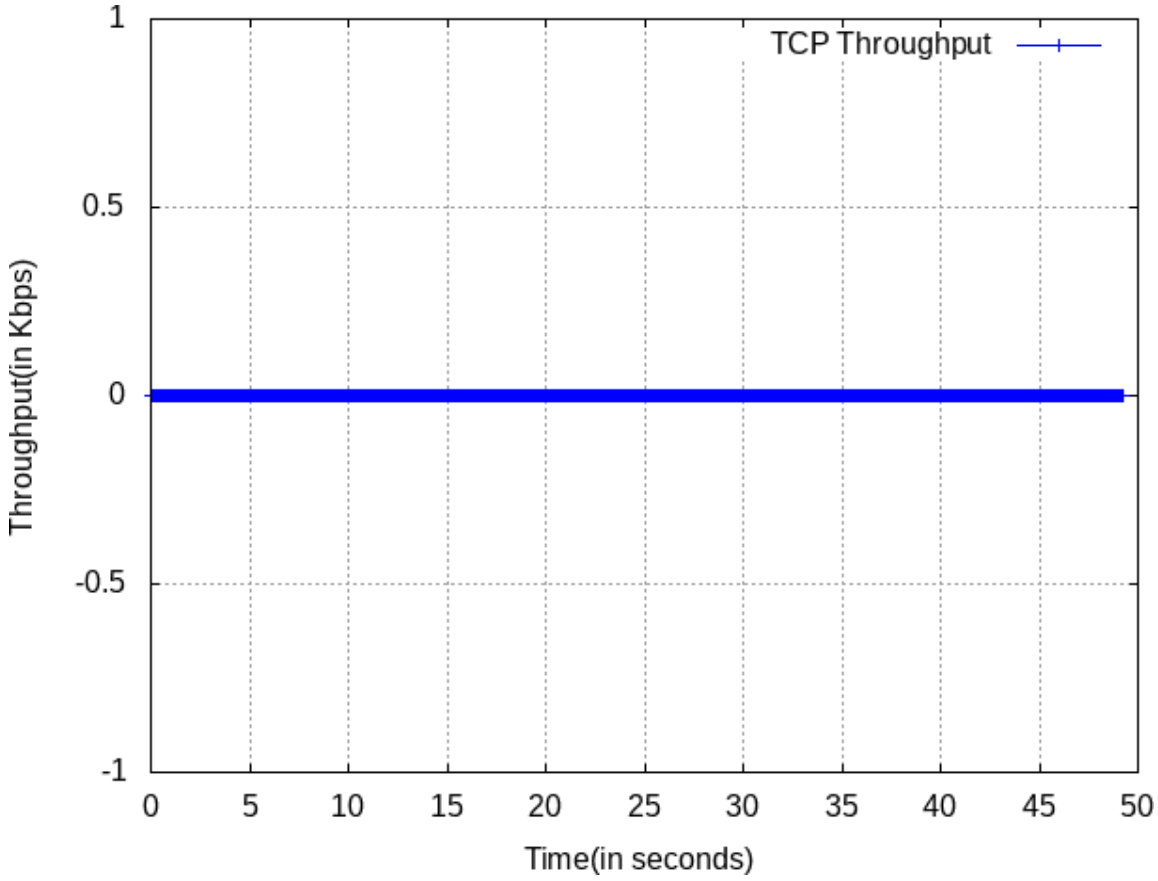
b) Packet Ratio Graph

```
set terminal png
#print output into
set output 'result1.png'
#set xrange
set xrange [0.00:2.00]
set xlabel "Time(in seconds)"
set autoscale
#set yrange
set yrange [0:1]
set ylabel "Packet loss ratio"
```

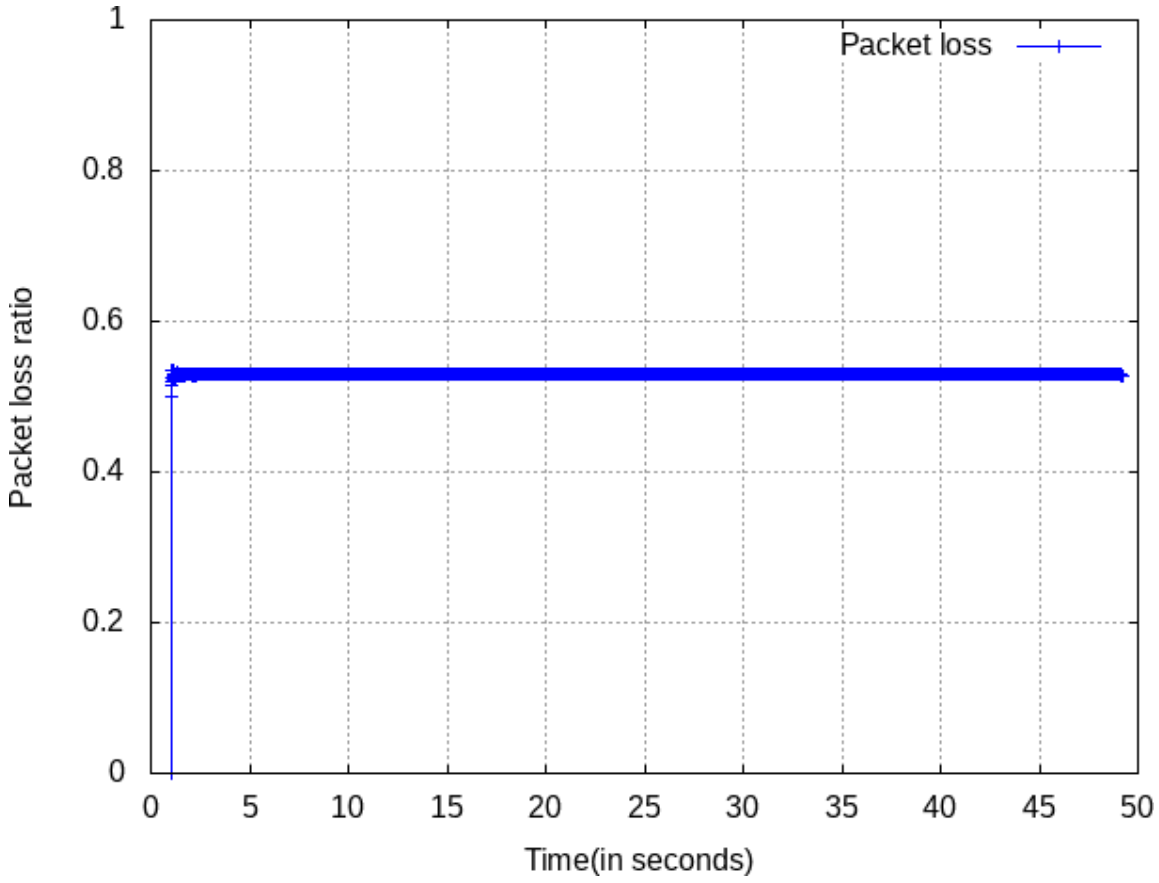
```
set grid
set style data linespoints
#plot the graph
plot "tp1" using 1:2 title "Packet loss" lt rgb "blue"
```

Graph:

a) Throughput



b) Packet Delivery Ratio



Mesh Topology

.tcl File:

```
#Create a simulator object
set ns [new Simulator]

#Define different colors and labels for data flows
$ns color 1 Blue
$ns color 2 Red

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

#Open the nam trace file
set tr [open mesh.tr w]
$ns trace-all $tr

#Define a 'finish' procedure
proc finish {} {
   global ns nf
   $ns flush-trace
   close $nf
   exec nam mesh.nam &
   exit 0
}

#Creating nodes
set n0 [$ns node]
set n1 [$ns node]
set n2 [$ns node]
set n3 [$ns node]

#Create link between nodes
$ns duplex-link $n0 $n1 1Mb 10ms DropTail
$ns duplex-link $n0 $n2 1Mb 10ms DropTail
$ns duplex-link $n0 $n3 1Mb 10ms DropTail
$ns duplex-link $n1 $n2 1Mb 10ms DropTail
```

```
$ns duplex-link $n1 $n3 1Mb 10ms DropTail
$ns duplex-link $n2 $n3 1Mb 10ms DropTail

#Setup a TCP connection
set tcp0 [new Agent/TCP]
$tcp0 set class_ 1
$ns attach-agent $n1 $tcp0

#Setup a TCP sink connection
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 packageSize_ 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 2 seconds of simulation time
$ns at 5.0 "finish"
#Run the simulation
$ns run
```

.nam File:

```
V -t * -v 1.0a5 -a 0
A -t * -n 1 -p 0 -o 0x7fffffff -c 30 -a 1
A -t * -h 1 -m 1073741823 -s 0
c -t * -i 1 -n Blue
c -t * -i 2 -n Red
n -t * -a 0 -s 0 -S UP -v circle -c black -i black
n -t * -a 1 -s 1 -S UP -v circle -c black -i black
n -t * -a 2 -s 2 -S UP -v circle -c black -i black
```

```
n -t * -a 3 -s 3 -S UP -v circle -c black -i black
l -t * -s 0 -d 1 -S UP -r 1000000 -D 0.01 -c black
l -t * -s 0 -d 2 -S UP -r 1000000 -D 0.01 -c black
l -t * -s 0 -d 3 -S UP -r 1000000 -D 0.01 -c black
```

.tr File:

```
+ 0.5 1 3 tcp 40--------1 1.0 3.0 0 0
- 0.5 1 3 tcp 40 ------- 1 1.0 3.0 0 0
r 0.51032 1 3 tcp 40--------1 1.0 3.0 0 0
+ 0.51032 3 1 ack 40--------1 3.0 1.0 0 1
- 0.51032 3 1 ack 40--------1 3.0 1.0 0 1
r 0.52064 3 1 ack 40 ------- 1 3.0 1.0 0 1
+ 0.52064 1 3 tcp 1040--------1 1.0 3.0 1 2
- 0.52064 1 3 tcp 1040--------1 1.0 3.0 1 2
+ 0.52064 1 3 tcp 1040--------1 1.0 3.0 2 3
- 0.52896 1 3 tcp 1040--------1 1.0 3.0 2 3
r 0.53896 1 3 tcp 1040 ------- 1 1.0 3.0 1 2
+ 0.53896 3 1 ack 40--------1 3.0 1.0 1 4
```

.awk File:

a) Throughput

```
#init variables
BEGIN {
recv=0;
gotime = 0;
time = 0;
packet_size = $6;
time_interval=0.01;
}
#body
{
 #refer variables to trace file format
     event = $1
          time = $2
          node_id = $4
          pktType = $5
             packet_size = $6
         #write throughput into .txt file
 if(time>gotime) {
```

```
 print gotime, (packet_size * recv * 8.0)/1000; #packet size * ... gives results in
kbps
 gotime+= time_interval;
 recv=0;
 }

#calculate throughput

 if (( event == "r") && ( pktType == "tcp" ))
 {
   recv++;
 }

} #body


END {
;
}
```

b) Packet Ratio

```
#init variables
BEGIN{
time=0;
time_interval=0.01;
packetsize=$6;
gotime=1;
a=0;
b=0;
}

{
        #refer variables to trace file format
        event=$1
        time=$2
        from=$3
        to=$4
```

```
        pktType=$5
        packet_size=$6
        #calculate packets sent and received
if(event=="+")
 {
   a++;
 }
else if(event=="-")
{
  b++;
}
#write packet delivery ratio into .txt file
 if(time>gotime) {
   print gotime,(a-b)/a;
   gotime+=time_interval;
 }

}

END{
;
}
```

