PART A

1.Write a program for error detecting code using 16 bits CRC-CCITT

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

# define MAX 30

void crc(char *data, char *gen, char *rem)
{
   int i, j ;
   int dwordSize = strlen(data)-(strlen(gen)-1) ; // dataword size
   char out[MAX]; // xored val after each step

   strcpy(out, data);

   /* Perform XOR on the msg */
   for(i=0; i<dwordSize ; i++)
   {
      if(out[i]=='0') continue ;
      for(int j = 0 ; j<strlen(gen) ; j++){
         out[i+j] = (gen[j] == out[i+j] ? '0' : '1') ;
      }
   }

   for(i=0;i<strlen(gen)-1;i++)
   {
        rem[i]=out[dwordSize+i];
   }
}

int main()
{
   int i, j;
   char dword[MAX]; // dataword k bits
   char augWord[MAX]; // augmented dataword
   char cword[MAX]; // codeword n bits
   char rem[MAX]; // remainder from crc
   char recv[MAX]; // received message
   char gen[MAX] = "10001000000100001"; // crc-16 (17 bits)

   printf("\nCRC-16 Generator : x^16 + x^12 + x^5 + 1 ");
   printf("\nBinary Form      : %s", gen);

   printf("\n\nEnter Dataword   : ");
   scanf("%s", dword);

   strcpy(augWord, dword);
   for(i=0; i<strlen(gen)-1; i++)
```

```
    {
        strcat(augWord, "0");
    }
    printf("\nAugmented dataword is   : %s",augWord);

    crc(augWord, gen, rem);

    strcpy(cword, dword);
    strcat(cword, rem);
    printf("\n\nFinal data transmitted  : %s", cword);

    printf("\n\nEnter the data received : ");
    scanf("%s", recv);
    if(strlen(recv) < strlen(cword))
    {
        printf("\n Invalid input \n");
        exit(0);
    }

    crc(recv, gen, rem);

    printf("\nSyndrome = %s ", rem);

    for(i=0; i<strlen(rem); i++)
    {
        if( rem[i] == '1')
        {
            printf("\nError occured !!! Corrupted data received.\n");
            exit(0);
        }
    }
    printf("\nNo Error. Data received successfully.\n");
}
```

2.For the given network graph, write a program to implement Link state routing algorithm to build a routing table for the given node.

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

#define INFINITY 999
#define MAX 100

int cost[MAX][MAX];
int distance[MAX];
int visited[MAX] = {0};
int parent[MAX];
int source;
int n;
```

```
void initialize()
{
   int i;
   visited[source] = 1;
   parent[source] = source;

   for(i=0; i<n; i++)
   {
      distance[i] = cost[source][i];
      if( cost[source][i] != INFINITY )
      {
         parent[i] = source;
      }
   }
}

int GetMin()
{
   int minIdx = -1;
   int minDist = INFINITY;

   int i;
   for(i=0; i<n; i++)
   {
      if( !visited[i] && minDist >= distance[i] )
      {
            minIdx = i;
            minDist = distance[i];
      }
   }
   return minIdx;
}

void updateTable(int node)
{
   int i;
   for(i=0; i<n; i++)
   {
      if( cost[node][i] != INFINITY && distance[i] > distance[node]+cost[node][i] )
      {
         distance[i] = distance[node] + cost[node][i];
         parent[i] = node;
      }
   }
}

void display()
{
   int i;
```

```
    int node;

    printf("\nNode \t Distance from source \t Path \n");
    for(i=0; i<n; i++)
    {
        printf("%d \t\t %d \t\t", i, distance[i]);

        node = i;
        printf("%d", node);
        while( node != source)
        {
            printf(" <- %d", parent[node]);
            node = parent[node];
        }
        printf("\n");
    }
}

int main()
{
    int i, j, node;

    printf("Enter the number of nodes: ");
    scanf("%d", &n);

    printf("Enter the source node    : ");
    scanf("%d", &source);

    printf("\nEnter the cost matrix: \n");
    for(i=0; i<n; i++)
    {
        for(j=0; j<n; j++)
        {
            scanf("%d", &cost[i][j]);
            if(cost[i][j]==0 && i!=j)
            {
                    cost[i][j]=INFINITY;
            }
        }
    }

    initialize();

    for(i=0; i<n-1; i++)
    {
        node = GetMin();
        visited[node] = 1;
        updateTable(node);
    }
    display();
    return 0;
}
```

