Course Code: P13ISL68	Semester : VI	L- T – P : 0 – 0 - 3	Credit: 1.5
Course Title: Networking Laboratory			
Contact period : Lecture : 39 Hrs, Exam: 3Hrs		Weightage: CIE:50%; SEE:50%	

Contents

PART - A

The following experiments shall be conducted using C/C++

1. Write a program for error detection using CRC-CCITT (16-bits).
2. Write a program for frame sorting technique used in buffers.
3. Write a program to generate Hamming Code for error detection and correction.
4. Write a program for congestion control using Leaky bucket algorithm.
5. Write a program for simple RSA algorithm to encrypt and decrypt the data.
6. 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.
7. Implement the above program using FIFOs as IPC channels.

PART - B

The following experiments shall be conducted using QualNet simulator

1. Design a star topology for LAN using hub or switch or both by taking at least five nodes and analyze the statistics.
2. Design a mesh topology for five nodes and analyze the TCP, UDP statistics and FIFO queuing mechanism.
3. Develop a scenario for wired network and implement VOIP and analyze the statistics.
4. Design a scenario to integrate wired network and wireless network using Wi-Fi infrastructure mode and analyze the statistics.
5. Develop a scenario for WIMAX using standard 802.16. Analyze the statistics.
6. Design a scenario for wireless sensor networks (total 9 devices). Analyze the Zigbee battery model along with the statistics.
7. Develop a scenario for MANET in wireless networks. Consider 7 devices and show the mobility for all the devices and analyze the statistics.

Note: Student is required to solve one problem from PART - A and one problem from PART - B.

Course Outcomes

The student is able to

CO1: Write C/C++ code to implement network related concepts.

CO2: Design/Develop network scenarios and analyze its performance by specifying parameters using QualNet simulator.

PART - A Programs

Experiment No 1

CRC

Problem Statement

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

Theory

It does error checking via polynomial division. In general, a bit string

$$b_{n-1}b_{n-2}b_{n-3}\ldots b_2b_1b_0$$

As

$$b_{n-1}X^{n-1} + b_{n-2}\,X^{n-2} + b_{n-3}\,X^{n-3} + \ldots b_2\,X^2 + b_1\,X^1 + b_0$$

Ex: -

10010101110

As

$$X^{10} + X^7 + X^5 + X^3 + X^2 + X^1$$

All computations are done in modulo 2

Algorithm:-

1. Given a bit string, append 0^S to the end of it (the number of 0^s is the same as the degree of the generator polynomial) let B(x) be the polynomial corresponding to B.
2. Divide B(x) by some agreed on polynomial G(x) (generator polynomial) and determine the remainder R(x). This division is to be done using Modulo 2 Division.
3. Define T(x) = B(x) – R(x)

   (T(x)/G(x) => remainder 0)

4. Transmit T, the bit string corresponding to T(x).
5. Let T' represent the bit stream the receiver gets and T'(x) the associated polynomial. The receiver divides $T^1(x)$ by G(x). If there is a 0 remainder, the receiver concludes T = T' and no error occurred otherwise, the receiver concludes an error occurred and requires a retransmission.

Program

```
#include<stdio.h>
#include<string.h>
#include<conio.h>
#define N strlen(g)

char t[128], cs[128], 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);
}
```

```
void main() {
  clrscr();
  printf("\nEnter poly : "); scanf("%s",t);
  printf("\nGenerating Polynomial is : %s",g);
  a=strlen(t);
  for(e=a;e<a+N-1;e++) t[e]='0';
  printf("\nModified t[u] is :  %s",t);
  crc();
  printf("\nChecksum is : %s",cs);
  for(e=a;e<a+N-1;e++) t[e]=cs[e-a];
  printf("\nFinal Codeword is : %s",t);
  printf("\nTest Error detection 0(yes) 1(no) ? : ");
  scanf("%d",&e);
  if(e==0) {
    printf("Enter position where error is to inserted : ");
    scanf("%d",&e);
    t[e]=(t[e]=='0')?'1':'0';
    printf("Errorneous data   : %s\n",t);
  }
  crc();
  for (e=0;(e<N-1)&&(cs[e]!='1');e++);
  if(e<N-1) printf("Error detected.");
  else printf("No Error Detected.");
  getch();
}
```

Output

```
Enter poly : 1011101
Generating Polynomial is : 10001000000100001
Modified t[u] is :  10111010000000000000000
Checksum is : 1000101101011000
Final Codeword is : 10111011000101101011000
Test Error detection 0(yes) 1(no) ? : 0
Enter position where you want to insert error : 3
Errorneous data   : 1010111011000101101011000
Error detected.

Enter poly : 1011101
Generating Polynomial is : 10001000000100001
Modified t[u] is :  10111010000000000000000
Checksum is : 1000101101011000
Final Codeword is : 10111011000101101011000
Test Error detection 0(yes) 1(no) ? : 1
No Error Detected.
```

Experiment No 2

Frame Sorting

Problem Statement

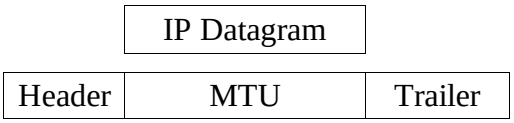
Write a program for frame sorting technique used in buffers.

Theory

The data link layer divides the stream of bits received from the network layer into manageable data units called frames.

If frames are to be distributed to different systems on the network, the Data link layer adds a header to the frame to define the sender and/or receiver of the frame.

Each Data link layer has its own frame format. One of the fields defined in the format is the maximum size of the data field. In other words, when datagram is encapsulated in a frame, the total size of the datagram must be less than this maximum size, which is defined by restriction imposed by the hardware and software used in the network.