.sh File:

a) Throughput Graph

```
set terminal png
#print output into
set output 'Result.png'
#set xrange [0.0:3.0]
set xlabel "Time(in seconds)"
set autoscale xfix
set autoscale
#set yrange [0:80]
set ylabel "Throughput(in Kbps)"
set grid
set style data linespoints
#print output into
set output "mesh.png"
```

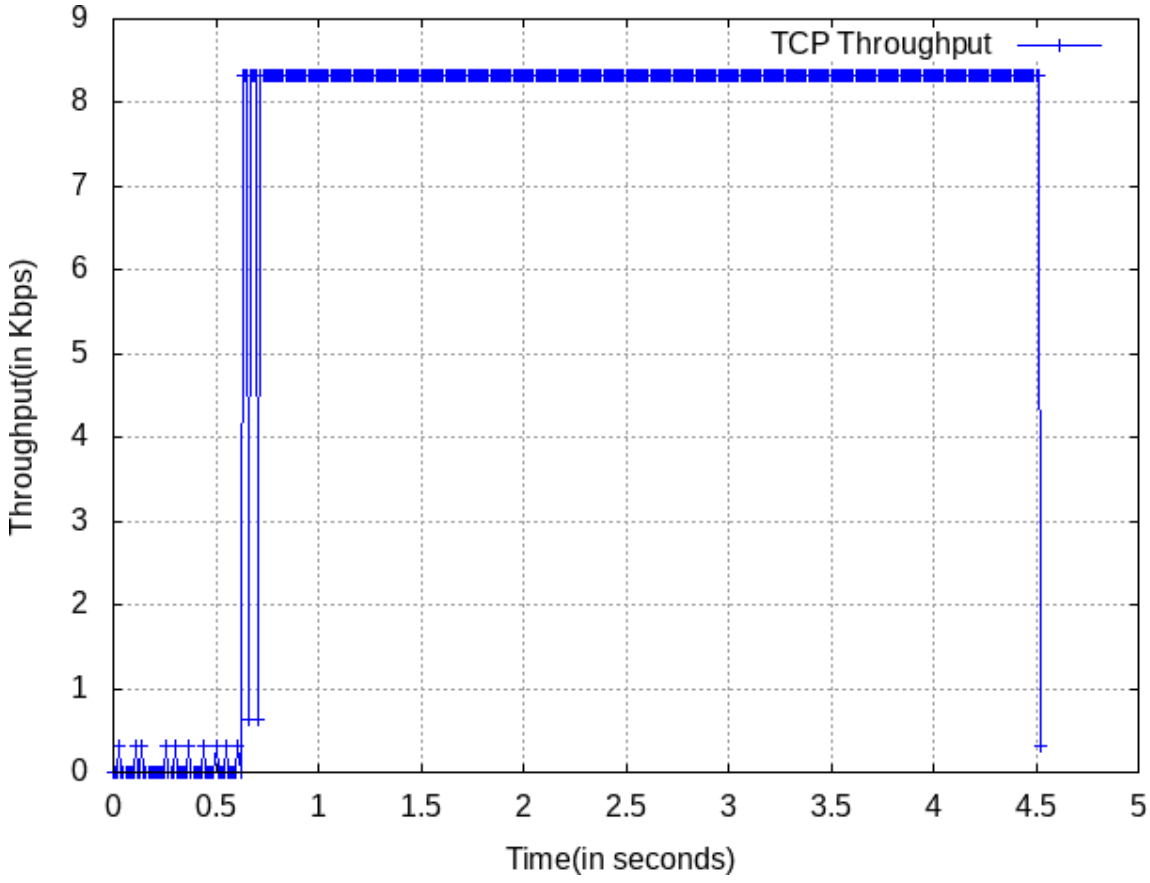
```
#plot the graph
plot "mesh" using 1:2 title "TCP Throughput" lt rgb "blue"
```

b) Packet Ratio Graph

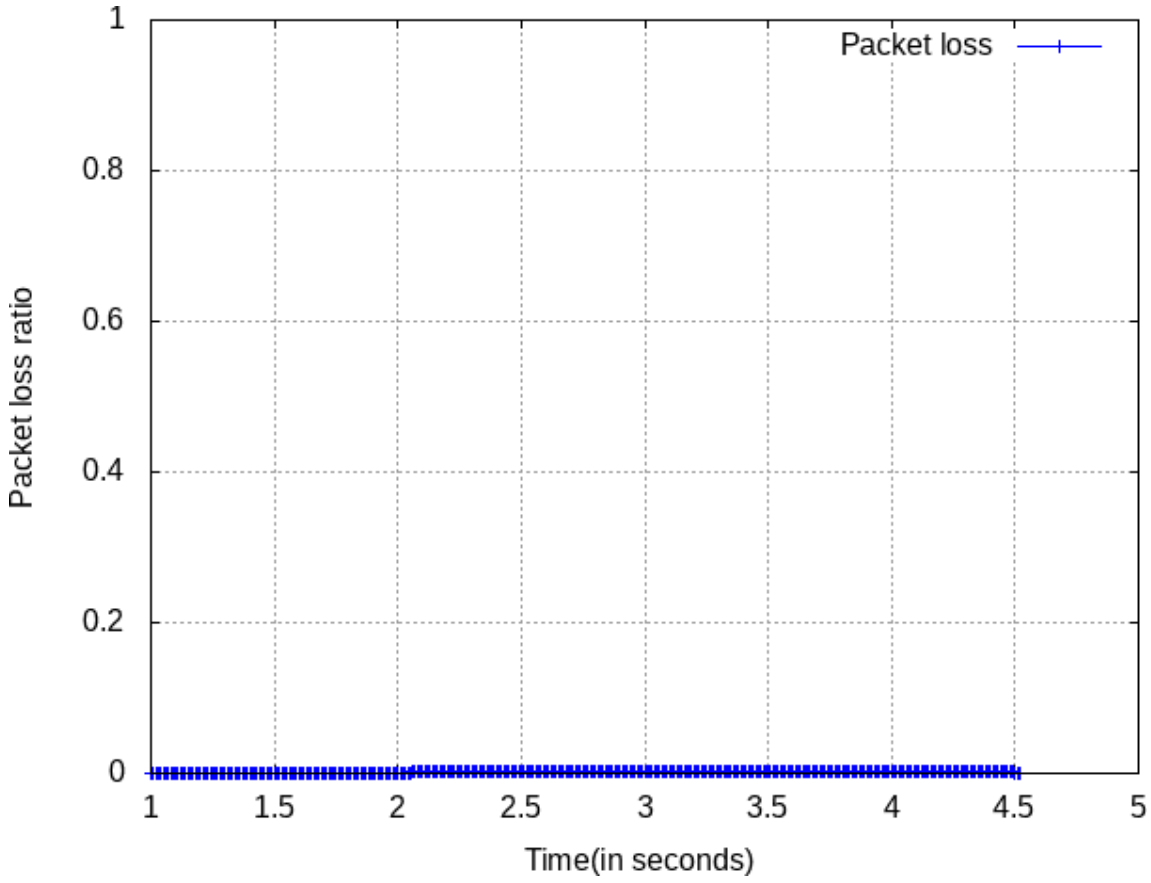
```
set terminal png
#print output into
set output 'result1.png'
#set xrange
set xrange [0.00:2.00]
set xlabel "Time(in seconds)"
set autoscale
#set yrange
set yrange [0:1]
set ylabel "Packet loss ratio"
set grid
set style data linespoints
#plot the graph
plot "tp1" using 1:2 title "Packet loss" lt rgb "blue"
```

Graph:

a) Throughput



b) Packet Delivery Ratio



Ring Topology

.tcl File:

```
#Create a simulator object
set ns [new Simulator]

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

#Open the nam trace file
set f [open ring.tr w]
$ns trace-all $f

#Define a 'finish' procedure
proc finish {} {
        global ns nf f
        $ns flush-trace
        close $nf
        close $f
        exec nam ring.nam &
        exit 0
}

#Creating 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 link between 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

#Setup a TCP connection
set tcp0 [new Agent/TCP]
$tcp0 set class_ 1
$ns attach-agent $n1 $tcp0

#Setup a TCP sink connection
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 File:

```
V -t * -v 1.0a5 -a 0
A -t * -n 1 -p 0 -o 0x7fffffff -c 30 -a 1
A -t * -h 1 -m 1073741823 -s 0
n -t * -a 4 -s 4 -S UP -v circle -c black -i black
n -t * -a 0 -s 0 -S UP -v circle -c black -i black
n -t * -a 5 -s 5 -S UP -v circle -c black -i black
n -t * -a 1 -s 1 -S UP -v circle -c black -i black
n -t * -a 2 -s 2 -S UP -v circle -c black -i black
```

```
n -t * -a 3 -s 3 -S UP -v circle -c black -i black
l -t * -s 0 -d 1 -S UP -r 1000000 -D 0.01 -c black
l -t * -s 1 -d 2 -S UP -r 1000000 -D 0.01 -c black
l -t * -s 2 -d 3 -S UP -r 1000000 -D 0.01 -c black
```

.tr File:

```
+ 0.5 1 2 tcp 40--------1 1.0 3.0 0 0
- 0.5 1 2 tcp 40 ------- 1 1.0 3.0 0 0
r 0.51032 1 2 tcp 40--------1 1.0 3.0 0 0
+ 0.51032 2 3 tcp 40 ------- 1 1.0 3.0 0 0
- 0.51032 2 3 tcp 40--------1 1.0 3.0 0 0
r 0.52064 2 3 tcp 40--------1 1.0 3.0 0 0
+ 0.52064 3 2 ack 40--------1 3.0 1.0 0 1
- 0.52064 3 2 ack 40--------1 3.0 1.0 0 1
r 0.53096 3 2 ack 40 ------- 1 3.0 1.0 0 1
+ 0.53096 2 1 ack 40--------1 3.0 1.0 0 1
- 0.53096 2 1 ack 40--------1 3.0 1.0 0 1
r 0.54128 2 1 ack 40 ------- 1 3.0 1.0 0 1
```

.awk File:

a) Throughput

```
#init variables
BEGIN {
recv=0;
gotime = 0;
time = 0;
packet_size = $6;
time_interval=0.01;
}
#body
{
 #refer variables to trace file format
     event = $1
          time = $2
          node_id = $4
          pktType = $5
             packet_size = $6
         #write throughput into .txt file
 if(time>gotime) {
```

```
  print gotime, (packet_size * recv * 8.0)/1000; #packet size * ... gives results in
kbps
  gotime+= time_interval;
  recv=0;
  }

#calculate throughput

  if (( event == "r") && ( pktType == "tcp" ))
  {
    recv++;
  }

} #body


END {
;
}
```

b) Packet Ratio

```
#init variables
BEGIN{
time=0;
time_interval=0.01;
packetsize=$6;
gotime=1;
a=0;
b=0;
}

{
        #refer variables to trace file format
        event=$1
        time=$2
        from=$3
        to=$4
```

```
        pktType=$5
        packet_size=$6
        #calculate packets sent and received
if(event=="+")
 {
   a++;
 }
else if(event=="-")
{
  b++;
}
#write packet delivery ratio into .txt file
 if(time>gotime) {
   print gotime,(a-b)/a;
   gotime+=time_interval;
 }

