**EX. NO: 1A IMPLEMENTATION OF STOP AND WAIT PROTOCOL**

**AIM:** To write a program to implement stop and wait protocol for transmitting data frames.

**ALGORITHM:**

1. Start the program.
2. Generate a random number that gives the total number of frames to be transmitted.
3. Transmit the first frame.
4. Receive the acknowledgement for the first frame.
5. Transmit the next frame
6. Find the remaining frames to be sent.
7. If an acknowledgement is not received for a particular frame retransmit that frame alone again.
8. Repeat the Steps 5 to 7 till the number of remaining frames to be send becomes zero.
9. Stop the program.

**PROGRAM:**

#include<stdio.h>

#include<stdlib.h>

int main()

{

int x,x1=10,x2,noframes,i=1,j=1,a=1;

printf("enter the maximum buffer size");

scanf("%d",&noframes);

printf("\n number of frames is %d",noframes);

while(a <= noframes)

{

printf("\n sending frame %d",i);

srand(x1++);

x=rand()%10;

if(x%2==0)

{

for(x2=1;x2<2;x2++)

{

printf("waiting for %d seconds \n",x2);

sleep(x2);

}

printf("\n resending frame %d",i);

srand(x1++);

x=rand()%10;

}

printf("\n ack for frame %d",j);

i++;

j++;

a++;

}

printf("\n end of stop and wait protocol");

}

**OUTPUT**

[s@localhost ~]$ vi nex1a.c

[s@localhost ~]$ cc nex1a.c

[s@localhost ~]$ ./a.out

enter the maximum buffer size 5

number of frames is 5

sending frame 1

ack for frame 1

sending frame 2

ack for frame 2

sending frame 3waiting for 1 seconds

resending frame 3

ack for frame 3

sending frame 4

ack for frame 4

sending frame 5

ack for frame 5

end of stop and wait protocol

**RESULT:**

**EX.NO:1B SIMULATION OF SLIDING WINDOW PROTOCOL**

**AIM:** To write a program in C to simulate the Sliding Window Protocol.

**ALGORITHM:**

1. Get the window size and number of frames to be sent from the user.
2. Get the frames to be sent.
3. For the window size, send the data.
4. If the receiver gets the data, send the acknowledgement and the received data.
5. If the data is sent, display the message else display the remaining frames.

**PROGRAM:**

#include<stdio.h>

#include<stdlib.h>

#include<string.h>

int i,j,n,m,win=0,sws,ch,rws,i,d,a[100],b[100];

int main()

{

retry:printf("enter the data size");

scanf("%d",&d);

printf("enter the data that is to be transmitted");

for(i=win;i<d;i++)

scanf("%d",&a[i]);

printf("enter the sender window size");

scanf("%d",&sws);

if(d<sws)

{

printf("\n error");

printf("\n retry");

goto retry;

}

do

{

printf("\n sending data to the receiver \n the data send is");

for(i=win;i<sws+win;i++)

{

b[i]=a[i];

if(a[i]!=0)

printf("%d",a[i]);

}

printf("\n");

printf("\n acknowledgement send for received data");

printf("\n data received is:");

for(i=win;i<sws+win;i++)

if(b[i]!=0)

printf("\t %d",b[i]);

win=win+sws;

j++;

printf("\n");

}

while(win<d);

}

**OUTPUT:**

[s@localhost ~]$ cc nex1b.c

[s@localhost ~]$ ./a.out

enter the data size 5

enter the data that is to be transmitted 1 4 7 2 8

enter the sender window size 3

sending data to the receiver

the data send is 147

acknowledgement send for received data

data received is: 1 4 7

sending data to the receiver

the data send is: 2 8

acknowledgement send for received data

data received is: 2 8

**RESULT**:

**EX.NO.2 STUDY OF SOCKET PROGRAMMING AND CLIENT – SERVER MODEL**

**AIM:** To discuss some of the basic functions used for socket programming.

**Important functions**

**1.socket()**

This function is called by both TCP server and client process to create an empty socket.

#include <sys/socket.h>

int socket (int family, int type, int protocol);

family: specifies the protocol family and is one of the constants below:

|  |  |
| --- | --- |
| **Family** | **Description** |
| AF\_INET | IPv4 protocols |
| AF\_INET6 | IPv6 protocols |
| AF\_LOCAL | Unix domain protocols |
| AF\_ROUTE | Routing sockets |
| AF\_KEY | Key sockets |
|  |  |

type: indicates communications semantics

SOCK\_STREAM - stream socket

SOCK\_DGRAM - datagram socket

SOCK\_RAW - raw socket

protocol: set to 0 except for raw sockets.

Returns on success: socket descriptor (a small nonnegative integer), on error: -1

**2. bind()**

The bind function assigns a local protocol address to a socket. The protocol address is the combination of either a 32-bit IPV4 address or a 128-bit IPV6 address, along with a 16-bit TCP or UDP port number.

#include <sys/socket.h>

int bind(int sockfd, const struct sockaddr \*myaddr, socklen\_t addrlen);

sockfd: a socket descriptor returned by the socket function.

\*myaddr: a pointer to a protocol-specific address.

addrlen: the size of the socket address structure.

Returns on success: 0, on error: -1

**3. connect()**

The connect function is used by a TCP client to establish a connection with a TCP server.

#include <sys/socket.h>

int connect(int sockfd, const struct sockaddr \*servaddr, socklen\_t addrlen);

sockfd: a socket descriptor returned by the socket function

\*servaddr: a pointer to a socket address structure

addrlen: the size of the socket address structure

Returns on success: 0, on error: -1

**4. listen()**

The listen function is called only by a TCP server to converts an unconnected socket into a passive socket, indicating that kernel should accept incoming connection requests directed to its socket.

#include<sys/socket.h>

int listen (int sockfd, int backlog);

sockfd: a socket descriptor returned by the socket function.

backlog: maximum number of connections that the kernel should queue for this socket.

Returns on success: 0, on error: -1

**5. accept()**

The accept function is called by the TCP server to return the next completed connection from the front of the completed connection queue.

#include<sys/socket.h>

int accept (int sockfd, struct sockaddr \*cliaddr, socklen\_t \*addrlen);

sockfd: This is the same socket descriptor as in listen call.

\*cliaddr: used to return the protocol address of the connected peer process

\*addrlen: length of the address.

Returns on success: a new (connected)socket descriptor, on error:-1

**6. close()**

The close function is used to close a socket and terminate a TCP connection.

#include <unistd.h>

int close (int sockfd);

sockfd: This socket descriptor is no longer useable.

Returns on success: 0, on error: -1

**7. read()**

The read function is used to receive data from the specified socket.

#include <unistd.h>

ssize\_t read(int sockfd, const void \* buf, size\_t nbytes);

sockfd: a socket descriptor returned by the socket function.

buf: buffer to store the data.

nbytes: size of the buffer

Returns: number of bytes read if OK,0 on EOF, -1 on error

**8. write()**

The write function is used to send the data through the specified socket.

#include <unistd.h>

ssize\_t write(int sockfd, const void \* buf, size\_t nbytes);

sockfd: a socket descriptor returned by the socket function.

buf: buffer to store the data.

nbytes: size of the buffer

Returns: number of bytes written if OK,0 on EOF, -1 on error

**9. sendto()**

This function is similar to the write function, but additional arguments are required.

#include<sys/socket.h>

ssize\_t sendto(int sockfd, const void \*buff, size\_t nbyte, int flag,

const struct sockaddr \*to, socklen\_t addrlen);

sockfd – socket descriptor

\*buff – pointer to buffer to write from.

nbytes – number of bytes to write.

to – socket address structure containing the protocol address of where the data is to be sent.

addrlen – size of the socket address structure

Returns: number of bytes read or written if OK,-1 on error

**10. recvfrom()**

This function is similar to the read function, but additional arguments are required.

#include<sys/socket.h>

ssize\_t recvfrom(int sockfd, void \*buff, size\_t nbyte, int flag,

struct sockaddr \*from, socklen\_t \*addrlen);

sockfd – socket descriptor

\*buff – pointer to buffer to read.

nbytes – number of bytes to read.

addrlen – size of the socket address structure

from - socket address structure of who sent the datagram.

Returns: number of bytes read or written if OK,-1 on error

**Socket functions for connection-oriented communication**

EOF notification

Data (reply)

Data (request)

Connection establishment

Blocks until connection from client

process request

**TCP Client**

**TCP Server**

socket()

socket()

bind()

listen()

accept()

read()

write()

read()

write()

read()

close()

close ()

conect()

**Socket functions for connection-less communication**

**UDP Client**

**UDP Server**

Blocks until datagram received from a client

Process request

Data (reply)

Data (request)

close()

recvfrom()

sendto()

sendto()

recvfrom()

bind()

socket()

socket()

**RESULT**:

**EX.NO.3**  **SIMULATING ARP / RARP PROTOCOLS**

**AIM:** To write a C program to implement Address Resolution Protocol/Reverse Address Resolution Protocol.

**ALGORITHM:**

**SERVER**

1. Include header files.
2. Create the socket address structure.
3. IP address, port number and family is initialized with server’s socket address structure
4. Socket address is manipulated using the byte manipulation function bzero().
5. Socket created using socket() function.
6. Using bind () function, socket is bound with the server’s well known port.
7. Enter number of client, accept those connection with those client with specified port.
8. The IP address is received from the client and the corresponding MAC address is sent to the client.
9. Socket is closed.

**CLIENT**

1. Include header files.
2. Socket address structure is created.
3. Server’s socket address structure is initialized with IP address, port number and family.
4. Server and the client’s address are manipulated using the byte manipulation functionbzero().
5. Socket is created using socket () function.
6. Socket is bound with the server’s well known port.
7. The IP address is sent to the server.
8. Corresponding MAC address is received from the server and it is printed.
9. Socket is closed.

**PROGRAM:**

**SERVER:**

#include<stdio.h>

#include<sys/types.h>

#include<sys/shm.h>

#include<string.h>

main()

{

int shmid, a, i;

char \*ptr, \*shmptr;

shmid=shmget(3000,10,IPC\_CREAT | 0666);

shmptr=shmat(shmid,NULL,0);

ptr=shmptr;

for(i=0;i<3;i++)

{

puts("Enter the Mac address");

scanf("%s",ptr);

a=strlen(ptr);

printf("string length:%d",a);

ptr[a]= ' ' ;

puts("Enter IP address");

ptr=ptr+a+1;

scanf("%s",ptr);

ptr[a]='\n' ;

ptr= ptr+a+1;

}

ptr[strlen(ptr)]= '\0';

printf("\n ARP table at serverside is=\n%s", shmptr);

shmdt(shmptr);

}

**CLIENT:**

#include<stdio.h>

#include<string.h>

#include<sys/types.h>

#include<sys/shm.h>

main()

{

int shmid,a;

char \*ptr, \*shmptr;

char ptr2[51], ip[12], mac[26];

shmid=shmget(3000,10,0666);

shmptr=shmat(shmid,NULL,0);

puts("ARP table is");

printf("%s",shmptr);

printf("\n1.ARP\n 2.RARP\n 3.EXIT\n");

scanf("%d",&a);

switch(a)

{

case 1:

puts("Enter IP address");

scanf("%s",ip);

ptr=strstr(shmptr, ip);

ptr-=8;

sscanf(ptr,"%s%\*s",ptr2);

printf("MAC addr is %s",ptr2);

break;

case 2:

puts("Enter MAC addr");

scanf("%s",mac);

ptr=strstr(shmptr, mac);

sscanf(ptr,"%\*s%s",ptr2);

printf("IP addr is %s",ptr2);

break;

case 3:

exit(1);

}

}

**OUTPUT:**

**SERVER SIDE**

[s@localhost ~]$ cc nex3ser.c

[s@localhost ~]$ ./a.out

Enter the Mac address

e.e.e.e

String length:7

Enter IP address

1.2.3.4

Enter the Mac address

f.g.h.k

String length:7

Enter IP address

5.6.7.8

Enter the Mac address

d.d.d.d

String length:7

Enter IP address

1.3.5.7

ARP table at serverside is=

e.e.e.e 1.2.3.4

f.g.h.k 5.6.7.8

d.d.d.d 1.3.5.7

**CLIENT SIDE**

s@localhost ~]$ cc nex3cli.c

[s@localhost ~]$ ./a.out

ARP table is

e.e.e.e 1.2.3.4

f.g.h.k 5.6.7.8

d.d.d.d 1.3.5.7

1.ARP

2.RARP

3.EXIT

Enter your choice 1

Enter IP address 1.2.3.4

MAC addr is e.e.e.e

1.ARP

2.RARP

3.EXIT

Enter your choice 2

Enter MAC addr f.g.h.k

IP addr is 5.6.7.8

1.ARP

2.RARP

3.EXIT

Enter your choice 3

**RESULT:**

**EX.NO.4 A SIMULATING** **PING COMMAND**

**AIM:** To write the java program for simulating ping command

**ALGORITHM:**

1. Start the program.
2. Include necessary package in java.
3. To create a process object p to implement the ping command.
4. Declare one BufferedReader stream class object.
5. Get thedetails of the server

5.1: length of the IP address.

5.2: time required to get the details.

5.3: send packets, receive packets and lost packets.

5.4: minimum, maximum and average times.

1. Print the results.
2. Stop the program.

**PROGRAM:**

package ping;

import java.io.\*;

import java.net.\*;

public class Ping

{

public static void main(String[] args)

{

try

{

String str;

System.out.print(" Enter the IP Address to be Ping : ");

BufferedReader buf1=new BufferedReader(new InputStreamReader(System.in));

String ip=buf1.readLine();

Runtime H=Runtime.getRuntime();

Process p=H.exec("ping " + ip);

InputStream in=p.getInputStream();

BufferedReader buf2=new BufferedReader(new

InputStreamReader(in));

while((str=buf2.readLine())!=null)

{

System.out.println(" " + str);

}

}

catch(Exception e)

{

System.out.println(e.getMessage());

}

}

}

**OUTPUT:**

Enter the IP Address to be Ping : 10.10.30.48

PING 10.10.30.48 (10.10.30.48) 56(84) bytes of data.

