NEW HORIZON
COLLEGE OF ENGINEERING
New Horizon Knowledge Park, Ring Road, Marathalli
Autonomous College Permanently Affiliated to VTU, Approved by AICTE & UGC
Accredited by NAAC with 'A' Grade, Accredited by NBA

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

LAB MANUAL

COMPUTER NETWORKS
20CSL67

Prepared by

Ms.Uma.N

Dr.Vinodha.K

Dr.Thirukumaran

Verified by

Dr.Pamela Vinitha

Approved by

Dr. Rajalakshmi B
HOD- CSE

COMPUTER NETWORKS LAB

Course Code : 20CSL67
L:T:P : 0:0:1.5
Exam Hours : 3

Credits : 1.5
CIE Marks : 25
SEE Marks : 25

Course Outcomes: At the end of the Course, the Student will be able to

CO1	Evaluate the functionalities of various protocols
CO2	Create the socket programming interface for client server programming
CO3	Design and develop efficient routing algorithms ,congestion algorithms
CO4	Implement the cryptographic algorithms for network security

Mapping of Course Outcomes to Program Outcomes

	PO1	PO2	PO3	PO4	PO5	PO6	PO7	PO8	PO9	PO10	PO11	PO12
CO1	3	3	3	3	2	-	-	-	-	-	-	-
CO2	3	3	3	3	-	-	-	-	1	3	-	-
CO3	3	3		3	-	-	-	-	-	3	-	-
CO4	3	3	3	3	-	-	3	-	3	3	-	3

Exp. No	Experiment	Hours	COs
1	Write a program for error detecting code-using CRC-CCITT (16-bits).	4	CO1
2	Write a program to implement Go/Back N and Selective repeat sliding window protocol.	3	CO1
3	Write a program for implementation of stop and wait.	3	CO1
4	Using TCP/IP Sockets, write a client-server program to make client sending the file name and the server to send back the contents of the requested file if present.	4	CO2
5	Design, develop, and execute a program in C under UNIX / LINUX environment to implement a simple echo server and demonstrate its working. Both the server and client are to be connectionless and use UDP. The system works as follows: Client reads a line from the standard input and writes the line to the server; the server reads a line from its network input and echoes the line back to the client; the client reads the echoed line and prints it on its standard output.	4	CO2
6	Write a program for Congestion control using the leaky bucket algorithm	4	CO3
7	Write a program for Distance Vector Algorithm to find suitable path for transmission.	4	CO3
8	Write a program for Link State Algorithm to find suitable path for transmission	4	CO3
9	Write a program for encryption and decryption using RSA algorithm.	3	CO4
10	Write a program to implement Diffie Hellman Key exchange.	3	CO4
11	a) Simulate Capturing and analyzing Ethernet	4	CO1

	frames. b) Simulate HTTP GET/POST interaction c) Simulate capturing a bulk TCP transfer from your computer to a remote server.		
12	Simulate a) Analysis of ICMP and PING messages b) Analysis of ICMP and Trace route	4	CO1

Reference Material(s):

1. Behrouz A. Forouzan: Data Communication and Networking, 5th Edition Tata McGraw-Hill, 2013. (M1,2,5)

2. Communication Networks – Fundamental Concepts & key architectures, Alberto Leon Garcia & Indra Widjaja, 2nd Edition, Tata McGraw-Hill, 2004, India. (M3)

3. Computer & Communication Networks, Nadir F Mir, Pearson Education, 2014, India. (M4,5)

CIE – Continuous Internal Evaluation: LAB (25 Marks)

Blooms Taxonomy	Tests
Marks (Out of 25)	25
L1: Remember	-
L2: Understand	05
L3: Apply	05
L4: Analyze	10
L5: Evaluate	05
L6: Create	-

SEE – Semester End Examination: LAB (25 Marks)

Blooms Taxonomy	Marks (Out of 25)
L1: Remember	-
L2: Understand	05
L3: Apply	05
L4: Analyze	10
L5: Evaluate	05
L6: Create	-

NEW HORIZON COLLEGE OF ENGINEERING, BANGALORE

Autonomous college permanently affiliated to VTU, Approved by AICTE & UGC
Accredited by NAAC with 'A' grade, Accredited by NBA

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

LAB RUBRICS

Course:COMPUTER NETWORKS LAB
Course Code:20CSL67
Sem: 6

Internal Assessment Marks: 25

Divided into two components:

1. Continuous Assessment: 10 marks
2. Internal Test: 15 marks

1. Continuous Assessment:

i) Will be carried out in every lab (for 12 labs -12 programs)

ii) Each program will be evaluated for 10 marks

iii) Totally for 12 lab programs it will be 120 marks. This will be scaled down to 10.

iv) During the semester, 2 internal tests will be conducted for 25 marks each. The total 50 marks for the internal tests, will be scaled down to 15.

Break up of 10 marks (in every lab):

Will be carried out in every lab (for 12 lab programs)

Attributes	Descriptors	Scores
Program Write-up(3)	Complete program with proper variable naming, proper commenting	3
	Complete program with not so proper variable naming, poor commenting	2
	Incomplete code	1
	Not written	0

Execution & Results (3)	Passes all specified test cases efficiently	3
	Fails in some test cases	2
	Incomplete execution	1
Viva Voce(2)	Answers correctly	2
	Answers satisfactorily	1
	Do not answer any question	0
Record completion and submission(2)	Submits in time and completed (during subsequent lab)	2
	Fails to submit the record in time / incomplete submission	0

2. **Internal Test:**

Break up of 25 marks (for each of the 2 internal tests) which is scaled down to 15marks after the conduction of 2 internal tests:

The 1st lab internal will comprise of the first 6 lab programs and the 2nd lab internal will comprise of the next 6 lab programs.

Attributes	Descriptors	Scores
Program Write-up(5)	Complete program with proper variable naming, proper commenting	5
	Complete program with not so proper variable naming, poor commenting	3-4
	Incomplete code	1-2
	Not written	0
Execution & Results (15)	Passes all specified test cases efficiently	15
	Fails in some test cases	10-14
	Incomplete execution	1-9
Viva Voce(5)	Answers 100% questions correctly	5
	Answers 75% questions correctly	3-4
	Answers satisfactorily	1-2
	Does not answer any question	0

SEE Assessment Marks: 25

Session End Examination is conducted for 50 marks which is scaled down to 25 marks.