3.Write a program to divide the message into variable length frames and sort them and display the message at the receiving side

```
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
#include<time.h>

#define MAX 100

typedef struct{
   int id;
   char data[MAX];
}frame;

// Fisher yates algorithm to shuffle the frame
void shuffleFrames(frame f[MAX], int n)
{
   int i;
   for(i=n-1; i>=0; i--)
   {
      int j = rand()%(i+1);

      frame temp = f[j];
      f[j] = f[i];
      f[i] = temp;
   }
}

void sortFrames(frame f[MAX], int n)
{
   int i, j;
   for(i=1; i<n; i++)
   {
      frame t = f[i];
      j = i-1;
      while(j>=0 && f[j].id > t.id)
      {
         f[j+1] = f[j];
         j = j-1;
      }
      f[j+1] = t;
   }
}

void showFrames(frame f[MAX] , int n ){
```

```
        int i;
    printf("\nframe_id \t frame_data \n");
    printf("----------------------------\n");
    for(i=0 ; i < n; i++)
    {
        printf("%d \t\t %s \n", f[i].id, f[i].data);
    }
}

int main()
{
    frame f[MAX];
    int n = 0;     // no of frames
    int fsize;     // size of frame

    char msg[MAX];
    int m = 0; // message iterator
    int i, j;

    printf("Enter a message : ");
    fgets(msg , MAX, stdin);
    msg[strlen(msg)-1] = '\0'; // to remove '\n' from string

    srand(time(NULL));
    // Divide the message into frames
    for(i=0 ; m < strlen(msg) ; i++)
    {
        f[i].id = i;
        fsize = rand()%5+1; // variable Frame size in range [1,5]
        n++;                // count number of frames

        strncpy(f[i].data , msg+m , fsize) ;
        m = m+fsize ;
    }

    shuffleFrames(f, n);
    printf("\nShuffled frames:");
    showFrames(f,n);

    sortFrames(f, n);
    printf("\nSorted frames:");
    showFrames(f,n);

     printf("\nfinal message : ");
    for(i=0; i< n; i++)
    {
        printf("%s", f[i].data);
    }
    printf("\n");
}
```

4.Using TCP/IP sockets, write a client – server program, the client sends the file name and the server sends back the requested text file if present.

Server.c

```
#include <stdio.h>
#include <stdlib.h>
#include <fcntl.h>
#include <arpa/inet.h>
#include <unistd.h>

int main()
{
   int sersock, sock, fd, n;
   char buffer[1024], fname[50];

   sersock = socket(AF_INET, SOCK_STREAM, 0);//Domain=AF_INET
(Ipv4),type=SOCK_STREAM(TCP),protocol=0(IP)
   struct sockaddr_in addr = { AF_INET, htons(1234), inet_addr("127.0.0.1") };

   // Forcefully connecting to same port everytime
   int reuse = 1;
         setsockopt(sersock, SOL_SOCKET, SO_REUSEADDR, (char *)&reuse, sizeof(reuse));

   /* attaching socket to port */
   bind(sersock, (struct sockaddr *) &addr, sizeof(addr));
   printf("\nServer is Online");

   listen(sersock, 5); // listen(int sockfd, int backlog)
   sock = accept(sersock, NULL, NULL);

   /*  receive the filename from client */
   recv(sock, fname, 50, 0);
   printf("\nRequesting for file: %s\n", fname);

   fd = open(fname, O_RDONLY); // open file in read-only mode
   if (fd < 0)
   {
      send(sock, "\nFile not found\n", 15, 0); // strlen(\nFile not found)=15
      exit(0);
   }

   while ((n = read(fd, buffer, sizeof(buffer))) > 0)
   {
      send(sock, buffer, n, 0);
   }
   printf("\nFile content sent\n");

   close(fd);
   return 0;
```

```
}
```

client.c

```
#include <stdio.h>
#include <stdlib.h>
#include <fcntl.h>
#include <arpa/inet.h>
#include <unistd.h>

int main()
{
   int sock, n;
   char buffer[1024], fname[50];

   sock = socket(AF_INET, SOCK_STREAM, 0);//Domain=AF_INET
(Ipv4),type=SOCK_STREAM(TCP),protocol=0(IP)
   struct sockaddr_in addr = { AF_INET, htons(1234), inet_addr("127.0.0.1") };

   /*  keep trying to esatablish connection with server */
   while(connect(sock, (struct sockaddr *) &addr, sizeof(addr)) < 0) ;
   printf("\nClient is connected to Server");

   /* send the filename to the server */
   printf("\nEnter file name: ");
   scanf("%s", fname);
   send(sock, fname, sizeof(fname), 0);

   printf("\nRecieved file data\n");
   printf("-----------------------------------------------------\n");

   /*  keep printing any data received from the server */
   while ((n = recv(sock, buffer, sizeof(buffer), 0)) > 0)
   {
      buffer[n] = '\0' ;
      printf("%s", buffer);
   }

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

5. Using FIFOs as IPC channels, write a client – server program, the client sends the file name and the server sends back the requested text file if present

server.c