64 bytes from 10.10.30.48: icmp\_seq=1 ttl=128 time=4.35 ms

64 bytes from 10.10.30.48: icmp\_seq=2 ttl=128 time=0.380 ms

64 bytes from 10.10.30.48: icmp\_seq=3 ttl=128 time=0.350 ms

64 bytes from 10.10.30.48: icmp\_seq=4 ttl=128 time=0.332 ms

64 bytes from 10.10.30.48: icmp\_seq=5 ttl=128 time=0.330 ms

**RESULT:**

**EX.NO.4 B SIMULATING** **TRACEROUTE COMMAND**

**AIM:** To write a program to simulate Traceroute command.

**ALGORITHM:**

1. Start the program and declare the variables.
2. Create the document file path.txt and give the all details
3. Open the document file path.txt and compare the input with the details in the path.txt
4. Trace the route
5. Stop the program

**PROGRAM:**

#include<stdio.h>

#include<string.h>

#include<stdlib.h>

int main()

{

char ip1[25],ip2[25],ip3[25],ip4[25],ip5[25];

char destn[25];

FILE \*fp;

printf("\n tracerroute");

scanf("%s",&destn);

fp=fopen("path.txt","r");

while(!feof(fp))

{

fscanf(fp,"%s\t\t%s\t\t%s\t\t%s\t\t%s\n",&ip1,&ip2,&ip3,&ip4,&ip5);

if((strcmp(destn,ip4)==0)||(strcmp(destn,ip5)==0))

{

printf("\n tracing route to %s\n over a maximum of 30hops",ip4);

printf("\n]%s\n2]%s[%s]\n",ip2,ip3,ip4,ip5);

printf("\n trace complete"); exit(0);

}

}

return 0;

}

**OUTPUT:**

path.txt

3.21.191.19 LocalGateway[67.195.160.76] 145.42.22.125 125.22.42.145 www.rediff.com

3.21.191.19 LocalGateway[67.195.160.76] 213.36.144.59 59.144.36.215 www.monster.com

3.21.191.19 LocalGateway[67.195.160.76] 216.115.96.52 76.13.0.191 www.facebook.com

tracerroute www.facebook.com

tracing route to 76.13.0.191

over a maximum of 30hops

]localgateway[67.195.160.76]

2]216.115.96.52 [76.13.0.191]

trace complete

**RESULT:**

**EX.NO.5 CREATE A SOCKET FOR HTTP FOR WEB PAGE UPLOAD AND DOWNLOAD**

**AIM:**

To write a java program for socket for HTTP for web page upload and download.

**ALGORITHM:**

1. Start the program.

2. Get the frame size from the user

3. To create the frame based on the user request.

4. To send frames to server from the client side.

5. If your frames reach the server it will send ACK signal to client otherwise it will send NACK signal to client.

6. Stop the program

**PROGRAM:**

**webpageclient.java**

import javax.imageio.\*;

import java.io.\*;

import java.awt.image.BufferedImage;

import java.io.ByteArrayOutputStream;

import java.io.File;

import java.io.IOException;

import javax.imageio.ImageIO;

public class Client

{

public static void main(String args[]) throws Exception

{

Socket soc;

BufferedImage img = null;

soc=new Socket("localhost",4015);

System.out.println("Client is running. ");

try

{

System.out.println("Reading image from disk. ");

img = ImageIO.read(new File("/home/s/image/karuppu.jpeg"));

ByteArrayOutputStream baos = new ByteArrayOutputStream();

ImageIO.write(img, "jpg", baos);

baos.flush();

byte[] bytes = baos.toByteArray();

baos.close();

System.out.println("Sending image to server. ");

OutputStream out = soc.getOutputStream();

DataOutputStream dos = new DataOutputStream(out);

dos.writeInt(bytes.length);

dos.write(bytes, 0, bytes.length);

System.out.println("Image sent to server. ");

dos.close();

out.close();

}

catch (Exception e)

{

System.out.println("Exception: " + e.getMessage());

soc.close();

}

soc.close();

}

}

**webpageserver.java**

package web;

import java.net.\*;

import java.io.\*;

import java.awt.image.\*;

import javax.imageio.\*;

import javax.swing.\*;

public class Server

{

public static void main(String args[]) throws Exception

{

ServerSocket server=null;

Socket socket;

server=new ServerSocket(4015);

System.out.println("Server Waiting for image");

socket=server.accept(); System.out.println("Client connected.");

InputStream in = socket.getInputStream();

DataInputStream dis = new DataInputStream(in);

int len = dis.readInt();

System.out.println("Image Size: " + len/1024 + "KB"); byte[] data = new byte[len];

dis.readFully(data);

dis.close();

in.close();

InputStream ian = new ByteArrayInputStream(data);

BufferedImage bImage = ImageIO.read(ian);

JFrame f = new JFrame("Server");

ImageIcon icon = new ImageIcon(bImage);

JLabel l = new Jlabel();

l.setIcon(icon);

f.add(l);

f.pack();

f.setVisible(true);

}

}

**OUTPUT:**

**CLIENT SIDE**

Client is running.

Reading image from disk.

Sending image to server.

Image sent to server.

**SERVER SIDE**

Server waiting for image

Client connected.

Image Size: 3KB



**RESULT:**

**EX.NO.6 IMPLEMENT RPC (REMOTE PROCEDURE CALL**)

**AIM:**  To perform addition and subtraction of two numbers using remote procedure call.

**ALGORITHM:**

1. Create the specification file (file.x).
2. Create the server file.(rpcserver.c)
3. Create the client file.(rpcclient.c)
4. Generate header files and stub files.
5. Compile Server file and run the server.
6. Compile client and Run the client application by specify argument list and hostname of the server.

**file.x**

1. This defines the protocol definition for the application.
2. Define two remote procedures - Each must be called with a single parameter, a structure that holds 2 integers.
3. The return value of each procedure is an int.

**CLIENT**

1. Define the wrapper function that takes care of calling the RPC procedure.
2. Gather everything into a single data structure to send to the server.
3. Call the client stub created by rpcgen.
4. Create a CLIENT data structure that reference the RPC procedure SIMP\_PROG, version SIMP\_VERSION running on the host specified by the first command line arg.
5. Get the 2 numbers that should be added, from the user.
6. Pass the numbers to the server for calculation and prints the output in the terminal.

**SERVER**

1. This file contains the definition of the remote add and subtract procedure used by simple RPC example
2. The return value of the procedure must be a pointer to integer.
3. Declare the variable result as static so we can return a pointer to it.
4. Define implementation of the method add\_1\_svc as add two arguments.
5. Define implementation of the method sub\_1\_svc as subtract second argument from first argument.

**PROGRAM**

**file.x**

#define VERSION\_NUMBER 1

%#define foo 127

struct operands

{

int x,y;

};

program SIMP\_PROG

{

version SIMP\_VERSION

{

int ADD(operands)=1;

int SUB(operands)=2;

}=VERSION\_NUMBER;

}=555555555;

**rpcserver.c**

#include"file.h"

int \*add\_1\_svc(operands \*argp,struct svc\_req \*rqstp)

{

static int result;

printf("Got request:adding %d,%d\n",argp->x,argp->y);

result=argp->x + argp->y;

return(&result);

}

int \*sub\_1\_svc(operands \*argp,struct svc\_req \*rqstp)

{

static int result;

printf("Got request:subtracting %d,%d\n",argp->x,argp->y);

result=argp->x - argp->y;

return(&result);

}

**rpcclient.c**

#include<stdio.h>

#include"file1.h"

int add(CLIENT \*clnt,int x,int y)

{

operands ops;

int \*result;

ops.x=x;

ops.y=y;

result=add\_1(&ops,clnt);

if(result==NULL)

{

fprintf(stderr,"trouble calling remote procedure\n");

exit(0);

}

return(\*result);

}

int sub(CLIENT \*clnt,int x,int y)

{

operands ops;

int \*result;

ops.x=x;

ops.y=y;

result=sub\_1(&ops,clnt);

if(result==NULL)

{

fprintf(stderr,"trouble calling remote procedure\n");

exit(0);

}

return(\*result);

}

int main(int argc,char\*argv[])

{

CLIENT \*clnt;

int x,y;

if(argc!=4)

{

fprintf(stderr,"usage:%s hostname num1 num \n",argv[0]);

exit(0);

}

clnt=clnt\_create(argv[1],SIMP\_PROG,SIMP\_VERSION,"udp");

if(clnt==(CLIENT \*)NULL)

{

clnt\_pcreateerror(argv[1]);

exit(1);

}

x=atoi(argv[2]);

y=atoi(argv[3]);

printf("%d+%d=%d\n",x,y,add(clnt,x,y));

printf("%d-%d=%d\n",x,y,sub(clnt,x,y));

return 0;

}

**OUTPUT:**

[s@localhost ~]$ rpcgen -C -a file.x

[s@localhost ~]$ ls

file\_server.c rpcserver.c file\_svc.c

file.x file\_xdr.c Makefile.file

file\_client.c rpcclient.c file\_clnt.c

file.h

[s@localhost ~]$ cc rpcserver.c file\_svc.c file\_xdr.c

[s@localhost ~]$ ./a.out

Got request:adding 20,10

Got request:subtracting 20,10

[s@localhost ~]$ cc rpcclient.c file\_clnt.c file\_xdr.c

[s@localhost ~]$ ./a.out 127.0.0.1 20 10

20+10=30

20-10=10

**RESULT:**

**EX.NO.7 IMPLEMENTATION** **OF SUBNETTING**

**Aim:** Write a java program to implement subnetting and find the subnet masks.

**ALGORITHM**:

1. Start the program
2. Declare the arguments as Host ip and Netmask
3. Calculate subnet-mask
4. Print the first address and last address
5. Compile and Run the program
6. Stop the program.

**PROGRAM:**

package subnet;

import java.util.Scanner;

import java.io.\*;

import java.net.\*;

public class Subnet {

public static void main(String[] args)

{

Scanner sc = new Scanner(System.in);

System.out.println("Enter the ip address");

String ip = sc.nextLine();

String split\_ip[]=ip.split("\\.");

//SPlit the string after every .

String split\_bip[] = new String[4];

//split binary ip

String bip="";

for(int i=0;i<4;i++){

split\_bip[i] = appendZeros(Integer.toBinaryString(Integer.parseInt(split\_ip[i])));

// “18” => 18 => 10010=>00010010

bip += split\_bip[i];

}

System.out.println("IP in binary is" +bip);

System.out.println("Enter the number of addresses:");

int n = sc.nextInt();

//Calculation of mask

int bits = (int)Math.ceil(Math.log(n)/Math.log(2));

/\*eg if address = 120, log 120/log 2 gives log to the base 2 => 6.9068, ceil gives us upper

integer \*/

System.out.println("Number of bits required for address=" +bits);

int mask = 32-bits;

System.out.println("The subnet mask is="+mask);

//Calculation of first address and last address

int fbip[]= new int[32];

for(int i=0; i<32;i++)

fbip[i] = (int)bip.charAt(i)-48;

//convert character 0,1 to integer 0,1

for(int i=31;i>31-bits;i--)

//Get first address by ANDing last n bits with 0

fbip[i] &= 0;

String fip[]={"","","",""};

for(int i=0;i<32;i++)

fip[i/8] = new String (fip[i/8] + fbip[i]);

System.out.print("First address is = ");

for(int i=0;i<4;i++)

{

System.out.print(Integer.parseInt(fip[i],2));

if(i!=3)

System.out.print(".");

}

System.out.println();

int lbip[] = new int[32];

for(int i=0; i<32;i++)

lbip[i] = (int)bip.charAt(i)-48;

//convert cahracter 0,1 to integer 0,1

for(int i=31;i>31-bits;i--)

//Get last address by ORing last n bits with 1

lbip[i] |= 1;

String lip[] ={"","","",""};

for(int i=0;i<32;i++)

lip[i/8] = new String(lip[i/8]+lbip[i]);

System.out.print("Last address is =");

for(int i=0;i<4;i++){

System.out.print(Integer.parseInt(lip[i],2));

if(i!=3)

System.out.print(".");

}

System.out.println();

}

static String appendZeros(String s){

String temp= "00000000";

return temp.substring(s.length())+ s;

}

}

**OUTPUT:**

Enter the ip address

110.12.14.1

IP in binary is01101110000011000000111000000001

Enter the number of addresses:

4

Number of bits required for address=2

The subnet mask is=30

First address is = 110.12.14.0

Last address is =110.12.14.3

**RESULT:**

**EX.NO.8A APPLICATIONS USING TCP SOCKETS LIKE ECHO CLIENT AND ECHO SERVER**

**AIM:** To write a program in C to implement TCP Echo Client Server (Iterative model).

**ALGORITHM:**

**SERVER:**

1. A TCP socket is created.
2. An Internet socket address structure is filled in with the wildcard address (INADDR\_ANY) and the server’s well-known port (PORT).
3. The socket is converted into a listening socket by calling the listen function.
4. The server blocks in the call to accept, waiting for the client connection to complete.
5. When the connection is established, the server reads the line from the client using readn and echoes it back to the client using written.
6. Finally, the server closes the connected socket.

**CLIENT:**

1. A TCP socket is created.
2. An Internet socket address structure is filled in with the server’s IP address and the same port number.
3. The connect function establishes the connection with the server.
4. The client reads a line of text from the standard input using fgets, writes it to the server using writen, reads back the server’s echo of the line using readline and outputs the echoed line to the standard output using fputs.

