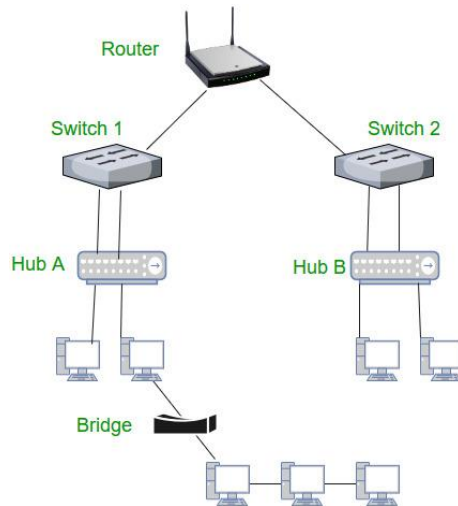


## Experiment : 1

**Aim:** Algorithm to implement study of Network devices in detail and connect the computers in Local Area Network.

### Description:

Network devices, also known as networking hardware, are physical devices that allow hardware on a computer network to communicate and interact with one another. For example Repeater, Hub, Bridge, Switch, Routers, Gateway, Brouter, and NIC, etc.



**Hub** – A hub is a basically multi-port repeater. A hub connects multiple wires coming from different branches, for example, the connector in star topology which connects different stations. Hubs cannot filter data, so data packets are sent to all connected devices.

**Bridge** – A bridge operates at the data link layer. A bridge is a repeater, with add on the functionality of filtering content by reading the MAC addresses of the source and destination. It is also used for interconnecting two LANs working on the same protocol.

**Switch** – A switch is a multiport bridge with a buffer and a design that can boost its efficiency(a large number of ports imply less traffic) and performance. A switch is a data link layer device.

## Experiment : 2

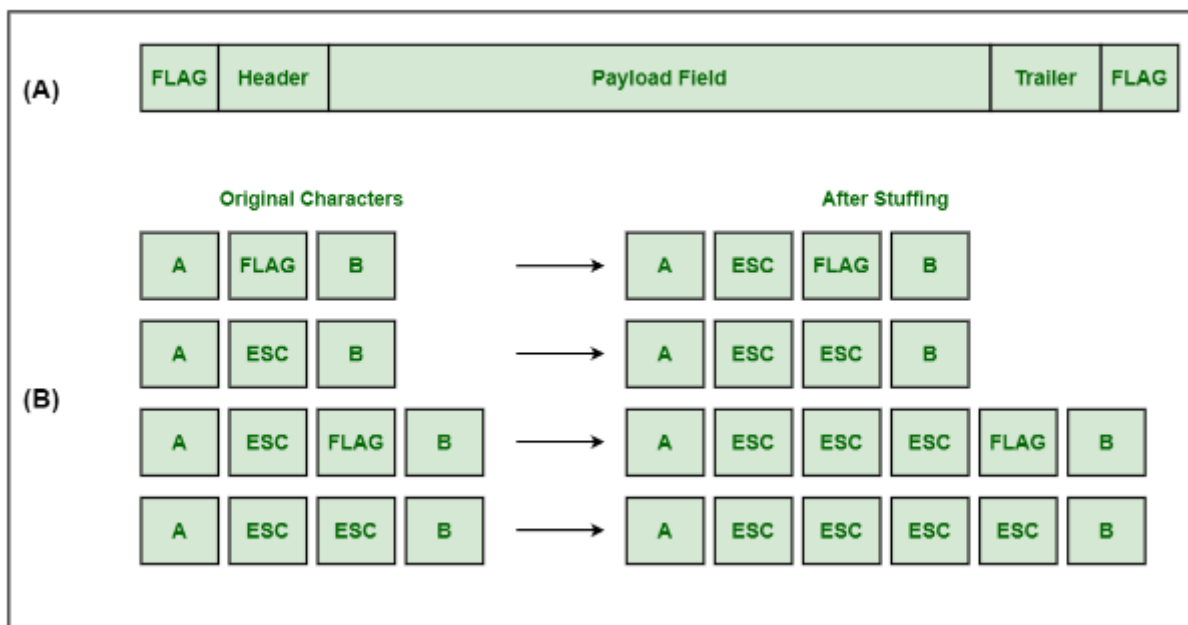
**Aim:** Algorithm to implement the data link layer framing methods such as i) Character stuffing ii) bit stuffing

### (i) Character stuffing

#### Description:

Character stuffing is also known as byte stuffing or character-oriented framing and is same as that of bit stuffing but byte stuffing actually operates on bytes whereas bit stuffing operates on bits. In byte stuffing, special byte that is basically known as ESC (Escape Character) that has predefined pattern is generally added to data section of the data stream or frame when there is message or character that has same pattern as that of flag byte.