```
#include <stdio.h>
#include <unistd.h>
#include <fcntl.h> // used for file handling
#include <sys/stat.h> // used for mkfifo function
#include <sys/types.h> // mkfifo() has dependency on both types.h and stat.h

int main()
{
   char fname[50], buffer[1025];
   int req, res, n, file;

   mkfifo("req.fifo", 0777);
   mkfifo("res.fifo", 0777);

   printf("Waiting for request...\n");
   req = open("req.fifo", O_RDONLY);
   res = open("res.fifo", O_WRONLY);

   read(req, fname, sizeof(fname));
   printf("Received request for %s\n", fname);

   file = open(fname, O_RDONLY);
   if (file < 0)
   {
      write(res, "File not found\n", 15);
   }
   else
   {
      while((n = read(file, buffer, sizeof(buffer))) > 0)
      {
         write(res, buffer, n);
      }
   }

   close(req);
   close(res);

   unlink("req.fifo");
   unlink("res.fifo");

   return 0;
}
```

client.c

```
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>
#include <fcntl.h>
#include <sys/stat.h>
#include <sys/types.h>

int main()
{
   char fname[50], buffer[1025];
   int req, res, n;

   req = open("req.fifo", O_WRONLY);
   res = open("res.fifo", O_RDONLY);

   if(req < 0 || res < 0)
   {
      printf("Please Start the server first\n");
      exit(-1);
   }

   printf("Enter filename to request : ");
   scanf("%s", fname);

   // write file name to request file
   write(req, fname, sizeof(fname));

   printf("Received response\n");
   printf("----------------------------------------\n");
   while((n = read(res, buffer, sizeof(buffer)))>0)
   {
      buffer[n]= '\0' ;
      printf("%s", buffer);
   }
   printf("----------------------------------------\n");

   close(req);
   close(res);
   return 0;
}
```

6. Using UDP, write a client – server program, to exchange messages between client and the server.

Server.c

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

#define MAX 1024

int main()
{
	char buffer[MAX], msg[MAX];
	struct sockaddr_in servaddr, cliaddr;

	int sock = socket(AF_INET, SOCK_DGRAM, 0);

	// Forcefully connecting to same port everytime
	int reuse = 1;
	setsockopt(sock, SOL_SOCKET, SO_REUSEADDR, (char *)&reuse, sizeof(reuse));

	// Initialize servaddr and cliaddr with zero
	memset(&servaddr, 0, sizeof(servaddr));
	memset(&cliaddr, 0, sizeof(cliaddr));
	int len = sizeof(cliaddr);

	// Filling server information
	servaddr.sin_family = AF_INET; // IPv4
	servaddr.sin_addr.s_addr = INADDR_ANY; //INADDR_ANY listen on all available
interfaces
	servaddr.sin_port = htons(1234);

	// Bind the socket with the server address
	bind(sock, (const struct sockaddr *)&servaddr, sizeof(servaddr));
	printf("Waiting for message from client...\n");

	while(1)
	{
	  int n = recvfrom(sock, (char *)buffer, sizeof(buffer), 0, ( struct sockaddr *) &cliaddr,
&len);
	  buffer[n] = '\0';
	  printf("Client : %s", buffer);
	  printf("Server : ");
	  fgets(msg, MAX, stdin);
```

```
        sendto(sock, (const char *)msg, strlen(msg), 0, (const struct sockaddr *) &cliaddr, len);
    }
    return 0; }
```

client.c

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

#define MAX 1024

int main()
{
    char buffer[MAX], msg[MAX];
    struct sockaddr_in servaddr;

    // Creating socket file descriptor
    int sock = socket(AF_INET, SOCK_DGRAM, 0);

    memset(&servaddr, 0, sizeof(servaddr));
    // Filling server information
    servaddr.sin_family = AF_INET;
    servaddr.sin_port = htons(1234); // htons(port)
    servaddr.sin_addr.s_addr = INADDR_ANY;

    while(1)
    {
        printf("Client : ");
        fgets(msg, MAX, stdin);
        sendto(sock, (const char *)msg, strlen(msg), 0, (const struct sockaddr *) &servaddr,
sizeof(servaddr));

