COMPUTER NETWORK LAB FILE

(KCS-663)

Of

Bachelor of Technology

In

Computer Science & Engineering

SESSION 2021-2022

Submitted To

Er. Sharad Yadav

Submitted By

Hardik Jindal
1904310026

BUNDELKHAND INSTITUTE OF ENGINEERING AND TECHNOLOGY,

Jhansi 284128

INDEX

Sr. No	Practical Name	Date	Sign
1.	Implementation of Stop and Wait Protocol and Sliding Window Protocol.		
2.	Study of Socket Programming and Client – Server model.		
3.	Write a code simulating ARP /RARP protocols.		
4.	Create a socket for HTTP for web page upload and download.		
5.	Implementation of Subnetting.		
6.	Applications using TCP and UDP Sockets like d. DNS e. SNMP f. File Transfer		
7.	Perform a case study about the different routing algorithms to select the network path with its optimum and economical during data transfer. i. Link State routing ii. Flooding iii. Distance vector		
8.	To learn handling and configuration of networking hardware like RJ-45 connector, CAT-6 cable, crimping tool, etc.		
9.	Configuration of router, hub, switch etc. (using real devices or simulators)		
10.	Running and using services/commands like ping, traceroute, nslookup, arp, telnet, ftp, etc.		
11.	Socket programming using UDP and TCP (e.g., simple DNS, data & time client/server, echo client/server, iterative & concurrent servers)		

Practical-1

Objective - C++ program to Implement Stop and Wait Flow Control Protocol

Implementation of Stop and Wait Protocol in C++

```
#include<iostream>
#include <time.h>
#include <cstdlib>
#include<ctime>
#include <unistd.h>
using namespace std;
class timer {
   private:
    unsigned long begTime;
   public:
    void start() {
     begTime = clock();
    }
  unsigned long elapsedTime() {
     return ((unsigned long) clock() - begTime) / CLOCKS_PER_SEC;
   }
  bool isTimeout(unsigned long seconds) {
     return seconds >= elapsedTime();
    }
};
int main()
{
 int frames[] = {1,2,3,4,5,6,7,8,9,10};
 unsigned long seconds = 5;
 srand(time(NULL));
 timer t;
 cout<<"Sender has to send frames : ";
 for(int i=0;i<10;i++)
    cout<<frames[i]<<" ";
 cout<<endl;
 int count = 0;
 bool delay = false;
 cout<<endl<<"Sender\t\t\t\t\tReceiver"<<endl;
 do
 {
    bool timeout = false;
    cout<<"Sending Frame : "<<frames[count];
    cout.flush();
    cout<<"\t\t";
    t.start();
```

```
    if(rand()%2)
    {
       int to = 24600 + rand()%(64000 - 24600) + 1;
       for(int i=0;i<64000;i++)
          for(int j=0;j<to;j++) {}
    }
    if(t.elapsedTime() <= seconds)
    {
       cout<<"Received Frame : "<<frames[count]<<" ";
       if(delay)
       {
          cout<<"Duplicate";
          delay = false;
       }
       cout<<endl;
       count++;
    }
    else
    {
       cout<<"---"<<endl;
       cout<<"Timeout"<<endl;
       timeout = true;
    }
    t.start();
    if(rand()%2 || !timeout)
    {
       int to = 24600 + rand()%(64000 - 24600) + 1;
       for(int i=0;i<64000;i++)
          for(int j=0;j<to;j++) {}
       if(t.elapsedTime() > seconds )
       {
          cout<<"Delayed Ack"<<endl;
          count--;
          delay = true;
       }
       else if(!timeout)
          cout<<"Acknowledgement : "<<frames[count]-1<<endl;
    }
 }while(count!=10);
 return 0;
}
```

OUTPUT

Sender has to send frames : 1 2 3 4 5 6 7 8 9 10

Sender Receiver
Sending Frame : 1 Received Frame : 1
Acknowledgement : 1
Sending Frame : 2 ---
Timeout
Sending Frame : 2 Received Frame : 2
Acknowledgement : 2
Sending Frame : 3 Received Frame : 3
Acknowledgement : 3
Sending Frame : 4 Received Frame : 4
Acknowledgement : 4
Sending Frame : 5 Received Frame : 5
Acknowledgement : 5
Sending Frame : 6 Received Frame : 6
Acknowledgement : 6
Sending Frame : 7 Received Frame : 7
Delayed Ack
Sending Frame : 7 Received Frame : 7 Duplicate
Delayed Ack
Sending Frame : 7 ---
Timeout
Sending Frame : 7 Received Frame : 7 Duplicate
Acknowledgement : 7
Sending Frame : 8 ---
Timeout
Delayed Ack
Sending Frame : 7 Received Frame : 7
Acknowledgement : 7
Sending Frame : 8 Received Frame : 8
Acknowledgement : 8
Sending Frame : 9 Received Frame : 9
Delayed Ack
Sending Frame : 9 ---
Timeout
Sending Frame : 9 Received Frame : 9 Duplicate
Delayed Ack
Sending Frame : 9 Received Frame : 9 Duplicate
Acknowledgement : 9
Sending Frame : 10 ---
Timeout
Sending Frame : 10 Received Frame : 10