#### Example:



#### Code:

```
#include<stdio.h>
#include<conio.h>
#include<string.h>
void main()
{
    int i=0,j=0,n,pos;
    char a[20],b[50],ch='z';
    clrscr();
    printf("enter string\n");
    scanf("%s",&a);
    n=strlen(a);
    printf("enter position\n");
    scanf("%d",&pos);
    if(pos>n)
    {
```

```

        printf("invalid position, Enter again less than %d:",n);
        scanf("%d",&pos);
    }
    b[0]='d';
    b[1]='l';
    b[2]='e';
    b[3]='s';
    b[4]='t';
    b[5]='x';
    j=6;
    while(i<n)
    {
        if(i==pos-1)
        {
            b[j]='d';
            b[j+1]='l';
            b[j+2]='e';
            b[j+3]=ch;
            b[j+4]='d';
            b[j+5]='l';
            b[j+6]='e';
            j=j+7;
        }
        if(a[i]=='d' && a[i+1]=='l' && a[i+2]=='e')
        {
            b[j]='d';
            b[j+1]='l';
            b[j+2]='e';
            j=j+3;
        }
        b[j]=a[i];
        i++;
        j++;
    }
    b[j]='d';
    b[j+1]='l';
    b[j+2]='e';
    b[j+3]='e';
    b[j+4]='t';
    b[j+5]='x';
    b[j+6]='\0';
    printf("\nframe after stuffing:\n");
    printf("%s",b);
    getch();
}

```

**Output:**

```
enter string
hellogautam
enter position
5

frame after stuffing:
dlestxhelldlezdleogautamdleetx

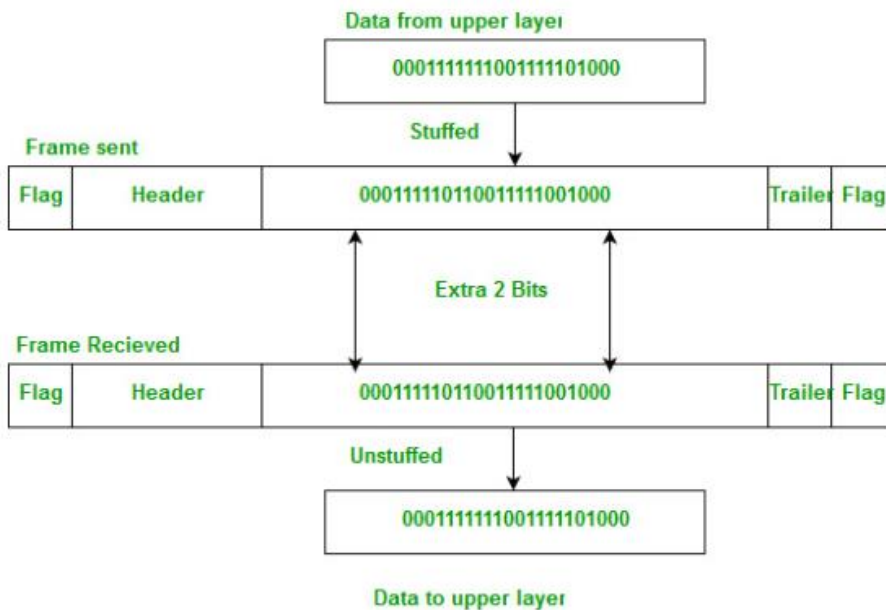
...Program finished with exit code 0
Press ENTER to exit console.
```

## (ii) Bit Stuffing

### Description:

Bit stuffing is also known as bit-oriented framing or bit-oriented approach. In bit stuffing, extra bits are being added by network protocol designers to data streams. It is generally insertion or addition of extra bits into transmission unit or message to be transmitted as simple way to provide and give signalling information and data to receiver and to avoid or ignore appearance of unintended or unnecessary control sequences.

### Example:



### Code:

```
#include<stdio.h>
#include<conio.h>
#include<string.h>
void main()
{
    int a[20],b[30],i,j,k,count,n;
    clrscr();
    printf("Enter frame length:");
    scanf("%d",&n);
    printf("Enter input frame (0's & 1's only):");
    for(i=0;i<n;i++)
        scanf("%d",&a[i]);
    i=0;
    count=1;
    j=0;
    while(i<n)
    {
        if(a[i]==1)
        {
            b[j]=a[i];
```

```

        for(k=i+1;a[k]==1 && k<n &&count<5;k++)
            { j++;b[j]=a[k];
              count++;
              if(count==5)
                  {
                      j++;
                      b[j]=0;
                  }
              i=k;
            }
    else
        {
            b[j]=a[i];
        }
    i++;
    j++;
}

printf("After stuffing the frame is:");
for(i=0;i<j;i++)
    printf("%d",b[i]);
getch();
}

```

### Output:

```

Enter frame length:10
Enter input frame (0's & 1's only):1
1
1
1
1
1
1
0
0
1
1
After stuffing the frame is:11111010011
...Program finished with exit code 255
Press ENTER to exit console.