        int n = recvfrom(sock, (char *)buffer, sizeof(buffer), 0, NULL, NULL);
        buffer[n] = '\0';
        printf("Server : %s", buffer);
    }
    return 0;
}
```

7. Write a socket program to demonstrate IP multicasting which provides the capability for an application to send a single IP datagram that a group of hosts in a network can receive.

Server.c

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

int main ()
{
   int sock;
   char msg[1024];
   struct sockaddr_in groupaddr;
   struct in_addr localInterface;

   sock = socket(AF_INET, SOCK_DGRAM, 0);

   memset(&groupaddr, 0, sizeof(groupaddr));
   groupaddr.sin_family = AF_INET;
   groupaddr.sin_addr.s_addr = inet_addr("226.1.1.1");
   groupaddr.sin_port = htons(1234);

   localInterface.s_addr = inet_addr("127.0.0.1");  // or system ip address
   setsockopt(sock, IPPROTO_IP, IP_MULTICAST_IF, (char *)&localInterface,
sizeof(localInterface));

   printf("Enter message : ");
   fgets(msg, 1024, stdin);
   msg[strlen(msg)-1] = '\0'; // to remove '\n' from string
   sendto(sock, msg, sizeof(msg), 0, (struct sockaddr*)&groupaddr, sizeof(groupaddr));
   printf("Message Sent.\n");

   return 0;
}
```

client.c

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

int main()
{
	int sock, reuse = 1;
	char msg[1024];
	struct sockaddr_in addr;
	struct ip_mreq group;

	sock = socket(AF_INET, SOCK_DGRAM, 0);

	setsockopt(sock, SOL_SOCKET, SO_REUSEADDR, (char *)&reuse, sizeof(reuse));

	memset(&addr, 0, sizeof(addr));
	addr.sin_family = AF_INET;
	addr.sin_port = htons(1234);
	addr.sin_addr.s_addr = INADDR_ANY; // to listen on all available interfaces.

	bind(sock, (struct sockaddr*)&addr, sizeof(addr));

	group.imr_multiaddr.s_addr = inet_addr("226.1.1.1");
	group.imr_interface.s_addr = inet_addr("127.0.0.1");
	setsockopt(sock, IPPROTO_IP, IP_ADD_MEMBERSHIP, (char *)&group, sizeof(group));
	printf("Waiting for message from server.....");

	// recvfrom(sock, msg, sizeof(msg), 0, NULL, NULL);
	read(sock, msg, sizeof(msg));
	printf("\nThe message from multicast server is : %s \n", msg);

	close(sock);
	return 0;
}
```

8.Write a program to implement sliding window protocol between two hosts.

Server.c

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

#define MAX 20

int main()
{
	int sersock = socket(AF_INET, SOCK_STREAM, 0);
	struct sockaddr_in addr = { AF_INET, htons(1234), inet_addr("127.0.0.1") };

	// Forcefully connecting to same port everytime
	int reuse = 1;
	setsockopt(sersock, SOL_SOCKET, SO_REUSEADDR, (char *)&reuse, sizeof(reuse));

	bind(sersock, (struct sockaddr *) &addr, sizeof(addr));
	printf("\nServer is Online\n");

	listen(sersock, 5);
	int sock = accept(sersock, NULL, NULL);

	char frame[MAX];
	char res[MAX]; // to store all bytes that are recieved successfully
	int ack;
	int k=0;		// iterator for res
	srand(time(NULL));

	while(1)
	{
		int recvsize = 5;
		memset(frame, 0, MAX); // re-initialise frame buffer with 0

		recv(sock, frame, recvsize, 0);  // recv(socket, buffer, length, flag)
		if(strcmp(frame, "Exit") == 0) break; // at end exit frame is recieved

		if(strlen(frame) < recvsize)
		{
			recvsize = strlen(frame);
		}
```

```
                int err_idx = rand()%8; // probability of byte to get corrupted = 50%
                if(err_idx < recvsize)
                {
                        recvsize = err_idx;
                        frame[err_idx]='x';
                        printf("\nError occured at = %d", err_idx+1);
                }

                int j;
                for(j=0; j<recvsize ; j++)
                {
                        res[k++] = frame[j];
                }
                printf("\nPacket received  = %s", frame);
                printf("\nReceiving window: ");
                printf("\n start seqno = %d", k-recvsize);
                printf("\n end seqno   = %d", k-1);

                ack = k ;
                printf("\nSending ack = %d", ack);
                send(sock, &ack, sizeof(ack), 0) ;
                printf("\n");
        }

        res[k] = '\0';
        printf("\n\nFinal string recieved at Destination = ");
        fputs(res, stdout);

        printf("\n\n");
        close(sock); close(sersock);
}
```

client.c

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

#define MAX 20

int main()
{
   int sock = socket(AF_INET, SOCK_STREAM, 0);
   struct sockaddr_in addr = { AF_INET, htons(1234), inet_addr("127.0.0.1") };

   /*  keep trying to establish connection with server */
   while(connect(sock, (struct sockaddr *) &addr, sizeof(addr))) ;
```