}

END{
;
}
```

.sh File:

a) Throughput Graph

```
set terminal png
#print output into
set output 'Result.png'
#set xrange [0.0:3.0]
set xlabel "Time(in seconds)"
set autoscale xfix
set autoscale
#set yrange [0:80]
set ylabel "Throughput(in Kbps)"
set grid
set style data linespoints
#print output into
set output "ring.png"
```

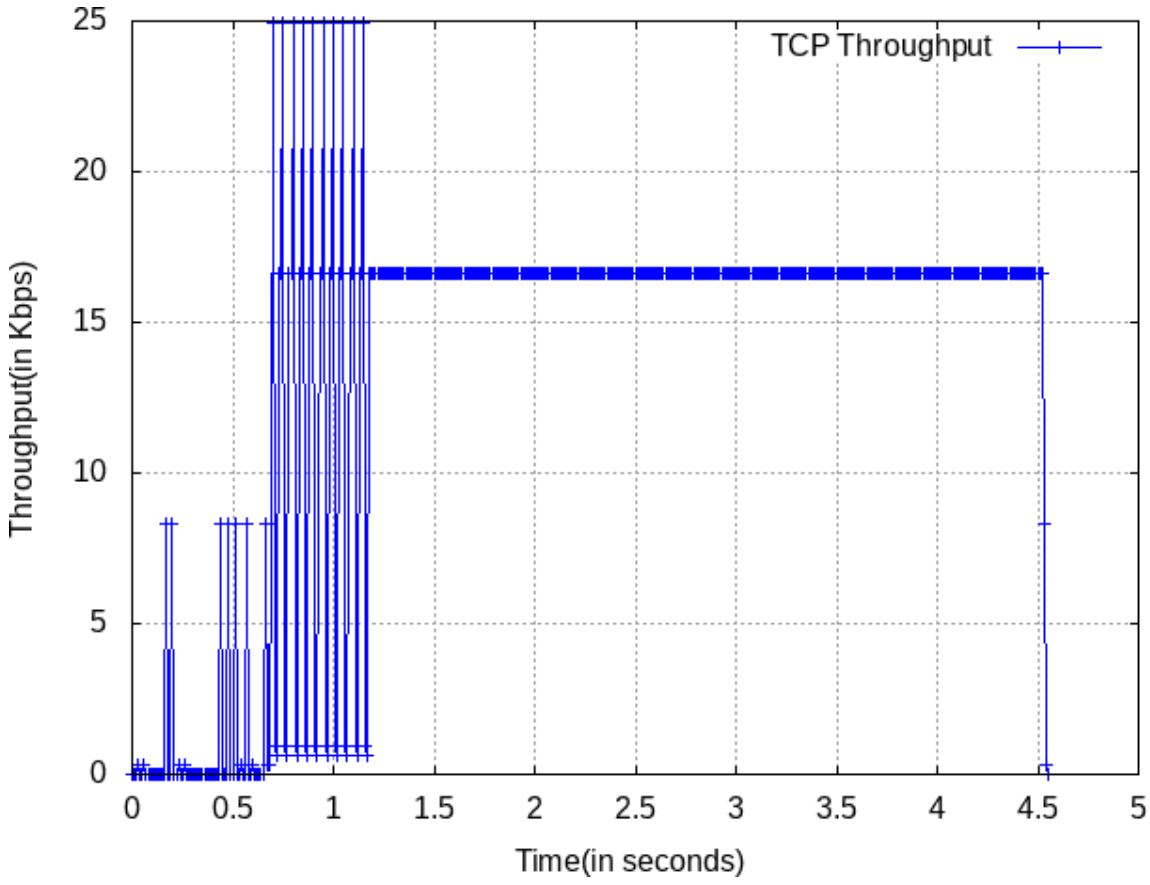
```
#plot the graph
plot "ring" using 1:2 title "TCP Throughput" lt rgb "blue"
```

b) Packet Ratio Graph

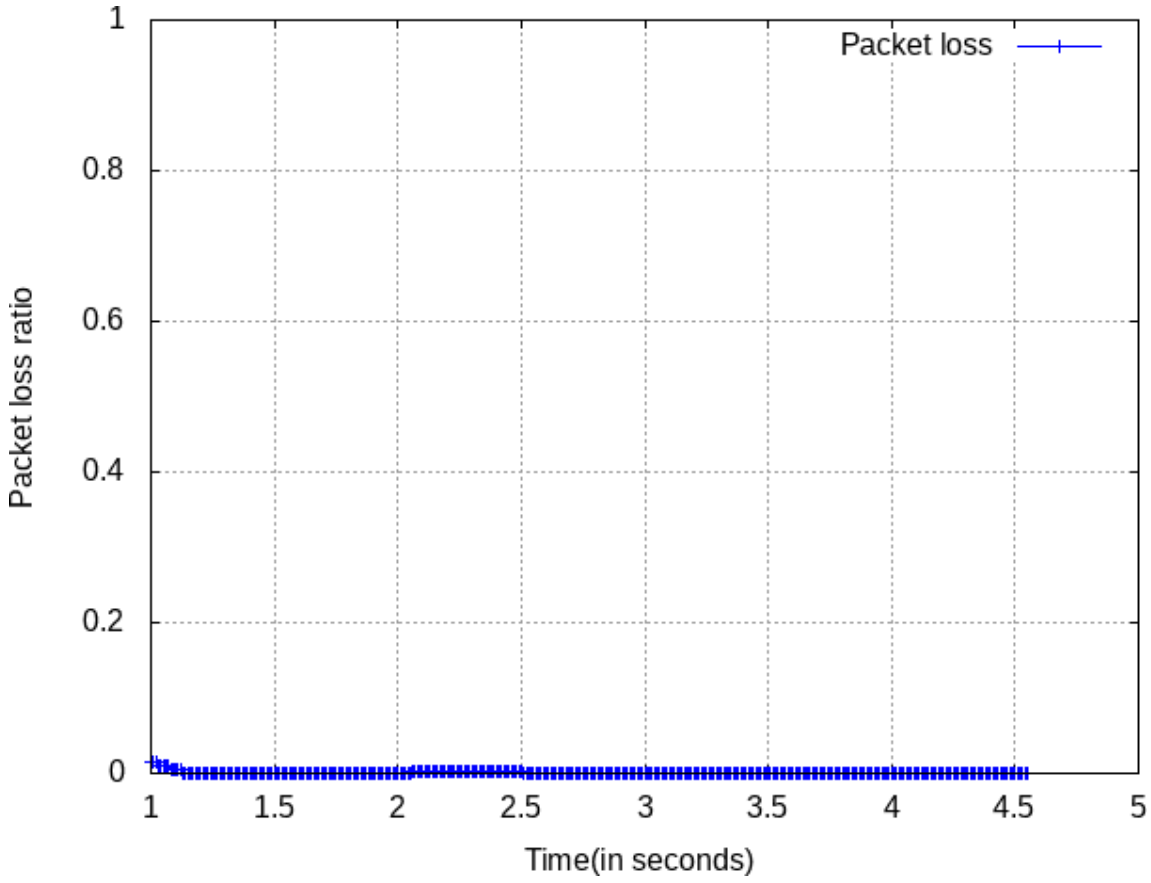
```
set terminal png
#print output into
set output 'result1.png'
#set xrange
set xrange [0.00:2.00]
set xlabel "Time(in seconds)"
set autoscale
#set yrange
set yrange [0:1]
set ylabel "Packet loss ratio"
set grid
set style data linespoints
#plot the graph
plot "tp1" using 1:2 title "Packet loss" lt rgb "blue"
```

Graph:

a) Throughput



b) Packet Delivery Ratio



TCP Tahoe

.tcl File:

```
#Create a simulator object
set ns [new Simulator]

#Define traffic percentage
puts "Give Traffic percentage N"
gets stdin N

#Open the data file
set ptf [open tahoe.dat w]
$ns trace-all $ptf

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

#Open the nam trace file
set f [open tahoe.tr w]
$ns trace-all $f

#Open the congestion data file
set cwnd [open cwnd.dat w]

#Define different colors and labels for data flows
set n(0) [$ns node]
$n(0) color white

#Creating nodes
for {set i 1} {$i <= 50} {incr i} {
set n($i) [$ns node]
$n($i) shape box
$n($i) color green
}

set K [expr (($N * 50)/100) ]
```

```
#Create link between nodes
for {set i 1} {$i < 50} {incr i} {
                set j [expr $i+1]
                $ns duplex-link $n($i) $n($j) 5Mb 10ms DropTail
                $ns duplex-link-op $n($i) $n($j) color "blue"
                $ns queue-limit $n($i) $n($j) 6

        if {$i <= 12} {
 $ns duplex-link-op $n($i) $n($j) orient right
} elseif {$i <= 24} {
        $ns duplex-link-op $n($i) $n($j) orient down
} elseif {$i <= 36} {
        $ns duplex-link-op $n($i) $n($j) orient left
} else {
$ns duplex-link-op $n($i) $n($j) orient up
}

}

for {set i 1} {$i <= $K} {incr i} {
        set j [expr $i + 1]
                #Setup a TCP connection
                set tcp($i,$j) [new Agent/TCP]
                $ns attach-agent $n($i) $tcp($i,$j)

                #Setup a TCP sink connection
                set sink($i,$j) [new Agent/TCPSink]
                $ns attach-agent $n($j) $sink($i,$j)
                $ns connect $tcp($i,$j) $sink($i,$j)
                #Setup a FTP connection
                set ftp($i,$j) [new Application/FTP]
                $ftp($i,$j) attach-agent $tcp($i,$j)

                #Schedule events for the FTP agents
                $ns at 0.0 "$ftp($i,$j) start"
                $ns at 50.0 "$ftp($i,$j) stop"
```

```
}

#Define a 'recWin' procedure
proc recWin {tcpSender f} {
        global ns
        set time 0.1
        set cTime [$ns now]
        set wnd [$tcpSender set cwnd_]
        puts $f "$cTime $wnd"
        $ns at [expr $cTime + $time] "recWin $tcpSender $f"
}

#Schedule events for the congestion control agents
$ns at 0.1 "recWin $tcp(1,2) $cwnd"
#Call the finish procedure after 50 seconds of simulation time
$ns at 50.0 "finish"
#Define a 'finish' procedure
proc finish {} {
        global ns nf ptf
        $ns flush-trace
        close $nf
        close $ptf
        exec nam tahoe.nam &
        exit 0
}

#Run the simulation
$ns run
```

.nam File:

```
V -t * -v 1.0a5 -a 0
A -t * -n 1 -p 0 -o 0x7fffffff -c 30 -a 1
A -t * -h 1 -m 1073741823 -s 0
n -t * -a 35 -s 35 -S UP -v box -c green -i green
n -t * -a 36 -s 36 -S UP -v box -c green -i green
n -t * -a 37 -s 37 -S UP -v box -c green -i green
n -t * -a 38 -s 38 -S UP -v box -c green -i green
n -t * -a 39 -s 39 -S UP -v box -c green -i green
```

```
n -t * -a 40 -s 40 -S UP -v box -c green -i green
n -t * -a 41 -s 41 -S UP -v box -c green -i green
n -t * -a 42 -s 42 -S UP -v box -c green -i green
n -t * -a 43 -s 43 -S UP -v box -c green -i green
```

.tr File:

```
+ 0 1 2 tcp 40 ------- 0 1.0 2.0 0 0
- 0 1 2 tcp 40--------0 1.0 2.0 0 0
+ 0 2 3 tcp 40 ------- 0 2.1 3.0 0 1
- 0 2 3 tcp 40--------0 2.1 3.0 0 1
+ 0 3 4 tcp 40 ------- 0 3.1 4.0 0 2
- 0 3 4 tcp 40--------0 3.1 4.0 0 2
+ 0 4 5 tcp 40 ------- 0 4.1 5.0 0 3
- 0 4 5 tcp 40--------0 4.1 5.0 0 3
+ 0 5 6 tcp 40 ------- 0 5.1 6.0 0 4
- 0 5 6 tcp 40--------0 5.1 6.0 0 4
+ 0 6 7 tcp 40 ------- 0 6.1 7.0 0 5
- 0 6 7 tcp 40--------0 6.1 7.0 0 5
```

.awk File:

a) Throughput

```
#init variables
BEGIN {
recv=0;
gotime = 0;
time = 0;
packet_size = $6;
time_interval=0.35;
}
#body
{
 #refer variables to trace file format
     event = $1
          time = $2
          node_id = $4
          pktType = $5
             packet_size = $6
         #write throughput into .txt file
 if(time>gotime) {
```

```
  print gotime, (packet_size * recv * 8.0)/1000; #packet size * ... gives results in
kbps
  gotime+= time_interval;
  recv=0;
  }

#calculate throughput

  if (( event == "r") && ( pktType == "tcp" ))
  {
    recv++;
  }

}


END {
;
}
```

b) Packet Ratio

```
#init variables
BEGIN{
time=0;
time_interval=0.01;
packetsize=$6;
gotime=1;
a=0;
b=0;
}

{
        #refer variables to trace file format
        event=$1
        time=$2
        from=$3
        to=$4
```

```
        pktType=$5
        packet_size=$6
#write packet delivery ratio into .txt file
if(time>gotime) {
   print gotime,(a-b)/a;
   gotime+=time_interval;
 }
        #calculate packets sent and received
if(event=="+")
 {
   a++;
 }
else if(event=="r")
{
  b++;
}


}
END{
;
}
```

.sh File:

a) Throughput Graph

```
set terminal png
#print output into
set output 'Result.png'
#set xrange [0.0:3.0]
set xlabel "Time(in seconds)"
set autoscale xfix
set autoscale
#set yrange [0:80]
set ylabel "Throughput(in Kbps)"
set grid
set style data linespoints
#print output into
```

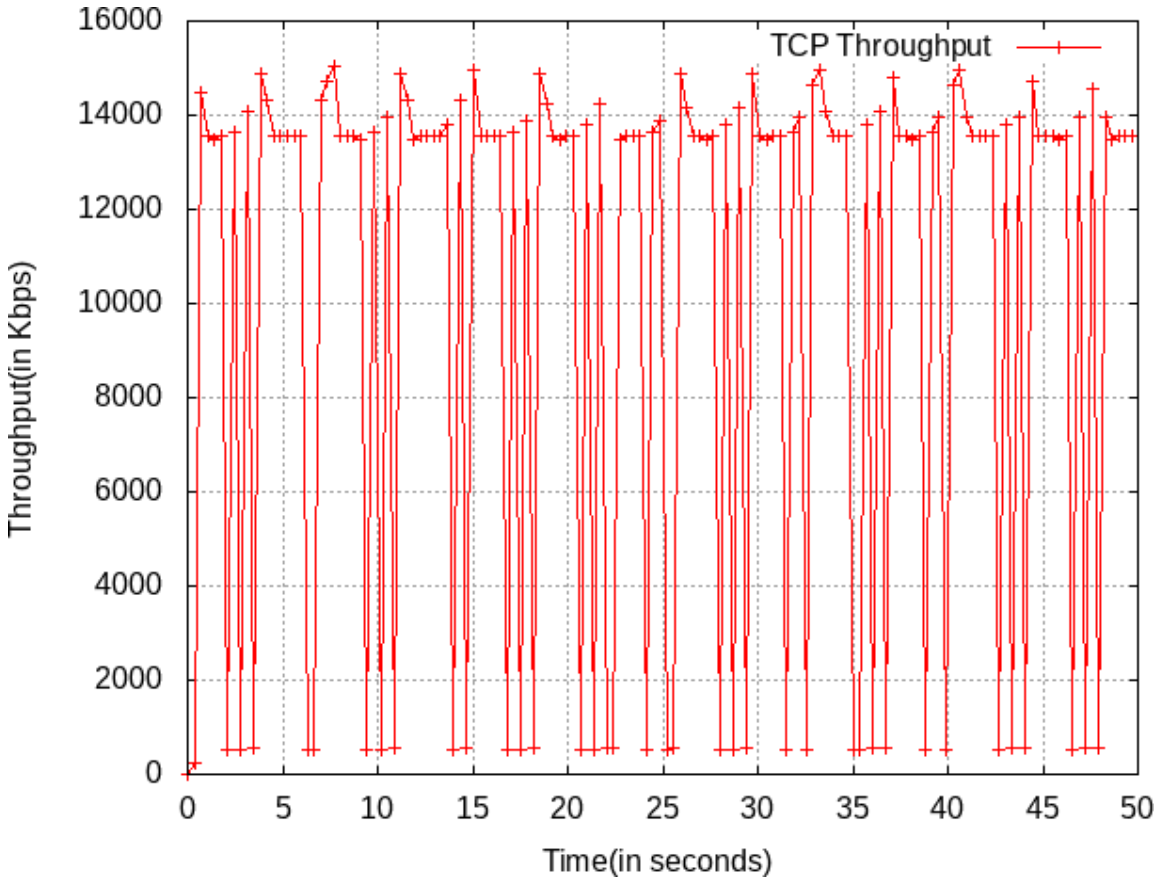
```
set output "tahoe.png"
#plot the graph
plot "tahoe" using 1:2 title "TCP Throughput" lt rgb "red"
```

b) Packet Ratio Graph

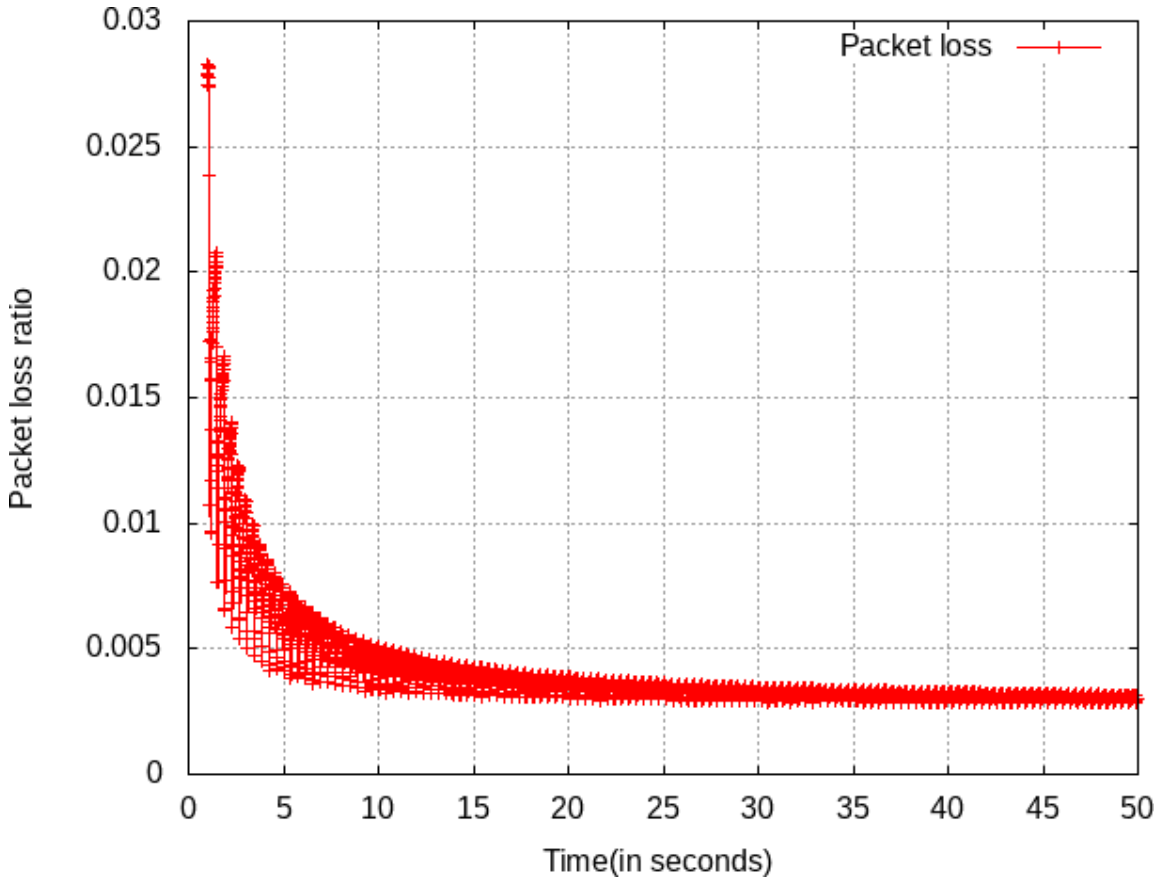
```
set terminal png
#print output into
set output 'result1.png'
#set xrange [0.00:2.00]
set xlabel "Time(in seconds)"
set autoscale
#set yrange [0:1]
set ylabel "Packet loss ratio"
set grid
set style data linespoints
#print output into
set output "tahoep.png"
#plot the graph
plot "tahoep" using 1:2 title "Packet loss" lt rgb "red"
```

Graph:

a) Throughput



b) Packet Delivery Ratio



.cwnd File:

0.10000000000000001 16
0.20000000000000001 5
0.30000000000000004 4
0.40000000000000002 7.78963
0.5 12.0464
0.59999999999999998 16.1061
0.69999999999999996 19.4833
0.79999999999999993 8
0.89999999999999991 12.463
0.99999999999999989 16.5412
1.0999999999999999 19.8946
1.2 8

TCP Reno

.tcl File:

```
#Create a simulator object
set ns [new Simulator]

#Define traffic percentage
puts "Give Traffic percentage N"
gets stdin N

#Open the data file
set ptf [open reno.dat w]
$ns trace-all $ptf

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

#Open the nam trace file
set f [open reno.tr w]
$ns trace-all $f

#Open the congestion data file
set cwnd [open cwnd.dat w]

#Define different colors and labels for data flows
set n(0) [$ns node]
$n(0) color white

#Creating nodes
for {set i 1} {$i <= 50} {incr i} {
set n($i) [$ns node]
$n($i) shape box
$n($i) color green
}

set K [expr (($N * 50)/100) ]
```

```
#Create link between nodes
for {set i 1} {$i < 50} {incr i} {
                set j [expr $i+1]
                $ns duplex-link $n($i) $n($j) 5Mb 10ms DropTail
                $ns duplex-link-op $n($i) $n($j) color "blue"
                $ns queue-limit $n($i) $n($j) 6

        if {$i <= 12} {
 $ns duplex-link-op $n($i) $n($j) orient right
} elseif {$i <= 24} {
        $ns duplex-link-op $n($i) $n($j) orient down
} elseif {$i <= 36} {
        $ns duplex-link-op $n($i) $n($j) orient left
} else {
$ns duplex-link-op $n($i) $n($j) orient up
}

}

for {set i 1} {$i <= $K} {incr i} {
        set j [expr $i + 1]
                #Setup a TCP connection
                set tcp($i,$j) [new Agent/TCP/Reno]
                $ns attach-agent $n($i) $tcp($i,$j)