```

### Experiment : 3

**Aim:** Algorithm to implement data link layer framing method checksum

**Description:**

The Checksum is an error-detecting method that is applied to the higher layer protocols. In this technique, the generator subdivides the data unit into equal segments of n bits. These segments are added using 1's complement method in a way such that the result is also n-bit long. The sum is then 1's complemented and appended to the data unit as redundant bits that are called the Checksum field.

**Example:**

```
10101001
00111001
11100010 ← sum
00011101 ← checksum
```

**Pattern sent is 10101001 00111001 00011101.**

**The receiver does the following operations :**

```
10101001
00111001
00011101
11111111
00000000 ← result
```

So in this case there is no error and the data is accepted.

In case of error suppose the received data is 10101111 11111001 00011101. Then the receiver will compute like this :

```
10101111
11111001
00011101
111000101
  1
11000110
00111001
```

**Program**

```
#include<stdio.h>
#include<string.h>
int main()
{
    char a[20],b[20];
    char sum[20],complement[20];
    int i,length;
    printf("Enter first binary string\n");
    scanf("%s",a);
    printf("Enter second binary string\n");
```

```

scanf("%s",b);
if(strlen(a)==strlen(b))
{
    length=strlen(a);
    char carry='0';
    for(i=length-1;i>=0;i--)
    {
        if(a[i]=='0'&&b[i]=='0'&&carry=='0')
        {
            sum[i]='0';
            carry='0';
        }
        else if(a[i]=='0'&&b[i]=='0'&&carry=='1')
        {
            sum[i]='1';
            carry='0';
        }
        else if(a[i]=='0'&&b[i]=='1'&&carry=='0')
        {
            sum[i]='1';
            carry='0';
        }
        else if(a[i]=='0'&&b[i]=='1'&&carry=='1')
        {
            sum[i]='0';
            carry='1';
        }
        else if(a[i]=='1'&&b[i]=='0'&&carry=='0')
        {
            sum[i]='1';
            carry='0';
        }
        else if(a[i]=='1'&&b[i]=='0'&&carry=='1')

```



```

        {
            sum[i]='0';
            carry='1';
        }
    else if(a[i]=='1'&&b[i]=='1'&&carry=='0')
    {
        sum[i]='0';
        carry='1';
    }
    else if(a[i]=='1'&&b[i]=='1'&&carry=='1')
    {
        sum[i]='1';
        carry='1';
    }
    else
        break;
}
printf("\n Carry=%c, Sum=%s",carry,sum);
for(i=length-1;i>0;i--)
{
    if(sum[i]=='1'&& carry=='1')
    {
        sum[i]='0';
        carry='1';
    }
    else if(sum[i]=='0'&& carry=='1')
    {
        sum[i]='1';
        break;
    }
    else
        break;
}

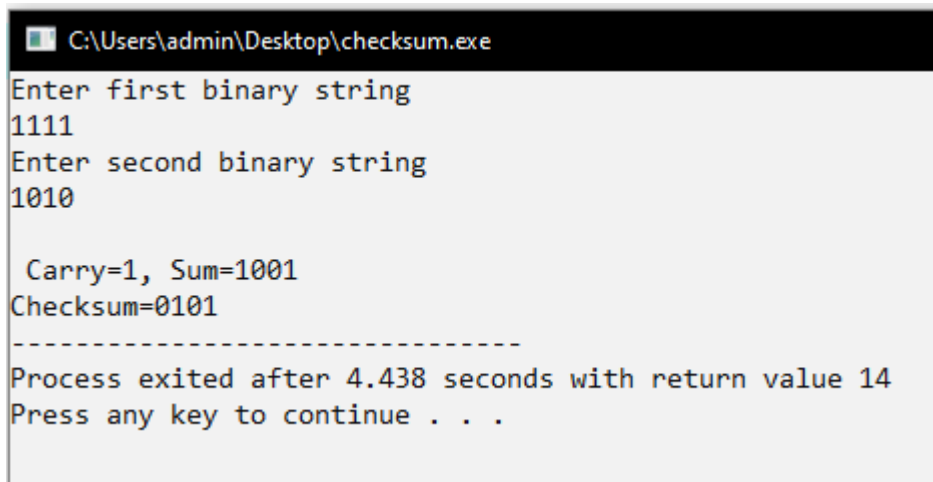
```

```

        for(i=0;i<length;i++)
        {
            if(sum[i]=='0')
                complement[i]='1';
            else
                complement[i]='0';
        }
        printf("\nChecksum=%s",complement);
    }
    else
    {
        printf("\n Wrong input Strings");
    }
}