```
    printf("\nClient is connected to Server\n");

    char msg[MAX];
    printf("\nEnter message to send : ");
    fgets(msg, MAX, stdin);
    msg[strlen(msg)-1] = '\0';

    char frame[MAX];
    int i = 0;
    int ack;

    while(i<strlen(msg))
    {
        int sendsize = 5;
        memset(frame, 0, MAX); // re-initialise frame buffer with 0

        // strncpy(destination , source , length)
        strncpy(frame, msg+i, sendsize); //copy msg to frame
        if( sendsize > strlen(frame) )
        {
            sendsize = strlen(frame);
        }
        printf("\n\nSending packet = %s", frame);
        printf("\nSending window: ");
        printf("\n start seqno = %d", i);
        printf("\n end seqno   = %d", i+sendsize-1);

        send(sock, frame, sendsize, 0);
        recv(sock, &ack, sizeof(ack), 0);
        printf("\nreceived ack no = %d ",ack);

        i = ack; // next data seq no = incoming ack no
    }
    send(sock, "Exit", strlen("Exit"), 0);
    close(sock); printf("\n\n");
}
```

PART B

1. Simulate a three nodes point – to – point network with duplex links between them.Set the queue size and vary the bandwidth and find the number of packets dropped.

```
# create a new simulator
set ns [new Simulator]

# open trace and NAM trace file in write mode
set tf [open out.tr w]
$ns trace-all $tf
set nf [open out.nam w]
$ns namtrace-all $nf

##### Decide a topology #########
#
#   [udp][cbr]
#     [0]------
#              |      [null]
#             [2]------[3]
#              |
#     [1]------
#  [udp][cbr]
#
###################################

# create 4 nodes
set n0 [$ns node]
set n1 [$ns node]
set n2 [$ns node]
set n3 [$ns node]

# create duplex links between nodes
$ns duplex-link $n0 $n2 10Mb 300ms DropTail
$ns duplex-link $n1 $n2 10Mb 300ms DropTail
$ns duplex-link $n2 $n3 1Mb 300ms DropTail

# set up queue size
$ns queue-limit $n0 $n2 10
$ns queue-limit $n1 $n2 10
$ns queue-limit $n2 $n3 10

# setup udp connection for transport layer
set udp0 [new Agent/UDP]
set udp1 [new Agent/UDP]
set null3 [new Agent/Null]

$ns attach-agent $n0 $udp0
$ns attach-agent $n1 $udp1
$ns attach-agent $n3 $null3
```

```
# setup cbr(constant bit rate) over udp for application layer
set cbr0 [new Application/Traffic/CBR]
set cbr1 [new Application/Traffic/CBR]
$cbr0 attach-agent $udp0
$cbr1 attach-agent $udp1

# connect source to destination
$ns connect $udp0 $null3
$ns connect $udp1 $null3

# set bandwidth (vary values for different output)
$cbr0 set packetSize_ 500Mb
$cbr1 set packetSize_ 500Mb
$cbr0 set interval_ 0.005
$cbr1 set interval_ 0.005

# define a finish procedure
proc finish {} {
        global ns nf tf
        $ns flush-trace
        exec nam out.nam &
        close $tf
        close $nf

        set count 0
        set tf [open out.tr r]
        while {[gets $tf line] != -1} {
                # d is event in the trace file which denotes dropped packets
                if { [string match "d*" $line] } {
                        set count [expr $count + 1]
                }
        }
        puts "Number of packets dropped: $count"
        exit 0
}

# schedule events
$ns at 0.01 "$cbr0 start"
$ns at 0.01 "$cbr1 start"
$ns at 5.0 "finish"
$ns run

############## output ##############

# Number of packets dropped: 700
```

---

2. Simulate the different types of Internet traffic such as FTP and TELNET over a network and analyze the throughput.

```
# create a new simulator
set ns [new Simulator]

# open trace and NAM trace file in write mode
set tf [open out.tr w]
$ns trace-all $tf
set nf [open out.nam w]
$ns namtrace-all $nf

######## Decide a topology #######
#
#    [ftp]
#    [tcp]
#     [0]------
#              |         [sink0]
#             [2]------[3]
#              |         [sink1]
#      [1]------
#     [tcp]
#   [telnet]
#
###################################

# create 4 nodes
set n0 [$ns node]
set n1 [$ns node]
set n2 [$ns node]
set n3 [$ns node]

# create duplex links between nodes
$ns duplex-link $n0 $n2 2Mb 1ms DropTail
$ns duplex-link $n1 $n2 2Mb 1ms DropTail
$ns duplex-link $n2 $n3 2Mb 1ms DropTail

# set n0 and n1 as tcp source
set tcp0 [new Agent/TCP]
set tcp1 [new Agent/TCP]
$ns attach-agent $n0 $tcp0
$ns attach-agent $n1 $tcp1