The value of MTU differs from one physical network to another. In order to make IP protocol portable/independent of the physical network, the packagers decided to make the maximum length of the IP datagram equal to the largest Maximum Transfer Unit (MTU) defined so far. However for other physical networks we must divide the datagrams to make it possible to pass through these networks. This is called fragmentation.

When a datagram is fragmented, each fragmented has its own header. A fragmented datagram may itself be fragmented if it encounters a network with an even smaller MTU. In another words, a datagram may be fragmented several times before it reached the final destination and also, the datagrams referred to as (frames in Data link layer) may arrives out of order at destination. Hence sorting of frames need to be done at the destination to recover the original data.

Program

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

#define FSize 3

typedef struct packet{int SeqNum; char Data[FSize+1];}packet;

struct packet *readdata, *transdata;

int divide(char *msg) {
  int msglen, NoOfPacket, i, j;
  msglen = strlen(msg);
  NoOfPacket = msglen/FSize;
  if ((msglen%FSize)!=0) NoOfPacket++;
  readdata = (struct packet *)malloc(sizeof(packet) * NoOfPacket);
  for(i = 0; i < NoOfPacket; i++) {
     readdata[i].SeqNum = i + 1;
     for (j = 0; (j < FSize) && (*msg != '\0'); j++, msg++)
         readdata[i].Data[j] = *msg;
     readdata[i].Data[j] = '\0';
  }
```

```
printf("\nThe Message has been divided as follows\n");
  printf("\nPacket No.         Data\n\n");

for (i = 0; i < NoOfPacket; i++)
    printf("   %2d              %s\n", readdata[i].SeqNum,
           readdata[i].Data);
  return NoOfPacket;
}

void shuffle(int NoOfPacket) {
  int *Status;
  int i, j, trans;
  randomize();
  Status=(int * )calloc(NoOfPacket, sizeof(int));
  transdata = (struct packet *)malloc(sizeof(packet) * NoOfPacket);
  for (i = 0; i < NoOfPacket;) {
     trans = rand()%NoOfPacket;
     if (Status[trans]!=1) {
         transdata[i].SeqNum = readdata[trans].SeqNum;
         strcpy(transdata[i].Data, readdata[trans].Data);
         i++;     Status[trans] = 1;
     }
  }
  free(Status);
}

void sortframes(int NoOfPacket) {
  packet temp;
  int i, j;
  for (i = 0; i < NoOfPacket; i++)
    for (j = 0; j < NoOfPacket - i-1; j++)
       if (transdata[j].SeqNum > transdata[j + 1].SeqNum) {
           temp.SeqNum = transdata[j].SeqNum;
           strcpy(temp.Data, transdata[j].Data);
           transdata[j].SeqNum = transdata[j + 1].SeqNum;
           strcpy(transdata[j].Data, transdata[j + 1].Data);
           transdata[j + 1].SeqNum = temp.SeqNum;
           strcpy(transdata[j + 1].Data, temp.Data);
       }

}

void receive(int NoOfPacket) {
  int i;
  printf("\nPackets received in the following order\n");
  for (i = 0; i < NoOfPacket; i++) printf("%4d", transdata[i].SeqNum);
  sortframes(NoOfPacket);
  printf("\n\nPackets in order after sorting..\n");
  for (i = 0; i < NoOfPacket; i++) printf("%4d", transdata[i].SeqNum);
  printf("\n\nMessage received is :\n");
  for (i = 0; i < NoOfPacket; i++) printf("%s", transdata[i].Data);
}

void main() {
  char *msg;
  int NoOfPacket;
  clrscr();
  printf("\nEnter The message to be Transmitted :\n");
  scanf("%[^\n]", msg);
  NoOfPacket = divide(msg);
  shuffle(NoOfPacket);
  receive(NoOfPacket);
```

```
  free(readdata);
  free(transdata);
  getch();
}
```

Output

```
Enter The messgae to be Transmitted :
hi, it was nice meeting u on sunday

The Message has been divided as follows
Packet No.          Data

    1               hi,
    2                it
    3                wa
    4               s n
    5               ice
    6                me
    7               eti
    8               ng
    9               u o
   10               n s
   11               und
   12               ay

Packets received in the following order
   4   2   6   3   5   1   8   9  11   7  12  10

Packets in order after sorting..
   1   2   3   4   5   6   7   8   9  10  11  12