Implementation of Sliding Window Protocol in C++

```
#include<stdio.h>
#include<conio.h>
void main()
{
char sender[50],receiver[50];
int i,winsize;
clrscr();
 printf("\n ENTER THE WINDOWS SIZE : ");
scanf("%d",&winsize);
 printf("\n SENDER WINDOW IS EXPANDED TO STORE MESSAGE OR WINDOW \n");
 printf("\n ENTER THE DATA TO BE SENT: ");
fflush(stdin);
gets(sender);
for(i=0;i<winsize;i++)
receiver[i]=sender[i];
receiver[i]=NULL;
 printf("\n MESSAGE SEND BY THE SENDER:\n");
 puts(sender);
 printf("\n WINDOW SIZE OF RECEIVER IS EXPANDED\n");
 printf("\n ACKNOWLEDGEMENT FROM RECEIVER \n");
for(i=0;i<winsize;i++);
printf("\n ACK:%d",i);
 printf("\n MESSAGE RECEIVED BY RECEIVER IS : ");
 puts(receiver);
 printf("\n WINDOW SIZE OF RECEIVER IS SHRINKED \n");
getch();
}
```

OUTPUT:

```
ENTER THE WINDOWS SIZE : 10

SENDER WINDOW IS EXPANDED TO STORE MESSAGE OR WINDOW

ENTER THE DATA TO BE SENT: ForgetCode.com

MESSAGE SEND BY THE SENDER:

ForgetCode.com
WINDOW SIZE OF RECEIVER IS EXPANDED
ACKNOWLEDGEMENT FROM RECEIVER
ACK:5
MESSAGE RECEIVED BY RECEIVER IS : ForgetCode
```

Practical-2

Objective- Study of Socket Programming and Client – Server model

Server:

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

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

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

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

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

// Driver function
```

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

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

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

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

    // Now server is ready to listen and verification
    if ((listen(sockfd, 5)) != 0) {
        printf("Listen failed...\n");
        exit(0);
    }
    else
        printf("Server listening..\n");
    len = sizeof(cli);

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

```
    // Function for chatting between client and server
    func(connfd);

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

OUTPUT-Socket successfully created..
Socket successfully binded..
Server listening..
server accept the client...
From client: hi
To client : hello
From client: exit
To client : exit
Server Exit...

Client:

```
        #include <netdb.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/socket.h>
#define MAX 80
#define PORT 8080
#define SA struct sockaddr
void func(int sockfd)
{
    char buff[MAX];
    int n;
    for (;;) {
        bzero(buff, sizeof(buff));
        printf("Enter the string : ");
        n = 0;
        while ((buff[n++] = getchar()) != '\n')
            ;
        write(sockfd, buff, sizeof(buff));
        bzero(buff, sizeof(buff));
        read(sockfd, buff, sizeof(buff));
        printf("From Server : %s", buff);
        if ((strncmp(buff, "exit", 4)) == 0) {
            printf("Client Exit...\n");
            break;
        }
    }
}
```

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

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

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

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

   // function for chat
   func(sockfd);

   // close the socket
   close(sockfd);
}
```

OUTPUT-

```
Socket successfully created..
connected to the server..
Enter the string : hi
From Server : hello
Enter the string : exit
From Server : exit
Client Exit...
```

Practical-3

Objective- Write a code simulating ARP/RARP protocol

Server Side:

arpserver.c

```
#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 name:");
scanf("%s",ptr);
a=strlen(ptr);
printf("String length:%d",a);
ptr[a]=' ';
puts("Enter ip:");
ptr=ptr+a+1;
scanf("%s",ptr);
ptr[a]='\n';
ptr=ptr+a+1;
}
ptr[strlen(ptr)]='\0';
printf("\nARP table at serverside is=\n%s",shmptr);
shmdt(shmptr);
}
```

Client Side:

arpclient.c

```
#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("The ARPtable is:");
printf("%s",shmptr);
printf("\n1.ARP\n2.RARP\n3.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("%s",ptr2);
break;
case  3:
exit(1);
}
}
```

Practical-4

Objective- Create a socket for HTTP for web page upload and download

Client

```
import javax.swing.*;
import java.net.*;
import java.awt.image.*;
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",4000);
System.out.println("Client is running. ");
 try {

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

img = ImageIO.read(new File("digital_image_processing.jpg")); 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();
```

```
}
}
```

Server

```
import java.net.*;
import java.io.*;
import java.awt.image.*;
import   javax.imageio.*;
import javax.swing.*;

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

  ServerSocket server=null;
Socket socket;
server=new ServerSocket(4000);
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);
}
}
```

Practical-5

Objective- Implementation of subnetting