# set n3 as tcp destination for n0 and n1
set TCPS0 [new Agent/TCPSink]
set TCPS1 [new Agent/TCPSink]
$ns attach-agent $n3 $TCPS0
$ns attach-agent $n3 $TCPS1

# set ftp over tcp0
set ftp0 [new Application/FTP]
$ftp0 attach-agent $tcp0
```

```
#set telnet over tcp1
set tel1 [new Application/Telnet]
$tel1 attach-agent $tcp1

$tel1 set packetSize_ 500Mb
$tel1 set interval_ 0.001

# connect source to destination
$ns connect $tcp0 $TCPS0
$ns connect $tcp1 $TCPS1

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

        # because time difference between start and finish is 2
        set time 2
        set fCount 0
        set tCount 0
        set tf [open out.tr r]
        while {[gets $tf line] != -1} {
                if { [string match "*tcp*0.0*3.0*" $line] } {
                        set fCount [expr $fCount + 1]
                }
                if { [string match "*tcp*1.0*3.1*" $line] } {
                        set tCount [expr $tCount + 1]
                }
        }
        puts "Throughput of FTP: [expr $fCount/$time]"
        puts "Throughput of TELNET: [expr $tCount/$time]"
        exit 0
}

# schedule events
$ns at 0.01 "$ftp0 start"
$ns at 0.01 "$tel1 start"
$ns at 2.01 "finish"
$ns run

################ output #################

# No of FTP packets: 767
# No of TELNET packets: 750
```

---

3. Simulate an Ethernet LAN using n nodes (6-10), change error rate and data rate and compare the throughput.

```
# Declare a new Simulator
set ns [new Simulator]

# Open nam and trace file in write mode
set tf [open out.tr w]
set nf [open out.nam w]
$ns trace-all $tf
$ns namtrace-all $nf

# Take value of error rate and data rate from std input
puts "Enter error rate (<1) : "
gets stdin erate

puts "Enter data rate (in Mbps) : "
gets stdin drate
############### Select a topology ######################
#           [udp1]              duplex-link
#.    [n0]    [n1]  [n2]    [n3]-------------
#       |     |     |     |             |
#       |     |     |     |             |
#   --------------------------------lan7     |
#                                     |
#   --------------------------------lan8     |
#       |      |       |               |
#       |      |       |               |
#       [n4]    [n5]     [n6]----------------
#             [null5]
#########################################################
# Create 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]
set n6 [$ns node]

# set label and color (OPTIONAL)
$n1 label "udp/source"
$n5 label "udp/null"
$n0 color "blue"
$n1 color "blue"
$n2 color "blue"
$n3 color "blue"
$n4 color "red"
$n5 color "red"
$n6 color "red"

# Create two lans
```

```
$ns make-lan "$n0 $n1 $n2 $n3" 10Mb 10ms LL Queue/DropTail Mac/802_3
$ns make-lan "$n4 $n5 $n6" 10Mb 10ms LL Queue/DropTail Mac/802_3

# Setup Links
$ns duplex-link $n3 $n6 10Mb 10ms DropTail

# Declare the transport layer protocols
set udp1 [new Agent/UDP]
set null5 [new Agent/Null]
$ns attach-agent $n1 $udp1
$ns attach-agent $n5 $null5

# Declare the application layer protocol
set cbr1 [new Application/Traffic/CBR]
$cbr1 attach-agent $udp1

# Connect the source and destination
$ns connect $udp1 $null5

# Create error model
set err [new ErrorModel]
$ns lossmodel $err $n3 $n6
$err set rate_ $erate

# Define the data rate
$cbr1 set packetSize_ $drate.Mb
$cbr1 set interval_ 0.001

# Define procedure
proc finish { } {
        global ns nf tf
        $ns flush-trace
        exec nam out.nam &
        close $nf
        close $tf

        set count 0
        set tr [open out.tr r]
        while {[gets $tr line] != -1} {
                # 8 denotes LAN at destination side and 5 denotes destination node
                if {[string match "* 8 5 *" $line]} {
                        set count [expr $count+1]
                }
        }
        set thr [expr $count/5]
        puts "Throughput : $thr"
        exit 0
}

$ns at 0.1 "$cbr1 start"
$ns at 5.1 "finish"
$ns run
```

4. Simulate an Ethernet LAN using n nodes and set multiple traffic nodes and determine the collision across different nodes.