Message received is :
hi, it was nice meeting u on sunday
```

Experiment No 3

Hamming Codes

Problem Statement

Write a program for Hamming Code generation for error detection and correction

Theory

Hamming codes are used for detecting and correcting single bit errors in transmitted data. This requires that 3 parity bits (check bits) be transmitted with every 4 data bits. The algorithm is called A(7, 4) code, because it requires seven bits to encode 4 bits of data.

Eg:

Bit String	Parity Bit	Verification
000	0	0 + 0 + 0 + 0 = 0
001	1	0 + 0 + 1 + 1 = 0
010	1	0 + 1 + 0 + 1 = 0
011	0	0 + 1 + 1 + 0 = 0
100	1	1 + 0 + 0 + 1 = 0
101	0	1 + 0 + 1 + 0 = 0
110	0	1 + 0 + 1 + 0 = 0
111	1	1 + 1 + 1 + 1 = 0

Parity types:

Even: - Even number of 1's is present, i.e. the modulo 2 sum of the bits is 0.

Odd: - Odd number of 1's is present, i.e. the modulo 2 sum of the bits is 1.

Given data bits D1, D2, D3 and D4, A (7, 4) Hamming code may define parity bits P1, P2 and P3 as,

$$P1=D2+D3+D4$$
$$P2=D1+D3+D4$$
$$P3=D1+D2+D4$$

There are 4 equations for a parity bit that may be used in Hamming codes,

$$P4=D1+D2+D3$$

Valid Hamming codes may use any 3 of the above 4 parity bit definitions. Valid Hamming codes may also place the parity bits in any location within the block of 7 data and parity bits.

One method for transferring 4 bits of data into a 7 bit Hamming code word is to use a 4x7 generate matrix [G] defined to be the 1x4 vector [D1 D2 D3 D4].

Its possible to create a 4x7 generator matrix [G] such that the product modulo 2 of d and [G] (D[G]) is the desired 1x7 Hamming code word.

For example each data bits can be represented with in a column vector as follows.

$$D1=\begin{vmatrix}1\\0\\0\\0\end{vmatrix} \quad D2=\begin{vmatrix}0\\1\\0\\0\end{vmatrix} \quad D3=\begin{vmatrix}0\\0\\1\\0\end{vmatrix} \quad D4=\begin{vmatrix}0\\0\\0\\1\end{vmatrix}$$

And represent each parity bit with a column vector continuing a 1 in the row corresponding to each data bit included in the computation and a zero in all other rows.

$$P1=\begin{vmatrix}0\\1\\1\\1\end{vmatrix} \quad P2=\begin{vmatrix}1\\0\\1\\1\end{vmatrix} \quad P3=\begin{vmatrix}1\\1\\0\\1\end{vmatrix}$$

And

$$G = \begin{vmatrix} 0 & 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 1 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 \end{vmatrix}$$

P1 P2 P3 D1 D2 D3 D4

Encoding

The process to encode the data value 1010 using the Hamming code defined by the G matrix is as follows: -

$$[1010] \begin{vmatrix} 0 & 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 1 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 \end{vmatrix}$$

$$= \begin{vmatrix} (1*0)+(0*1)+(1*1)+(0*1) \\ (1*1)+(0*0)+(1*1)+(0*1) \\ (1*1)+(0*1)+(1*0)+(0*1) \\ (1*1)+(0*0)+(1*0)+(0*0) \\ (1*0)+(0*1)+(1*0)+(0*0) \\ (1*0)+(0*0)+(1*1)+(0*0) \\ (1*0)+(0*0)+(1*0)+(0*1) \end{vmatrix} = \begin{vmatrix} 0+0+1+0 \\ 1+0+1+0 \\ 1+0+0+0 \\ 1+0+0+0 \\ 0+0+0+0 \\ 0+0+1+0 \\ 0+0+0+0 \end{vmatrix} = \begin{vmatrix} 1 \\ 0 \\ 1 \\ 1 \\ 0 \\ 1 \\ 0 \end{vmatrix}$$

Therefore 1010 encodes into 1011010. Equivalent Hamming codes represented by different generator matrices will produce different results.

Decoding

The first step is to check the parity bits to determine if there is an error. Arithmetically, parity may be checked as follows: -

P1 = D2+D3+D4 = 0+1+1 = 0
P2 = D1+D3+D4 = 1+1+1 = 1
P3 = D1+D2+D4 = 1+0+1 = 0

Parity may also be validated by using matrix operation. A 3x7 parity check matrix [H] may be constructed such that row 1 contains 1^s in the position of the first parity bits and all the data bits that are included in its parity calculation. Using this, matrix [H] may be defined as follows: -

$$H = \begin{vmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 1 & 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 1 & 1 & 1 & 0 & 1 \end{vmatrix}$$

3^{rd} Bit Parity

2^{nd} Bit Parity

1^{st} Bit Parity

Multiplying the 3x7 matrix [H] by a 7x1 matrix representing the encoded data produces a 3x1 matrix called the **Syndrome**.

If the syndrome is all 0's the encoded data is error free. If the syndrome has a non-zero value, flipping the encoded bit that is in the position of the column matching the syndrome will result in a valid code word.

Ex: -

Suppose the data received is 1011011,

Then

$$\begin{vmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 1 & 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 1 & 1 & 1 & 0 & 1 \end{vmatrix} \begin{vmatrix} 1 \\ 0 \\ 1 \\ 1 \\ 0 \\ 1 \\ 1 \end{vmatrix} = \begin{vmatrix} (1*1)+(0*0)+(0*1)+(0*1)+(1*0)+(1*1)+(1*1) \\ (0*1)+(1*0)+(0*1)+(1*1)+(0*0)+(1*1)+(1*1) \\ (0*1)+(0*0)+(1*1)+(1*1)+(1*0)+(0*1)+(1*1) \end{vmatrix} = \begin{vmatrix} \mathbf{1} \\ \mathbf{1} \\ \mathbf{1} \end{vmatrix}$$

Looking back at the matrix [H] the 7^{th} column is all 1^{s} so the 7^{th} bit is the bit that **has an error.**

Program

```
#include<iostream.h>
#include<conio.h>
#include<stdlib.h>
#include<stdio.h>

char data[5];
int encoded[8], edata[7], syndrome[3];
int hmatrix[3][7]= { 1,0,0,0,1,1,1,
                     0,1,0,1,0,1,1,
                     0,0,1,1,1,0,1};

char gmatrix[4][8]={ "0111000", "1010100", "1100010", "1110001"};

void main() {
  int i,j;
  clrscr();
  cout<<"Hamming Code --- Encoding\n";
  cout<<"Enter 4 bit data : ";
  cin>>data;

  cout<<"Generator Matrix\n";
  for(i=0;i<4;i++) cout<<"\t"<<gmatrix[i]<<"\n";

  cout<<"Encoded Data : ";
  for(i=0;i<7;i++) {
    for(j=0;j<4;j++)
      encoded[i]+=((data[j]- '0')*(gmatrix[j][i]- '0'));
    encoded[i]=encoded[i]%2;
    cout<<encoded[i]<<" ";
  }
  cout<<"\nHamming code --- Decoding\n";
  cout<<"Enter Encoded bits as received : ";
  for(i=0;i<7;i++) cin>>edata[i];

  for(i=0;i<3;i++) {
    for(j=0;j<7;j++)
      syndrome[i]=syndrome[i]+(edata[j]*hmatrix[i][j]);
    syndrome[i]=syndrome[i]%2;
  }
```