Attributes	Descriptors	Scores
Program Write-up(10)	Complete program with proper variable naming, proper commenting	10
	Complete program with not so proper variable naming, poor commenting	8-9
	Incomplete code	4-5
	Not written	0
Execution & Results (30)	Passes all specified test cases efficiently	30
	Fails in some test cases	20-25
	Incomplete execution	18-20
Viva Voce(10)	Answers 100% questions correctly	10
	Answers 75% questions correctly	8
	Answers satisfactorily	4-5
	Does not answer any question	0

Lab Course Faculty

Course Coordinator

CSE62 COMPUTER NETWORKS LAB MANUAL

CONTENTS

	i) Analysis of ICMP and PING messages ii) Analysis of ICMP and Traceroute
	Sample VIVA Questions

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

```
#include<stdio.h>
#include<string.h>
intcrc(char *input, char *output, char *gp)
{
        inti, j;
        for(i=0; i<strlen(input); i++)
        if(output[i] == '1')
        for(j=0; j<strlen(gp); j++)
        {
                if (output[i+j]==gp[j])
                        output[i+j]='0';
                else
                        output[i+j]='1';
        }
        for(i=0; i<strlen(output); i++)
        if(output[i] == '1')
                return 1;
        return 0;
}
int main()
{
        char input[50],output[50];
        charrecv[50], gp[50];
        inti;
        printf("\n Enter the input message in binary\n");
        scanf("%s",input);
        printf("\n Enter the generator polynomial\n");
        scanf("%s",gp);
        strcpy(output, input);
        for(i=1; i<strlen(gp); i++)
                strcat(output,"0");
        crc(input,output,gp);
        printf("\n The transmitted message is %s %s\n",input, output+strlen (input));
        printf("\n\n Enter the received message in binary \n");
        scanf("%s",recv);
```

```
        if(crc(input,recv,gp))
                printf("\n Error in data transmission has occurred \n");
        else
                printf("\nNo error in data \n");
}
```

2. Write a program to implement Go-Back N and Selective repeat sliding window protocol.

```
#include<stdio.h>
#include<stdlib.h>
#include<math.h>
intn,r;
struct frame
{
Char ack;
int data;
}frm[10];
int sender(void);
voidrecvfrm(void);
void resend(void);
void resend1(void);
voidgoback(void);
void selective(void);
int main()
{
int c;
```

```
do
{
printf("\n\n1.Selective repeat ARQ\n2.Goback ARQ\n3.exit");
printf("\nEnterur choice:");
scanf("%d",&c);
switch(c)
{
case 1:selective();
break;
case 2:goback();
break;
case 3:exit(0);
break;
}
}while(c>=4);
}
Voidgoback()
{
sender();
recvfrm();
resend1();
printf("\n all packets sent successfully\n");
}
void selective()
{
sender();
recvfrm();
resend();
printf("\nAll packets sent successfully");
```

```
}
int sender()
{
inti;
printf("\nEnter the no. of packets to be sent:");
scanf("%d",&n);
for(i=1;i<=n;i++)
{
printf("\nEnter data for packets[%d]",i);
scanf("%d",&frm[i].data);
frm[i].ack='y';
}
return 0;
}
voidrecvfrm()
{
inti;
rand();
r=rand()%n;
frm[r].ack='n';
for(i=1;i<=n;i++)
{
if(frm[i].ack=='n')
printf("\nThe packet number %d is not received\n",r);
}
}
void resend()
{
printf("\nresending packet %d",r);
```

```
sleep(2);
frm[r].ack='y';
printf("\nThe received packet is %d",frm[r].data);
}
void resend1()
{
inti;
printf("\n resending from packet %d",r);
for(i=r;i<=n;i++)
{
sleep(2);
frm[i].ack='y';
printf("\nReceived data of packet %d is %d",i,frm[i].data);
}
}
```

3. Write a program for implementation of stop and wait.

```
#include <conio.h>
#include <dos.h>
#include <stdio.h>
#include <stdlib.h>
#define TIMEOUT 5
#define MAX_SEQ 1
#define TOT_PACKETS 8
#define inc(k) if(k<MAX_SEQ) k++; else k=0;
typedefstruct
{
int data;
}packet;
typedefstruct
{
int kind;
intseq;
intack;
packet info;
int err;
}frame;
```

```
frame DATA;
typedefenum{frame_arrival,err,timeout,no_event} event_type;

voidfrom_network_layer(packet *);
voidto_network_layer(packet *);
voidto_physical_layer(frame *);
voidfrom_physical_layer(frame *);
voidwait_for_event_sender(event_type *);
voidwait_for_event_reciever(event_type *);
voidreciever();
voidsender();

inti=1;        //Data to be sent by sender
char turn;      //r , s
int DISCONNECT=0;
/*_____________________________________________________________________*/
voidmain()
{
clrscr();
randomize();
while(!DISCONNECT)
{
sender();
delay(400);
reciever();
}
getch();
}
/*_____________________________________________________________________*/
voidsender()
{
staticintframe_to_send=0;
static frame s;
packet buffer;
event_type event;
staticint flag=0;

if(flag==0)
{
from_network_layer(&buffer);
s.info = buffer;
s.seq = frame_to_send;
printf("SENDER :  Info = %d    Seq No = %d     ",s.info,s.seq);
turn = 'r';
to_physical_layer(&s);
flag = 1;
}
wait_for_event_sender(&event);
if(turn=='s')
```

```
{
if(event==frame_arrival)
{
from_network_layer(&buffer);
inc(frame_to_send);
s.info = buffer;
s.seq = frame_to_send;
printf("SENDER :  Info = %d    Seq No = %d     ",s.info,s.seq);
turn = 'r';
to_physical_layer(&s);
}
if(event==timeout)
{
printf("SENDER : Resending Frame            ");
turn = 'r';
to_physical_layer(&s);
}
}
}
/*_____________________________________________________________________*/
voidreciever()
{
staticintframe_expected=0;
framer,s;
event_type event;

wait_for_event_reciever(&event);
if(turn=='r')
{
if(event==frame_arrival)
{
from_physical_layer(&r);
if(r.seq==frame_expected)
{
to_network_layer(&r.info);
inc(frame_expected);
}
else
printf("RECIEVER : Acknowledgement Resent\n");

turn = 's';
to_physical_layer(&s);
}
if(event==err)
{
printf("RECIEVER : Garbled Frame\n");
turn = 's';     //if frame not recieved
}               //sender shold send it again
}
```

```
}
/*_____________________________________________________________________*/
voidfrom_network_layer(packet *buffer)
{
(*buffer).data = i;
i++;
}
/*_____________________________________________________________________*/
voidto_physical_layer(frame *s)
{                   // 0 means error
s->err = random(4);  //non zero means no error
DATA = *s;          //probability of error = 1/4
}
/*_____________________________________________________________________*/
voidto_network_layer(packet *buffer)
{
printf("RECIEVER :Packet %d recieved , Ack Sent\n",(*buffer).data);
if(i>TOT_PACKETS)          //if all packets recieved then disconnect
{
DISCONNECT = 1;
printf("\nDISCONNECTED");
}
}
/*_____________________________________________________________________*/
voidfrom_physical_layer(frame *buffer)
{
*buffer = DATA;
}
/*_____________________________________________________________________*/
voidwait_for_event_sender(event_type * e)
{
staticint timer=0;

if(turn=='s')
{
timer++;
if(timer==TIMEOUT)
{
*e = timeout;
printf("SENDER : Ack not recieved=> TIMEOUT\n");
timer = 0;
return;
}
if(DATA.err==0)
*e = err;
else
{
timer = 0;
*e = frame_arrival;
```

```
}
}
}
/*_____________________________________________________________________*/
voidwait_for_event_reciever(event_type * e)
{
if(turn=='r')
{
if(DATA.err==0)
*e = err;
else
*e = frame_arrival;
}
}
```