**PROGRAM**

**SERVER**

#include<stdio.h>

#include<sys/types.h>

#include<netinet/in.h>

#include<sys/socket.h>

#include<string.h>

#include<unistd.h>

#define PORT 8000

int main(int argc,char\*argv[])

{

char buffer[20];

int sockfd,connfd,a,len;

struct sockaddr\_in servaddr,cliaddr;

sockfd=socket(AF\_INET,SOCK\_STREAM,0);

if(sockfd==-1)

printf("ERROR CREATING SOCKET!");

bzero(&servaddr,sizeof(servaddr))

servaddr.sin\_family=AF\_INET;

servaddr.sin\_port=htons(PORT);

servaddr.sin\_addr.s\_addr=htonl(INADDR\_ANY);

bind(sockfd,(struct sockaddr\*)&servaddr,sizeof(servaddr));

if((a=listen(sockfd,5))<0)

printf("error in listen function");

while(1)

{

len=sizeof(cliaddr);

connfd=accept(sockfd,(struct sockaddr \*)&cliaddr,&len);

strcpy(buffer,"");

read(connfd,buffer,10);

printf("message received and echoed:%s",buffer);

write(connfd,buffer,sizeof(buffer));

}

close(sockfd);

}

**CLIENT**

#include<stdio.h>

#include<sys/types.h>

#include<netinet/in.h>

#include<sys/socket.h>

#include<arpa/inet.h>

#include<string.h>

#include<unistd.h>

#define PORT 8000

int main(int argc,char \*argv[])

{

int sockfd;

struct sockaddr\_in serv;

char buff[20];

sockfd=socket(AF\_INET,SOCK\_STREAM,0);

memset(&serv,0,sizeof(serv));

serv.sin\_family=AF\_INET;

serv.sin\_port=htons(PORT);

serv.sin\_addr.s\_addr=inet\_addr(argv[1]);

if(connect(sockfd,(struct sockaddr \*)&serv,sizeof(serv))<0)

printf("ERROR IN CONNECT");

printf("ENTER THE STRING TO ECHO :");

fgets(buff,sizeof(buff),stdin);

write(sockfd,buff,sizeof(buff));

strcpy(buff," ");

read(sockfd,buff,sizeof(buff));

fputs(buff,stdout);

close(sockfd);

return;

}

**OUTPUT:**

**CLIENTSIDE**

[s@localhost ~]$ cc nex8acli.c

[s@localhost ~]$ ./a.out 127.0.0.1

ENTER THE STRING TO ECHO :hello

hi

**SERVERSIDE**

[s@localhost ~]$ cc nex8aser.c

[s@localhost ~]$ ./a.out

message received and echoed:hello

**RESULT:**

**EX.NO.8B APPLICATIONS USING TCP SOCKETS LIKE CHAT**

**AIM:**

To perform the full duplex chat by sending and receiving the message from the client to server and vice versa using TCP sockets.

**ALGORITHM:**

**SERVER**

1. A TCP socket is created.
2. An Internet socket address structure is filled in with the wildcard address (INADDR\_ANY) and the server’s well-known port (PORT).
3. The socket is converted into a listening socket by calling the listen function.
4. The server blocks in the call to accept, waiting for the client connection to complete.
5. When the connection is established, the server reads the line from the client using connected socket and display the message in the standard output using fputs.
6. Then again the server reads a line of text from the standard input and writes it back to the client through the connected socket.
7. The server went through the Steps (5) and (6) until it receives 'bye' either from the standard input or client.
8. Finally, the server closes the connected socket.

**CLIENT**

1. A TCP socket is created.
2. An Internet socket address structure is filled in with the server’s IP address and the same port number.
3. The connect function establishes the connection with the server.
4. When the connection is established, the client reads the line from the standard input using fgets and sends the message to the server through the socket.
5. Then again the client reads a line of text from the server through the connected socket and writes it back to the standard output using fputs.
6. The client went through the Steps (4) and (5) until it receives 'bye' either from the standard input or server.
7. Finally, the client closes the connected socket.

**PROGRAM**

**SERVER**

**//tcpchatser.c -- TCP CHAT SERVER**

#include<stdio.h>

#include<sys/types.h>

#include<netinet/in.h>

#include<sys/socket.h>

#include<string.h>

#include<unistd.h>

#include<stdlib.h>

#define PORT 4775

int main(int argc,char \*argv[])

{

char buffer[100];

int sockfd,a,connfd,len,i=0;

pid\_t pid;

struct sockaddr\_in servaddr,cliaddr;

sockfd=socket(AF\_INET,SOCK\_STREAM,0);

if(sockfd==-1)

{

printf("Error creating socket\n");

exit(0);

}

printf("Server Socket Created Successfully Connected!\n");

bzero((struct sockaddr \*)&servaddr,sizeof(servaddr));

servaddr.sin\_family=AF\_INET;

servaddr.sin\_port=htons(PORT);

servaddr.sin\_addr.s\_addr=htonl(INADDR\_ANY);

if(bind(sockfd,(struct sockaddr \*)&servaddr,sizeof(servaddr))<0)

{

printf("Error in BIND function");

exit(0);

}

printf("Server Socket Binded.\n");

if((a=listen(sockfd,5))==-1)

{

printf("Error in LISTEN function");

exit(0);

}

printf("Server Socket Listened...\n");

len=sizeof(cliaddr);

if((connfd=accept(sockfd,(struct sockaddr \*)&cliaddr,&len))<0)

{

printf("Error in ACCEPT");

exit(0);

}

do

{

strcpy(buffer," ");

read(connfd,buffer,100);

printf("From client: %s",buffer);

if(strcmp(buffer,"bye\n")==0)

{

printf("From Client: %s",buffer);

goto stop;

}

printf("Server: ");

fgets(buffer,sizeof(buffer),stdin);

write(connfd,buffer,100);

} while(strcmp(buffer,"bye\n")!=0);

stop:

exit(0);

close(connfd);

return 0;

}

**CLIENT**

**//tcpchatcli.c -- TCP CHAT CLIENT**

#include<stdio.h>

#include<sys/types.h>

#include<netinet/in.h>

#include<sys/socket.h>

#include<arpa/inet.h>

#include<string.h>

#include<unistd.h>

#include<stdlib.h>

#define PORT 4775

int main(int argc,char \*argv[])

{

int sockfd;

struct sockaddr\_in serv;

char buff[100];

sockfd=socket(AF\_INET,SOCK\_STREAM,0);

printf("Client Socket Created Successfully.\n");

memset(&serv,0,sizeof(serv));

serv.sin\_family=AF\_INET;

serv.sin\_port=htons(PORT);

serv.sin\_addr.s\_addr=inet\_addr(argv[1]);

if (connect(sockfd,(struct sockaddr \*)&serv, sizeof(serv))<0)

{

printf("error in connect");

exit(0);

}

printf("Client Socket with Server Successfully Connected! \n");

do

{

strcpy(buff," ");

printf("Client: ");

fgets(buff,100,stdin);

write(sockfd,buff,100);

if(strcmp(buff,"bye\n")==0)

{

printf("Client: %s",buff);

goto stop;

}

strcpy(buff," ");

read(sockfd,buff,sizeof(buff));

printf("From Server: %s\n",buff);

} while(strcmp(buff,"bye\n")!=0);

stop:

exit(0);

close(sockfd);

return 0;

}

**OUTPUT:**

**TCP CHAT SERVER**

[s@localhost ~]$ cc tcpchatser.c

[s@localhost ~]$ ./a.out

Server Socket Created Successfully Connected!

Server Socket Binded.

Server Socket Listened...

From client: hi

Server: good morning

From client: have a nice day

Server: bye

**TCP CHAT CLIENT**

[s@localhost ~]$ cc tcpchatcli.c

[s@localhost ~]$ ./a.out 127.0.0.1

Client Socket Created Successfully.

Client Socket with Server Successfully Connected!

Client: hi

From Server: good morning

Client: have a nice day

From Server: bye

**RESULT:**

**EX.NO.8C APPLICATIONS USING TCP SOCKETS LIKE FILE TRANSFER**

**AIM:** To write a C program to transfer file using TCP Sockets

**ALGORITHM:**

**SERVER:**

1. Start the program, declare variables
2. Create a socket using the socket structure socket(AF\_INET, SOCK\_STREAM,0)
3. Set the socket family, IP address and the port using the server address
4. Set the socket address of 8 bytes to zero using the memset() function
5. Bind and listen the socket structure
6. Accept the client connection using the socket descriptor and the server address
7. Get the file name to be transferred, the server search the file and send the file into the client and then close the connection
8. Compile and Execute the program

**CLIENT**

1. Start the program
2. Create a socket using socket(AF\_INET, SOCK\_STREAM,0)
3. Set the 8 bytes address to be zero using memset() function set the socket address family and port using servaddr
4. Establish connection with the server, read the filename to be retrieved from server
5. Receive the file from the server and read the contents of the received file

**PROGRAM**

**SERVER**

#include<stdio.h>

#include<stdlib.h>

#include<unistd.h>

#include<error.h>

#include<string.h>

#include<sys/types.h>

#include<netinet/in.h>

#include<arpa/inet.h>

#include<sys/wait.h>

#include<signal.h>

#include<sys/socket.h>

#define MYPORT 7014

#define BACKLOG 10

int main(void)

{

char buf[100],fname[30]; int n,nbytes;

int sockfd,new\_fd,size,des; struct sockaddr\_in maddr; struct sockaddr\_in taddr;

if((sockfd=socket(AF\_INET,SOCK\_STREAM,0))==-1)

{

perror("SOCKET");

exit(1);

}

maddr.sin\_family=AF\_INET; maddr.sin\_port=htons(MYPORT); maddr.sin\_addr.s\_addr=INADDR\_ANY; memset(&(maddr.sin\_zero),'\0',8);

if(bind(sockfd,(struct sockaddr\*)&maddr,sizeof(struct sockaddr))==-1)

{

perror("BIND");

exit(1);

}

if(listen(sockfd,BACKLOG)==-1)

{

perror("LISTEN");

exit(1);

}

printf("FILE TRANSFER\n");

while(1)

{

size=sizeof(struct sockaddr\_in);

if((new\_fd=accept(sockfd,(struct sockaddr\*)&taddr,&size))==-1)

{

perror("ACCEPT");

continue;

}

if((nbytes=recv(new\_fd,fname,114,0))==-1)

{

perror("ERROR IN RECEIVING\n"); exit(1);

}

if((des=open(fname,0))==-1)

{

perror("ERROR");

exit(0);

}

while((n=read(des,buf,100))>0)

{

if(send(new\_fd,buf,n,0)==-1)

perror("ERROR IN SENDING\n");

}

printf("FILE IS SENT AND READ SUCCESSFULLY\n"); fflush(stdout);

close(des);

}

return 0;

}

**CLIENT**

#include<stdio.h>

#include<stdlib.h>

#include<unistd.h>

#include<errno.h>

#include<string.h>

#include<sys/types.h>

#include<netinet/in.h>

#include<arpa/inet.h>

#include<sys/wait.h>

#include<signal.h>

#define MYPORT 7014

int main(int argc,char \*argv[])

{

char buf[100],fname[14]; int nbytes,sockfd;

struct sockaddr\_in taddr;

if(argc!=2)

{

fprintf(stderr,"usage:ClientHost Name\n"); exit(1);

}

if((sockfd=socket(AF\_INET,SOCK\_STREAM,0))==-1)

{

perror("SOCKET");

exit(1);

}

taddr.sin\_family=AF\_INET; taddr.sin\_port=htons(MYPORT); taddr.sin\_addr.s\_addr=htonl(INADDR\_ANY); memset(&(taddr.sin\_zero),'\0',8);

if(connect(sockfd,(struct sockaddr \*)&taddr,sizeof(struct sockaddr))==-1)

{

perror("CONNECTING ERROR");

exit(1);

}

fflush(stdout);

printf("FILE TRANSFER\n");

fflush(stdout); printf("INPUT\n");

printf("ENTER THE FILE NAME\n");

scanf("%s",fname);

if(send(sockfd,fname,14,0)==- 1)

perror("SENDING ERROR");

printf("OUTPUT\n");

while(1)

{

if((nbytes=recv(sockfd,buf,100,0))!=0)

{

buf[nbytes]='\0';

printf("RECEIVED FROM CLIENT\n");

printf("THE FILE CONTENTS ARE %s",buf);

fflush(stdout);

}

else

break;

}

close(sockfd);

return 0;

}

**OUTPUT:**

**SERVER SIDE**

[s@localhost ~]$ vi file

[s@localhost ~]$ cc nex8cser.c

[s@localhost ~]$ ./a.out

FILE TRANSFER

**CLIENT SIDE**

[s@localhost ~]$ cc nex8ccli.c

[s@localhost ~]$ ./a.out 127.0.0.1

FILE TRANSFER

INPUT

ENTER THE FILE NAME

ftpfile

OUTPUT

RECEIVED FROM CLIENT

THE FILE CONTENTS ARE Hello!

**SERVER SIDE**

[s@localhost ~]$ vi file

[s@localhost ~]$ cc nex8cser.c

[s@localhost ~]$ ./a.out

FILE TRANSFER

FILE IS SENT AND READ SUCCESSFULLY

**RESULT:**

**EX.NO.9A APPLICATIONS USING TCP AND UDP SOCKETS LIKE DNS**

**AIM:** To write a C program to implement Domain Name System using TCP/UDP Sockets.

**ALGORITHM:**

**SERVER**

1. Start the program.
2. Create the Socket for the Server.   
   Bind the Socket to the Port.
3. Listen for the incoming client connection.
4. Receive the IP address from the client to be resolved.
5. Get the domain name from the client.
6. Check the existence of the domain in the server.
7. If domain matches then send the corresponding address to the client.
8. Stop the program execution.

**CLIENT**

1. Start the program.
2. Create the Socket for the client.   
   Connect the Socket to the server.
3. Send the hostname to the server to be resolved.
4. If the server responds the print the address and terminates the process.