```
for(j=0;j<7;j++)
    if ((syndrome[0]==hmatrix[0][j])&&(syndrome[1]==hmatrix[1][j])&&
      (syndrome[2]==hmatrix[2][j]))
      break;
  if(j==7)
    cout<<"Data is error free!!\n";
  else {
    cout<<"Error received at bit number "<<j+1<<" of the data\n";
    edata[j]=!edata[j];
    cout<<"The Correct data Should be : ";
    for(i=0;i<7;i++) cout<<edata[i]<<" ";
    }
  }
}
```

Output

```
Hamming Code --- Encoding
Enter 4 bit data: 1 0 1 0
Generator Matrix
        0111000
        1010100
        1100010
        1110001
Encoded Data: 1 0 1 1 0 1 0
Hamming code --- Decoding
Enter Encoded bits as received: 1 0 1 1 0 1 1
Error received at bit number 7 of the data
The Correct data Should be: 1 0 1 1 0 1 0

Hamming Code --- Encoding
Enter 4 bit data: 1 0 1 0
Generator Matrix
        0111000
        1010100
        1100010
        1110001
Encoded Data: 1 0 1 1 0 1 0
Hamming code --- Decoding
Enter Encoded bits as received: 1 0 1 1 0 1 0
Data is error free!!
```

Experiment No 4

Leaky Bucket

Problem Statement

Write a program for congestion control using Leaky bucket algorithm.

Theory

The congesting control algorithms are basically divided into two groups: open loop and closed loop. Open loop solutions attempt to solve the problem by good design, in essence, to make sure it does not occur in the first place. Once the system is up and running, midcourse corrections are not made. Open loop algorithms are further divided into ones that act at source versus ones that act at the destination.

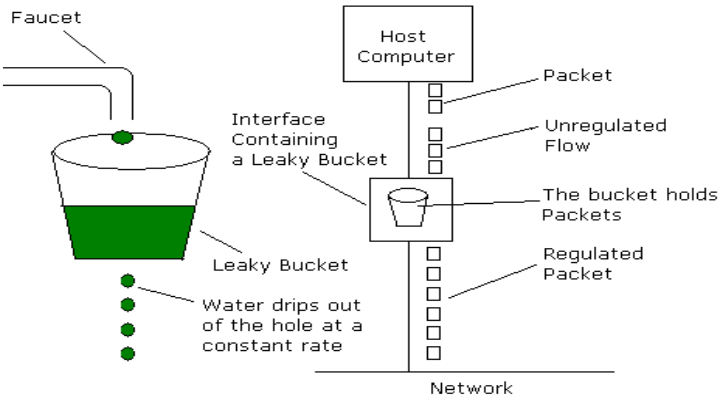
In contrast, closed loop solutions are based on the concept of a feedback loop if there is any congestion. Closed loop algorithms are also divided into two sub categories: explicit feedback and implicit feedback. In explicit feedback algorithms, packets are sent back from the point of congestion to warn the source. In implicit algorithm, the source deduces the existence of congestion by making local observation, such as the time needed for acknowledgment to come back.

The presence of congestion means that the load is (temporarily) greater than the resources (in part of the system) can handle. For subnets that use virtual circuits internally, these methods can be used at the network layer.

Another open loop method to help manage congestion is forcing the packet to be transmitted at a more predictable rate. This approach to congestion management is widely used in ATM networks and is called **traffic shaping**.

The other method is the leaky bucket algorithm. Each host is connected to the network by an interface containing a leaky bucket, that is, a finite internal queue. If a packet arrives at the queue when it is full, the packet is discarded. In other words, if one or more process are already queued, the new packet is unceremoniously discarded. This arrangement can be built into the hardware interface or simulate d by the host operating system. In fact it is nothing other than a single server queuing system with constant service time.

The host is allowed to put one packet per clock tick onto the network. This mechanism turns an uneven flow of packet from the user process inside the host into an even flow of packet onto the network, smoothing out bursts and greatly reducing the chances of congestion.



Program

```
#include<iostream.h>
#include<dos.h>
#include<stdlib.h>
#define bucketSize 512
void bktInput(int a,int b) {
  if(a>bucketSize)
    cout<<"\n\t\tBucket overflow";
  else {
    delay(500);
    while(a>b){
      cout<<"\n\t\t"<<b<<" bytes outputted.";
      a-=b;
      delay(500);
    }
    if (a>0) cout<<"\n\t\tLast "<<a<<" bytes sent\t";
    cout<<"\n\t\tBucket output successful";
  }
}

void main() {
  int op, pktSize;
  randomize();
  cout<<"Enter output rate : "; cin>>op;
  for(int i=1;i<=5;i++){
    delay(random(1000));
    pktSize=random(1000);
    cout<<"\nPacket no "<<i<<"\tPacket size = "<<pktSize;
    bktInput(pktSize,op);
  }
}
```

Output