4. Using TCP/IP Sockets, write a client-server program to make client sending the file name and the server to send back the contents of the requested file if present.

Header Filename: inet.h

```
#include<stdio.h>
#include<sys/types.h>
#include<sys/socket.h>
#include<netinet/in.h>
#include<arpa/inet.h>
#include<fcntl.h>
#include<string.h>
#include<stdlib.h>
#define SERV_TCP_PORT 6880
#define SERV_HOST_ADDR "127.0.0.1"
```

Program-Name: Client.c

```
#include"inet.h"

int main()
{
intsockfd;
structsockaddr_inserv_addr, cli_addr;
char filename[100],buf[1000];
int n;
serv_addr.sin_family=AF_INET;
serv_addr.sin_addr.s_addr=inet_addr(SERV_HOST_ADDR);
serv_addr.sin_port=htons(SERV_TCP_PORT);
if((sockfd=(socket(AF_INET,SOCK_STREAM,0)))<0)
```

```
{
printf("Client: can’t open stream socket\n");
exit(0);
}
else
printf("Client: stream socket opened successfully\n");
if(connect(sockfd,(structsockaddr *)&serv_addr, sizeof(serv_addr))<0)
{
printf("Client: cant connect to server\n");
exit(0);
}

else
printf("Client: connected to server successfully\n");
printf("\n Enter the file name to be displayed :");
scanf("%s",filename);
write(sockfd, filename, strlen(filename));
printf("\n filename transferred to server\n");
n=read(sockfd,buf,1000);
buf[n]=’\0’;
printf("\n Client : Displaying file content of %s\n", filename);
puts(buf);
close(sockfd);
exit(0);
}
```

Program-Name: Server.c

```
#include"inet.h"
intmai n()
{
intsockfd,newsockfd,clilen;
structsockaddr_incli_addr,serv_addr;
char filename[25],buf[1000];
intn,m,fd,size;
serv_addr.sin_family=AF_INET;
serv_addr.sin_addr.s_addr=htonl(INADDR_ANY);
serv_addr.sin_port=htons(SERV_TCP_PORT);
if((sockfd=(socket(AF_INET,SOCK_STREAM,0)))<0)
{
printf("Server: can’t open stream socket\n");
exit(0);
}
else
printf("Server: stream socket opened successfully\n");
if((bind(sockfd,(structsockaddr *) &serv_addr, sizeof(serv_addr)))<0)
{
printf("Server:cant bind local address\n");
exit(0);
```

```
}
else
printf("Server: bind to local address\n");
listen(sockfd,5);
printf("\n SERVER : Waiting for client...\n");
clilen=sizeof(cli_addr);
newsockfd=accept(sockfd,(structsockaddr*)&cli_addr,&clilen);


if(newsockfd<0)
{
printf("server:accept error\n");
exit(0);
}
else
printf("Server: accepted\n");
n=read(newsockfd,filename,25);
filename[n]='\0';
printf("\n SERVER : %s is found and ready to transfer \n",filename);
fd=open(filename,O_RDONLY);
if(fd==-1)
{
write(newsockfd,"File doesn't exists",25);
exit(0);
}
size=lseek(fd,0,2);
lseek(fd,0,0);
n=read(fd,buf,size);
buf[n]='\0';
write(newsockfd,buf,n);
printf("\n transfer success\n");
puts(buf);
close(newsockfd);
exit(0);
}
```

5. Design, develop, and execute a program in C under UNIX / LINUX environment to implement a simple echo server and demonstrate its working. Both the server and client are to be connectionless and use UDP. The system works as follows: Client reads a line from the standard input and writes the line to the server; the server reads a line from its network input and echoes the line back to the client; the client reads the echoed line and prints it on its standard output.

Server.c

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

```
#include <stdlib.h>

int main(){
intudpSocket, nBytes;
char buffer[1024];
structsockaddr_inserverAddr, clientAddr;
structsockaddr_storageserverStorage;
socklen_taddr_size, client_addr_size;
int i;

/*Create UDP socket*/
udpSocket = socket(PF_INET, SOCK_DGRAM, 0);

/*Configure settings in address struct*/
serverAddr.sin_family = AF_INET;
serverAddr.sin_port = htons(7891);
serverAddr.sin_addr.s_addr = inet_addr("127.0.0.1");
memset(serverAddr.sin_zero, '\0', sizeofserverAddr.sin_zero);

/*Bind socket with address struct*/
bind(udpSocket, (structsockaddr *) &serverAddr, sizeof(serverAddr));

/*Initialize size variable to be used later on*/
addr_size = sizeofserverStorage;

while(1){
/* Try to receive any incoming UDP datagram. Address and port of
requesting client will be stored on serverStorage variable */
nBytes = recvfrom(udpSocket,buffer,1024,0,(structsockaddr *)&serverStorage, &addr_size);

/*Convert message received to uppercase*/
for(i=0;i<nBytes-1;i++)
buffer[i] = toupper(buffer[i]);

/*Send uppercase message back to client, using serverStorage as the address*/
sendto(udpSocket,buffer,nBytes,0,(structsockaddr *)&serverStorage,addr_size);
}
return 0;
}
```