**PROGRAM**

**SERVER**

#include<stdio.h>

#include<sys/types.h>

#include<netinet/in.h>

#include<string.h>

main()

{

int sd,sd2,nsd,clilen,sport,len,i;

char sendmsg[20],recvmsg[20];

char ipid[20][20]={"172.15.64.66","172.15.44.55","172.15.33.44","172.15.22.33"};

char hostid[20][20]={"www.yahoo.com","www.google.com","www.hotmail.com"};

struct sockaddr\_in servaddr,cliaddr;

printf("DNS Server Side\n");

printf("Enter the Port\n");

scanf("%d",&sport);

sd=socket(AF\_INET,SOCK\_STREAM,0);

if(sd<0)

printf("Can't Create \n");

else

printf("Socket is Created\n");

servaddr.sin\_family=AF\_INET;

servaddr.sin\_addr.s\_addr=htonl(INADDR\_ANY);

servaddr.sin\_port=htons(sport);

sd2=bind(sd,(struct sockaddr\*)&servaddr,sizeof(servaddr));

if(sd2<0)

printf("Can't Bind\n");

else

printf("\n Binded\n");

listen(sd,5);

clilen=sizeof(cliaddr);

nsd=accept(sd,(struct sockaddr\*)&cliaddr,&clilen);

if(nsd<0)

printf("Can't Accept\n");

else

printf("Accepted\n");

recv(nsd,recvmsg,20,0);

for(i=0;i<4;i++)

{

if(strcmp(recvmsg,hostid[i])==0)

{

send(nsd,ipid[i],20,20);

break;

}

}

}

**CLIENT**

#include<stdio.h>

#include<sys/types.h>

#include<netinet/in.h>

main()

{

int csd,cport,len;

char sendmsg[20],recvmsg[20];

struct sockaddr\_in servaddr;

printf("DNS Client Side\n");

printf("Enter the Client port\n");

scanf("%d",&cport);

csd=socket(AF\_INET,SOCK\_STREAM,0);

if(csd<0)

printf("Can't Create\n");

else

printf("Socket is Created\n");

servaddr.sin\_family=AF\_INET;

servaddr.sin\_addr.s\_addr=htonl(INADDR\_ANY);

servaddr.sin\_port=htons(cport);

if(connect(csd,(struct sockaddr\*)&servaddr,sizeof(servaddr))<0)

printf("Can't Connect\n");

else

printf("Connected\n");

printf("Enter the host address\n");

scanf("%s",sendmsg);

send(csd,sendmsg,20,0); recv(csd,recvmsg,20,20);

printf("The Corresponding IP Address is\n");

printf("%s",recvmsg);

}

**OUTPUT**

**SERVER**

[s@localhost ~]$ cc nex9aser.c

[s@localhost ~]$ ./a.out

DNS Server Side

Enter the Port

6771

Socket is Created

Binded

Accepted

**CLIENT**

[s@localhost ~]$ cc nex9acli.c

[s@localhost ~]$ ./a.out

DNS Client Side

Enter the Client port

6771

Socket is Created

Connected

Enter the host address

www.hotmail.com

The Corresponding IP Address is

172.15.22.33

**RESULT:**

**EX.NO.9B APPLICATIONS USING TCP AND UDP SOCKETS LIKE SNMP**

**AIM:** To write a C program for simulation of Simple Network management Protocols.

**ALGORITHM**

**MANAGER:**

1. Start the program.
2. Create an unnamed socket for client using socket ( ) system.
3. Call with parameters AF\_INET as domain and SOCK\_STREAM as type. Step 4: Name the socket using bind ( ) system call.
4. Now connect the socket to agent using connect ( ) system call.
5. Get the input for the type of information needed from the agent.
6. If the input is equal to „TCP connection‟ then goto next Step else If it is equal to „system‟ Goto Step 9.
7. Read the input for the object, send it and receive the details of the TCP connection of that object from the agent. Go to Step 10.
8. Read the input for the object, send it and receive the details of the system from the agent. Go to Step 10.
9. Receive the message, print and terminate the process.

**AGENTS**

1. Start the program.
2. Create an unnamed socket for the server using the parameters AF\_INET as domain and the SOCK\_STREAM as type.
3. Name the socket using bind( ) system call with the parameters server\_sockfd and the manager address(sin\_addr and sin\_sport).
4. Create a connection queue and wait for manager using the listen ( ) system call with the number of manager request as parameters.
5. Accept the connection using accept( ) system call when manager requests for connection.
6. Receive the message from the manager. If the request is for „TCP connections‟ then send the details of the requested object, else if the request is for „System‟ then send the details of the requested system.
7. Stop the program execution.

**PROGRAM:**

**AGENT1**

#include<stdio.h>

#include<sys/types.h>

#include<netinet/in.h>

#include<string.h>

main()

{

int i,sd,sd2,nsd,clilen,sport,len;

char sendmsg[20],recvmsg[100];

char oid[5][10]={"client1","client2","client3","cleint4","client5"};

char wsize[5][5]={"5","10","15","3","6"};

struct sockaddr\_in servaddr,cliaddr;

printf("I'm the Agent - TCP Connection\n");

printf("\nEnter the Server port");

printf("\n\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\n");

scanf("%d",&sport);

sd=socket(AF\_INET,SOCK\_STREAM,0);

if(sd<0)

printf("Can't Create \n");

else

printf("Socket is Created\n");

servaddr.sin\_family=AF\_INET;

servaddr.sin\_addr.s\_addr=htonl(INADDR\_ANY);

servaddr.sin\_port=htons(sport);

sd2=bind(sd,(struct sockaddr\*)&servaddr,sizeof(servaddr));

if(sd2<0)

printf(" Can't Bind\n");

else

printf("\n Binded\n");

listen(sd,5);

clilen=sizeof(cliaddr);

nsd=accept(sd,(struct sockaddr\*)&cliaddr,&clilen);

if(nsd<0)

printf("Can't Accept\n");

else

printf("Accepted\n");

recv(nsd,recvmsg,100,0);

for (i=0;i<5;i++)

{

if(strcmp(recvmsg,oid[i])==0)

{

send(nsd,wsize[i],100,0); break;

}

}

}

**AGENT2**

#include<stdio.h>

#include<sys/types.h>

#include<netinet/in.h>

#include<string.h>

main()

{

int i,sd,sd2,nsd,clilen,sport,len;

char sendmsg[20],recvmsg[100];

char oid[5][10]={"System1","System2","System3","System4","System5"};

char mdate[5][15]={"1-10-05","10-03-11","14.03.14","11.07.13","17.12.10"};

char time[5][15]={"9am","10pm","11am","12.30pm","11.30am"};

struct sockaddr\_in servaddr,cliaddr;

printf("Enter the Server port");

printf("\n\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\n");

scanf("%d",&sport);

sd=socket(AF\_INET,SOCK\_STREAM,0);

if(sd<0)

printf("Can't Create \n");

else

printf("Socket is Created\n");

servaddr.sin\_family=AF\_INET;

servaddr.sin\_addr.s\_addr=htonl(INADDR\_ANY);

servaddr.sin\_port=htons(sport);

sd2=bind(sd,(struct sockaddr\*)&servaddr,sizeof(servaddr));

if(sd2<0)

printf(" Can't Bind\n");

else

printf("\n Binded\n");

listen(sd,5);

clilen=sizeof(cliaddr);

nsd=accept(sd,(struct sockaddr\*)&cliaddr,&clilen);

if(nsd<0)

printf("Can't Accept\n");

else

printf("Accepted\n");

recv(nsd,recvmsg,100,0);

for(i=0;i<5;i++)

{

if(strcmp(recvmsg,oid[i])==0)

{

send(nsd,mdate[i],100,0);

send(nsd,time[i],100,0);

break;

}

}

}

**MANAGER**

#include<stdio.h>

#include<sys/types.h>

#include<netinet/in.h>

main()

{

int csd,cport,len,i;

char sendmsg[20],rcvmsg[100],rmsg[100],oid[100];

struct sockaddr\_in servaddr;

printf("Enter the port\n");

scanf("%d",&cport);

csd=socket(AF\_INET,SOCK\_STREAM,0);

if(csd<0)

printf("Can't Create\n");

else

printf("Socket is Created\n");

servaddr.sin\_family=AF\_INET;

servaddr.sin\_addr.s\_addr=htonl(INADDR\_ANY);

servaddr.sin\_port=htons(cport);

if(connect(csd,(struct sockaddr\*)&servaddr,sizeof(servaddr))<0)

printf("Can't Connect\n");

else

printf("Connected\n");

printf("\n 1.TCP Connection\n"); printf("\n 2. System \n");

printf("Enter the number for the type of information needed....\n");

scanf("%d",&i);

if(i==1)

{

printf("Enter the Object ID for Client\n");

scanf("%s",oid);

send(csd,oid,100,0);

recv(csd,rmsg,100,0);

printf("\n The window size of %s is %s",oid,rmsg);

}

else

{

printf("\nEnter the Object ID for the System\n");

scanf("%s",oid);

send(csd,oid,100,0);

recv(csd,rmsg,100,0);

printf("\nThe Manufacturing date for %s is %s",oid,rmsg); recv(csd,rmsg,100,0);

printf("\nThe time of Last Utilization for %s is %s",oid,rmsg);

}

}

**OUTPUT**

**AGENT 1**

[s@localhost ~]$ cc nex9ba1.c

[s@localhost ~]$ ./a.out

I'm the Agent - TCP Connection

Enter the Server port

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

8400

Socket is Created

Binded

Accepted

**MANAGER**

[s@localhost ~]$ cc nex9bman.c

[s@localhost ~]$ ./a.out

Enter the port

8400

Socket is Created

Connected

1. TCP Connection

2. System

Enter the number for the type of information needed....

1

Enter the Object ID for Client

client2

The window size of client1 is 10

**AGENT 2**

[s@localhost ~]$ cc nex9ba2.c

[s@localhost ~]$ ./a.out

Enter the Server port

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

8600

Socket is Created

Binded

Accepted

**MANAGER**

[s@localhost ~]$ cc nex9bman.c

[s@localhost ~]$ ./a.out

Enter the port

8600

Socket is Created

Connected

1. TCP Connection

2. System

Enter the number for the type of information needed....

2

Enter the Object ID for the System

System2

The Manufacturing date for System1 is 1-10-05

The time of Last Utilization for System1 is 9pm

**RESULT:**

**EX: NO: 10 STUDY OF NETWORK SIMULATOR (NS)**

**AIM:** To study of ns2 in detail.

**DESCRIPTION:**

*ns* is an object oriented simulator, written in C++, with an OTcl interpreter as a frontend. The simulator supports a class hierarchy in C++ (also called the compiled hierarchy in this document), and a similar class hierarchy within the OTcl interpreter (also called the interpreted hierarchy in this document). The two hierarchies are closely related to each other; from the user’s perspective, there is a one-to-one correspondence between a class in the interpreted hierarchy and one in the compiled hierarchy.

The root of this hierarchy is the class TclObject. Users create new simulator objects through the interpreter; these objects are instantiated within the interpreter, and are closely mirrored by a corresponding object in the compiled hierarchy. The interpreted class hierarchy is automatically established through methods defined in the class TclClass. User instantiated objects are mirrored through methods defined in the class TclObject. There are other hierarchies in the C++ code and OTcl scripts; these other hierarchies are not mirrored in the manner of TclObject.

**CONCEPT OVERVIEW:**

ns uses two languages because simulator has two different kinds of things it needs to do. On one hand, detailed simulations of protocols require a systems programming language which can efficiently manipulate bytes, packet headers, and implement algorithms that run over large data sets. For these tasks run-time speed is important and turn-around time (run simulation, find bug, fix bug, recompile, re-run) is less important. On the other hand, a large part of network research involves slightly varying parameters or configurations, or quickly exploring a number of scenarios.

In these cases, iteration time (change the model and re-run) is more important. Since configuration runs once (at the beginning of the simulation), run-time of this part of the task is less important. ns meets both of these needs with two languages, C++ and OTcl. C++ is fast to run but slower to change, making it suitable for detailed protocol implementation. OTcl runs much slower but can be changed very quickly (and interactively), making it ideal for simulation configuration. ns (via tclcl) provides glue to make objects and variables appear on both languages.

**SIMULATOR INITIALIZATION:**

When a new simulation object is created in tcl, the initialization procedure performs the following operations:

• initialize the packet format (calls create\_packetformat)

• create a scheduler (defaults to a calendar scheduler)

• create a “null agent” (a discard sink used in various places)

**SCHEDULERS AND EVENTS:**

The simulator is an event-driven simulator. There are presently four schedulers available in the simulator, each of which is implemented using a different data structure: a simple linked-list, heap, calendar queue (default), and a special type call called “real-time”. Each of these is described below. The scheduler runs by selecting the next earliest event, executing it to completion and returning to execute the next event. Unit of time used by scheduler is seconds. Presently, the simulator is single-threaded and only one event in execution at any given time. If more than one event are scheduled to execute at the same time, their execution is performed on the first scheduled – first dispatched manner. Simultaneous events are not reordered anymore by schedulers (as it was in earlier versions) and all schedulers should yeild the same order of dispatching given the same input.

**NODE BASICS:**

The basic primitive for creating a node is

set ns [new Simulator]

$ns node

The instance procedure node constructs a node out of simpler classifier objects (Section 5.4). The Node itself is a standalone class in OTcl. However, most of the components of the node are themselves TclObjects. The typical structure of a (unicast) node is as shown in Figure 5.1. This simple structure consists of two TclObjects: an address classifer (classifer\_) and a port classifier (dmux\_). The function of these classifiers is to distribute incoming packets to the correct agent or outgoing link.

All nodes contain at least the following components:

• an address or id\_, monotonically increasing by 1 (from initial value 0) across the simulation namespace as nodes are created,

• a list of neighbors (neighbor\_)

**BASIC COMMANDS IN NS2:**

* set *ns* [new Simulator]: generates an NS simulator object instance, and assigns it to variable *ns*.The "Simulator" object has member functions that do the following:
  + Create compound objects such as nodes and links (described later)
  + Connect network component objects created (ex. attach-agent)
  + Set network component parameters (mostly for compound objects)
  + Create connections between agents (ex. make connection between a "tcp" and "sink")
  + Specify NAM display options
  + Most of member functions are for simulation setup (referred to as plumbing functions in the Overview section) and scheduling, however some of them are for the NAM display. The "Simulator" object member function implementations are located in the file "ns-2/tcl/lib/ns-lib.tcl"
* *$ns* color *fid color*: is to set color of the packets for a flow specified by the flow id (fid). This member function of "Simulator" object is for the NAM display, and has no effect on the actual simulation.
* *$ns* namtrace-all *file-descriptor*: This member function tells the simulator to record simulation traces in NAM input format. It also gives the file name that the trace will be written to later by the command *$ns* flush-trace. Similarly, the member function trace-all is for recording the simulation trace in a general format
* proc *finish* {}: is called after this simulation is over by the command *$ns* at 5.0 "*finish*". In this function, post-simulation processes are specified.
* set *n0* [*$ns* node]: The member function node creates a node. A node in NS is compound object made of address and port classifiers (described in a later section). Users can create a node by separately creating an address and a port classifier objects and connecting them together. However, this member function of Simulator object makes the job easier.
* To see how a node is created, look at the files: "ns-2/tcl/libs/ns-lib.tcl"andns-2/tcl/libs/ns-node.tcl".  
  *$ns* duplex-link *node1 node2 bandwidth delay queue-type*: creates two simplex links of specified bandwidth and delay, and connects the two specified nodes. In NS, the output queue of a node is implemented as a part of a link, therefore users should specify the queue-type when creating links. In the above simulation script, DropTail queue is used. If the reader wants to use a RED queue, simply replace the word DropTail with RED. The NS implementation of a link is shown in a later section. Like a node, a link is a compound object, and users can create its sub-objects and connect them and the nodes. Link source codes can be found in "ns-2/tcl/libs/ns-lib.tcl" and "ns-2/tcl/libs/ns-link.tcl" files. One thing to note is that you can insert error modules in a link component to simulate a lossy link (actually users can make and insert any network objects)
* *$ns* queue-limit *node1 node2 number*: This line sets the queue limit of the two simplex links that connect node1 and node2 to the number specified. At this point, the authors do not know how many of these kinds of member functions of Simulator objects are available and what they are. Please take a look at "ns-2/tcl/libs/ns-lib.tcl" and "ns-2/tcl/libs/ns-link.tcl", or NS.
* $ns duplex-link-op node1 node2 ...: The next couple of lines are used for the NAM display. To see the effects of these lines, users can comment these lines out and try the simulation.