```
import java.io.*;
import java.net.InetAddress;
public class Subnet1 {

	public static void main(String[] args) throws IOException {
	System.out.println("ENTER IP:");
	BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
	String ip = br.readLine();
	String checkclass = ip.substring(0, 3);
	int cc = Integer.parseInt(checkclass);
	String mask = null;
	if(cc>0)
	{
	if(cc<=127)
	{
		mask = "255.0.0.0";
	 System.out.println("Class A IP Address");
	 System.out.println("SUBNET MASK:\n"+mask);
	}
	if(cc>=128 && cc<=191)
	{
		mask = "255.255.0.0";
	 System.out.println("Class B IP Address");
	 System.out.println("SUBNET MASK:\n"+mask);
	}
	if(cc>=192 && cc<=223)
	{
		mask = "255.255.255.0";
	 System.out.println("Class C IP Address");
	 System.out.println("SUBNET MASK:\n"+mask);
	}
	if(cc>=224 && cc<=239)
	{
	 mask = "255.0.0.0";
		System.out.println("Class D IP Address Used for multicasting");
	}
	if(cc>=240 && cc<=254)
	{
	 mask = "255.0.0.0";
		System.out.println("Class E IP Address Experimental Use");
	}
	}
```

```
        String networkAddr="";
         String lastAddr="";
        String[] ipAddrParts=ip.split("\\.");
        String[] maskParts=mask.split("\\.");
        for(int i=0;i<4;i++){
        int x=Integer.parseInt(ipAddrParts[i]);
        int y=Integer.parseInt(maskParts[i]);
        int z=x&y;
        networkAddr+=z+".";
   int w=z|(y^255);
   lastAddr+=w+".";
        }

        System.out.println("First IP of block: "+networkAddr);
        System.out.println("Last IP of block: "+lastAddr);
  }

}
```

OUTPUT

```
iotlab@iotlab-Veriton-M200-B360:~$ javac Subnet1.java
iotlab@iotlab-Veriton-M200-B360:~$ java Subnet1
ENTER IP:
226.35.65.23
Class D IP Address Used for multicasting
First IP of block: 226.0.0.0.
Last IP of block: 226.255.255.255.
iotlab@iotlab-Veriton-M200-B360:~$ java Subnet1
ENTER IP:
192.168.100.5
Class C IP Address
SUBNET MASK:
255.255.255.0
First IP of block: 192.168.100.0.
Last IP of block: 192.168.100.255.
iotlab@iotlab-Veriton-M200-B360:~$
```

Practical-6

Objective- Applications using TCP and UDP Sockets like d. DNS

e. SNMP f. File Transfer

UDP DNS Server

```
.java import java.io.*; import java.net.*;
public class udpdnsserver
{
private static int indexOf(String[] array, String str)
{
str = str.trim();
for (int i=0; i < array.length; i++)
{
if (array[i].equals(str)) return i;
}
return -1;
}

public static void main(String arg[])throws IOException
{
String[] hosts = {"yahoo.com", "gmail.com","cricinfo.com", "facebook.com"};
String[] ip = {"68.180.206.184", "209.85.148.19","80.168.92.140", "69.63.189.16"};
System.out.println("Press Ctrl + C to Quit");
while (true)
{
DatagramSocket serversocket=new DatagramSocket(1362); byte[] senddata = new byte[1021];
byte[] receivedata = new byte[1021];
DatagramPacket recvpack = new DatagramPacket(receivedata, receivedata.length);
serversocket.receive(recvpack);
String sen = new String(recvpack.getData()); InetAddress ipaddress = recvpack.getAddress(); int
port = recvpack.getPort();
String capsent;
System.out.println("Request for host " + sen);
if(indexOf (hosts, sen) != -1) capsent = ip[indexOf (hosts, sen)]; else capsent = "Host Not Found";
senddata = capsent.getBytes();
DatagramPacket pack = new DatagramPacket (senddata, senddata.length,ipaddress,port);
serversocket.send(pack);
serversocket.close();
}
}
}
```

//UDP DNS Client –

```
.java import java.io.*;
import java.net.*;
public class udpdnsclient
{
public static void main(String args[])throws IOException
{
BufferedReader br = new BufferedReader(new InputStreamReader(System.in)); DatagramSocket
clientsocket = new DatagramSocket();

InetAddress ipaddress; if (args.length == 0)
ipaddress = InetAddress.getLocalHost(); else
ipaddress = InetAddress.getByName(args[0]); byte[] senddata = new byte[1024];
byte[] receivedata = new byte[1024];
int portaddr = 1362;
System.out.print("Enter the hostname : ");
String sentence = br.readLine();
Senddata = sentence.getBytes();
DatagramPacket pack = new DatagramPacket(senddata,senddata.length, ipaddress,portaddr);
clientsocket.send(pack);
DatagramPacket recvpack =new DatagramPacket(receivedata,receivedata.length);
clientsocket.receive(recvpack);