Client.c

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

int main(){
int clientSocket, portNum, nBytes;
char buffer[1024];
structsockaddr_inserverAddr;
```

```
socklen_taddr_size;

/*Create UDP socket*/
clientSocket = socket(PF_INET, SOCK_DGRAM, 0);

/*Configure settings in address struct*/
serverAddr.sin_family = AF_INET;
serverAddr.sin_port = htons(7891);
serverAddr.sin_addr.s_addr = inet_addr("127.0.0.1");
memset(serverAddr.sin_zero, '\0', sizeofserverAddr.sin_zero);

/*Initialize size variable to be used later on*/
addr_size = sizeofserverAddr;

while(1){
printf("Type a sentence to send to server:\n");
fgets(buffer,1024,stdin);
printf("You typed: %s",buffer);

nBytes = strlen(buffer) + 1;

/*Send message to server*/
sendto(clientSocket,buffer,nBytes,0,(structsockaddr *)&serverAddr,addr_size);

/*Receive message from server*/
nBytes = recvfrom(clientSocket,buffer,1024,0,NULL, NULL);

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

}

return 0;
}
```

6.Write a program for Congestion control using the leaky bucket algorithm

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

int min(int x, int y)
{
        if(x<y)
                return x;
```

```
        else
                return y;
}
int main()
{
        int drop=0,mini,nsec,cap,count=0,i,inp[25],process;
        system("clear");
        printf("Enter The Bucket Size\n");
        scanf("%d",&cap);
        printf("Enter The Processing Rate\n");
        scanf("%d",&process);
        printf("Enter The No. Of Seconds Packets are arriving\n");
        scanf("%d",&nsec);
        for(i=1;i<=nsec;i++)
        {
                printf("Enter Number of packets entering at %d sec\n",i);
                scanf("%d",&inp[i]);
        }
        printf("\nSecond|PacketsRecieved|PacketsSent|PacketsLeft|P
        ackets Dropped|\n");
        printf("-----------------------------------------------------------------------------
        ---\n");
        for(i=1;i<=nsec;i++)
        {
                count=count+inp[i];
                if(count>cap)
                {
                        drop=count-cap;
                        count=cap;
                }
                printf("%d",i);
                printf("\t%d",inp[i]);
                mini=min(count,process);
                printf("\t\t%d",mini);
                count=count-mini;
                printf("\t\t%d",count);
                printf("\t\t%d\n",drop);
```

```
            drop=0;
        }
        for(;count!=0;i++)
        {
            if(count>cap)
            {
            drop=count-cap;
            count=cap;
            }
            printf("%d",i);
            printf("\t0");
            mini=min(count,process);
            printf("\t\t%d",mini);
            count=count-mini;
            printf("\t\t%d",count);
            printf("\t\t%d\n",drop);
        }
}
```

7.Write a program for Distance Vector Algorithm to find suitable path for transmission.

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

int main()
{
        int copy[20][20], dist[20][20], via[20][20];
        int n=0, i=0, j=0, k=0,count=0;
        system("clear");
        printf("Enter The Number Of Nodes\n");
        scanf("%d",&n);
        printf("Enter The Cost Matrix\n");
        for(i=1;i<=n;i++)
        for(j=1;j<=n;j++)
        {
            scanf("%d",&dist[i][j]);
```

```
            dist[i][i]=0;
            copy[i][j]=dist[i][j];
            via[i][j]=j;
        }

        do
        {
            count=0;
            for(i=1;i<=n;i++)
            for(j=1;j<=n;j++)
            for(k=1;k<=n;k++)
            if(copy[i][j]>(dist[i][k]+copy[k][j]))
            {
                copy[i][j]=dist[i][k]+copy[k][j];
                via[i][j]=k;
                count++;
            }
        }while (count!=0);
        for(i=1;i<=n;i++)
        {
            printf("Shortest path from Router %d \n",i);
            for(j=1;j<=n;j++)
            {
                printf("\t\t To %d Via %d Is %d\n", j, via[i][j], copy[i][j]);
            }
        }
}
```

8.Write a program for Link State Algorithm to find suitable path for transmission.

```
#include <stdio.h>
#include <conio.h>
#define infinity 999

void dij(intn,intv,int cost[10][10],intdist[])
{
inti,u,count,w,flag[10],min;
```

```
 for(i=1;i<=n;i++)
  flag[i]=0,dist[i]=cost[v][i];
 count=2;
 while(count<=n)
 {
  min=99;
  for(w=1;w<=n;w++)
   if(dist[w]<min && !flag[w])
    min=dist[w],u=w;
  flag[u]=1;
  count++;
  for(w=1;w<=n;w++)
   if((dist[u]+cost[u][w]<dist[w]) && !flag[w])
dist[w]=dist[u]+cost[u][w];
 }
}

void main()
{
intn,v,i,j,cost[10][10],dist[10];
clrscr();
printf("n Enter the number of nodes:");
scanf("%d",&n);
printf("n Enter the cost matrix:n");
 for(i=1;i<=n;i++)
  for(j=1;j<=n;j++)
  {
scanf("%d",&cost[i][j]);
   if(cost[i][j]==0)
    cost[i][j]=infinity;
  }
printf("n Enter the source matrix:");
scanf("%d",&v);
dij(n,v,cost,dist);
printf("n Shortest path:n");
 for(i=1;i<=n;i++)
  if(i!=v)
printf("%d->%d,cost=%dn",v,i,dist[i]);
getch();
}
```

9.Write a program for encryption and decryption using RSA algorithm.