Now that the basic network setup is done, the next thing to do is to setup traffic agents such as TCP and UDP, traffic sources such as FTP and CBR, and attach them to nodes and agents respectively.

* set *tcp* [new *Agent/TCP*]: This line shows how to create a TCP agent. But in general, users can create any agent or traffic sources in this way. Agents and traffic sources are in fact basic objects (not compound objects), mostly implemented in C++ and linked to OTcl. Therefore, there are no specific Simulator object member functions that create these object instances. To create agents or traffic sources, a user should know the class names these objects (Agent/TCP, Agnet/TCPSink, Application/FTP and so on). This information can be found in the NS documentation or partly in this documentation. But one shortcut is to look at the "ns2/tcl/libs/ns-default.tcl" file. This file contains the default configurable parameter value settings for available network objects. Therefore, it works as a good indicator of what kind of network objects is available in NS and what are the configurable parameters.
* *$ns* attach-agent *node agent*: The attach-agent member function attaches an agent object created to a node object. Actually, what this function does is call the attach member function of specified node, which attaches the given agent to itself. Therefore, a user can do the same thing by, for example, $n0 attach $tcp. Similarly, each agent object has a member function .
* *$ns* connect *agent1 agent2*: After two agents that will communicate with each other are created, the next thing is to establish a logical network connection between them. This line establishes a network connection by setting the destination address to each others' network and port address pair.

Assuming that all the network configuration is done, the next thing to do is write a simulation scenario (i.e. simulation scheduling). The Simulator object has many scheduling member functions. However, the one that is mostly used is the following:

* *$ns* at *time "string"*: This member function of a Simulator object makes the scheduler (scheduler\_ is the variable that points the scheduler object created by [new Scheduler] command at the beginning of the script) to schedule the execution of the specified string at given simulation time. For example, *$ns* at *0.1 "$cbr start"* will make the scheduler call a start member function of the CBR traffic source object, which starts the CBR to transmit data. In NS, usually a traffic source does not transmit actual data, but it notifies the underlying agent that it has some amount of data to transmit, and the agent, just knowing how much of the data to transfer, creates packets and sends them.

After all network configurations, scheduling and post-simulation procedure specifications are done, the only thing left is to run the simulation. This is done by *$ns* run.

**RESULT:**

**EX NO: 11 CASE STUDY ON DIFFERENT ROUTING ALGORITHMS**

**INRODUCTION**

Routing algorithms can be differentiated based on several key characteristics. First, the particular goals of the algorithm designer affect the operation of the resulting routing protocol. Second, various types of routing algorithms exist, and each algorithm has a different impact on network and router resources. Finally, routing algorithms use a variety of metrics that affect calculation of optimal routes. The following sections analyze these routing algorithm attributes.

### DESIGN GOALS

Routing algorithms often have one or more of the following design goals:

* Optimality
* Simplicity and low overhead
* Robustness and stability
* Rapid convergence
* Flexibility

Optimality refers to the capability of the routing algorithm to select the best route, which depends on the metrics and metric weightings used to make the calculation. For example, one routing algorithm may use a number of hops and delays, but it may weigh delay more heavily in the calculation. Naturally, routing protocols must define their metric calculation algorithms strictly.

Routing algorithms also are designed to be as simple as possible. In other words, the routing algorithm must offer its functionality efficiently, with a minimum of software and utilization overhead. Efficiency is particularly important when the software implementing the routing algorithm must run on a computer with limited physical resources. Routing algorithms must be robust, which means that they should perform correctly in the face of unusual or unforeseen circumstances, such as hardware failures, high load conditions, and incorrect implementations. Because routers are located at network junction points, they can cause considerable problems when they fail. The best routing algorithms are often those that have withstood the test of time and that have proven stable under a variety of network conditions.

In addition, routing algorithms must converge rapidly. Convergence is the process of agreement, by all routers, on optimal routes. When a network event causes routes to either go down or become available, routers distribute routing update messages that permeate networks, stimulating recalculation of optimal routes and eventually causing all routers to agree on these routes. Routing algorithms that converge slowly can cause routing loops or network outages.

In the routing loop displayed in [Figure: Slow Convergence and Routing Loops Can Hinder Progress](http://docwiki.cisco.com/wiki/Routing_Basics#Figure:_Slow_Convergence_and_Routing_Loops_Can_Hinder_Progress), a packet arrives at Router 1 at time t1. Router 1 already has been updated and thus knows that the optimal route to the destination calls for Router 2 to be the next stop. Router 1 therefore forwards the packet to Router 2, but because this router has not yet been updated, it believes that the optimal next hop is Router 1. Router 2 therefore forwards the packet back to Router 1, and the packet continues to bounce back and forth between the two routers until Router 2 receives its routing update or until the packet has been switched the maximum number of times allowed.

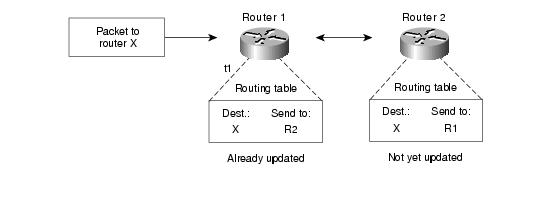
[](http://docwiki.cisco.com/wiki/File:CT840501.jpg)

Figure: Slow Convergence and Routing Loops Can Hinder Progress

Routing algorithms should also be flexible, which means that they should quickly and accurately adapt to a variety of network circumstances. Assume, for example, that a network segment has gone down. As many routing algorithms become aware of the problem, they will quickly select the next-best path for all routes normally using that segment. Routing algorithms can be programmed to adapt to changes in network bandwidth, router queue size, and network delay, among other variables.

#### LINK-STATE VERSUS DISTANCE VECTOR

Link-state algorithms (also known as shortest path first algorithms) flood routing information to all nodes in the internetwork. Each router, however, sends only the portion of the routing table that describes the state of its own links. In link-state algorithms, each router builds a picture of the entire network in its routing tables. Distance vector algorithms (also known as Bellman-Ford algorithms) call for each router to send all or some portion of its routing table, but only to its neighbors. In essence, link-state algorithms send small updates everywhere, while distance vector algorithms send larger updates only to neighboring routers. Distance vector algorithms know only about their neighbors.

Because they converge more quickly, link-state algorithms are somewhat less prone to routing loops than distance vector algorithms. On the other hand, link-state algorithms require more CPU power and memory than distance vector algorithms. Link-state algorithms, therefore, can be more expensive to implement and support. Link-state protocols are generally more scalable than distance vector protocols.

**FLOODING**

Flooding is a simple routing algorithm. Flooding is used in bridging and in systems such as Usenet and peer-to-peer file sharing and as part of some routing protocols, including OSPF, DVMRP, and those used in ad-hoc wireless networks. There are generally two types of flooding available, Uncontrolled Flooding and Controlled Flooding. Each node acts as both a transmitter and a receiver. Each node tries to forward every message to every one of its neighbours except the source node. This results in every message eventually being delivered to all reachable parts of the network. Algorithms may need to be more complex than this, since, in some case, precautions have to be taken to avoid wasted duplicate deliveries and infinite loops, and to allow messages to eventually expire from the system. A variant of flooding called selective flooding partially addresses these issues by only sending packets to routers in the same direction. In selective flooding the routers don't send every incoming packet on every line but only on those lines which are going approximately in the right direction.

**EX NO: 11A IMPLEMENTATION OF DISTANCE VECTOR ROUTING PROTOCOL**

**AIM:** To perform the simulation of the distance vector routing protocol using NS2.

**ALGORITHM:**

1. Start the program
2. Create the trace file and NAM file
3. Setup the topology object
4. Create mobile nodes and attach them to the channel
5. Configure the nodes and provide initial location of mobile nodes
6. Set up a TCP Connection between nodes
7. Define and initialize positions for the NAM window
8. Specify the end of simulation
9. Stop.

**PROGRAM:**

set ns [new Simulator]

#Define different colors for data flows (for NAM)

$ns color 1 Blue

$ns color 2 Yellow

#Open the Trace file

set file1 [open out.tr w]

$ns trace-all $file1

#Open the NAM trace file

set file2 [open out.nam w]

$ns namtrace-all $file2

#Define a 'finish' procedure

proc finish {} {

global ns file1 file2

$ns flush-trace

close $file1

close $file2

exec nam out.nam &

exit 0

}

# Next line should be commented out to have the static routing

$ns rtproto DV

#Create six nodes

set n0 [$ns node]

set n1 [$ns node]

set n2 [$ns node]

set n3 [$ns node]

set n4 [$ns node]

set n5 [$ns node]

#Create links between the nodes

$ns duplex-link $n0 $n4 0.3Mb 10ms DropTail

$ns duplex-link $n0 $n1 0.3Mb 10ms DropTail

$ns duplex-link $n1 $n2 0.3Mb 10ms DropTail

$ns duplex-link $n2 $n3 0.3Mb 10ms DropTail

$ns duplex-link $n1 $n4 0.3Mb 10ms DropTail

$ns duplex-link $n4 $n3 0.5Mb 10ms DropTail

$ns duplex-link $n3 $n5 0.5Mb 10ms DropTail

$ns duplex-link $n4 $n5 0.5Mb 10ms DropTail

#Give node position (for NAM)

$ns duplex-link-op $n0 $n4 orient up

$ns duplex-link-op $n0 $n1 orient right

$ns duplex-link-op $n1 $n2 orient right

$ns duplex-link-op $n2 $n3 orient up

$ns duplex-link-op $n1 $n4 orient up-left

$ns duplex-link-op $n4 $n3 orient right

$ns duplex-link-op $n3 $n5 orient left-up

$ns duplex-link-op $n4 $n5 orient right-up

#Setup a TCP connection

set tcp [new Agent/TCP/Newreno]

$ns attach-agent $n0 $tcp

set sink [new Agent/TCPSink/DelAck]

$ns attach-agent $n5 $sink

$ns connect $tcp $sink

$tcp set fid\_ 1

#Setup a FTP over TCP connection

set ftp [new Application/FTP]

$ftp attach-agent $tcp

$ftp set type\_ FTP

$ns rtmodel-at 1.0 down $n0 $n4

$ns rtmodel-at 4.5 up $n0 $n4

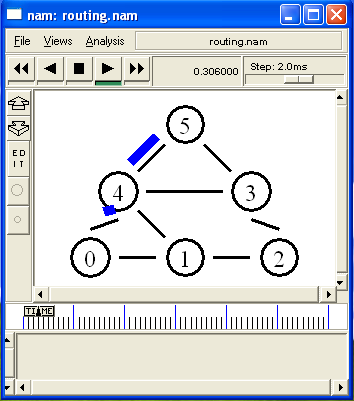
$ns at 0.1 "$ftp start"

$ns at 6.0 "finish"

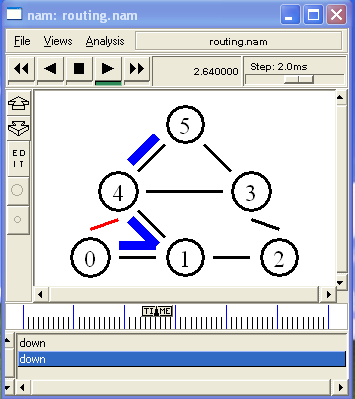
$ns run

**OUTPUT:**

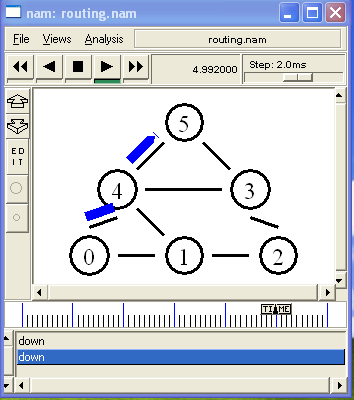
**Up:**



**Down:**



**Up:**



**RESULT:**

**EX: NO: 11B IMPLEMENTATION OF ROUTING ALGORITHM LINK STATE ROUTING**

**AIM:**  To perform the simulation of the link state routing protocol using NS2.

**ALGORITHM:**

1. Define new simualtor
2. Define different colors for data flows (for NAM)
3. Define a new Trace file and open it
4. Define a new NAM Trace file and open it
5. Define a 'finish' procedure – to flush trace record in the `trace and trace output files.
6. Define the routing protocol as Link State (LS)
7. Create six nodes – n0,n1,..n5
8. Create links between the nodes with 0.3Mb and 10 ms Link with DropTail option
9. Give node position (for NAM) to place six nodes in the layout
10. Setup a TCP connection – attach TCP Source Agent to node n0 and TCP sink agent to node n5
11. Setup a FTP over TCP connection
12. Define configuration such that link between nodes n1 and n4 to be failed at 1.0 interval, and up again at 4.5 interval
13. Start the simulation

**PROGRAM**

#routing2.tcl

set ns [new Simulator]

#Define different colors for data flows (for NAM)

$ns color 1 Blue

$ns color 2 Red

#Open the Trace file

set file1 [open routing2.tr w]

$ns trace-all $file1

#Open the NAM trace file

set file2 [open routing2.nam w]

$ns namtrace-all $file2

#Define a 'finish' procedure

proc finish {}

{

global ns file1 file2

$ns flush-trace

close $file1

close $file2

exec nam routing2.nam &

exit 0

}

# Next line should be commented out to have the static routing

$ns rtproto LS

#Create six nodes

set n0 [$ns node]

set n1 [$ns node]

set n2 [$ns node]

set n3 [$ns node]

set n4 [$ns node]

set n5 [$ns node]

#Create links between the nodes

$ns duplex-link $n0 $n1 0.3Mb 10ms DropTail

$ns duplex-link $n1 $n2 0.3Mb 10ms DropTail

$ns duplex-link $n2 $n3 0.3Mb 10ms DropTail

$ns duplex-link $n1 $n4 0.3Mb 10ms DropTail

$ns duplex-link $n3 $n5 0.5Mb 10ms DropTail

$ns duplex-link $n4 $n5 0.5Mb 10ms DropTail

#Give node position (for NAM)