```
# Declare a new Simulator
set ns [new Simulator]

# Open the trace and nam file in write mode
set tf [open out.tr w]
set nf [open out.nam w]
$ns trace-all $tf
$ns namtrace-all $nf

# Decide the topology: [tcp(0->2)], [udp(2->1)], [tcp(1->3)]
#
#      [tcp0]   [tcp1][null1]
#       [n0]       [n1]
#        |          |
#        |          |
#   ------------------------------ lan4
#            |           |
#            |           |
#           [n3]         [n2]
#         [sink3]     [udp2][sink2]

# Create 4 nodes
set n0 [$ns node]
set n1 [$ns node]
set n2 [$ns node]
set n3 [$ns node]

# Create lan and setup the link
$ns make-lan -trace on "$n0 $n1 $n2 $n3" 100Mb 10ms LL Queue/DropTail Mac/802_3

# Declare the required transport layer Protocols
set tcp0 [new Agent/TCP]
set tcp1 [new Agent/TCP]
set udp2 [new Agent/UDP]
set null1 [new Agent/Null]
set sink2 [new Agent/TCPSink]
set sink3 [new Agent/TCPSink]

# Attach these Protocols to their respective nodes
$ns attach-agent $n0 $tcp0
$ns attach-agent $n1 $tcp1
$ns attach-agent $n2 $udp2
$ns attach-agent $n1 $null1
$ns attach-agent $n2 $sink2
$ns attach-agent $n3 $sink3

# Declare Application layer protocols and attach them with their transport layer protocols
set ftp0 [new Application/FTP]
```

```
set ftp1 [new Application/FTP]
set cbr2 [new Application/Traffic/CBR]
$ftp0 attach-agent $tcp0
$ftp1 attach-agent $tcp1
$cbr2 attach-agent $udp2

# connect source to destination
$ns connect $tcp0 $sink2
$ns connect $udp2 $null1
$ns connect $tcp1 $sink3

# set the interval
$ftp0 set interval_ 0.001
$ftp1 set interval_ 0.001
$cbr2 set interval_ 0.01

# define finish procedure
proc finish {} {
        global ns nf tf
        $ns flush-trace
        exec nam out.nam &
        close $tf
        close $nf

        set count 0
        set tr [open out.tr r]
        while {[gets $tr line] !=-1 } {
                if { [string match "c*" $line] } {
                        set count [expr $count + 1]
                }
        }
        puts "No of packets collided: $count"
        exit 0
}

# schedule the events
$ns at 0.1 "$cbr2 start"
$ns at 0.1 "$ftp0 start"
$ns at 0.1 "$ftp1 start"
$ns at 5.0 "finish"
$ns run

############ output ############

# No of packets collided: 242
```

5. Simulate the transmission of ping messages over a network topology consisting of 6 nodes and find the number of packets dropped due to congestion.

```
# Declare new Simulator
set ns [new Simulator]

# Open trace and nam file in write mode
set tf [open out.tr w]
set nf [open out.nam w]
$ns trace-all $tf
$ns namtrace-all $nf

# Decide the topology
#
#   [s0][ping]     [ping]     [ping]
#      [n0]         [n1]       [n3]
#         '.        |        .'
#           '.      |      .'
#             '.    |    .'
#               '.  |  .'
#                 '.|.'
#                  [n2]
#                 .'|'.
#               .'  |  '.
#             .'    |    '.
#           .'      |      '.
#         .'        |        '.
#       .'          |          '.
#      [n4]         [n5]        [n6]
#    [ping][d0]   [s1][ping]   [ping][d1]

# Create the 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]
set n6 [$ns node]

# set up links
$ns duplex-link $n0 $n2 100Mb 300ms DropTail
$ns duplex-link $n5 $n2 100Mb 300ms DropTail
$ns duplex-link $n1 $n2 100Mb 300ms DropTail
$ns duplex-link $n3 $n2 100Mb 300ms DropTail
$ns duplex-link $n2 $n4 100Mb 300ms DropTail
$ns duplex-link $n2 $n6 100Mb 300ms DropTail

# Declare the agents/protocols
set ping0 [new Agent/Ping]
set ping4 [new Agent/Ping]
set ping5 [new Agent/Ping]
```

```
set ping6 [new Agent/Ping]

# Attach the ping with the respective nodes
$ns attach-agent $n0 $ping0
$ns attach-agent $n4 $ping4
$ns attach-agent $n5 $ping5
$ns attach-agent $n6 $ping6

# Connect the ping from source to destination
$ns connect $ping0 $ping4
$ns connect $ping5 $ping6

# Write proc for ping agent
Agent/Ping instproc recv {from rtt} {
        $self instvar node_
        puts "The node [$node_ id] recieved $from with round trip time $rtt"
}

# Write the proc function
proc finish { } {
        global ns nf tf
        $ns flush-trace
        exec nam out.nam &
        close $nf
        close $tf

        set count 0
        set tr [open out.tr r]
        while {[gets $tr line]!=-1} {
                if {[string match "d*" $line]} {
                        set count [expr $count + 1]
                }
        }
        puts "No. of packet dropped : $count"
        exit 0
}