String modified = new String(recvpack.getData()); System.out.println("IP Address: " + modified);
clientsocket.close();
}
}
```

OUTPUT Server

```
javac udpdnsserver.java
java udpdnsserver
Press Ctrl + C to Quit Request for host yahoo.com
Request for host cricinfo.com
Request for host youtube.com
```

Client

```
javac udpdnsclient.java
java udpdnsclient
Enter the hostname : yahoo.com
IP Address: 68.180.206.184
java udpdnsclient
Enter the hostname : cricinfo.com
IP Address: 80.168.92.140
java udpdnsclient
Enter the hostname : youtube.com
```

IP Address: Host Not Found

//SNMP
Implementation of SNMP

```
#include<stdio.h>
#include<sys/types.h>
#include<netinet/in.h>
#include<string.h>
main()
{
int i,sd,sd2,nsd,clilen,sport,len;
charsendmsg[20],recvmsg[100];
charoid[5][10]={"client1","client2","client3","cleint4","client5"};
charwsize[5][5]={"5","10","15","3","6"};
structsockaddr_inservaddr,cliaddr;
printf("I'm the Agent-TCP Connection\n");
printf("\nEnter the Serverport");
scanf("%d",&sport);
sd=socket(AF_INET,SOCK_STREAM,0);
if(sd<0)
printf("Can't Create\n");
else
printf("Socket isCreated\n");
servaddr.sin_family=AF_INET;
servaddr.sin_addr.s_addr=htonl(INADDR_ANY);
servaddr.sin_port=htons(sport);
sd2=bind(sd,(structsockaddr*)&servaddr,sizeof(servaddr));
if(sd2<0)
printf("Can'tBind\n");
else
printf("\nBinded\n");
listen(sd,5);
clilen=sizeof(cliaddr);
nsd=accept(sd,(structsockaddr*)&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);
else
break;
}
```

Manager

```
#include<stdio.h>
#include<sys/types.h>
#include<netinet/in.h>
main()
{
intcsd,cport,len,i;
charsendmsg[20],rcvmsg[100],rmsg[100],oid[100];
structsockaddr_inservaddr;
printf("Enter theport\n");
scanf("%d",&cport);
csd=socket(AF_INET,SOCK_STREAM,0);

if(csd<0)
printf("Can't Create\n");
else
printf("Scocket isCreated\n");
servaddr.sin_family=AF_INET;
servaddr.sin_addr.s_addr=htonl(INADDR_ANY);
servaddr.sin_port=htons(cport);
if(connect(csd,(structsockaddr*)&servaddr,sizeof(servaddr))<0)
printf("Can't Connect\n");
else
printf("Connected\n");
printf("\n 1.TCPConnection\n");
printf("\n 2. System\n");
printf("Enter thenumberforthe typeofinformtion needed. ..\n");
scanf("%d",&i);
if(i==1)
{
printf("Enter theObjectIDfor 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 ObjectIDfor the System\n");
scanf("%s",oid);
send(csd,oid,100,0);
recv(csd,rmsg,100,0);
printf("\nTheManufacturingdate for%s is%s",oid,rmsg);
recv(csd,rmsg,100,0);
printf("\n The time of last utilization for %s is %s",oid,rmsg); } }
```

OUTPUT

Agent

```
[cse012@CSLab 2~]$viA1.c
[cse012@CSLab 2~]$ccAt1.c
[cse012@CSLab 2~]$./a.out
I'm theAgent - TCPConnection
Enter theServer port
8202
Socket is created BindedAccepted
```

Manager