$ns duplex-link-op $n0 $n1 orient right

$ns duplex-link-op $n1 $n2 orient right

$ns duplex-link-op $n2 $n3 orient up-down

$ns duplex-link-op $n1 $n4 orient up-left

$ns duplex-link-op $n3 $n5 orient left-up

$ns duplex-link-op $n4 $n5 orient right-up

#Setup a TCP connection

set tcp [new Agent/TCP/Newreno]

$ns attach-agent $n0 $tcp

set sink [new Agent/TCPSink/DelAck]

$ns attach-agent $n5 $sink

$ns connect $tcp $sink

$tcp set fid\_ 1

#Setup a FTP over TCP connection

set ftp [new Application/FTP]

$ftp attach-agent $tcp

$ftp set type\_ FTP

$ns rtmodel-at 1.0 down $n1 $n4

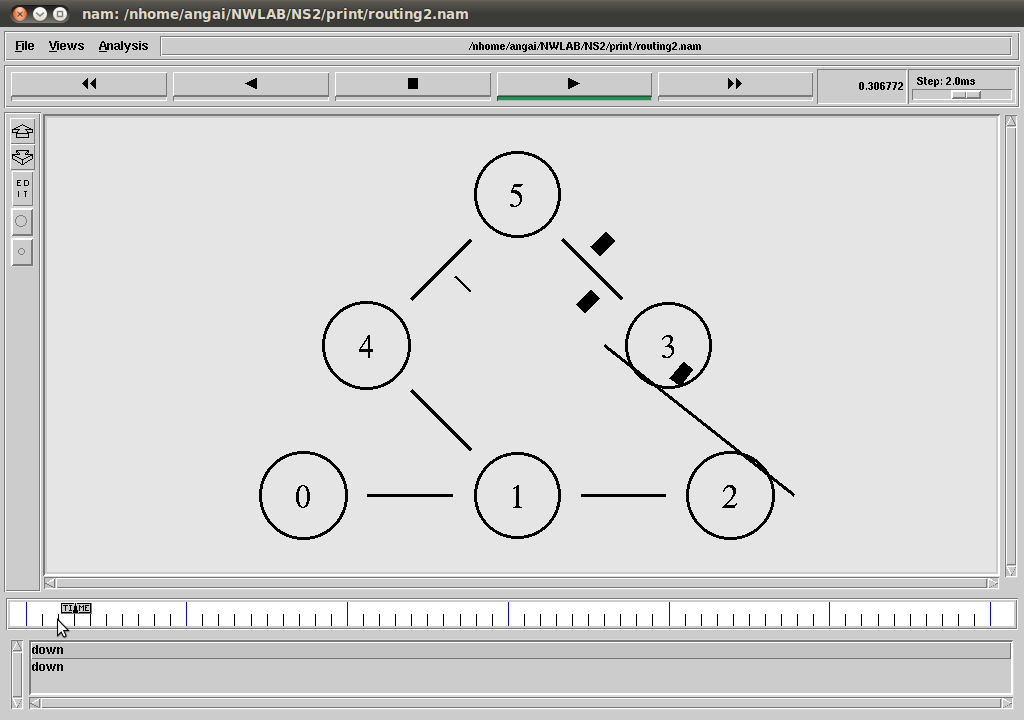
$ns rtmodel-at 3.0 up $n1 $n4

$ns at 0.1 "$ftp start"

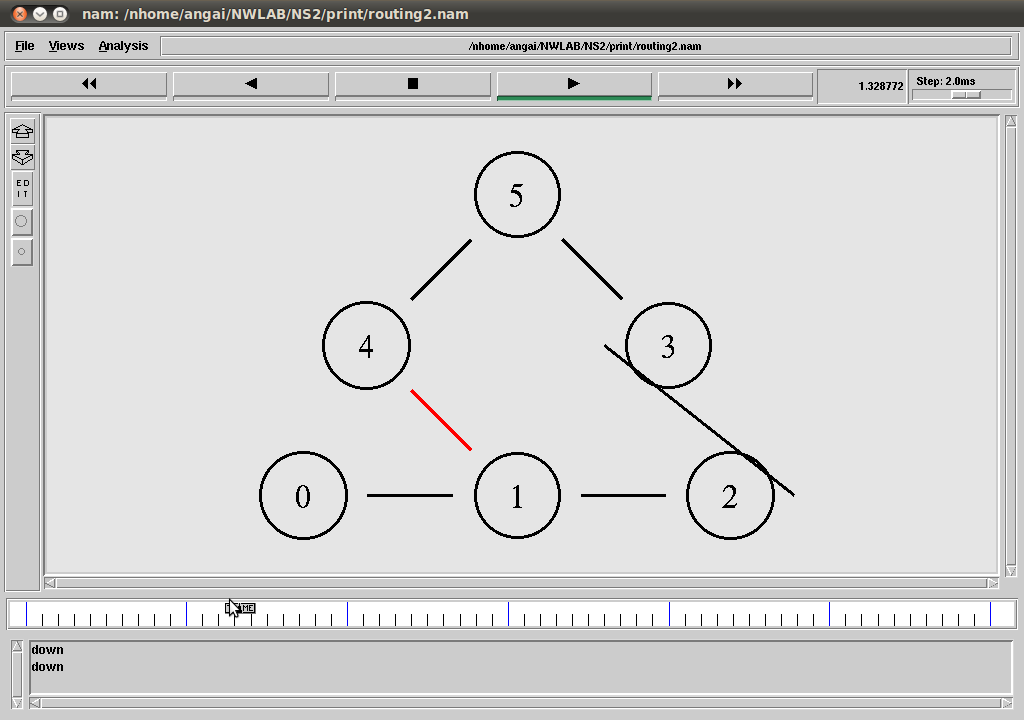
$ns at 6.0 "finish"

$ns run

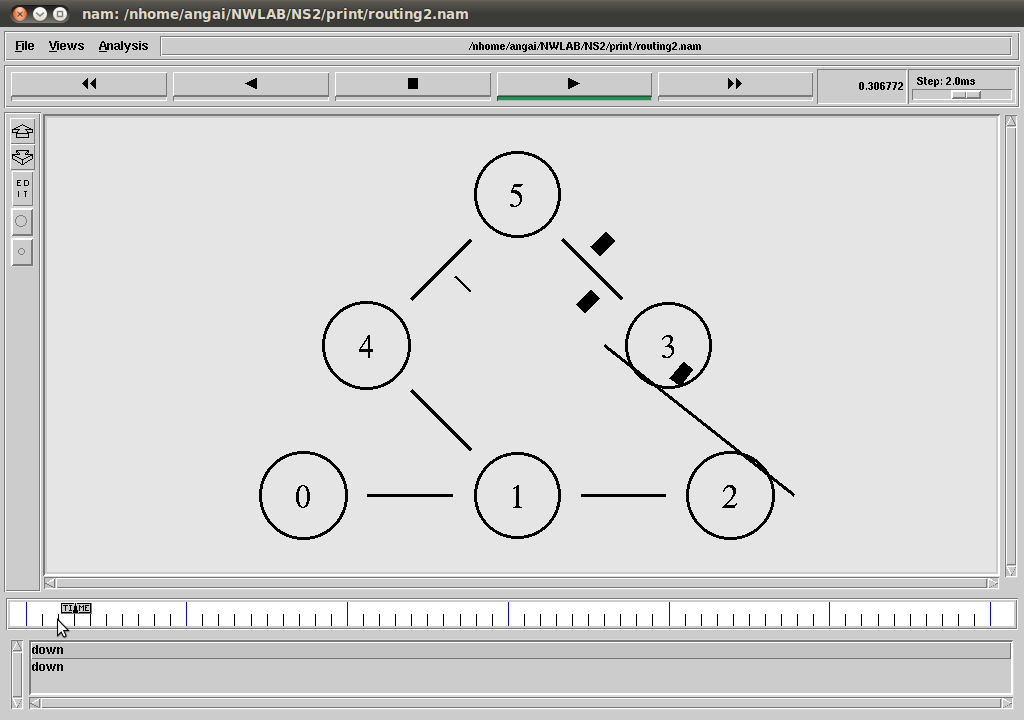
**OUTPUT:**

****

**Before Link failure between Nodes n1 and n4**

****

**While Link failure between Nodes n1 and n4**



**After failed link between Nodes n1 and n4 up**

**RESULT:**

**EX: NO: 11C(i) FLOODING**

**ALGORITHM:**

1. Start the program

2. Create the trace file and NAM file.

3. Setup the flooding agent

4. Create nodes and set duplex links between the nodes.

5. Create nodes and attach them to the queue, DROPTAIL

6. Configure the nodes and provides initial location of nodes. Generation of movements is done.

7. Setup connection between the nodes

8. Define a initialize positions for the nam window.

9. Telling nodes when the simulation ends

10. Ending nam and the simulation

**PROGRAM**

Mac/Simple set bandwidth\_ 1Mb

set MESSAGE\_PORT 42

set BROADCAST\_ADDR -1

# variables which control the number of nodes and how they're grouped

# (see topology creation code below)

set group\_size 4

set num\_groups 6

set num\_nodes [expr $group\_size \* $num\_groups]

set val(chan) Channel/WirelessChannel ;#Channel Type

set val(prop) Propagation/TwoRayGround ;# radio-propagation model

set val(netif) Phy/WirelessPhy ;# network interface type

#set val(mac) Mac/802\_11 ;# MAC type

#set val(mac) Mac ;# MAC type

set val(mac) Mac/Simple

set val(ifq) Queue/DropTail/PriQueue ;# interface queue type

set val(ll) LL ;# link layer type

set val(ant) Antenna/OmniAntenna ;# antenna model

set val(ifqlen) 50 ;# max packet in ifq

# DumbAgent, AODV, and DSDV work. DSR is broken

set val(rp) DumbAgent

#set val(rp) DSDV

#set val(rp) DSR

#set val(rp) AODV

# size of the topography

set val(x) [expr 120\*$group\_size + 500]

set val(y) [expr 240\*$num\_groups + 200]

set ns [new Simulator]

set f [open wireless-flooding-$val(rp).tr w]

$ns trace-all $f

set nf [open wireless-flooding-$val(rp).nam w]

$ns namtrace-all-wireless $nf $val(x) $val(y)

$ns use-newtrace

# set up topography object

set topo [new Topography]

$topo load\_flatgrid $val(x) $val(y)

#

# Create God

#

create-god $num\_nodes

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) \

-topoInstance $topo \

-agentTrace ON \

-routerTrace OFF \

-macTrace ON \

-movementTrace OFF \

-channel $chan\_1\_

# subclass Agent/MessagePassing to make it do flooding

Class Agent/MessagePassing/Flooding -superclass Agent/MessagePassing

Agent/MessagePassing/Flooding instproc recv {source sport size data} {

$self instvar messages\_seen node\_

global ns BROADCAST\_ADDR

# extract message ID from message

set message\_id [lindex [split $data ":"] 0]

puts "\nNode [$node\_ node-addr] got message $message\_id\n"

if {[lsearch $messages\_seen $message\_id] == -1} {

lappend messages\_seen $message\_id

$ns trace-annotate "[$node\_ node-addr] received {$data} from $source"

$ns trace-annotate "[$node\_ node-addr] sending message $message\_id"

$self sendto $size $data $BROADCAST\_ADDR $sport

} else {

$ns trace-annotate "[$node\_ node-addr] received redundant message $message\_id from $source"

}

}

Agent/MessagePassing/Flooding instproc send\_message {size message\_id data port} {

$self instvar messages\_seen node\_

global ns MESSAGE\_PORT BROADCAST\_ADDR

lappend messages\_seen $message\_id

$ns trace-annotate "[$node\_ node-addr] sending message $message\_id"

$self sendto $size "$message\_id:$data" $BROADCAST\_ADDR $port

}

# create a bunch of nodes

for {set i 0} {$i < $num\_nodes} {incr i} {

set n($i) [$ns node]

$n($i) set Y\_ [expr 230\*floor($i/$group\_size) + 160\*(($i%$group\_size)>=($group\_size/2))]

$n($i) set X\_ [expr (90\*$group\_size)\*($i/$group\_size%2) + 200\*($i%($group\_size/2))]

$n($i) set Z\_ 0.0

$ns initial\_node\_pos $n($i) 20

}

# attach a new Agent/MessagePassing/Flooding to each node on port $MESSAGE\_PORT

for {set i 0} {$i < $num\_nodes} {incr i} {

set a($i) [new Agent/MessagePassing/Flooding]

$n($i) attach $a($i) $MESSAGE\_PORT

$a($i) set messages\_seen {}

}

# now set up some events

$ns at 0.2 "$a(1) send\_message 200 1 {first message} $MESSAGE\_PORT"

$ns at 0.4 "$a([expr $num\_nodes/2]) send\_message 600 2 {some big message} $MESSAGE\_PORT"

$ns at 0.7 "$a([expr $num\_nodes-2]) send\_message 200 3 {another one} $MESSAGE\_PORT"