```
Enter output rate: 100

Packet no 0 Packet size = 3
                Bucket output successful
                Last 3 bytes sent
Packet no 1 Packet size = 33
                Bucket output successful
                Last 33 bytes sent
Packet no 2 Packet size = 117
                Bucket output successful
                100 bytes outputted.
                Last 17 bytes sent
Packet no 3 Packet size = 95
                Bucket output successful
                Last 95 bytes sent
Packet no 4 Packet size = 949
                Bucket overflow
```

Experiment No 5

RSA Algorithm

Problem Statement

Write a program for simple RSA algorithm to encrypt and decrypt the data.

Theory

Cryptography has a long and colorful history. The message to be encrypted, known as the plaintext, are transformed by a function that is parameterized by a key. The output of the encryption process, known as the ciphertext, is then transmitted, often by messenger or radio. The enemy, or intruder, hears and accurately copies down the complete ciphertext. However, unlike the intended recipient, he does not know the decryption key and so cannot decrypt the ciphertext easily. The art of breaking ciphers is called **cryptanalysis** the art of devising ciphers (cryptography) and breaking them (cryptanalysis) is collectively known as cryptology.

There are several ways of classifying cryptographic algorithms. They are generally categorized based on the number of keys that are employed for encryption and decryption, and further defined by their application and use. The three types of algorithms are as follows:

1. Secret Key Cryptography (SKC): Uses a single key for both encryption and decryption. It is also known as symmetric cryptography.
2. Public Key Cryptography (PKC): Uses one key for encryption and another for decryption. It is also known as asymmetric cryptography.
3. Hash Functions: Uses a mathematical transformation to irreversibly "encrypt" information

Public-key cryptography has been said to be the most significant new development in cryptography. Modern PKC was first described publicly by Stanford University professor Martin Hellman and graduate student Whitfield Diffie in 1976. Their paper described a two-key crypto system in which two parties could engage in a secure communication over a non-secure communications channel without having to share a secret key.

Generic PKC employs two keys that are mathematically related although knowledge of one key does not allow someone to easily determine the other key. One key is used to encrypt the plaintext and the other key is used to decrypt the ciphertext. The important point here is that it does not matter which key is applied first, but that both keys are required for the process to work. Because pair of keys is required, this approach is also called asymmetric cryptography.

In PKC, one of the keys is designated the public key and may be advertised as widely as the owner wants. The other key is designated the private key and is never revealed to another party. It is straight forward to send messages under this scheme.

The RSA algorithm is named after Ron Rivest, Adi Shamir and Len Adleman, who invented it in 1977. The RSA algorithm can be used for both public key encryption and digital signatures. Its security is based on the difficulty of factoring large integers.

Algorithm

1. Generate two large random primes, P and Q, of approximately equal size.
2. Compute $N = P \times Q$
3. Compute $Z = (P\text{-}1) \times (Q\text{-}1)$.
4. Choose an integer E, $1 < E < Z$, such that GCD $(E, Z) = 1$
5. Compute the secret exponent D, $1 < D < Z$, such that $E \times D \equiv 1 \pmod{Z}$
6. The public key is (N, E) and the private key is (N, D).

Note: The values of P, Q, and Z should also be kept secret.
The message is encrypted using public key and decrypted using private key.

An example of RSA encryption

1. Select primes $P = 11$, $Q = 3$.
2. $N = P \times Q = 11 \times 3 = 33$
   $Z = (P-1) \times (Q-1) = 10 \times 2 = 20$
3. Lets choose $E = 3$
   Check GCD(E, P-1) = GCD(3, 10) = 1 (i.e. 3 and 10 have no common factors except 1),
   and check GCD(E, Q-1) = GCD(3, 2) = 1
   therefore GCD(E, Z) = GCD(3, 20) = 1
4. Compute D such that $E \times D \equiv 1 \pmod{Z}$
   compute $D = E^{\wedge -1} \bmod Z = 3^{\wedge -1} \bmod 20$
   find a value for D such that Z divides (($E \times D$)-1)
   find D such that 20 divides 3D-1.
   Simple testing (D = 1, 2, ...) gives D = 7
   Check: ($E \times D$)-1 = 3.7 - 1 = 20, which is divisible by Z.
5. Public key = (N, E) = (33, 3)
   Private key = (N, D) = (33, 7).

Now say we want to encrypt the message m = 7,
Cipher code = $M^{\wedge E} \bmod N$
$= 7^{\wedge 3} \bmod 33$
$= 343 \bmod 33$
$= 13.$
Hence the ciphertext c = 13.

To check decryption we compute Message' = $C^{\wedge D} \bmod N$
$= 13^{\wedge 7} \bmod 33$
$= 7.$

Note that we don't have to calculate the full value of 13 to the power 7 here. We can make use of the fact that a = bc mod n = (b mod n).(c mod n) mod n so we can break down a potentially large number into its components and combine the results of easier, smaller calculations to calculate the final value.

Program

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

int mult(unsigned int x, unsigned int y, unsigned int n) {
  unsigned long int k=1;
  int j;
  for (j=1; j<=y; j++) k = (k * x) % n;
  return (unsigned int) k;
}

void main ()
{
  char msg[100];
  unsigned int pt[100], ct[100], n, d, e, p, q, i;
  printf("Enter message : "); gets(msg);
  //strcpy(pt, msg);
  for(i=0;i<strlen(msg);i++)
   pt[i]=msg[i];
```

```
  n = 253; d = 17; e = 13;
  printf("\nCT = ");
  for(i=0; i<strlen(msg); i++) ct[i] = mult(pt[i], e,n);
  for(i=0; i<strlen(msg); i++) printf("%d ", ct[i]);

  printf("\nPT = ");
  for(i=0; i<strlen(msg); i++) printf("%c", pt[i]);
  for(i=0; i<strlen(msg); i++) pt[i] = mult(ct[i], d,n) ;
}
```

Output

```
Enter message: alpha
CT = 113 3 129 213 113
PT = alpha
```

Experiment No 6

TCP Socket

Problem Statement

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.