```
#include<stdio.h>
int phi, M, n, e, d, C;
void encrypt()
{
Inti;
C = 1;
```

```
for(i=0;i<e;i++)
C=C*M%n;
C = C%n;
printf("\n\tEncrypted keyword : %d",C);
}
void decrypt()
{
inti;
M = 1;
for(i=0;i<d;i++)
M=M*C%n;
M = M%n;
printf("\n\tDecrypted keyword : %d",M);
}
int main()
{
Inti, p,q,rem;
printf("\nEnter Two Distinct Prime Numbers\t: ");
scanf("%d%d",&p,&q);
n = p*q;
phi=(p-1)*(q-1);
printf("\n\tPhi\t= %d",phi);
printf("\n\nEnter e coprime with Phi(1<e<%d)\t: ",phi);
scanf("%d",&e);
if((e<phi)&&(e>1))
{
        for(i=2;i<phi;i++)

        if(e%i==0&&phi%i==0)
        {
                printf("\nInvalid value of e\n");
                exit(0);
        }

        else
        {
        d =0;

        do
        {
                d++;
                rem = (d*e)%phi;
        }while(rem!=1);
        }
}
else
{
        printf("\nInvalid value of e\n");
        exit(0);
```

```
}

printf("\n\tPublic Key\t:{%d,%d}",e,n);
printf("\n\tPrivate Key\t:{%d,%d}",d,n);
printf("\n\nEnter The Plain Text(0<PT<%d)\t: ",n);
scanf("%d",&M);
if(M<n)
{
encrypt();
decrypt();
}
else
        printf("\nInvalid Plain text\n") ;

}
```

10.Write a program to implement Diffie Hellman Key exchange.

```
/* This program calculates the Key for two persons

using the Diffie-Hellman Key exchange algorithm */
#include<stdio.h>
#include<math.h>

// Power function to return value of a ^ b mod P
longintpower(longinta, longintb,
                        longintP)
{
   if(b == 1)
      return a;
   else
      return(((longint)pow(a, b)) % P);
}
//Driver program
Intmain()
{
   LongintP, G, x, a, y, b, ka, kb;

   // Both the persons will be agreed upon the
      // public keys G and P
   P = 23; // A prime number P is taken
   printf("The value of P : %lld\n", P);
   G = 9; // A primitve root for P, G is taken
   printf("The value of G : %lld\n\n", G);
   // Alice will choose the private key a
```

```
    a = 4; // a is the chosen private key
    printf("The private key a for Alice : %lld\n", a);
    x = power(G, a, P); // gets the generated key

    // Bob will choose the private key b
    b = 3; // b is the chosen private key
    printf("The private key b for Bob : %lld\n\n", b);
    y = power(G, b, P); // gets the generated key
    // Generating the secret key after the exchange
       // of keys
    ka = power(y, a, P); // Secret key for Alice
    kb = power(x, b, P); // Secret key for Bob

    printf("Secret key for the Alice is : %lld\n", ka);
    printf("Secret Key for the Bob is : %lld\n", kb);

    return0;
}
```

11a)Simulate Capturing and analysing Ethernet frames.

1. Review the Ethernet II header field descriptions and lengths.

Preamble	Destination Address	Source Address	Frame Type	Data	FCS
8 Bytes	6 Bytes	6 Bytes	2 Bytes	46 – 1500 Bytes	4 Bytes

2. Examine the network configuration of the PC.

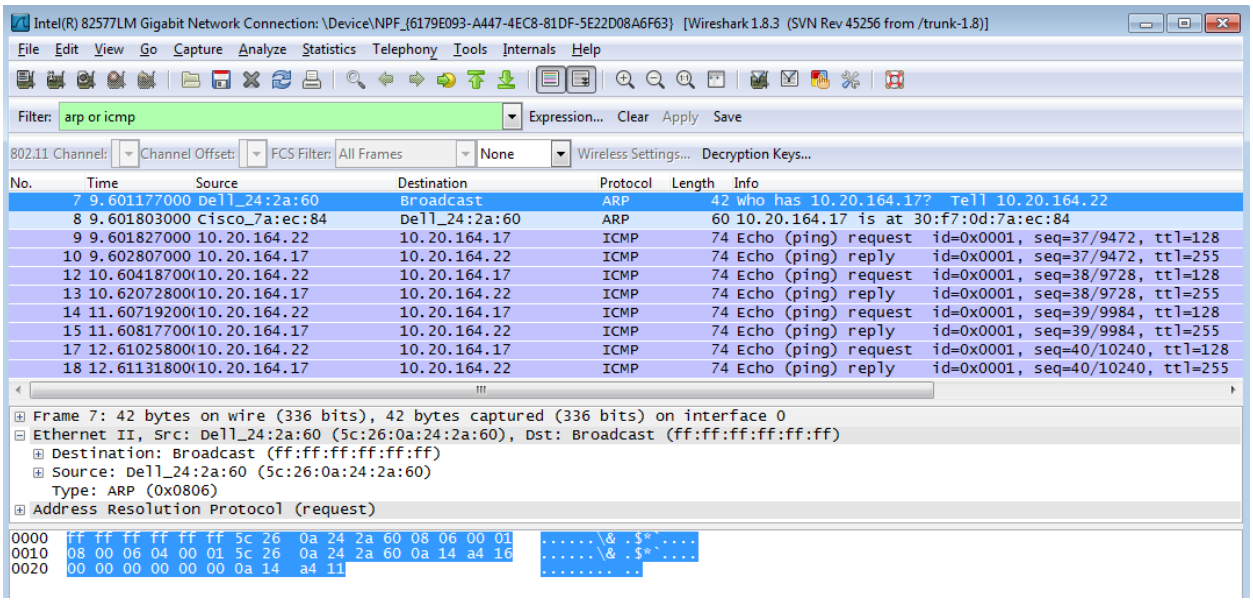
This PC host IP address is 10.20.164.22 and the default gateway has an IP address of 10.20.164.17.

```
Ethernet adapter Local Area Connection:

   Connection-specific DNS Suffix  . : cisco.com
   Link-local IPv6 Address . . . . . : fe80::b875:731b:3c7b:c0b1%10
   IPv4 Address. . . . . . . . . . . : 10.20.164.22
   Subnet Mask . . . . . . . . . . . : 255.255.255.240
   Default Gateway . . . . . . . . . : 10.20.164.17
```

3. Examine Ethernet frames in a Wireshark capture.

The Wireshark capture below shows the packets generated by a ping being issued from a PC host to its default gateway. A filter has been applied to Wireshark to view the ARP and ICMP protocols only. The session begins with an ARP query for the MAC address of the gateway router, followed by four ping requests and replies.



4. Examine the Ethernet II header contents of an ARP request.

The following table takes the first frame in the Wireshark capture and displays the data in the Ethernet II header fields.

Field	Value	Description
Preamble	Not shown in capture	This field contains synchronizing bits, processed by the NIC hardware.
Destination Address	Broadcast (ff:ff:ff:ff:ff:ff)	Layer 2 addresses for the frame. Each address is 48 bits long, or 6 octets, expressed as 12 hexadecimal digits, 0-9,A-F. A common format is 12:34:56:78:9A:BC. The first six hex numbers indicate the manufacturer of the network interface card (NIC), the last six hex numbers are the serial number of the NIC. The destination address may be a broadcast, which contains all ones, or a unicast. The source address is always unicast.
Source Address	Dell_24:2a:60 (5c:26:0a:24:2a:60)	
Frame Type	0x0806	For Ethernet II frames, this field contains a hexadecimal value that is used to indicate the type of upper-layer protocol in the data field. There are numerous upper-layer protocols supported by Ethernet II. Two common frame types are: Value Description 0x0800 IPv4 Protocol 0x0806 Address resolution protocol (ARP)

Data	ARP	Contains the encapsulated upper-level protocol. The data field is between 46 – 1,500 bytes.
FCS	Not shown in capture	Frame Check Sequence, used by the NIC to identify errors during transmission. The value is computed by the sending machine, encompassing frame addresses, type, and data field. It is verified by the receiver.

What is significant about the contents of the destination address field?

All host on the LAN will receive this broadcast frame, the host with the IP address of 192.168.1.1 (default gateway) will send a unicast reply to the source (PC host). This reply contains the Mac address of the NIC of the Default Gateway.

Why does the PC send out a broadcast ARP prior to sending the first ping request? Before the PC can send a ping request to a host ,it needs to determine the destination MAC address before it can build the frame header for that ping request. The ARP broadcast is used to request the MAC address of the host with the IP address contained in the ARP.

- What is the MAC address of the source in the first frame?(2c:60:48:f2:28)
- What is the Vendor ID (OUI) of the Source's NIC? __Quantaco 48:f2:28
- What portion of the MAC address is the OUI?

The first 3 octets, or 6 hexadecimal numbers, or 24 bits of the MAC address indicate the OUI

- What is the Source's NIC serial number? (2c:60:0c:48:f2:28)
  (Use Wireshark to Capture and Analyze Ethernet Frames

In Part 2, you will use Wireshark to capture local and remote Ethernet frames. You will then examine the information that is contained in the frame header fields.

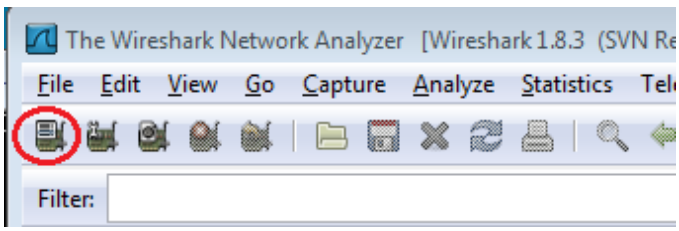
5. **Determine the IP address of the default gateway on your PC.**

Open a command prompt window and issue the **ipconfig** command.

What is the IP Address of the PC Default Gateway? 192.168.1.1

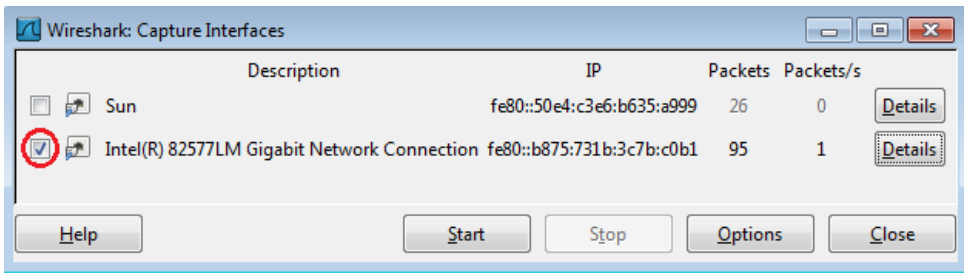
6. **Start capturing traffic on your PC's NIC.**

   1. Open Wireshark.
   2. On the Wireshark Network Analyzer toolbar, click the **Interface List** icon.

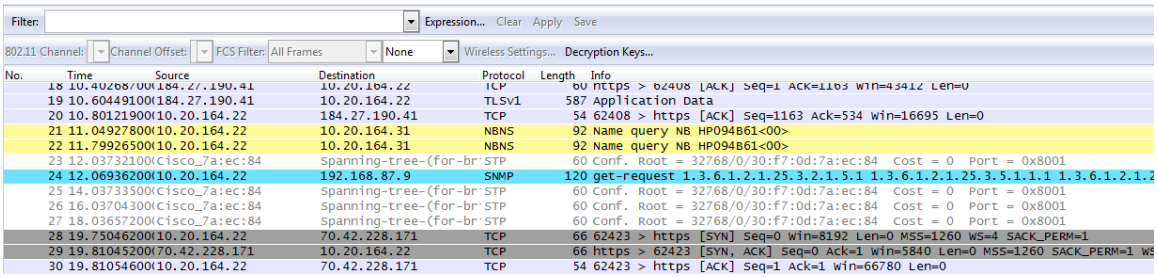


   3. On the Wireshark: Capture Interfaces window, select the interface to start

traffic capturing by clicking the appropriate check box, and then click **Start**. If you are uncertain of what interface to check, click **Details** for more information about each interface listed.



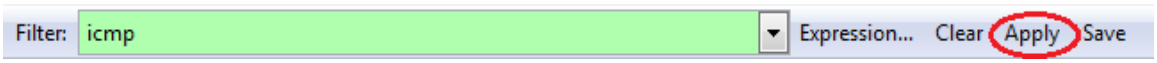
4. Observe the traffic that appears in the Packet List window.



7. **Filter Wireshark to display only ICMP traffic.**

You can use the filter in Wireshark to block visibility of unwanted traffic. The filter does not block the capture of unwanted data; it only filters what to display on the screen. For now, only ICMP traffic is to be displayed.

In the Wireshark **Filter** box, type **icmp**. The box should turn green if you typed the filter correctly. If the box is green, click **Apply** to apply the filter.

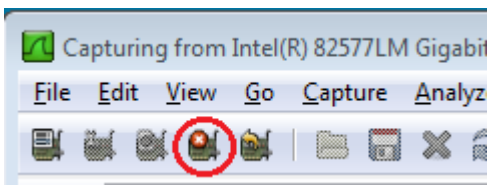


8. **From the command prompt window, ping the default gateway of your PC.**

From the command window, ping the default gateway using the IP address that you recorded in Step 1.

9. **Stop capturing traffic on the NIC.**

Click the **Stop Capture** icon to stop capturing traffic.



10. **Examine the first Echo (ping) request in Wireshark.**

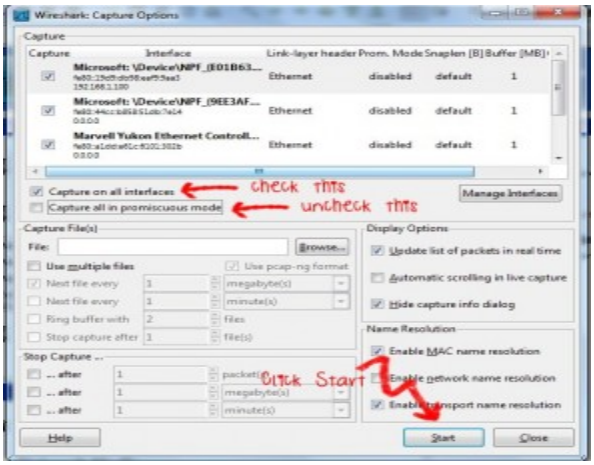
The Wireshark main window is divided into three sections: the Packet List pane (top),

the Packet Details pane (middle), and the Packet Bytes pane (bottom). If you selected the correct interface for packet capturing in Step 3, Wireshark should display the ICMP information in the Packet List pane of Wireshark, similar to the following example.

11b)Simulate HTTP GET/POST interaction

You will now need to configure the capture options. Follow these steps :

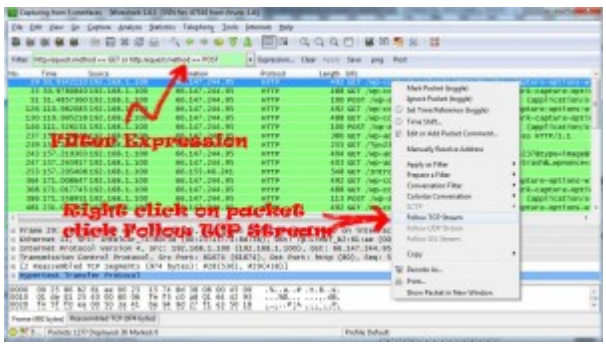