$ns at 1.0 "finish"

proc finish {} {

global ns f nf val

$ns flush-trace

close $f

close $nf

# puts "running nam..."

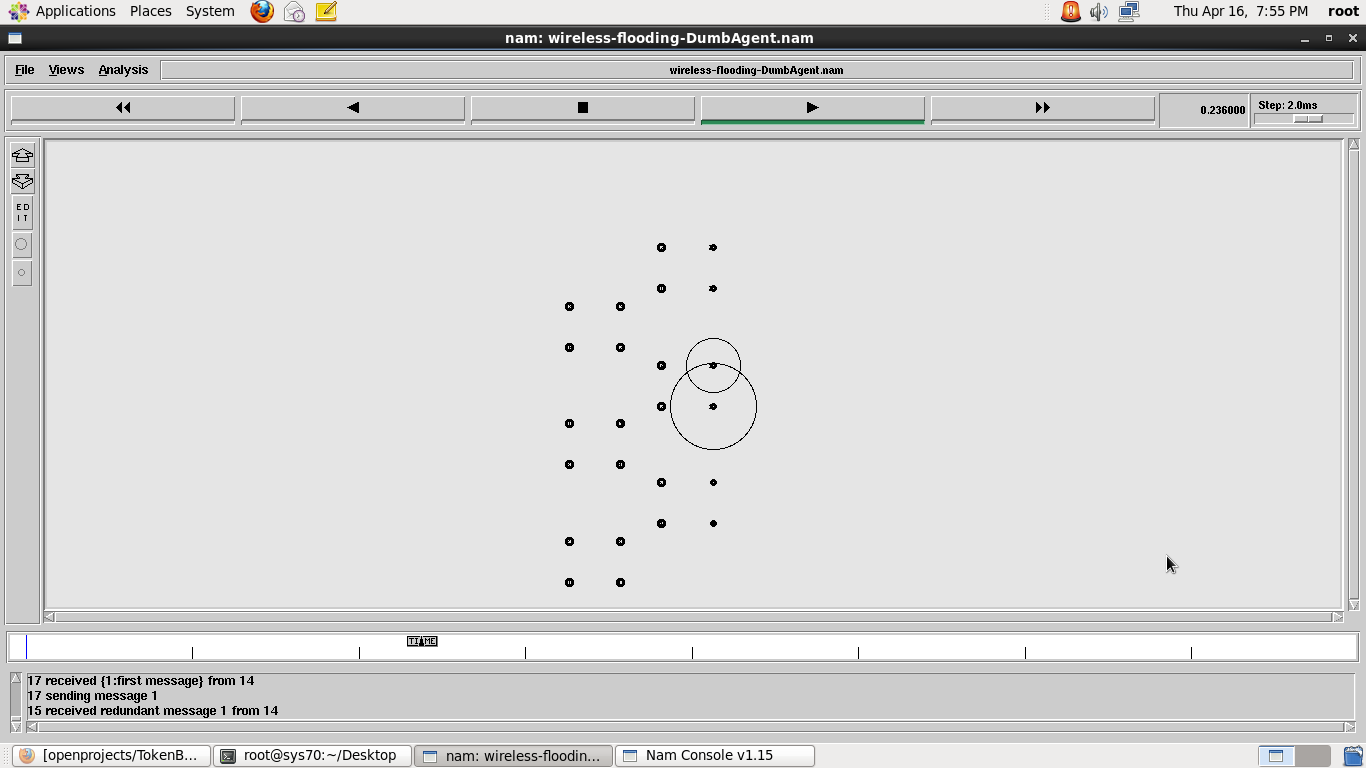
exec nam wireless-flooding-$val(rp).nam &

exit 0

}

$ns run

**OUTPUT**



**RESULT:**

**EX: NO: 11C(ii) SIMULATION OF CONGESTION CONTROL ALGORITHMS USING NS2**

**AIM:** To simulate the performance Stop wait protocol under congestion using ns2.

**ALGORITHM:**

1. Start the program
2. Create the trace file and NAM file.
3. Setup the topology object
4. Create nodes and set duplex links between the nodes.
5. Create nodes and attach them to the queue, DROPTAIL
6. Configure the nodes and provides initial location of nodes. Generation of movements is done.
7. Setup a TCP Connection between nodes
8. Define a initialize positions for the nam window.
9. Telling nodes when the simulation ends
10. Ending nam and the simulation

**PROGRAM:**

set ns [new Simulator]

set n0 [$ns node]

set n1 [$ns node]

set n2 [$ns node]

set n3 [$ns node]

set n4 [$ns node]

set n5 [$ns node]

$n0 color "purple"

$n1 color "purple"

$n2 color "violet"

$n3 color "violet"

$n4 color "chocolate"

$n5 color "chocolate"

set f [open stopwait.tr w]

$ns trace-all $f

set nf [open stopwait.nam w]

$ns namtrace-all $nf

$ns at 0.0 "$n0 label SYS0"

$ns at 0.0 "$n1 label SYS1"

$ns at 0.0 "$n2 label SYS2"

$ns at 0.0 "$n3 label SYS3"

$ns at 0.0 "$n4 label SYS4"

$ns at 0.0 "$n5 label SYS5"

$ns duplex-link $n0 $n2 0.2Mb 20ms DropTail

$ns duplex-link $n1 $n2 0.2Mb 20ms DropTail

$ns duplex-link $n2 $n3 0.2Mb 20ms DropTail

$ns duplex-link $n3 $n4 0.2Mb 20ms DropTail

$ns duplex-link $n3 $n5 0.2Mb 20ms DropTail

$ns duplex-link-op $n0 $n2 orient right-down

$ns duplex-link-op $n1 $n2 orient right-up

$ns duplex-link-op $n2 $n3 orient right

$ns duplex-link-op $n3 $n4 orient right-up

$ns duplex-link-op $n3 $n5 orient right-down

$ns queue-limit $n0 $n2 10

Agent/TCP set\_nam\_tracevar\_true

set tcp [new Agent/TCP]

$tcp set window 1

$tcp set maxcwnd 1

$tcp set fid 1

$ns attach-agent $n0 $tcp

set sink [new Agent/TCPSink]

$ns attach-agent $n5 $sink

$ns connect $tcp $sink

set ftp [new Application/FTP]

$ftp attach-agent $tcp

$ns add-agent-trace $tcp tcp

$ns monitor-agent-trace $tcp

$tcp tracevar cwnd

$ns at 0.1 "$ftp start"

$ns at 0.53 "$ns queue-limit $n3 $n5 0"

$ns at 0.80 "$ns queue-limit $n3 $n5 5"

$ns at 2.0 "$ns detach-agent $n0 $tcp ; $ns detach-agent $n5 $sink"

$ns at 2.5 "finish"

#$ns at 0.0 "$ns trace-annotate \"STOPWAIT\""

$ns at 0.01 "$ns trace-annotate \"FTP starts at 0.01\""

$ns at 0.10 "$ns trace-annotate \"Send Request SYS0 to SYS5\""

$ns at 0.18 "$ns trace-annotate \"Receive Request SYS5 to SYS0\""

$ns at 0.24 "$ns trace-annotate \"Send Packet\_0 SYS0 to SYS5\""

$ns at 0.42 "$ns trace-annotate \"Receive Ack\_0\""

$ns at 0.48 "$ns trace-annotate \"Send Packet\_1\""

$ns at 0.60 "$ns trace-annotate \"Disconnect N2 So loss the packet1\""

$ns at 0.67 "$ns trace-annotate \"Waiting for Ack\_1\""

$ns at 0.95 "$ns trace-annotate \"Send Packet\_1 again\""

$ns at 1.95 "$ns trace-annotate \"Deattach SYS3,Packet\_1 again\""

$ns at 2.09 "$ns trace-annotate \"Receive Ack\_1\""

$ns at 2.10 "$ns trace-annotate \"SEnd Packet\_2\""

$ns at 2.38 "$ns trace-annotate \"Receive Ack\_2\""

$ns at 2.5 "$ns trace-annotate \"FTP stops\""

proc finish {}

{

global ns nf

$ns flush-trace

close $nf

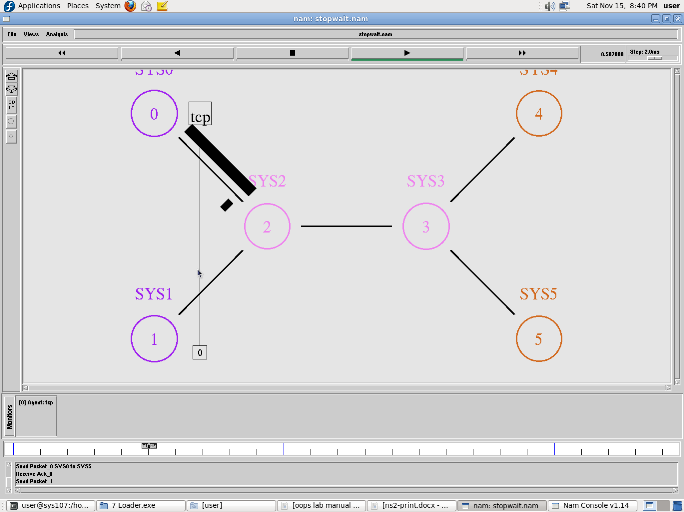
exec nam stopwait.nam &

exit 0

}

$ns run

**OUTPUT:**



**RESULT:**

**Ex.No:12 ERROR DETECTION TECHNIQUE- CYCLIC REDUNDANCY CHECK CRC)**

**AIM:** To implement the error detection technique cyclic redundancy check.

**ALGORITHM:**

1. Start the program
2. Enter the data and generate the polynomial.
3. Call the CRC function with performs XOR between the divisor and generated.
4. Display the checksum.
5. Once the final code is generated , error detection test can be made.
6. Enter the position where error is to be inserted and display the erroneous data.
7. Call the CRC and to check the error.
8. If the error exists, display “error detected” else “no error detected”.
9. Stop the program.

**PROGRAM:**

#include<stdio.h>

#include<string.h>

#define N strlen(g)

char t[28],cs[28],g[]="10001000000100001";

int a,e,c;

void xor()

{

for(c = 1;c < N; c++)

cs[c] = (( cs[c] == g[c])?'0':'1');

}

void crc()

{

for(e=0;e<N;e++)

cs[e]=t[e];

do

{

if(cs[0]=='1')

xor();

for(c=0;c<N-1;c++)

cs[c]=cs[c+1];

cs[c]=t[e++];

}while(e<=a+N-1);

}

int main()

{

printf("\nEnter data : ");

scanf("%s",t);

printf("\n----------------------------------------");

printf("\nGeneratng polynomial : %s",g);

a=strlen(t);

for(e=a;e<a+N-1;e++)

t[e]='0';

printf("\n----------------------------------------");

printf("\nModified data is : %s",t);

printf("\n----------------------------------------");

crc();

printf("\nChecksum is : %s",cs);

for(e=a;e<a+N-1;e++)

t[e]=cs[e-a];

printf("\n----------------------------------------");

printf("\nFinal codeword is : %s",t);

printf("\n----------------------------------------");

printf("\nTest error detection 0(yes) 1(no)? : ");

scanf("%d",&e);

if(e==0)

{

do

{

printf("\nEnter the position where error is to be inserted : ");

scanf("%d",&e);

}while(e==0 || e>a+N-1);

t[e-1]=(t[e-1]=='0')?'1':'0';

printf("\n----------------------------------------");

printf("\nErroneous data : %s\n",t);

}

crc();

for(e=0;(e<N-1) && (cs[e]!='1');e++);

if(e<N-1)

printf("\nError detected\n\n");

else

printf("\nNo error detected\n\n");

printf("\n----------------------------------------\n");

return 0;

}

**OUTPUT:**

Enter data : 1101

----------------------------------------

Generating polynomial : 10001000000100001

----------------------------------------

Modified data is : 11010000000000000000

----------------------------------------

Checksum is : 1101000110101101

----------------------------------------

Final codeword is : 11011101000110101101

----------------------------------------

Test error detection 0(yes) 1(no)? : 0

Enter the position where error is to be inserted : 3

----------------------------------------

Erroneous data : 11111101000110101101

Error detected

**----------------------------------------**

**RESULT:**

**VIVA QUESTION & ANSWERS**

**1. What is an address resolution protocol?**

Address resolution protocol maps an IPV4 addresses into an Ethernet address (Hardware address).

**2. What is the use of ARP?**

A host in an Ethernet network can communicate with another host, only if it knows the Ethernet address (MAC address) of that host. The higher level protocols like IP use a different kind of addressing scheme (like IP address) from the lower level hardware addressing scheme like MAC address. ARP is used to get the Ethernet address of a host from its IP address. ARP is extensively used by all the hosts in an Ethernet network.

**3. How does ARP works?**

In networking, ARP (Address Resolution Protocol) associates the IP address to a network node of the physical hardware address. The ARP creates ARP CACHE/TABLE that maps the hardware address of the nodes with the IP address on the local network. Based on the IP address the ARP check’s whether the hardware address exists in its ARP TABLE. It matches found then the transmission to that IP address is done quickly as destination is known.

**4. What is the reverse address resolution protocol?**

Reverse address resolution protocol maps an Ethernet address into an IPV4 address.

**5. Why is RARP needed?**

Normally, the IP address of a system is stored in a configuration file in the local disk. When the system is started, it determines its IP address from this file. In the case of a diskless workstation, its IP address cannot be stored in the system itself. In this case, RARP can be used to get the IP address from a RARP server.

**6. What is Domain Name System?**

The Domain Name System, or DNS, is used to map between hostnames and IP addresses. A host name can either be a simple name, or a FQDN (Fully Qualified Domain Name).

**7. What is the use of ‘socket’ function?**

Socket function is used to create a socket that is bound to a specific transport service provide. It returns a file descriptor that can be used in later function calls that operate on sockets.

**8. What is the use of ‘connect’ function?**

The connect function establishes a connection to a specified socket. If the socket has not already been bound to a local address, connect() shall bind it to an address which is an unused local address.

**9. What is the need for 'bind' function?**

The bind system call associates a local network transport address with a socket. In case of client it is not necessary to call bind before the process attempts to communicate; the kernel selects and implicitly binds a local address to the socket as needed. The server process must call bind before accepting connections or receiving datagram’s, because the client establish connection or send datagram’s to the well-known address.

**10. Define a trace route?**

An Internet utility that describes the path in real time from the client machine to the remote host being contacted. It reports the IP addresses of all the routers in between.

**11. What is the use of trace route program?**

Trace route lets us see the route that IP datagram’s follow from one host to another. Trace route also lets us use the IP source route option

**12. Mention the advantage of trace route**

Trace route lets us determine the path that IP datagrams follow from our host to some other destination.

**13. What are the 2 sockets needed in a trace route program**

A raw socket on which we read all returned ICMP messages .

An UDP socket on which we send the probe packets with the increasing TTLs.

**13. What are the uses of transport layer?**

Reliable data exchange , Independent of network being used, Independent of application

**14. What is protocol data unit (PDU)?**

Protocols are used to communicate and Control information is added to user data at each layer. Transport layer may fragment user data. Each fragment has a transport header added and header consists of Destination SAP, Sequence number and Error detection code.

**15. What are the uses of internet layer in TCP/IP?**

Systems may be attached to different networks, Routing functions across multiple networks, Implemented in end systems and routers

**16. What is a layered Network Architecture?**

A layer is created when a different level of abstraction occurs at protocol. Each layer should perform a well defined function.

Function of each layer should be chosen using internationality standardized protocols. Boundaries between should be chosen to minimize information flow across the interfaces.

**17. What is the need for layering?**

It reduces the design complexity.

It decomposes the problem of building a network into more manageable components.