                #Setup a TCP sink connection
                set sink($i,$j) [new Agent/TCPSink]
                $ns attach-agent $n($j) $sink($i,$j)
                $ns connect $tcp($i,$j) $sink($i,$j)
                #Setup a FTP connection
                set ftp($i,$j) [new Application/FTP]
                $ftp($i,$j) attach-agent $tcp($i,$j)

                #Schedule events for the FTP agents
                $ns at 0.0 "$ftp($i,$j) start"
                $ns at 50.0 "$ftp($i,$j) stop"
```

```
}

#Define a 'recWin' procedure
proc recWin {tcpSender f} {
        global ns
        set time 0.1
        set cTime [$ns now]
        set wnd [$tcpSender set cwnd_]
        puts $f "$cTime $wnd"
        $ns at [expr $cTime + $time] "recWin $tcpSender $f"
}

#Schedule events for the congestion control agents
$ns at 0.1 "recWin $tcp(1,2) $cwnd"
#Call the finish procedure after 50 seconds of simulation time
$ns at 50.0 "finish"

#Define a 'finish' procedure
proc finish {} {
        global ns nf ptf
        $ns flush-trace
        close $nf
        close $ptf
        exec nam reno.nam &
        exit 0
}

#Run the simulation
$ns run
```

.nam File:

```
V -t * -v 1.0a5 -a 0
A -t * -n 1 -p 0 -o 0x7fffffff -c 30 -a 1
A -t * -h 1 -m 1073741823 -s 0
n -t * -a 35 -s 35 -S UP -v box -c green -i green
n -t * -a 36 -s 36 -S UP -v box -c green -i green
n -t * -a 37 -s 37 -S UP -v box -c green -i green
n -t * -a 38 -s 38 -S UP -v box -c green -i green
```

```
n -t * -a 39 -s 39 -S UP -v box -c green -i green
n -t * -a 40 -s 40 -S UP -v box -c green -i green
n -t * -a 41 -s 41 -S UP -v box -c green -i green
n -t * -a 42 -s 42 -S UP -v box -c green -i green
n -t * -a 43 -s 43 -S UP -v box -c green -i green
```

.tr File:

```
+ 0 1 2 tcp 40 ------- 0 1.0 2.0 0 0
- 0 1 2 tcp 40--------0 1.0 2.0 0 0
+ 0 2 3 tcp 40 ------- 0 2.1 3.0 0 1
- 0 2 3 tcp 40--------0 2.1 3.0 0 1
+ 0 3 4 tcp 40 ------- 0 3.1 4.0 0 2
- 0 3 4 tcp 40--------0 3.1 4.0 0 2
+ 0 4 5 tcp 40 ------- 0 4.1 5.0 0 3
- 0 4 5 tcp 40--------0 4.1 5.0 0 3
+ 0 5 6 tcp 40 ------- 0 5.1 6.0 0 4
- 0 5 6 tcp 40--------0 5.1 6.0 0 4
+ 0 6 7 tcp 40 ------- 0 6.1 7.0 0 5
- 0 6 7 tcp 40--------0 6.1 7.0 0 5
```

.awk File:

a) Throughput

```
#init variables
BEGIN {
recv=0;
gotime = 0;
time = 0;
packet_size = $6;
time_interval=0.35;
}
#body
{
 #refer variables to trace file format
     event = $1
          time = $2
          node_id = $4
          pktType = $5
             packet_size = $6
         #write throughput into .txt file
```

```
 if(time>gotime) {

  print gotime, (packet_size * recv * 8.0)/1000; #packet size * ... gives results in
kbps
  gotime+= time_interval;
  recv=0;
  }

#calculate throughput

  if (( event == "r") && ( pktType == "tcp" ))
  {
    recv++;
  }

}

END {
;
}
```

b) Packet Ratio

```
#init variables
BEGIN{
time=0;
time_interval=0.01;
packetsize=$6;
gotime=1;
a=0;
b=0;
}

{
        #refer variables to trace file format
        event=$1
        time=$2
        from=$3
        to=$4
```

```
        pktType=$5
        packet_size=$6
#write packet delivery ratio into .txt file
if(time>gotime) {
   print gotime,(a-b)/a;
   gotime+=time_interval;
 }
        #calculate packets sent and received
if(event=="+")
 {
   a++;
 }
else if(event=="r")
{
  b++;
}


}
END{
;
}
```

.sh File:

a) Throughput Graph

```
set terminal png
#print output into
set output 'Result.png'
#set xrange [0.0:3.0]
set xlabel "Time(in seconds)"
set autoscale xfix
set autoscale
#set yrange [0:80]
set ylabel "Throughput(in Kbps)"
set grid
set style data linespoints
#print output into
```

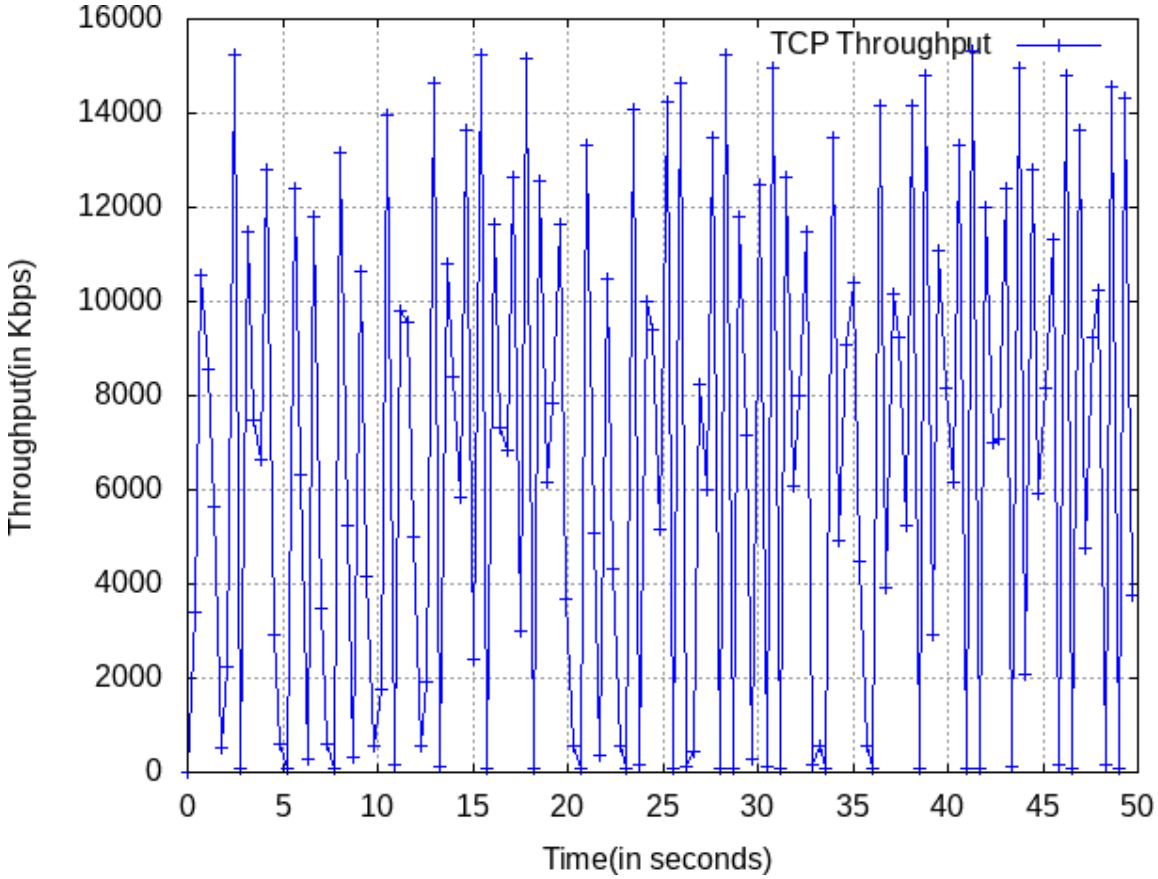
```
set output "reno.png"
#plot the graph
plot "reno" using 1:2 title "TCP Throughput" lt rgb "blue"
```

b) Packet Ratio Graph

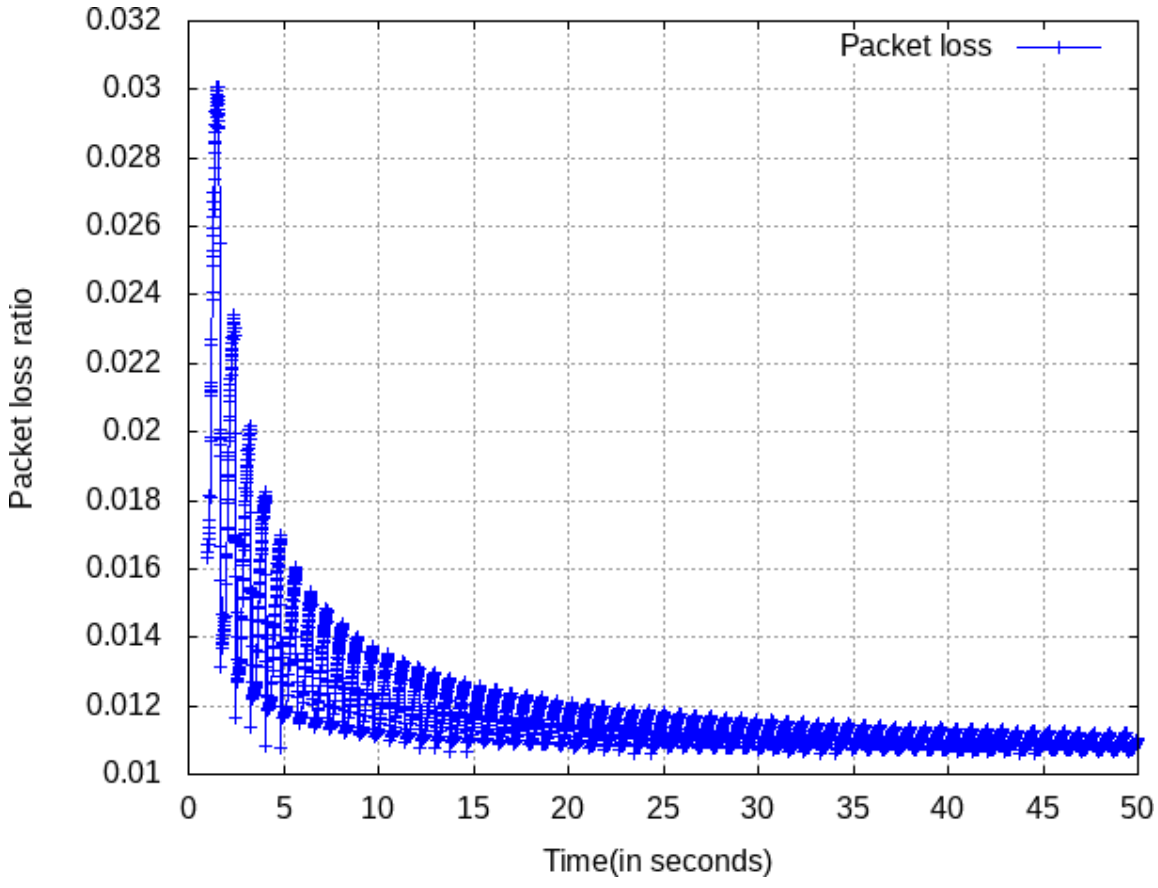
```
set terminal png
#print output into
set output 'result1.png'
#set xrange [0.00:2.00]
set xlabel "Time(in seconds)"
set autoscale
#set yrange [0:1]
set ylabel "Packet loss ratio"
set grid
set style data linespoints
#print output into
set output "renop.png"
#plot the graph
plot "renop" using 1:2 title "Packet loss" lt rgb "blue"
```

Graph:

a) Throughput



b) Packet Delivery Ratio



.cwnd File:

0.10000000000000001 16
0.20000000000000001 10.1
0.30000000000000004 10.1
0.40000000000000002 3
0.5 7.5777
0.59999999999999998 11.7415
0.69999999999999996 15.8797
0.79999999999999993 19.2966
0.89999999999999991 10.6614
0.99999999999999989 5.51828
1.0999999999999999 5.51828
1.2 3.55301

Queue Monitoring

.tcl File:

```
#This program will create a Star Topolgy using for loop in tcl in order to use less statements

#Create a simulator object
set ns  [new Simulator]

#Define different colors and labels for data flows
$ns color 1 green
$ns color 2 black

$ns rtproto DV
#Open the nam trace file
set tracefile [open star.tr w]
$ns trace-all $tracefile

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

#Define a 'finish' procedure
proc finish {} {
	global ns nf
	$ns flush-trace
	close $nf
	exec nam star.nam
	exit 0
	}

#Creating nodes
for {set i 0} {$i<7} {incr i} {
set n($i) [$ns node]
}

#Coloring nodes
for {set i 0} {$i<3} {incr i} {
```

```
$n($i) color red
}
for {set i 3} {$i<6} {incr i} {
$n($i) color blue
}
$n(6) color pink

#Create link between nodes
for {set i 1} {$i<7} {incr i} {
$ns duplex-link $n(0) $n($i) 512Kb 10ms SFQ
}

#Orienting the nodes
$ns duplex-link-op $n(0) $n(1) orient left-up
$ns duplex-link-op $n(0) $n(2) orient right-up
$ns duplex-link-op $n(0) $n(3) orient right
$ns duplex-link-op $n(0) $n(4) orient right-down
$ns duplex-link-op $n(0) $n(5) orient left-down
$ns duplex-link-op $n(0) $n(6) orient left

#TCP_Config
set tcp0 [new Agent/TCP]
$tcp0 set class_ 1
$ns attach-agent $n(1) $tcp0

#Setup a TCP sink connection
set sink0 [new Agent/TCPSink]
$ns attach-agent $n(4) $sink0

#Connect the traffic sources with the traffic sink
$ns connect $tcp0 $sink0

#UDP_Config
set udp0 [new Agent/UDP]
$udp0 set class_ 2
$ns attach-agent $n(2) $udp0
```

```
#Create a Null agent (a traffic sink) and attach it to node n(5)
set null0 [new Agent/Null]
$ns attach-agent $n(5) $null0

#Connect the traffic sources with the traffic sink
$ns connect $udp0 $null0

#CBR Config
set cbr0 [new Application/Traffic/CBR]
$cbr0 set rate_ 256Kb
$cbr0 attach-agent $udp0

#FTP Config
set ftp0 [new Application/FTP]
$ftp0 attach-agent $tcp0


#Scheduling Events
$ns rtmodel-at 0.5 down $n(0) $n(5)
$ns rtmodel-at 0.6 up $n(0) $n(5)

$ns rtmodel-at 0.7 down $n(0) $n(4)
$ns rtmodel-at 1.2 up $n(0) $n(4)

#Schedule events for the FTP agents
$ns at 0.1 "$ftp0 start"
$ns at 1.5 "$ftp0 stop"

#Schedule events for the CBR agents
$ns at 0.2 "$cbr0 start"
$ns at 1.3 "$cbr0 stop"

#Call the finish procedure after 2 seconds of simulation time
$ns at 2.0 "finish"
#Run the simulation
$ns run
```

.nam File:

```
V -t * -v 1.0a5 -a 0
A -t * -n 1 -p 0 -o 0x7fffffff -c 30 -a 1
A -t * -h 1 -m 1073741823 -s 0
c -t * -i 1 -n green
c -t * -i 2 -n black
n -t * -a 4 -s 4 -S UP -v circle -c blue -i blue
n -t * -a 0 -s 0 -S UP -v circle -c red -i red
n -t * -a 5 -s 5 -S UP -v circle -c blue -i blue
n -t * -a 1 -s 1 -S UP -v circle -c red -i red
n -t * -a 6 -s 6 -S UP -v circle -c pink -i pink
n -t * -a 2 -s 2 -S UP -v circle -c red -i red
n -t * -a 3 -s 3 -S UP -v circle -c blue -i blue
```

.tr File:

```
+ 0.00017 0 1 rtProtoDV 7-------- 0 0.1 1.2 -1 0
- 0.00017 0 1 rtProtoDV 7 ------- 0 0.1 1.2 -1 0
+ 0.00017 0 2 rtProtoDV 7-------- 0 0.1 2.2 -1 1
- 0.00017 0 2 rtProtoDV 7 ------- 0 0.1 2.2 -1 1
+ 0.00017 0 3 rtProtoDV 7-------- 0 0.1 3.1 -1 2
- 0.00017 0 3 rtProtoDV 7 ------- 0 0.1 3.1 -1 2
+ 0.00017 0 4 rtProtoDV 7-------- 0 0.1 4.2 -1 3
- 0.00017 0 4 rtProtoDV 7 ------- 0 0.1 4.2 -1 3
+ 0.00017 0 5 rtProtoDV 7-------- 0 0.1 5.2 -1 4
- 0.00017 0 5 rtProtoDV 7 ------- 0 0.1 5.2 -1 4
+ 0.00017 0 6 rtProtoDV 7-------- 0 0.1 6.1 -1 5
- 0.00017 0 6 rtProtoDV 7 ------- 0 0.1 6.1 -1 5
```

.awk File:

a) Queue Monitoring

```
#init variables
BEGIN{
   queued=0;
   dequed=0;
   dropped=0;
   interval=0;
   prev=0;
}
#body
{
```

```
 #refer variables to trace file format
  event = $1
  time =  $2
  source = $3
  destination = $4
  protocol = $5
  packet_size = $6
       #write queued, dequeued packets into .txt file
  if((interval>=0.001)){
  print time, queued, dequed, dropped;
  interval=0;
  }
  #calculate queuing and dequeuing packets
  else{
     if((event == "+"))
     {
        queued++;
     }
     if((event=="d"))
     {
        dropped+=10;
     }
     if((event=="-"))
     {
        dequed++;
     }
     interval+=(time-prev);
  }
  prev=time;
} #body

END{
  ;
}
```

.sh File:

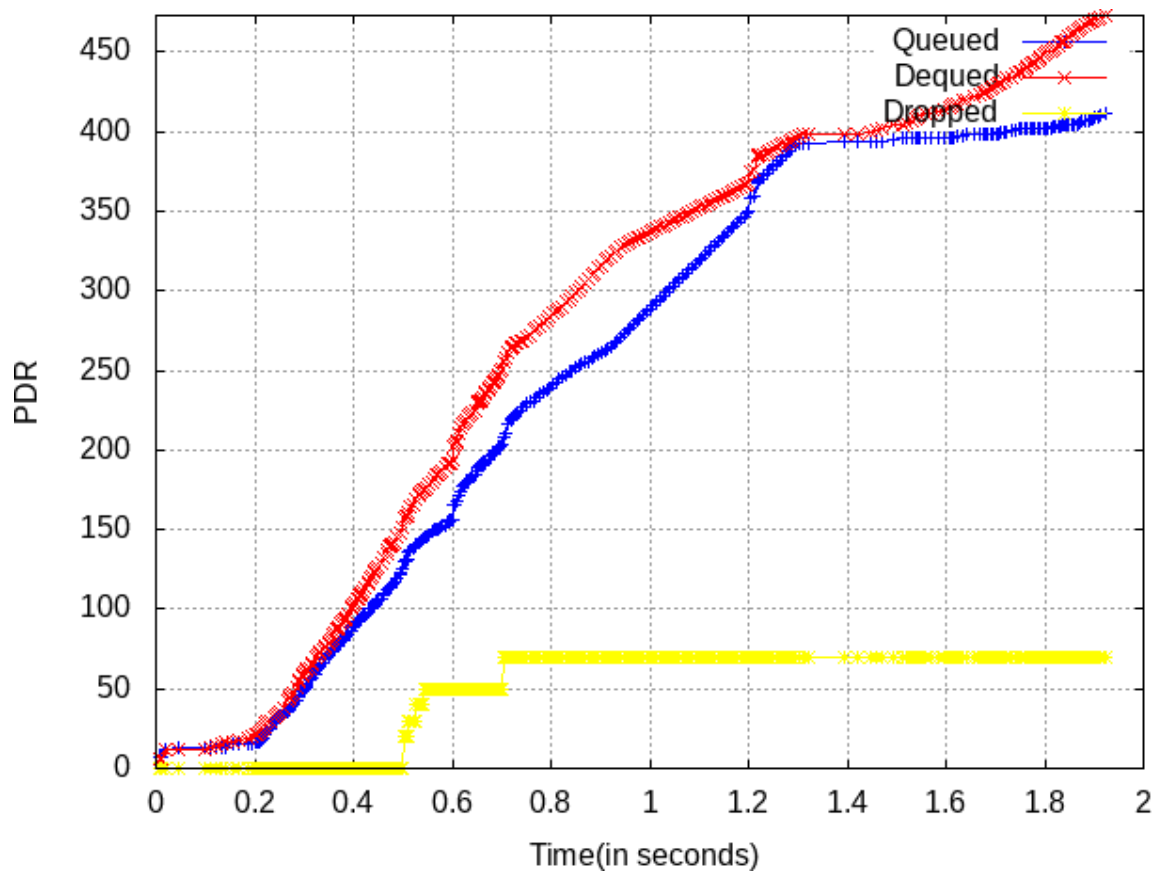
a) Queue Monitoring

```
set terminal png
```

```
#print output into
set output 'queue.png'
#set xrange
set xlabel "Time(in seconds)"
#set yrange
set ylabel "PDR"
set autoscale yfix
set grid
set style data linespoints
#plot the graph
plot "star" using 1:2 title "Queued" lt rgb "blue", "star" using 1:3 title "Dequed"
lt rgb "red", "star" using 1:4 title "Dropped" lt rgb "yellow"
```

Graph:

a) Queue Monitoring



Flow Monitoring

.tcl File:

```
#This program will create a Star Topolgy using for loop in tcl in order to use less statements

#Create a simulator object
set ns  [new Simulator]

#Define different colors and labels for data flows
$ns color 1 green
$ns color 2 black

$ns rtproto DV
#Open the nam trace file
set tracefile [open star.tr w]
$ns trace-all $tracefile

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

#Define a 'finish' procedure
proc finish {} {
	global ns nf
	$ns flush-trace
	close $nf
	exec nam out.nam
	exit 0
	}

#Creating nodes
for {set i 0} {$i<7} {incr i} {
set n($i) [$ns node]
}

#Coloring nodes
for {set i 0} {$i<3} {incr i} {
```

```
$n($i) color red
}
for {set i 3} {$i<6} {incr i} {
$n($i) color blue
}
$n(6) color pink

#Create link between nodes
for {set i 1} {$i<7} {incr i} {
$ns duplex-link $n(0) $n($i) 512Kb 10ms SFQ
}

#Orienting the nodes
$ns duplex-link-op $n(0) $n(1) orient left-up
$ns duplex-link-op $n(0) $n(2) orient right-up
$ns duplex-link-op $n(0) $n(3) orient right
$ns duplex-link-op $n(0) $n(4) orient right-down
$ns duplex-link-op $n(0) $n(5) orient left-down
$ns duplex-link-op $n(0) $n(6) orient left