# shut down link between n2 and n6 for 1 sec(2 - 1 = 1) to show packet dropping
$ns rtmodel-at 1 down $n2 $n6
$ns rtmodel-at 2 up $n2 $n6

# schedule events
for {set i 0.1} {$i<2} {set i [expr $i+0.1]} {
   $ns at $i "$ping0 send"
   $ns at $i "$ping5 send"
}

$ns at 5.0 "finish"
$ns run
```

6. A Simple ESS with transmitting nodes in Wireless LAN
7. A simple ad-hoc network with transmitting nodes

```
# Declare new Simulator
set ns [new Simulator]

# Open the trace file in write mode
set tf [open out.tr w]
$ns trace-all $tf
# Set name-trace for wireless network
set nf [open out.nam w]
$ns namtrace-all-wireless $nf 500 500

# Set new topography
set topo [new Topography]
$topo load_flatgrid 500 500

# Configure for a wireless node.
$ns node-config -adhocRouting DSDV \
-llType LL \
-macType Mac/802_11 \
-ifqType Queue/DropTail \
-ifqLen 50 \
-phyType Phy/WirelessPhy \
-channelType Channel/WirelessChannel \
-propType Propagation/TwoRayGround \
-antType Antenna/OmniAntenna \
-topoInstance $topo \
-agentTrace ON \
-routerTrace ON \
-macTrace OFF

# Create a god object
create-god 3
#################### Decide the topology ###############
#   500
#   |
#   400                    [sink2]
#   |                     [n2]
#   |                    .'
#   |                   .'
#   |                  .'
#   |                 .'
#   |                .'
#   |               .'
#   100           [n1]
#   |         .' [sink1]
#   |       .'   [tcp1]
#   10   [n0]     [ftp1]
#   |  [tcp0]
#   |  [ftp0]
#   |
#   |____10______100_________________400_________500
```

```
# Create nodes
set n0 [$ns node]
set n1 [$ns node]
set n2 [$ns node]

# Locate the nodes on load_flatgrid
$n0 set X_ 10
$n0 set Y_ 10
$n0 set Z_ 0

$n1 set X_ 100
$n1 set Y_ 100
$n1 set Z_ 0

$n2 set X_ 400
$n2 set Y_ 400
$n2 set Z_ 0

# initial state
$ns at 0.0 "$n0 setdest 10 10 15"
$ns at 0.0 "$n1 setdest 100 100 15"
$ns at 0.0 "$n2 setdest 400 400 15"

# move n1 near to node n2 at 50s and come back near to node n0 at 100s
$ns at 50 "$n1 setdest 300 300 15"
$ns at 100 "$n1 setdest 100 100 15"

# Declare and attach transport layer protocol
set tcp0 [new Agent/TCP]
set tcp1 [new Agent/TCP]
$ns attach-agent $n0 $tcp0
$ns attach-agent $n1 $tcp1

set sink1 [new Agent/TCPSink]
set sink2 [new Agent/TCPSink]
$ns attach-agent $n1 $sink1
$ns attach-agent $n2 $sink2

# Declare and attach appliction layer protocol
set ftp0 [new Application/FTP]
set ftp1 [new Application/FTP]
$ftp0 attach-agent $tcp0
$ftp1 attach-agent $tcp1

# connect source to destination
$ns connect $tcp0 $sink1
$ns connect $tcp1 $sink2
```

```
proc finish { } {
    global ns nf tf
    $ns flush-trace
    exec nam out.nam &
    close $tf

    set ctr1 0
    set ctr2 0
    set tf [open out.tr r]

    while {[gets $tf line] != -1} {
        # r->received, _1_ -> destination node
        if {[string match "r*_1_*AGT*" $line]} {
            set ctr1 [expr $ctr1 + 1]
        }
        if {[string match "r*_2_*AGT*" $line]} {
            set ctr2 [expr $ctr2 + 1]
        }
    }
    puts "\nThroughput from n0 to n1: $ctr1"
    puts "Throughput from n1 to n2: $ctr2"
    exit 0
}

# schedule events

# start ftp traffic
$ns at 1 "$ftp0 start"
$ns at 1 "$ftp1 start"
$ns at 150 "finish"
$ns run

########################## output ############################

# num_nodes is set 3
# INITIALIZE THE LIST xListHead
# channel.cc:sendUp - Calc highestAntennaZ_ and distCST_
# highestAntennaZ_ = 1.5,  distCST_ = 550.0
# SORTING LISTS ...DONE!
#
# Throughput from n0 to n1: 8438
# Throughput from n1 to n2: 3000

##############################################################
```