Algorithm (Client Side)

1. Start.
2. Create a socket using socket( ) system call.
3. Connect the socket to the address of the server using connect( ) system call.
4. Send the filename of required file using send( ) system call.
5. Read the contents of the file sent by server by recv( ) system call.
6. Stop.

Algorithm (Server Side)

1. Start.
2. Create a socket using socket( ) system call.
3. Bind the socket to an address using bind( ) system call.
4. Listen to the connection using listen( ) system call.
5. accept connection using accept( )
6. Receive filename and transfer contents of file with client.
7. Stop.

Program

/*Server*/

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

int main()
{
  int cont,create_socket,new_socket,addrlen,fd;
  int bufsize = 1024;
  char *buffer = malloc(bufsize);
  char fname[256];
  struct sockaddr_in address;

  if ((create_socket = socket(AF_INET,SOCK_STREAM,0)) > 0)
    printf("The socket was created\n");

  address.sin_family = AF_INET;
  address.sin_addr.s_addr = INADDR_ANY;
  address.sin_port = htons(15000);

  if (bind(create_socket,(struct sockaddr *)&address,sizeof(address)) == 0)
    printf("Binding Socket\n");
  listen(create_socket,3);
  addrlen = sizeof(struct sockaddr_in);
  new_socket = accept(create_socket,(struct sockaddr *)&address,&addrlen);
```

```
if (new_socket > 0)
     printf("The Client %s is Connected...\n",
             inet_ntoa(address.sin_addr));
  recv(new_socket,fname, 255,0);
  printf("A request for filename %s Received..\n", fname);
  if ((fd=open(fname, O_RDONLY))<0)
    {perror("File Open Failed"); exit(0);}
  while((cont=read(fd, buffer, bufsize))>0) {
     send(new_socket,buffer,cont,0);
  }
  printf("Request Completed\n");
  close(new_socket);
  return close(create_socket);
}
```

/*Client*/

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

int main(int argc,char *argv[])
{
  int create_socket;
  int bufsize = 1024;
  char *buffer = malloc(bufsize);
  char fname[256];
  struct sockaddr_in address;

  if ((create_socket = socket(AF_INET,SOCK_STREAM,0)) > 0)
    printf("The Socket was created\n");
  address.sin_family = AF_INET;
  address.sin_port = htons(15000);
  inet_pton(AF_INET,argv[1],&address.sin_addr);

  if (connect(create_socket,(struct sockaddr *) &address,
              sizeof(address)) == 0)
    printf("The connection was accepted with the server %s...\n",
            argv[1]);
  printf("Enter The Filename to Request : "); scanf("%s",fname);
  send(create_socket, fname, sizeof(fname), 0);
  printf("Request Accepted... Receiving File...\n\n");
  printf("The contents of file are...\n\n");
  while((cont=recv(create_socket, buffer, bufsize, 0))>0) {
    write(1, buffer, cont);
  }
  printf("\nEOF\n");
  return close(create_socket);
}
```

Output (Server)

```
[root@localhost CN Lab] ./s.o
Socket Created..
Binding Socket..
Now Listening for Request..

The Client 127.0.0.1 is trying to connect…
A request for filename alpha received..
Requested completed..
[root@localhost CN Lab]
```

Output (Client)

```
[root@localhost CN Lab] ./c.o 127.0.0.1
Socket Created..
Connetion is accepted by 127.0.0.1 ..
Enter File Name to Request : alpha
Requestion for file alpha.. Request accepted. Receiving file…

The contents of file are :-
This a demo of client server using Sockets
Just for trial.
Now End of file

EOF

[root@localhost CN Lab]
```

Experiment No 7

FIFO IPC

Problem Statement

Implement the above program using as message queues or FIFO as IPC channels.

Algorithm (Client Side)

1. Start.
2. Open well known server FIFO in write mode.
3. Write the pathname of the file in this FIFO and send the request.
4. Open the client specified FIFO in read mode and wait for reply.
5. When the contents of the file are available in FIFO, display it on the terminal
6. Stop.

Algorithm (Server Side)

1. Start.
2. Create a well known FIFO using mkfifo()
3. Open FIFO in read only mode to accept request from clients.
4. When client opens the other end of FIFO in write only mode, then read the contents and store it in buffer.
5. Create another FIFO in write mode to send replies.
6. Open the requested file by the client and write the contents into the client specified FIFO and terminate the connection.
7. Stop.

Program

/*Server*/

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

#define FIFO1 "fifo1"
#define FIFO2 "fifo2"
#define PERMS 0666
char fname[256];
int main() {
  int readfd, writefd, fd;
  ssize_t n;
  char buff[512];
  if (mkfifo(FIFO1, PERMS)<0)
    printf("Cant Create FIFO Files\n");
  if (mkfifo(FIFO2, PERMS)<0)
    printf("Cant Create FIFO Files\n");
  printf("Waiting for connection Request..\n");
  readfd =open(FIFO1, O_RDONLY, 0);
  writefd=open(FIFO2, O_WRONLY, 0);
  printf("Connection Established..\n");
  read(readfd, fname, 255);
  printf("Client has requested file %s\n", fname);
  if ((fd=open(fname,O_RDWR))<0) {
    strcpy(buff,"File does not exist..\n");
    write(writefd, buff, strlen(buff));
  } else {
    while((n=read(fd, buff,512))>0)
      write(writefd, buff, n);
  }
```