```
[cse012@CSLab 2~]$viManager.c
[cse012@CSLab 2~]$ccManger.c
[cse012@CSLab 2~]$./a.out
Enter theport
8202
Socket isCreated
Connected
1.TCPConnection
2. System
Enter thenumberforthetypeof information needed: 1
Enter theObjectIDforClient: Client1
Thewindowsize ofclient1 is 5
```

Practical-7

Objective- Perform a case study about the different routing algorithms to select the network path with its optimum and economical during data transfer. i. Link State routing ii. Flooding iii. Distance vector

Link state routing-

Link-state routing protocols are one of the two main classes of routing protocols used in packet switching networks for computer communications, the other being distance-vector routing protocols. Examples of link-state routing protocols include Open Shortest Path First (OSPF) and Intermediate System to Intermediate System (IS-IS).

The link-state protocol is performed by every switching node in the network (i.e., nodes that are prepared to forward packets; in the Internet, these are called routers). The basic concept of link-state routing is that every node constructs a map of the connectivity to the network, in the form of a graph, showing which nodes are connected to which other nodes. Each node then independently calculates the next best logical path from it to every possible destination in the network. Each collection of best paths will then form each node's routing table.

This contrasts with distance-vector routing protocols, which work by having each node share its routing table with its neighbours, in a link-state protocol the only information passed between nodes is connectivity related. Link-state algorithms are sometimes characterized informally as each router, "telling the world about its neighbors."

Flooding-

Flooding is used in computer networks routing algorithm in which every incoming packet is sent through every outgoing link except the one it arrived on.[1]

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 (WANETs).

There are several variants of flooding algorithms. Most work roughly as follows:

1. Each node acts as both a transmitter and a receiver.
2. Each node tries to forward every message to every one of its neighbors except the source node.
3. 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.

Distance Vector Routing-

A distance-vector routing (DVR) protocol requires that a router inform its neighbors of topology changes periodically. Historically known as the old ARPANET routing algorithm (or known as Bellman-Ford algorithm).

Bellman Ford Basics – Each router maintains a Distance Vector table containing the distance between itself and ALL possible destination nodes. Distances,based on a chosen metric, are computed using information from the neighbors' distance vectors.

Information kept by DV router -
Each router has an ID
Associated with each link connected to a router,
there is a link cost (static or dynamic).
Intermediate hops

Distance Vector Table Initialization -
Distance to itself = 0
Distance to ALL other routers = infinity number.

Distance Vector Algorithm –

A router transmits its distance vector to each of its neighbors in a routing packet.
Each router receives and saves the most recently received distance vector from each of its neighbors.
A router recalculates its distance vector when:
It receives a distance vector from a neighbor containing different information than before.
It discovers that a link to a neighbor has gone down.
The DV calculation is based on minimizing the cost to each destination

Dx(y) = Estimate of least cost from x to y
C(x,v) = Node x knows cost to each neighbor v
Dx = [Dx(y): y ∈ N] = Node x maintains distance vector
Node x also maintains its neighbors' distance vectors
– For each neighbor v, x maintains Dv = [Dv(y): y ∈ N]

Result- Thus, the case study about the different routing algorithms to select the network path with its optimal and economical during data transfer was performed.

Practical-8

Objective- To learn handling and configuration of networking hardware like RJ-45 connector, CAT-6 cable, crimping tool, etc

RJ45-RJ45 is a type of connector commonly used for Ethernet networking. It looks similar to a telephone jack, but is slightly wider. Since Ethernet cables have an RJ45 connector on each end, Ethernet cables are sometimes also called RJ45 cables.
The "RJ" in RJ45 stands for "registered jack," since it is a standardized networking interface. The "45" simply refers to the number of the interface standard. Each RJ45 connector has eight pins, which means an RJ45 cable contains eight separate wires. If you look closely at the end of an Ethernet cable, you can actually see the eight wires, which are each a different color. Four of them are solid colors, while the other four are striped.

The T-568B wiring scheme is by far the most common, though many devices support the T-568A wiring scheme as well. Some applications require a crossover Ethernet cable, which has a T-568A connector on one end and a T-568B connector on the other. This type of cable is typically used for direct computer-to-computer connections

CAT-6- Category 6 cable (Cat 6) is a standardized twisted pair cable for Ethernet and other network physical layers that is backward compatible with the Category 5/5e and Category 3 cable standards.

Cat 6 must meet more stringent specifications for crosstalk and system noise than Cat 5 and Cat 5e. The cable standard specifies performance of up to 250 MHz, compared to 100 MHz for Cat 5 and Cat 5e.

Whereas Category 6 cable has a reduced maximum length of 55 metres (180 ft) when used for 10GBASE-T, Category 6A cable is characterized to 500 MHz and has improved alien crosstalk characteristics, allowing 10GBASE-T to be run for the same 100-metre (330 ft) maximum distance as previous Ethernet variants.Cat 6 cable can be identified by the printing on the side of the cable sheath. Cable types, connector types and cabling topologies are defined by ANSI/TIA-568.

Cat 6 patch cables are normally terminated in 8P8C modular connectors, using either T568A or T568B pin assignments; performance is comparable provided both ends of a cable are terminated identically.
If Cat 6-rated patch cables, jacks and connectors are not used with Cat 6 wiring, overall performance is degraded and may not meet Cat 6 performance specifications.
The category 6 specification requires conductors to be pure copper. The industry has seen a rise in non-compliant / counterfeit cables, especially of the Copper Clad Aluminum (CCA) variety. This has exposed the manufacturers or installers of such fake cable to legal liabilities.

Crimping Tool-

Crimp tools are a varied collection of devices used to join materials or components by pressing them together and creating a seal or crimp. One of the most common uses of crimping tools is the attachment of connectors to the end of electrical cables.

Many crimping tools are multi-functional, with compression available alongside bending, cutting, stripping and similar actions. They are designed to work with particular cable, wire or pipe sizes - or gauges – and some models work with several.
o attach a connector with a network cable or the end of a phone is the main objective of the crimping tool. The most common connectors, RJ-11 and RJ-45 that can be attached to the end of a cable. These connectors are more important for cable and can be attached to the cable with the help of a crimping tool. The flat rocks, hammers, or other improvised tools are used by many homeowners for attaching the connectors to the cables, but it can be led to bad quality or failure. Combining two metal pieces together with the help of deforming one of them is the primary function of the crimping. For use, with the help of attaching a connector on the wire, it prepares a phone cable or a network. The wire that is previously ripped with another particular tool.

The connector is a very important part of the crimping, which prevent the wires from falling out, and helps to transmit data to the wires. If anyone uses the wrong tool, it can become the reason for more damage. There are many different kinds of crimping tools available, but they all have the same function. The built-in ratchet crimping tool is considered the best crimping tool that helps to prevent the jaws from breaking. The ratchet releases the crimped part when it gets the required pressure-the ratchet ensuring that enough pressure is applied and helps to improve the efficiency significantly.

Furthermore, the main purpose of designing the crimping tool with a wider jaw is to cover bigger surface areas. The operators require to match the wire size and the terminal size to crimp a cable with crimping. Once the terminal size and the wire size are matched, the operator will be able to strip the wire.

Practical-9

Objective- Configuration of router, hub, switch etc. (using real devices or simulators

Router:

A router is a networking device that forwards data packets between computer networks. Routers perform the traffic directing functions on the Internet. Data sent through the internet, such as a web page or email, is in the form of data packets. A packet is typically forwarded from one router to another router through the networks that constitute an internetwork (e.g. the Internet) until it reaches its destination node.

A router is connected to two or more data lines from different IP networks.When a data packet comes in on one of the lines, the router reads the network address information in the packet header to determine the ultimate destination. Then, using information in its routing table or routing policy, it directs the packet to the next network on its journey.

The most familiar type of IP routers are home and small office routers that simply forward IP packets between the home computers and the Internet. More sophisticated routers, such as enterprise routers, connect large business or ISP networks up to the powerful core routers that forward data at high speed along the optical fiber lines of the Internet backbone.

Hub:

A network hub is a node that broadcasts data to every computer or Ethernet-based device connected to it. A hub is less sophisticated than a switch, the latter of which can isolate data transmissions to specific devices.

Network hubs are best suited for small, simple local area network (LAN) environments. Hubs cannot provide routing capabilities or other advanced network services. Because they operate by forwarding packets across all ports indiscriminately, network hubs are sometimes referred to as "dumb switches."

With limited capabilities and poor scalability, network hubs had primarily one competitive advantage over switches: lower prices. As switch prices fell in the early to mid-2000s, hubs began getting phased out of use. Today, hubs are far less commonly deployed. But network hubs have some niche uses and continue to offer a simple means of networking.

Switch:

A network switch (also called switching hub, bridging hub, and, by the IEEE, MAC bridge) is networking hardware that connects devices on a computer network by using packet switching to receive and forward data to the destination device.

A network switch is a multiport network bridge that uses MAC addresses to forward data at the data link layer (layer 2) of the OSI model. Some switches can also forward data at the network layer (layer 3) by additionally incorporating routing functionality. Such switches are commonly known as layer-3 switches or multilayer switches.

Switches for Ethernet are the most common form of network switch. The first MAC Bridge was invented in 1983 by Mark Kempf, an engineer in the Networking Advanced Development group of Digital Equipment Corporation.

Practical-10

Objective- Running and using services/commands like ping, traceroute, nslookup, arp, telnet, ftp, etc.

PING:

```
import java.io.*;
import java.net.*;

class pingserver
{
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 the ping:192.168.0.1
Pinging 192.168.0.1: with bytes of data =32
Reply from 192.168.0.11:bytes=32 time<1ms TTL =128
Reply from 192.168.0.11:bytes=32 time<1ms TTL =128
Reply from 192.168.0.11:bytes=32 time<1ms TTL =128
Reply from 192.168.0.11:bytes=32 time<1ms TTL =128

Ping statistics for 192.168.0.1
Packets: sent=4,received=4,lost=0(0% loss),approximate round trip time in milli seconds:
```

Minimum=1
ms,maximum=4ms,average=2ms

TRACEROUTE commands-

```
import java.io.BufferedReader;
import java.io.InputStreamReader;
public class traceroutecmd
{
    public static void runSystemCommand(String command)
    {

        try
        {
            Process p = Runtime.getRuntime().exec(command);
            BufferedReader inputStream = new BufferedReader(
                    new InputStreamReader(p.getInputStream()));
            String s = "";
            while ((s = inputStream.readLine()) != null)
                System.out.println(s);
        }
        catch (Exception e)
        {
        }
    }

    public static void main(String[] args)
    {
        // String ip = "www.google.co.in";
        // String ip = "127.0.0.1";
        String ip = "www.cp-algorithms.com";
        runSystemCommand("tracert " + ip);
    }
}
```

NSLookUp:

```
import java.io.*;
public class NSLookup {
       public static void main(String[] args) throws Exception {
            //Process p = Runtime.getRuntime().exec("/usr/sbin/nslookup "+args[0]);
            // OR YOU CAN DO FOLLOWING
            Process p = Runtime.getRuntime().exec(new String[]{"nslookup",args[0]});
            //BufferedReader bi = new BufferedReader(new
InputStreamReader(p.getInputStream()));
            BufferedReader bi = new BufferedReader(new
InputStreamReader(p.getErrorStream()));
            p.waitFor();
            String line = "";
```

```
            line = bi.readLine();
            while ( line != null ) {
                  //System.out.println(line);
                  if ( line.indexOf("Non-existent") != -1 ) {
                        System.out.println("The host/domain :"+args[0]+": doesn't exist");
                        break;
                  }
                  line = bi.readLine();
            }
            bi.close();
      }
}
```

FTP:

```
import java.io.File;
import java.io.FileInputStream;
import java.io.IOException;
import java.io.InputStream;
import java.io.OutputStream;

import org.apache.commons.net.ftp.FTP;
import org.apache.commons.net.ftp.FTPClient;

/**
 * A program that demonstrates how to upload files from local computer
 * to a remote FTP server using Apache Commons Net API.
 * @author www.codejava.net
 */
public class FTPUploadFileDemo {

   public static void main(String[] args) {
      String server = "www.myserver.com";
      int port = 21;
      String user = "user";
      String pass = "pass";

      FTPClient ftpClient = new FTPClient();
      try {

         ftpClient.connect(server, port);
         ftpClient.login(user, pass);
         ftpClient.enterLocalPassiveMode();

         ftpClient.setFileType(FTP.BINARY_FILE_TYPE);

         // APPROACH #1: uploads first file using an InputStream
         File firstLocalFile = new File("D:/Test/Projects.zip");
```

```
            String firstRemoteFile = "Projects.zip";
            InputStream inputStream = new FileInputStream(firstLocalFile);

            System.out.println("Start uploading first file");
            boolean done = ftpClient.storeFile(firstRemoteFile, inputStream);
            inputStream.close();
            if (done) {
                System.out.println("The first file is uploaded successfully.");
            }

            // APPROACH #2: uploads second file using an OutputStream
            File secondLocalFile = new File("E:/Test/Report.doc");
            String secondRemoteFile = "test/Report.doc";
            inputStream = new FileInputStream(secondLocalFile);

            System.out.println("Start uploading second file");
            OutputStream outputStream = ftpClient.storeFileStream(secondRemoteFile);
            byte[] bytesIn = new byte[4096];
            int read = 0;

            while ((read = inputStream.read(bytesIn)) != -1) {
                outputStream.write(bytesIn, 0, read);
            }
            inputStream.close();
            outputStream.close();

            boolean completed = ftpClient.completePendingCommand();
            if (completed) {
                System.out.println("The second file is uploaded successfully.");
            }

        } catch (IOException ex) {
            System.out.println("Error: " + ex.getMessage());
            ex.printStackTrace();
        } finally {
            try {
                if (ftpClient.isConnected()) {
                    ftpClient.logout();
                    ftpClient.disconnect();
                }
            } catch (IOException ex) {
                ex.printStackTrace();
            }
        }
    }
}
```

Practical-11

Objective- Socket programming using UDP and TCP (e.g., simple DNS, data & time client/server, echo client/server, iterative & concurrent servers

Iterative, Connectionless Servers (UDP)
Creating a Passive UDP Socket

```
int passiveUDP(const char *service)
{
   return passivesock(service, "udp", 0);
}
u_short portbase = 0;

int passivesock(const char *service, const char *transport, int qlen)
{
   struct servent *pse;
   struct protoent *ppe;
   struct sockaddr_in sin;
   int     s, type;
   memset(&sin, 0, sizeof(sin));
   sin.sin_family = AF_INET;
   sin.sin_addr.s_addr = INADDR_ANY;
   /* Map service name to port number */
   if(pse = getservbyname(service, transport))
      sin.sin_port = htons(ntohs((u_short)pse->s_port) + portbase);
   else if((sin.sin_port = htons((u_short)atoi(service))) == 0)
      errexit("can't get \"%s\" service entry\n", service);
   /* Map protocol name to protocol number */
   if((ppe = getprotobyname(transport)) == 0)
      errexit("can't get \"%s\" protocol entry\n", transport);
   /* Use protocol to choose a socket type */
   if(strcmp(transport, "udp") == 0)
      type = SOCK_DGRAM;
   else
      type = SOCK_STREAM;
   /* Allocate a socket */
   s = socket(PF_INET, type, ppe->p_proto);
   if(s < 0)
      errexit("can't create socket: %s\n", strerror(errno));
   /* Bind the socket */
   if(bind(s, (struct sockaddr *)&sin, sizeof(sin)) < 0)
      errexit("can't bind to %s port: %s\n", service, strerror(errno));
```