#TCP_Config
set tcp0 [new Agent/TCP]
$tcp0 set class_ 1
$ns attach-agent $n(1) $tcp0

#Setup a TCP sink connection
set sink0 [new Agent/TCPSink]
$ns attach-agent $n(4) $sink0

#Connect the traffic sources with the traffic sink
$ns connect $tcp0 $sink0

#UDP_Config
set udp0 [new Agent/UDP]
$udp0 set class_ 2
$ns attach-agent $n(2) $udp0
```

```
#Create a Null agent (a traffic sink) and attach it to node n(5)
set null0 [new Agent/Null]
$ns attach-agent $n(5) $null0

#Connect the traffic sources with the traffic sink
$ns connect $udp0 $null0

#CBR Config
set cbr0 [new Application/Traffic/CBR]
$cbr0 set rate_ 256Kb
$cbr0 attach-agent $udp0

#FTP Config
set ftp0 [new Application/FTP]
$ftp0 attach-agent $tcp0


#Scheduling Events
$ns rtmodel-at 0.5 down $n(0) $n(5)
$ns rtmodel-at 0.6 up $n(0) $n(5)

$ns rtmodel-at 0.7 down $n(0) $n(4)
$ns rtmodel-at 1.2 up $n(0) $n(4)

#Schedule events for the FTP agents
$ns at 0.1 "$ftp0 start"
$ns at 1.5 "$ftp0 stop"

#Schedule events for the CBR agents
$ns at 0.2 "$cbr0 start"
$ns at 1.3 "$cbr0 stop"

#Call the finish procedure after 2 seconds of simulation time
$ns at 2.0 "finish"
#Run the simulation
$ns run
```

.nam File:

```
V -t * -v 1.0a5 -a 0
A -t * -n 1 -p 0 -o 0x7fffffff -c 30 -a 1
A -t * -h 1 -m 1073741823 -s 0
c -t * -i 1 -n green
c -t * -i 2 -n black
n -t * -a 4 -s 4 -S UP -v circle -c blue -i blue
n -t * -a 0 -s 0 -S UP -v circle -c red -i red
n -t * -a 5 -s 5 -S UP -v circle -c blue -i blue
n -t * -a 1 -s 1 -S UP -v circle -c red -i red
n -t * -a 6 -s 6 -S UP -v circle -c pink -i pink
n -t * -a 2 -s 2 -S UP -v circle -c red -i red
n -t * -a 3 -s 3 -S UP -v circle -c blue -i blue
```

.tr File:

```
+ 0.00017 0 1 rtProtoDV 7-------- 0 0.1 1.2 -1 0
- 0.00017 0 1 rtProtoDV 7 ------- 0 0.1 1.2 -1 0
+ 0.00017 0 2 rtProtoDV 7-------- 0 0.1 2.2 -1 1
- 0.00017 0 2 rtProtoDV 7 ------- 0 0.1 2.2 -1 1
+ 0.00017 0 3 rtProtoDV 7-------- 0 0.1 3.1 -1 2
- 0.00017 0 3 rtProtoDV 7 ------- 0 0.1 3.1 -1 2
+ 0.00017 0 4 rtProtoDV 7-------- 0 0.1 4.2 -1 3
- 0.00017 0 4 rtProtoDV 7 ------- 0 0.1 4.2 -1 3
+ 0.00017 0 5 rtProtoDV 7-------- 0 0.1 5.2 -1 4
- 0.00017 0 5 rtProtoDV 7 ------- 0 0.1 5.2 -1 4
+ 0.00017 0 6 rtProtoDV 7-------- 0 0.1 6.1 -1 5
- 0.00017 0 6 rtProtoDV 7 ------- 0 0.1 6.1 -1 5
```

.awk File:

a) Flow Monitoring

```
#init variables
BEGIN{
   queued=0;
   interval=0;
   prev=0;
}
#body
{
 #refer variables to trace file format
 event = $1
```

```
   time =  $2
   source = $3
   destination = $4
   protocol = $5
   packet_size = $6
         #write queued, dequeued packets into .txt file
   if((interval>=0.001)){
   print time, queued;
   interval=0;
   queued=0;
   }
   #calculate queuing and dequeuing packets
   else{
      if((event == "+"))
      {
         queued++;
      }
      interval+=(time-prev);
   }
   prev=time;
} #body

END{
   ;
}
```

.sh File:

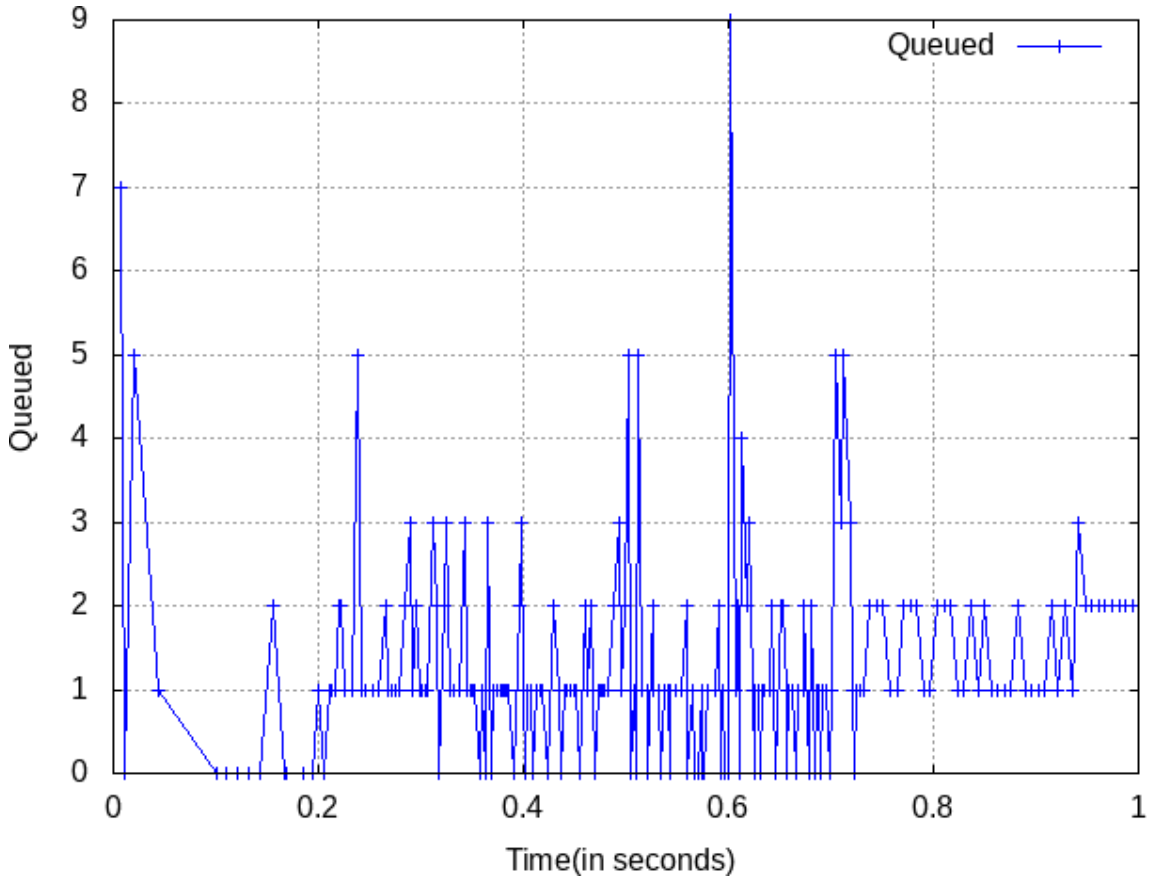
a) Flow Monitoring

```
set terminal png
#print output into
set output 'flow.png'
#set xrange
set xlabel "Time(in seconds)"
#set yrange
set ylabel "Queued"
set autoscale yfix
set xrange [0:1]
set grid
```

```
set style data linespoints
#plot the graph
plot "star" using 1:2 title "Queued" lt rgb "blue"
```

Graph:

a) Flow Monitoring



Ad-hoc Vehicle On Demand

.tcl File:

```
#Define options
set val(chan)  Channel/WirelessChannel;
 set val(prop) Propagation/TwoRayGround;
 set val(netif) Phy/WirelessPhy;
 set val(mac) Mac/802_11;
 set val(ifq) Queue/DropTail/PriQueue;
 set val(ll) LL;
 set val(ant) Antenna/OmniAntenna;
 set val(ifqlen) 50;
 set val(rp) AODV;
 set val(nn) 11;
 set val(x) 500;
 set val(y) 400;
 set val(stop) 3;

 set val(energymodel) EnergyModel;
 set val(initialenergy) 1000;

#Create a simulator object/
 set ns [new Simulator]

#Open the nam trace file
 set tf [open ns_aodv.tr w]
 $ns trace-all $tf

#Open the nam file
 set nf [open ns_aodv.nam w]
 $ns namtrace-all-wireless $nf $val(x) $val(y)

#Define network topography
 set topo [new Topography]
 $topo load_flatgrid $val(x) $val(y)

#Set node API configuration
 create-god $val(nn)
```

```
set chan_1_ [new $val(chan)]

$ns node-config -adhocRouting $val(rp) \
      -llType $val(ll) \
      -macType $val(mac) \
      -ifqType $val(ifq) \
      -ifqLen $val(ifqlen) \
      -antType $val(ant) \
      -propType $val(prop) \
      -phyType $val(netif) \
      -channel $chan_1_ \
      -topoInstance $topo \
      -agentTrace ON \
      -routerTrace ON \
      -macTrace OFF \
      -movementTrace ON \
      -energyModel $val(energymodel) \
      -initialEnergy $val(initialenergy) \
      -rxPower 0.4 \
      -txPower 1.0 \
      -idlePower 0.6 \
      -sleepPower 0.1 \
      -transitionPower 0.4 \
      -transitionTime 0.1


for {set i 0} {$i < $val(nn)} {incr i} {
     #Creating nodes
      set node_($i) [$ns node]
     #Set values on the XY plane
      $node_($i) set X_ [ expr 10+round(rand()*480) ]
      $node_($i) set Y_ [ expr 10+round(rand()*380) ]
      $node_($i) set Z_ 0.0
}

#Set random node motions
```

```
 for {set i 0} {$i < $val(nn)} {incr i} {
       $ns at [ expr 0.2+round(rand()) ] "$node_($i) setdest [ expr
10+round(rand()*480) ] [expr 10+round(rand()*380) ] [expr
60+round(rand()*30) ]"
 }

#Setup a UDP connection
 set udp [new Agent/UDP]
 $ns attach-agent $node_(5) $udp

#Create a Null agent (a traffic sink) and attach it to node_(2)
 set null [new Agent/Null]
 $ns attach-agent $node_(2) $null

# Create a CBR traffic source and attach it to udp
 set cbr [new Application/Traffic/CBR]
 $cbr attach-agent $udp
 $cbr set packetSize_ 512
 $cbr set interval_ 0.1
 $cbr set rate_ 1mb
 $cbr set maxpkts_ 10000

#Connect the traffic sources with the traffic sink
 $ns connect $udp $null

#Schedule events for the CBR agents
 $ns at 0.4 "$cbr start"

#Set initial node positions
 for {set i 0} {$i < $val(nn)} {incr i} {
       $ns initial_node_pos $node_($i) 30
 }