1) Check Capture on all interfaces
2) Uncheck Capture all in promiscuous mode
3) Click Start



Immediately the packets start getting captured and you can view them in the Wireshark window. Observe the protocol of the packets, it tells us what protocol is being used to transfer the packet. This helps us filter out only those packets that we need and leave the rest. Now as we need to find the GET and POST packets (which follow the HTTP protocol) we need to set an appropriate filter for it.

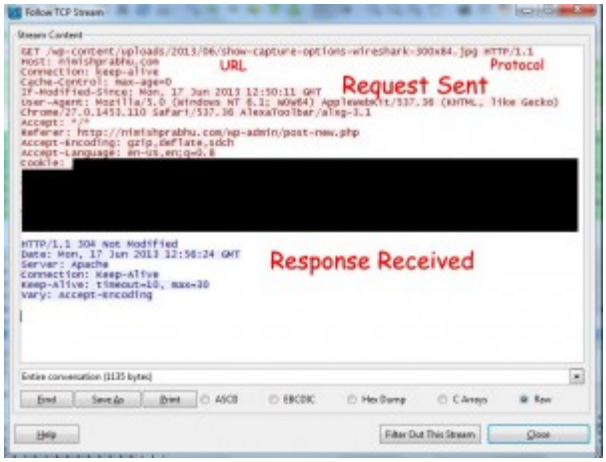
Notice the filter text box. Paste the following expression in it, **http.request.method == GET or http.request.method == POST** and hit enter. You can save it for future use as well, so that you don't need to remember it everytime you wish to filter packets.

Here's how the final result looks like:



Right click on the packet which you wish to analyze and click on "Follow TCP Stream". A new window will open with all the details of data sent and received. To be more accurate,

the “request” sent and the “response” received. Here’s how to read the details from the new window.



I will just try to explain the packet that I chose to analyze in this example. The one you choose might be completely different but the basics remain the same. Observe the following details in the text that is visible.

1) GET indicates the method used (GET or POST)
2) URL indicates the URL to which the request is being sent.
3) The protocol in this case will obviously be HTTP. It also shows the version, i.e. 1.1
4) If-Modified-Since is one of the header messages, it indicates that the request is just to check if the URL is modified since the time specified. (As you can see, in Response message, all we get back is “Not Modified”)
5) User-Agent contains information about the browser used.
6) Referer indicates the URL from which the request was referred.
7) Accept-Encoding is also one of the header message which indicates the different encoding methods that can be decoded by the browser from which the request is being sent.
8) Cookie, contains the data that is being stored in cookies of your current browser.

We can read the response in a similar manner :

1) HTTP/1.1 indicates the protocol/version used.
2) 304 is the status code for “Not Modified”. You can find all HTTP status codes on w3.org page.
3) Date indicates the time during which the response was generated.

11c)Simulate capturing a bulk TCP transfer from your computer to a remote server.

In order to perform an exploration of TCP, you’ll need to use Wireshark to obtain a packet trace of the TCP transfer of a large file from yourcomputer to a remote server. You’ll do so by accessing a Web page that will allow you to enter the name of a file stored on your computer (which contains the ASCII text of Alice in Wonderland), and then transfer the file to a Web server using the HTTP POST.You’re going to use the POST method rather than theGET method to transfer a large amount of data from your computer to another

computer. Of course, you'll be running Wireshark during this time to obtain the trace of the TCP segments sent and received from your computer.
Do the following:
•Start up your web browser.
•Go to:
http://gaia.cs.umass.edu/wireshark-labs/alice.txt
Retrieve an ASCII copy of Alice in Wonderland. Store this file somewhere on your computer.
•Open the file alice.txtusing the editing programNotepadcopy the whole text andappend it to (paste after) the existing content sothat the file will be twice as large as the original downloaded file. Save the file withoutchanging the name.
•Next go to:http://gaia.cs.umass.edu/wireshark-labs/TCP-wiresha
rk-file1.html
•You should see a screen (on next page) that looks like: .
"Tell me and I forget. Show me and I remember. Involve me and I understand." Chinese proverb.
Analyze a trace of the TCP segments sent and received in transferring a large file (containing the textof Lewis Carrol's Alice's Adventures in Wonderland)from your computer to a remote server. You will investigate the behavior of the celebratedTCP protocol in detail.You'll study TCP's use of sequence andacknowledgement numbers for providing reliable datatransfer; you'll see TCP's congestion control algorithm, slow start and probably congestion avoidance in action; and you'll look at TCP's receiver-advertised flow controlmechanism. You'll also briefly consider TCP connection setup and investigate the performance (throughput andround-trip time) of the TCP connection between yourcomputer and a remote server.
•Use theBrowsebutton in this form to enter the name of the file (full path name) on your computer containingAlice in Wonderland(or do so manually). Don't yet press the "Upload alice.txt file" button.
•Now start up Wireshark and begin packet capture with "tcp" as filter.
•Returning to your browser, press the "Upload alice.txt file" button to upload thefile to the gaia.cs.umass.eduserver. Once the file has been uploaded, a short congratulations message will be displayed in your browser window.
•Stop Wireshark packet capture

12.Simulate

a)Analysis of ICMP and PING messages

To capture ICMP Echo traffic:

1) Start a Wireshark capture.
2) Use ping <default gateway address> to ping the default gateway address.
3) Stop the Wireshark capture.

To analyze ICMP Echo Request traffic:

1. Observe the traffic captured in the top Wireshark packet list pane. Look for traffic with ICMP listed as the protocol. To view only ICMP traffic, type **icmp** (lower case) in the Filter box and press **Enter.**
2. Select the first ICMP packet, labeled**Echo (ping) request**.