```
    if(type == SOCK_STREAM && listen(s, qlen) < 0)
        errexit("can't listen on %s port: %s\n", service, strerror(errno));
    return s;
}
```

A TIME Server

```
/* main() - Iterative UDP server for TIME service */
int main(int argc, char *argv[ ])
{
    struct sockaddr_in fsin;
    char *service = "time";
    char buf[1];
    int sock;
    time_t now;
    int alen;
    sock = passiveUDP(service);
 while (1)
{
     alen = sizeof(fsin);
     if(recvfrom(sock, buf, sizeof(buf), 0, (struct sockaddr *)&fsin, &alen) < 0)
        errexit("recvfrom: %s\n", strerror(errno));
     time(&now);
     now = htonl((u_long)now);
     sendto(sock, (char *)&now, sizeof(now), 0, (struct sockaddr *)&fsin, sizeof(fsin));
 }
}
```

Iterative, Connection-Oriented Servers (TCP)
A DAYTIME Server

```
int passiveTCP(const char *service, int qlen)
{
    return passivesock(service, "tcp", qlen);
}
int main(int argc, char *argv[ ])
{
    struct sockaddr_in fsin;
    char *service = "daytime";
    int msock, ssock;
    int alen;
    msock = passiveTCP(service, 5);
    while (1) {
```

```
        ssock = accept(msock, (struct sockaddr *)&fsin, &alen);
        if(ssock < 0)
           errexit("accept failed: %s\n", strerror(errno));
        TCPdaytimed(ssock);
        close(ssock);
  }
}
void TCPdaytimed(int fd)
{
    char *pts;
    time_t now;
    char *ctime();
    time(&now);
    pts = ctime(&now);
    write(fd, pts, strlen(pts));
    return;
}
```

Concurrent, Connection-Oriented Servers (TCP)

Concurrent Echo Server Using fork()

Concurrent Echo Server Using fork()

```
int main(int argc, char *argv[ ])
{
    char *service = "echo"; /* service name or port number */
    struct sockaddr_in fsin; /* the address of a client */
    int alen;                /* length of client's address */
    int msock;                 /* master server socket  */
    int ssock;                /* slave server socket */
    msock = passiveTCP(service, QLEN);
    signal(SIGCHLD, reaper);
 while (1)
 {
     alen = sizeof(fsin);
     ssock = accept(msock, (struct sockaddr *)&fsin, &alen);
     if(ssock < 0) {
        if(errno == EINTR)
            continue;
       errexit("accept: %s\n", strerror(errno));
  }
  switch (fork())
 {
     /* child */
     case 0:
       close(msock);
```

```
    exit(TCPechod(ssock));
    /* parent */
    default:
      close(ssock);
      break;
    case -1:
       errexit("fork: %s\n", strerror(errno));
  }
  }
}
int TCPechod(int fd)
{
   char buf[BUFSIZ];
   int  cc;
 while (cc = read(fd, buf, sizeof buf))
 {
       if(cc < 0)
          errexit("echo read: %s\n", strerror(errno));
       if(write(fd, buf, cc) < 0)
          errexit("echo write: %s\n", strerror(errno));
  }
   return 0;
}
void reaper(int sig)
{
   int status;
   while (wait3(&status, WNOHANG, (struct rusage *)0) >= 0)
   /* empty ;*/
}
```

An ECHO Server using a Single Process

```
int main(int argc, char *argv[ ])
{
   char *service = "echo";
   struct sockaddr_in fsin;
   int    msock;
   fd_set  rfds;
   fd_set  afds;
   int  alen;
   int fd, nfds;
   msock = passiveTCP(service, QLEN);

   nfds = getdtablesize();
   FD_ZERO(&afds);
   FD_SET(msock, &afds);
```

```
   while (1) {
      memcpy(&rfds, &afds, sizeof(rfds));
     if(select(nfds, &rfds, (fd_set *)0, (fd_set *)0, (struct timeval *)0) < 0)
         errexit("select: %s\n", strerror(errno));
      if(FD_ISSET(msock, &rfds))
{
   int ssock;
   alen = sizeof(fsin);
   ssock = accept(msock, (struct sockaddr *)&fsin, &alen);
   if(ssock < 0)
     errexit("accept: %s\n", strerror(errno));
    FD_SET(ssock, &afds);
  }
  for(fd=0; fd < nfds; ++fd)
    if(fd != msock && FD_ISSET(fd, &rfds))
      if(echo(fd) == 0)
        {
          (void) close(fd);
          FD_CLR(fd, &afds);
         }
   }
}
int echo(int fd)
{
   char buf[BUFSIZ];
   int cc;
   cc = read(fd, buf, sizeof buf);
   if(cc < 0)
      errexit("echo read: %s\n", strerror(errno));
   if(cc && write(fd, buf, cc) < 0)
      errexit("echo write: %s\n", strerror(errno));
   return cc;
}
```