#Set final node co-ordinates
 for {set i 0} {$i < $val(nn)} {incr i} {
       $ns at $val(stop) "$node_($i) reset";
 }
```

```
$ns at $val(stop) "finish"
$ns at 3.1  "puts \"end simulation\"; $ns halt"

#Define a 'finish' procedure
proc finish {} {
        global ns tf nf
        $ns flush-trace
        close $tf
        close $nf
        exec nam ns_aodv.nam &
        exit 0
}
#Display CBR packets value
puts "CBR packet size = [$cbr set packetSize_]"
puts "CBR interval = [$cbr set interval_]"

#Run the simulation
$ns run
```

.nam File:

```
n -t * -s 0 -x 198 -y 275 -Z 0 -z 30 -v circle -c green
n -t * -s 1  -x 80 -y 68 -Z 0 -z 30  -v circle -c green
n -t * -s 2  -x 471 -y 31 -Z 0 -z 30  -v circle -c green
n -t * -s 3  -x 11 -y 138 -Z 0 -z 30  -v circle -c green
n -t * -s 4 -x 188 -y 237 -Z 0 -z 30 -v circle -c green
n -t * -s 5  -x 167 -y 96 -Z 0 -z 30  -v circle -c green
n -t * -s 6  -x 65 -y 100 -Z 0 -z 30  -v circle -c green
n -t * -s 7 -x 188 -y 230 -Z 0 -z 30 -v circle -c green
n -t * -s 8 -x 422 -y 338 -Z 0 -z 30 -v circle -c green
n -t * -s 9  -x 394 -y 254 -Z 0 -z 30  -v circle -c green
n -t * -s 10 -x 197 -y 304 -Z 0 -z 30 -v circle -c green
V -t * -v 1.0a5 -a 0
```

.tr File:

```
M 0.20000 0 (198.00, 275.00, 0.00), (28.00, 32.00), 87.00
M 0.20000 5 (167.00, 96.00, 0.00), (107.00, 313.00), 65.00
M 0.20000 6 (65.00, 100.00, 0.00), (181.00, 323.00), 84.00
M 0.20000 7 (188.00, 230.00, 0.00), (446.00, 346.00), 67.00
```

```
M 0.20000 10 (197.00, 304.00, 0.00), (104.00, 73.00), 83.00
s 0.400000000 _5_ AGT  --- 0 cbr 512 [0 0 0 0] [energy 1000.000000 ei 0.000 es
0.000 et 0.000 er 0.000] ------- [5:0 2:0 32 0] [0] 0 0
r 0.400000000 _5_ RTR ---- 0 cbr 512 [0 0 0 0] [energy 1000.000000 ei 0.000 es
0.000 et 0.000 er 0.000] ------- [5:0 2:0 32 0] [0] 0 0
s 0.400000000 _5_ RTR --- 0 AODV 48 [0 0 0 0] [energy 1000.000000 ei 0.000
es 0.000 et 0.000 er 0.000]--------[5:255 -1:255 30 0] [0x2 1 1 [2 0] [5 4]]
(REQUEST)
N -t 0.400535 -n 6 -e 999.759340
N -t 0.400535 -n 1 -e 999.759340
N -t 0.400535 -n 4 -e 999.759340
N -t 0.400535 -n 7 -e 999.759340
```

.awk File:

a) Throughput

```
#init variables
BEGIN {
      recvdSize = 0
      startTime = 2.0
      stopTime = 0
      sent=0
      received=0
      dropped=0
      forwarded=0
      gotime=0
      time_interval=0.01;
}
  {
    #refer variables to trace file format
    # Trace line format: new
          event = $1
       time = $2
       node_id = $3
       pkt_id = $6
       pkt_size = $8
          level = $4
  }
{
```

```
 # Store start time
 if(time>gotime) {

 print gotime, (recvdSize * received * 8.0)/1000; #packet size * ... gives results
in kbps
 gotime+= time_interval;
 received=0;
 }
 if ((level == "AGT" && event == "s") && pkt_size >= 512) {
        sent++
        if (time < startTime) {
         startTime = time
         }
 }
 if (event == "D" && pkt_size >= 512) {
     dropped++
 }
 if (event == "f" && pkt_size >= 512) {
     forwarded++
 }

 # Update total received packets' size and store packets arrival time
 if (level == "AGT" && event == "r" && pkt_size >= 512) {
    if (time > stopTime) {
         stopTime = time
    }
    received++
    # Rip off the header
    hdr_size = pkt_size % 512
    pkt_size -= hdr_size
    # Store received packet's size
    recvdSize += pkt_size
 }
}
#Print throughput
 END {
```

```
    printf("Average Throughput[kbps] = %.2f\t\t
StartTime=%.2f\tStopTime=%.2f\n",(recvdSize/(stopTime-
startTime))*(8/1000),startTime,stopTime)
    print("Sent - ",sent)
    print("Received - ",received)
    print("Dropped - ",dropped)
    print("Forwarded",forwarded)
}
```

b) Packet Ratio

```
#init variables
BEGIN {
    sends=0;
    recvs=0;
    routing_packets=0;
    droppedPackets=0;
    highest_packet_id =0;
    sum=0;
    time_interval=0.01;
    recvnum=0;
    gotime=1;
   }
 {
   #refer variables to trace file format
 time = $2;
 packet_id = $6;
 event =$1;
 #write packet delivery ratio into .txt file
 if(time>gotime) {
  print gotime,(sends-recvs)/sends;
  gotime+=time_interval;
 }
 # CALCULATE PACKET DELIVERY FRACTION
 if (( $1 == "s") &&  ( $7 == "cbr" ) && ( $4=="AGT" )) {  sends++; }
 if (( $1 == "r") && ( $7 == "cbr" ) && ( $4=="AGT" ))  {  recvs++; }
 # CALCULATE DELAY
 if ( start_time[packet_id] == 0 )  start_time[packet_id] = time;
```

```
 if (( $1 == "r") && ( $7 == "cbr" ) && ( $4=="AGT" )) { end_time[packet_id] =
time;  }
     else {  end_time[packet_id] = -1;  }
 # CALCULATE TOTAL AODV OVERHEAD
 if (($1 == "s" || $1 == "f" || $1="r") && $4 == "RTR" && ($7 =="AODV" ||$7
=="AOMDV")) routing_packets++;
 # DROPPED AODV PACKETS
 if (event == "D") droppedPackets++;
 }
 END {
 for ( i in end_time )
 {
 start = start_time[i];
 end = end_time[i];
 packet_duration = end - start;
 if ( packet_duration > 0 )
 {   sum += packet_duration;
     recvnum++;
 }
 }
 #Print packet delivery ratio
   delay=sum/recvnum;
   NRL = routing_packets/recvs; #normalized routing load
   PDF = (recvs/sends)*100; #packet delivery ratio[fraction]
   printf("Send Packets = %.2f\n",sends);
   printf("Received Packets = %.2f\n",recvs);
   printf("Routing Packets = %.2f\n",routing_packets++);
   printf("Packet Delivery Function = %.2f\n",PDF);
   printf("Normalised Routing Load = %.2f\n",NRL);
   printf("Average end to end delay(ms)= %.2f\n",delay*1000);
   print("No. of dropped packets = ",droppedPackets);

}
```

.sh File:

a) Throughput Graph

```
set terminal png
#print output into
```

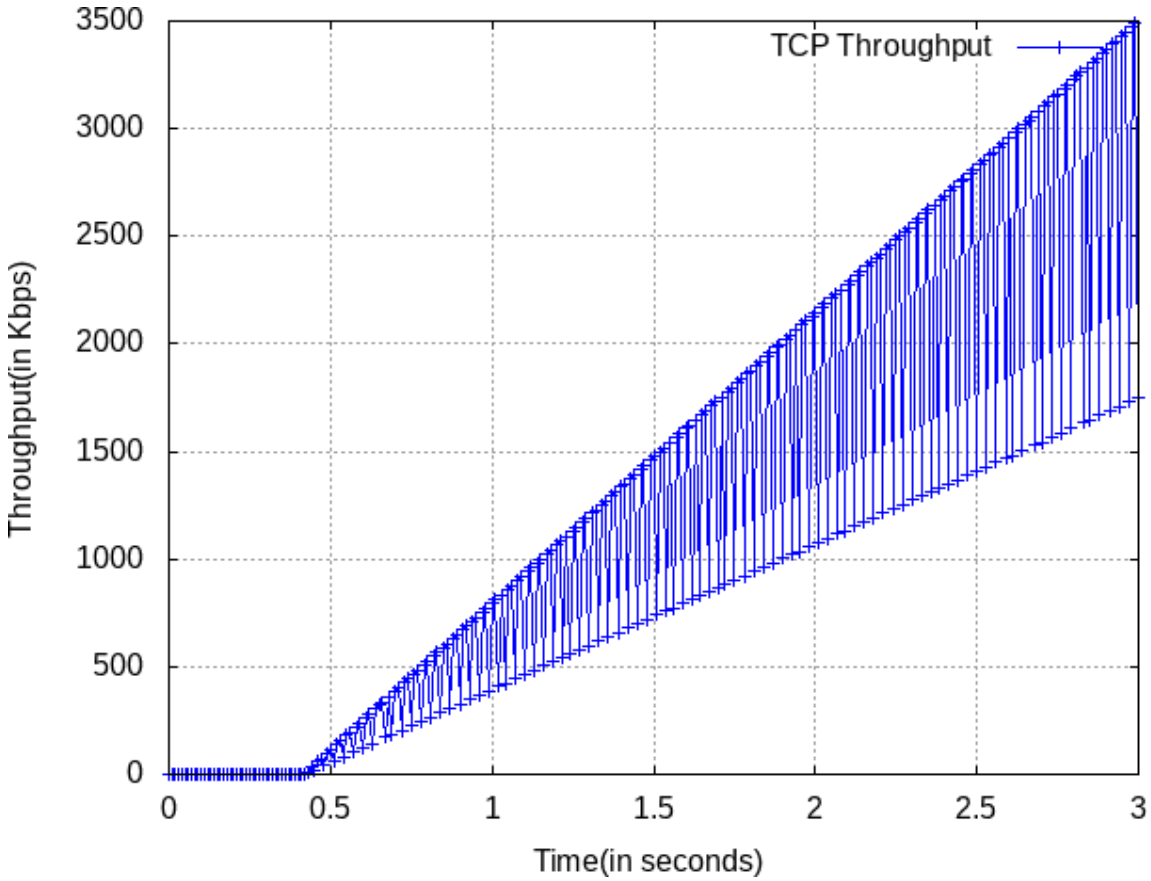
```
set output 'Result.png'
#set xrange [0.0:3.0]
set xlabel "Time(in seconds)"
set autoscale xfix
set autoscale
#set yrange [0:80]
set ylabel "Throughput(in Kbps)"
set grid
set style data linespoints
#print output into
set output "aodv.png"
#plot the graph
plot "aodv" using 1:2 title "TCP Throughput" lt rgb "blue"
```

b) Packet Ratio Graph

```
set terminal png
#print output into
set output 'result1.png'
#set xrange [0.00:2.00]
set xlabel "Time(in seconds)"
set autoscale
#set yrange [0:1]
set ylabel "Packet loss ratio"
set grid
set style data linespoints
#print output into
set output "aodvp.png"
#plot the graph
plot "aodvp" using 1:2 title "Packet loss" lt rgb "blue"
```

Graph:

a) Throughput



b) Packet Delivery Ratio