```
  close(readfd);  unlink(FIFO1);
  close(writefd);  unlink(FIFO2);
}
```

/*Client*/

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

#define FIFO1 "fifo1"
#define FIFO2 "fifo2"
#define PERMS 0666

char fname[256];

int main()
{
  ssize_t n;
  char buff[512];
  int readfd,writefd;
  printf("Trying to Connect to Server..\n");
  writefd = open(FIFO1, O_WRONLY, 0);
  readfd  = open(FIFO2, O_RDONLY, 0);
  printf("Connected..\n");
  printf("Enter the filename to request from server: ");
  scanf("%s",fname);
  write(writefd, fname, strlen(fname));
  printf("Waiting for Server to reply..\n");
  while((n=read(readfd,buff,512))>0)
    write(1,buff,n);
  close(readfd);
  close(writefd);
  return 0;
}
```

Output (Server)

```
[root@localhost CN Lab] ./s.o
Waiting for connection Request..
Connection Established..
Client has requested file alpha
[root@localhost CN Lab]
```

Output (Client)

```
[root@localhost CN Lab] ./c.o
Trying to Connect to Server..
Connected..
Enter the filename to request from server: alpha
Waiting for Server to reply..

This a demo of client server using Sockets
Just for trial.
Now End of file

[root@localhost CN Lab]
```

PART - B Programs

General Approach

In general, a simulation study comprises the following phases:

- The first phase is to create and prepare the simulation scenario based on the system description and metrics of interest.
- The next step is to execute, visualize, and analyze the created scenario and collect simulation results. Simulation results can include scenario animations, runtime statistics, final statistics, and output traces.
- The last phase is to analyze the simulation results. Typically, users may need to adjust the scenarios based on the collected simulation results.

This general procedure is illustrated in Fig. 1

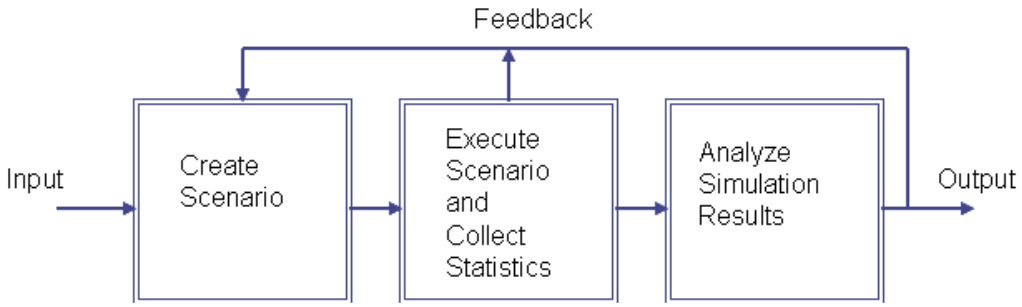


Fig. 1: Scenario-based Simulation

Experiment No 1

Problem Statement

Design a star topology for LAN using hub or switch or both by taking at least five nodes and analyze the statistics.

Topology:

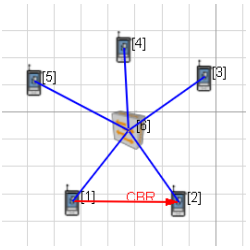


Fig. 2: Star topology for LAN using switch

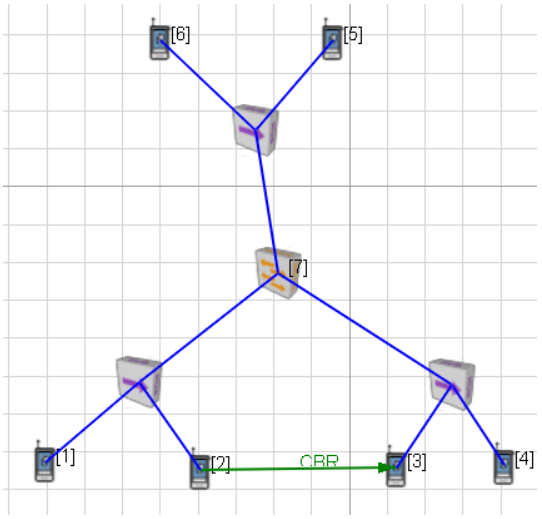


Fig. 3: Star topology for LAN using hub and switch

Experiment No 2

Problem Statement

Design a mesh topology for five nodes and analyze the TCP, UDP statistics and FIFO queuing mechanism.

Topology:

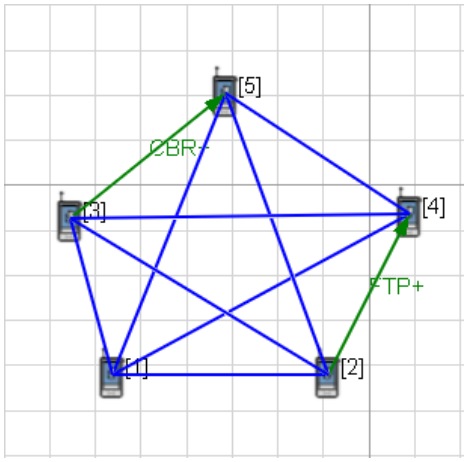


Fig. 4: Mesh topology

Experiment No 3

Problem Statement

Develop a scenario for wired network and implement VOIP and analyze the statistics.

Topology:

Fig. 5: Simple wired network

Experiment No 4

Problem Statement

Design a scenario to integrate wired network and wireless network using Wi-Fi infrastructure mode and analyze the statistics.