**18. What is a protocol?**

Protocol is used for communications between entities in a system and must speak the same language. Protocol is the set of rules governing the exchange of data between 2 entities. It defines what is communicated, how it is communicated, when it is communicated

**19. Name a few fields that are present in TCP Packet**.

Source port no, destination port no, seq.no, ack no, header length, rsvd, window size, data, checksum, urgent pointer, options, spl fields etc,

**20.What is the use of data link layer in OSI?**

**Frame synchronization**: Data is divided by data link layer as frames ,a manageable unit.

**Flow Control**: Sending station does not overwhelm receiving station.

**Error Control**: Any error in bits must be detected and corrected using some mechanism.

**21. Why is flow control and error control duplicated in different layers?**

Like the data link layer, the transport layer is responsible for flow and error control. Flow control and error control at data link layer is node-to-node level. But at transport layer, flow control and error control is performed end-end rather than across a single link.

**22. What are the functions of physical layer?**

Encoding/ decoding of signals, preamble generation/removal (for synchronization) and Bit transmission/ reception are the functions of physical layer.

**23. What are the functions of presentation layer?**

Translation, Encryption / Decryption, Authentication and Compression are the functions of presentation layer.

**24. List the advantages of a centralized scheme.**

It may afford greater control over access for priorities, overrides, and guaranteed capacity.

It enables the use of relatively simple access logic at each station.

It avoids problems of distributed coordination among peer entities.

**25. What is meant by Ethernet and MAC control?**

Ethernet is a networking topology developed in 1970 which is governed by the IEEE 802.3 specification. **MAC control** field contains any protocol control information needed for the functioning of the MAC protocol. For example, a priority level could be indicated here.

1. **What is CSMA/CD?**

It is a protocol used to sense whether a medium is busy before transmission but it has the ability to check whether a transmission has collided with another.

1. **List the rules for CSMA/CD.**

1. If the medium is idle, transmit; otherwise go to step 2.

2. If the medium is busy, continue to listen until the channel is idle, and then transmit immediately.

3. If a collision detected during transmission, transmit a brief jamming signal to all station to indicate collision has occurred and then cease transmission.

4. After transmitting a jamming signal, wait for some time, then transmit again.

1. **What is preamble?**

A 7-octet pattern of alternating 0s and 1s is used by the receiver to establish bit synchronization is called as preamble.

1. **When a transmitting station will insert a new token on the ring?**

It will insert a new token when the station has completed transmission of its frame.

The leading edge of the transmitted frame has returned to the station.

1. **What is a bridge?**

Bridge is a hardware networking device used to connect two LANs. A bridge operates at data link layer of the OSI layer.

1. **List the reason for using bridges in LAN.**

Reliability, performance, security, and geography are the reason for using bridges in LAN

1. **What are the functions of a bridge?**

1. A bridge should have enough buffer space to store the frames until it is transmitted.

2. It should be able to distinguish addresses of host on different LAN.

3. It can contain information about other bridges.

. 4. It should follow congestion control mechanisms to overcome congestion.

5. It works at layer 1 and layer 2 level..

1. **What is spanning tree routing?**

The spanning tree approach is a mechanism in which bridges automatically develop a routing table and update that table in response to changing topology.

**34.Give the applications of wireless LANs.**

LAN extension, cross building interconnect, nomadic access, and advantages hoc networks.

**35.What does IEEE 10 Base 5 standard signify?**

10 represents data rate 10 Mbps. 5 refers to segment length 5\* 100 m that can run without repeaters

Base represents Base band communication

**36.Define Repeater.**

Repeaters and hubs are interconnecting devices. Repeaters extends the Ethernet segment and it repeats the signal. It does not amplify the signal.

**37. Define Hub.**

Hub has several point to point segments coming out. It is a multi way repeater. It broadcasts any signal through all outgoing lines.

**38. Define a switch.**

**Switch**es are hardware or software device capable of creating temporary connections between more devices which are not directly connected. It is a multi input/output port device. It transfers data coming from one input port to one or more output ports.

**39.What are different types of bridge?**

* Simple Bridge connect 2 LAN
* Multi port Bridge connect more than 2 LANs
* Transparent Bridge it learns on its own about connected LANs.

**40. What are the limitations of bridges?**

Scalability and Hetrogenity

**41. What is Token ring?**

Token ring is a set of nodes are connected together in a ring. Data flow always in a particular direction around the ring.

**42. What is the use of bit stuffing?**

**Bit stuffing** is bit oriented protocol. It is used to detect the error during the transmission of the stream of bits.

**43. What is Virtual circuit Switching?**

In the virtual circuit approach, the relationship between all packets belonging to a message or session is preserved single route is chosen between sender and receiver at the beginning of the session.

**44. What is Switched virtual circuit?**

Here virtual circuit is created wherever is needed and exist only for duration of the specific exchange. It can be used with connection establishment and connection termination.

**45. What are the different types of source routing approach?**

Rotation, Stripping off and Using pointers are the different types of source routing approach.

**46. Define Unicasting.**

Unicasting is transmitting data from a single sender to a single receiver.

**47. Define Broadcasting.**

Broadcasting is transmitting data from a single source to all the other nodes in the network

4**8. Define Multicasting.**

**.**Multicasting is transmitting data from a single source to a group of destination nodes.

**49. What is IP address?**

An Internet Address is made of four bytes (32 bits) that define a host’s connection to a network. These are designed to cover the needs of different types of organizations, class A, B, C, D, E.

**50. Differentiate Physical Address and Logical Address.**

**Physical Address Logical Address**

1. It is implemented by data link layer. It is implemented by n/w layer.

2. It contains 48 bits. It contains 32 bits

3. It is a local addressing system. It is an universal address system.

4. Another name MAC address. Another name IP address.

5. It is flat in nature Hierarchical in nature

6. Does not give any clue for routing Its structure gives clue for routing

**51. Define Router.**

* A router operates as the physical, data link and network layer of the OSI model ,
* A router is termed as an intelligent device. Therefore, its capabilities are much more than those of a repeater or a bridge.

**52. Define ARP.**

Associates an IP address with physical address. It is used to find the physical address of the node when its Internet address is known. Any time a host/router needs to find the physical address of another host on its network, it formats an ARP query packet that includes the IP address and broadcasts it.

**53. Define RARP**

Allows a host to discover its internet address when it knows only its physical address ( a diskless computer). The host wishing to retrieve its internet address broadcasts an RARP query packet that contains its physical address to every host on its physical network.

**54. What do you mean by ICMP?**

ICMP is an error reporting mechanism. It does not specify the action to be taken for each possible error. The source must relate the error to an individual application program and take other actions to correct the problem.

**55. List out functions of IP.**

IP services unreliable, best-effort, connectionless packet system.

It defines basic unit of data transfer through TCP/IP.

IP s/w performs routing function – finds a path from source to destination.

IP includes a set of rules that embody the idea of unreliable packet delivery

**56. What is internetworking?**

**I**nternet is an interconnected set of networks. From outside it looks like a simply layer n/w. A collection of communication networks interconnected by routers is called internetworking.

**57. What are t**he principle of inter networks?

It provide a link between networks and routing for delivery of packets.

**58. What are the important fields in a routing table?**

Destination, Cost and Next Hop

**59.What is Trace route option?**

Here source creates an empty list of IP addresses and each router on the path of the datagram adds its IP address to the list whereas a router get a datagram that has record route option, it adds its addresses to the list.

**60.** **List some of the unicast routing protocols.**

Routing Information Protocol (RIP) for IP

Open Shortest Path First (OSPF)

**61. To whom ICMP reports error message?**

**ICMP** allows routers to send error messages to other router or hosts. ICMP is an error reporting mechanism. It does not specify the action to be taken for each possible error. It is informing the source that the error has occurred and the source has to take actions to rectify the errors.

**62. Which class IP addresses are used for multicast and unicast?**

Unicast : Class A, Class B , Class C Multicast: Class D

**63. What is Permanent virtual circuit** ‎?

In this technique the same virtual circuit provided between two users on a continuous basis. The circuit is dedicated to a specific user. No one else can use it. Because it always in place, it can be used without connection establishment and connection termination

**64. What is the use of a router?**

A router is useful for interconnecting two or more heterogeneous networks that differ in their physical characteristics such as frame size, transmission rates, topologies, addressing etc. A router has to determine the best possible transmission path among several available paths.

* + 1. **What are the various adaptive retransmission policy of TCP.**

Simple average, Exponential / weighted average, Exponential RTT backoff, Jacobson’s Algorithm

**66.What is the wrap around time for TCP Sequence Number?**

Once a segment with sequence x survives in Internet, TCP cannot use the same sequence no. How fast 32-bit sequence no space can be consumed? 32-bit sequence no is adequate for today’s network.

**67.What do you mean by congestion?**

Any given node has a number of I/O ports attached to it. There are two buffers at each port—one to accept arriving packets & another one to hold packets that are waiting to depart. If packets arrive too fast node than to process them or faster than packets can be cleared from the outgoing buffers, then there will be no empty buffer.

**68.Name the policies that can prevent congestion.**

* + Additive Increase Multiplicative decrease
  + Slowstart mechanism
  + Fast retransmit and fast recovery.

**69.What are the different phases in TCP state machine?**Connection Establishment, Data transfer and Connection Release.

**70. What is SYN segment?**

It is used to start a TCP connection and provides agreement between sender and receiver on sequence number

**71. Name the policies that can prevent congestion.**

DEC bit, Random Early Detection(RED) and Source based congestion avoidance.

**72. How do transport services differ from the data link layer services?**

The data link layer services are at node to node level. But the transport layer services are end to end level. Both the layers are having the flow control and error control mechanisms. The data link layer offers at node to node level. But the transport layer offers at end to end level. Data link layer is responsible for node to node delivery of the frames while transport layer is responsible for end to end delivery of the entire message.

**73.What are the TCP services to provide reliable communication?**

Error control, Flow control, Connection control and Congestion control.

**74. Define the DEC bit mechanism.**

Each router under this mechanism monitors the load and explicitly notifies the end nodes when congestion is going to occur. This notification is implemented by setting a binary congestion bit which is known as DEC bit, in the header of the packet that follows to the router.

**75. Give some examples for situations using UDP.**

It is very useful for audio or video delivery which does not need acknowledgement. It is useful in the transmission of multimedia data.

**76.** **What is the main difference between TCP & UDP?**

|  |  |
| --- | --- |
| **TCP** | **UDP** |
| It provides Connection oriented service | Provides connectionless service. |
| Connection Establishment delay will be there | No connection establishment delay |
| Provides reliable service | Provides unreliable, but fast service |
| It is used by FTP, SMTP | It is used by audio, video and multimedia applications. |

**77. What is a bus topology?**

Bus topology uses a multipoint medium and all stations are attached through appropriate hardware interfacing known as a tap. A full duplex operation is used for transmission and reception of data in a bus.

**78. What is tree topology?**

Tree topology is generalization of bus topology. Transmission medium is a branching cable with no closed loops. It begins at a point known as headend, where one or more cables start, and each of these may have branches.

**79. What is ring topology?**

In the ring topology, the network consists of a set of repeaters joined by point-to-point links in a closed loop. The repeater is a device which receives data in one link and transmits them in other link.

**80. What is star topology? List the advantages of it.**

In star topology, each station is directly connected to a common central node. Central node is referred as star coupler which uses two point-to-point links, one for transmission in each direction. Its advantages are scalable and easy to identify the fault.

**81. What is ARQ?**

Automatic Repeat Request is used to retransmit the information automatically when the packet is lost while transmit.

**82. What is peer to peer process?**

Communication between peer to peer process, a protocol defines a communication service that it exports locally, along with a set of rules governing the messages that the protocol to implement this service.

**83. List the two types of DNS message.**

DNS messages are: Query and Response. The query message consists of the header and the question records. The response message consists of a header, question record, answer record, authoritative record and additional record.

**84. Define threats.**

Information access threats intercept or modify data on behalf of users who should not have access to that data. Service threats exploit service flaws in computers to inhibit use by legitimate users.

**85. What is meant by attack?**

An attack on system security that derives from an intelligent threat: that is an intelligent act that is a deliberate attempt to evade security services and violate the security policy of a system.

**86. What is use of digital signature?**

Data appended to, or a data unit that allows a recipient of the data unit to prove the source and integrity if the data unit and protect against forgery.

**87. What are the basic functions of e-mail?**

Basic functions of e-mail: composition, Transfer, Reporting, Displaying, and Disposition.

1. **What is a URL?**

Uniform Resource Locator is a string identifier that identifies a page on the World Wide Web.

1. **Which protocol support email and give details about that protocol?**

**SMTP**  is a standard protocol for transferring mails using TCP/IP

SMTP standardization for message character is 7 bit ASCII

SMTP adds log info to the start (i.e.) path of the message

**90. What is TFTP?**

Trivial FTP is designed for transferring bootstrap and configuration files. It is so simple and it can fit into ROM of a diskless memory.

**92. What is the use of MIME Extension?**

MIME converts binary files, executed files into text files. Then only it can be transmitted using SMTP.SMTP cannot transmit text data including national language characters. MIME translates all these non ASCII codes to SMTP 7 bit ASCII code.

**93. How will you categorize DNS.**

Internet is divided into many top level domains. Each domain is divided into sub domain and so on.Topmost domains are categorized into generic and countries**.**

**94. Mention some categories of Generic domain.**

Com-commercial, gov-US government, edu-educational, org-profile organization, mil-US military, net-network providers.

**95. Name some of the country category domains**

uk - United kingdom, sg –singapore, jp -Japan, in –India.

**96. What an application program of DNS does?**

The application program interested in obtaining IP address of a domain name calls a library program "Resolver". Resolver sends UDP packet to nearest DNS server (local DNS server).

**97. Why is HTTP designed as a stateless protocol?**

Maintaining state across request-response connections significantly increase the initial interactions in a connection since the identity of each party needs to be established and any saved state must be retrieved. HTTP is therefore stateless to ensure that the Internet is scalable since state is not contained in the HTTP request/response pairs by default.

1. **What is a web browser?**

Web browser is a software program that interprets and displays the contents of HTML web pages.

**99. What is rlogin? What are the two methods of HTTP?**

Remote login is used to login into remote system and access its contents.**Two methods of HTTP** are GetMethod( ) and PostMethod( ).

**100. What is the main difference between FTP & HTTP?**

FTP – Out – of – band, HTTP – In – band.FTP uses two parallel TCP connections to transfer a file. They are Control Connection and Data connection.