3. Observe the packet details in the middle Wireshark packet details pane. Notice that it is an Ethernet II / Internet Protocol Version 4 / Internet Control Message Protocol frame.
4. Expand Internet Control Message Protocol to view ICMP details.
5. Observe the Type. Notice that the type is 8 (Echo (ping) request).
6. Select Data in the middle Wireshark packet details pane to highlight the data portion of the frame.
7. Observe the packet contents in the bottom Wireshark packet bytes pane. Notice that Windows sends an alphabet sequence during ping requests.

To analyze ICMP Echo Reply traffic:

1. In the top Wireshark packet list pane, select the second ICMP packet, labeled**Echo (ping) reply**.
2. Observe the packet details in the middle Wireshark packet details pane. Notice that it is an Ethernet II / Internet Protocol Version 4 / Internet Control Message Protocol frame.
3. Expand Internet Control Message Protocol to view ICMP details.
4. Observe the Type. Notice that the type is 0 (Echo (ping) reply).
5. Select Data in the middle Wireshark packet details pane to highlight the data portion of the frame.
6. Observe the packet contents in the bottom Wireshark packet bytes pane. Notice that the reply echoes the request sequence.
7. Close Wireshark to complete this activity. **Quit without Saving** to discard the captured traffic.

b)Analysis of ICMP and Traceroute

To capture ICMP tracert traffic:

- Start a Wireshark capture.
- Open a command prompt.
- Type **tracert -d 8.8.8.8** and press **Enter** to trace the route to one of Google's public DNS servers. The -d option prevents DNS name resolution, which in this case will improve performance and reduce the amount of captured traffic.
- When the trace is complete, close the command prompt.
- Stop the Wireshark capture.

To analyze tracert traffic:

1. Observe the traffic captured in the top Wireshark packet list pane. Look for traffic with ICMP listed as the protocol. To view only ICMP traffic, type **icmp** (lower case) in the Filter box and press **Enter.**
2. Select the first ICMP packet, labeled **Echo (ping) request**.
3. Observe the packet details in the middle Wireshark packet details pane. Notice that it is an Ethernet II / Internet Protocol Version 4 / Internet Control Message Protocol frame.
4. Expand Internet Protocol Version 4 to view IPv4 details.
5. Observe the Time to live. Notice that the time to live is set to 1.

6. Expand Internet Control Message Protocol to view ICMP details.
7. Observe the Type. Notice that the type is 8 (Echo (ping) request). Tracert is performed through a series of ICMP Echo requests, varying the Time-To-Live (TTL) until the destination is found.
8. In the top Wireshark packet list pane, select the second ICMP packet, labeled **Time-to-live exceeded**.
9. Observe the packet details in the middle Wireshark packet details pane. Notice that it is an Ethernet II / Internet Protocol Version 4 / Internet Control Message Protocol frame.
10. Expand Internet Protocol Version 4 to view IPv4 details.
11. Observe the Source. This is the IP address of the router where the time was exceeded.
12. Expand Internet Control Message Protocol to view ICMP details.
13. Observe the Type. Notice that the type is 11 (Time-to-live exceeded).
14. Observe the Code. Notice that the code is 0 (Time to live exceeded in transit).
15. Observe the fields that follow. Notice that the contents of the request packet are returned with the time exceeded error.
16. Continue selecting alternate ICMP Echo Request and ICMP Time-To-Live Exceeded packets. Notice that the request is repeated three times for each time-to-live count, and each reply indicates the IP address of the router where the time to live was exceeded.
17. Close Wireshark to complete this activity. **Quit without Saving** to discard the captured traffic.

SAMPLE VIVA-VOCE QUESTIONS:

1) Distinguish between Straight-wired cable and Cross-wired cable.

2) What are the various types of network cables?

3) What is RJ-45?

4) What is the advantage of using Twisted pair cable and what are its types?

5) Define Transmission media. What are the various transmission media?

6) What are the various networking devices available?

7) In which layer of OSI model following devices are implemented?

1) Repeater
2) Bridge
3) Switch
4) Router

8) Why a repeater is used?

9) Distinguish between Bridge and Router.

10) What is the advantage of gateways in networking?

11) What is a cyclic code?

12) State the advantages of using CRC.

13) In CRC, show the relationship between the following entities (size means the number of bits):

1) The size of the dataword and the size of the codeword
2) The size of the divisor and the remainder

14) Can CRC correct error? Justify your answer.

15) Why Checksum is used?

16) What is the function of Serversocketclass.
17) Define a Socket.
18) What are the methods of Serversocketclass.
19) What are the information that the client must know in socket programming.
20) Describe the Client-Server model.
21) Define Sliding window concept.
22) What is the size of send window and receive window.
23) What are the variables in send window and its functions.
24) Receive window slides only one slot at a time. Justify.
25) Name two protocols where sliding window concept is implemented.
26) What is the benefit of using Go-Back-N Automatic Repeat Request protocol.
27) Size of send window in Go-Back-N Automatic Repeat Request must be less than 2^m. Justify.
28) Differentiate between Go-Back-N ARQ and Stop-and- Wait ARQ.
29) Name one demerit of Go-Back-N ARQ protocol.
30) Why the name of this protocol is Go-Back-N ARQ.