Topology:

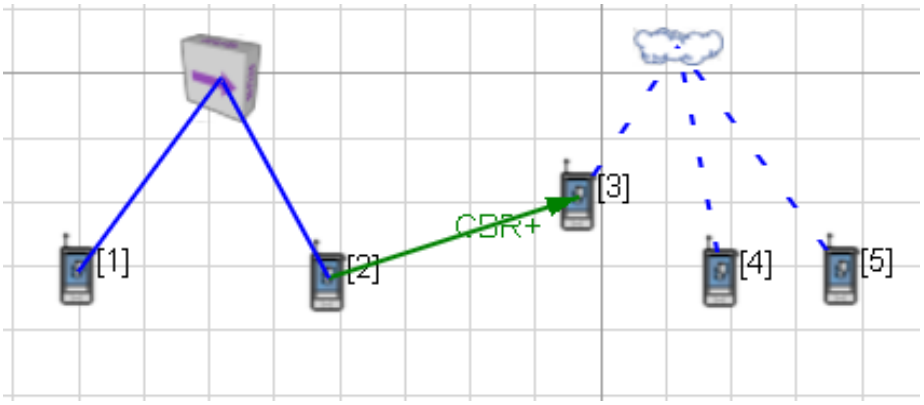


Fig. 6: Simple scenario to integrate wired network and wireless network

Experiment No 5

Problem Statement

Develop a scenario for WIMAX using standard 802.16. Analyze the statistics.

Topology:

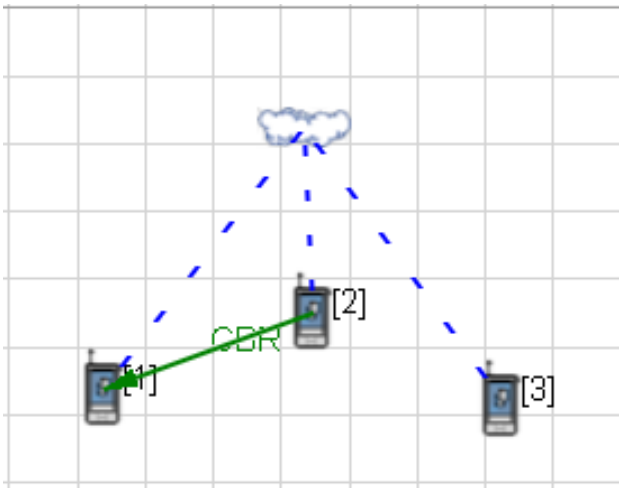


Fig. 7: Simple scenario for WIMAX

Experiment No 6

Problem Statement

Design a scenario for wireless sensor networks (total 9 devices). Analyze the Zigbee battery model along with the statistics.

Topology:

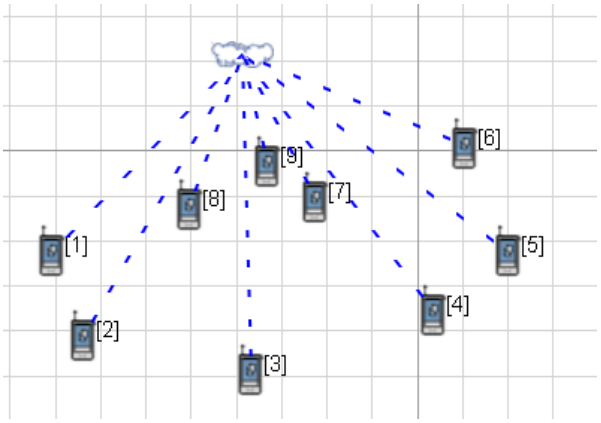


Fig. 8: Simple scenario for Zigbee

Experiment No 7

Problem Statement

Develop a scenario for MANET in wireless networks. Consider 7 devices and show the mobility for all the devices and analyze the statistics.

Topology:

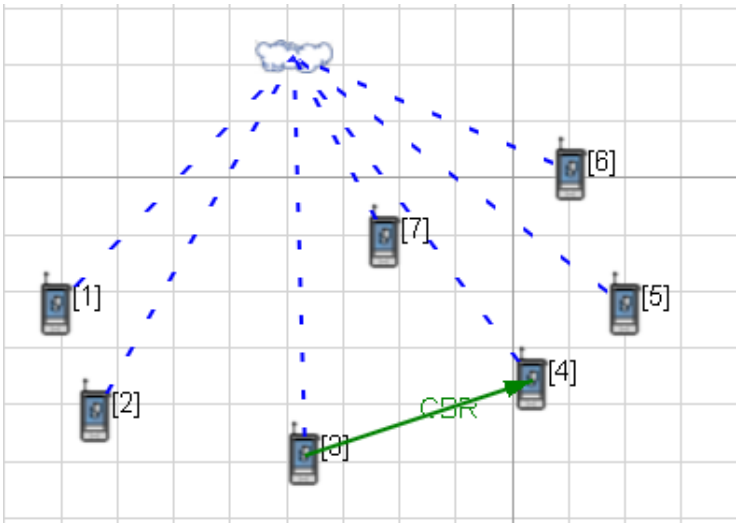


Fig. 9: Simple scenario for MANET

Viva Questions:

1. What are functions of different layers?
2. Differentiate between TCP/IP Layers and OSI Layers
3. Why header is required?
4. What is the use of adding header and trailer to frames?
5. What is encapsulation?
6. Why fragmentation requires?
7. What is MTU?
8. Which layer imposes MTU?
9. Differentiate between flow control and congestion control.
10. Differentiate between Point-to-Point Connection and End-to-End connections.
11. What are protocols running in different layers?
12. What is Protocol Stack?
13. Differentiate between TCP and UDP.
14. Differentiate between Connectionless and connection oriented connection.
15. Why frame sorting is required?
16. What is meant by subnet?
17. What is meant by Gateway?
18. What is an IP address?
19. What is MAC address?
20. Why IP address is required when we have MAC address?
21. What is meant by port?
22. What are ephemerical port number and well known port numbers?
23. What is a socket?
24. What are the parameters of socket()?
25. Describe bind(), listen(), accept(),connect(), send() and recv().
26. What are system calls? Mention few of them.
27. What is IPC? Name three techniques.
28. Explain mkfifo(), open(), close() with parameters.
29. What is meant by file descriptor?
30. What is meant by traffic shaping?