```

## OUTPUT:



The screenshot shows a Windows command prompt window with the title bar "C:\Users\admin\Desktop\checksum.exe". The program prompts the user to "Enter first binary string" and "Enter second binary string". The user enters "1111" and "1010" respectively. The program then displays the calculation: "Carry=1, Sum=1001" and "Checksum=0101". A dashed line separates this from the final output: "Process exited after 4.438 seconds with return value 14" and "Press any key to continue . . .".

```

C:\Users\admin\Desktop\checksum.exe
Enter first binary string
1111
Enter second binary string
1010

Carry=1, Sum=1001
Checksum=0101
-----
Process exited after 4.438 seconds with return value 14
Press any key to continue . . .

```

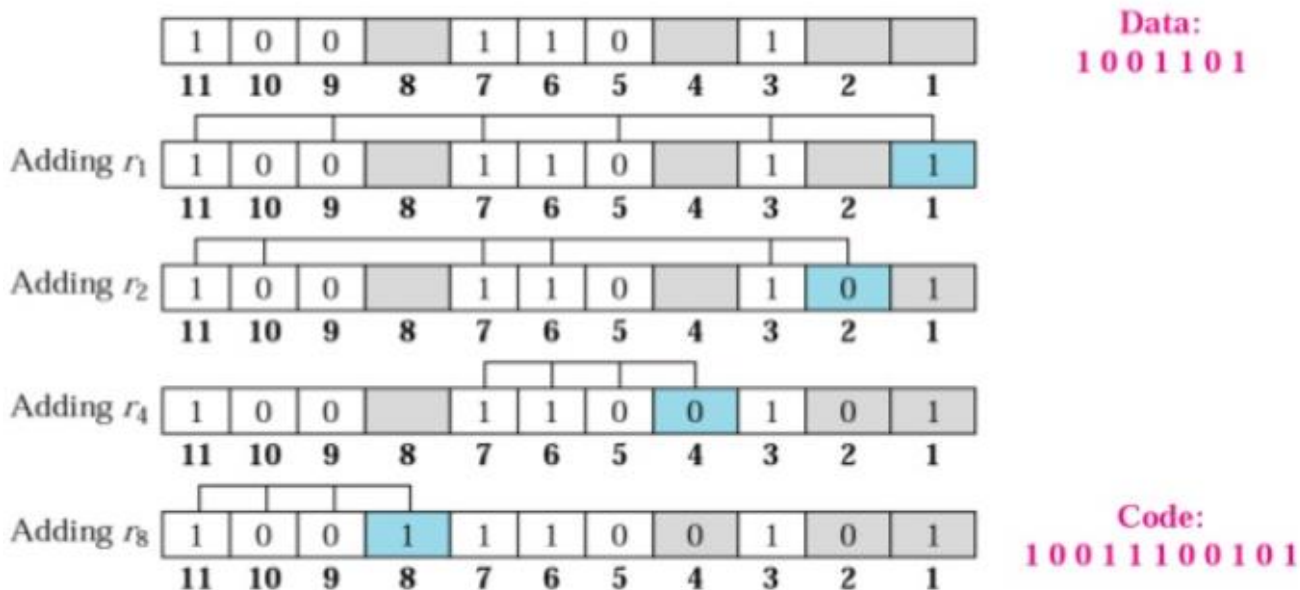
## Experiment : 4

**Aim:** Algorithm to implement Hamming Code generation for error detection and correction

### Description:

Hamming code is a linear code that is useful for error detection up to two immediate bit errors. It is capable of single-bit errors. In Hamming code, the source encodes the message by adding redundant bits in the message. These redundant bits are mostly inserted and generated at certain positions in the message to accomplish error detection and correction process.

Example:



### Program:

```
#include<stdio.h>
void main()
{
    int data[10];
    int dataatrec[10],c,c1,c2,c3,i;
    printf("Enter 4 bits of data one by one\n");
    scanf("%d",&data[0]);
    scanf("%d",&data[1]);
    scanf("%d",&data[2]);
    scanf("%d",&data[4]);
    //calculation of even parity
    data[6]=data[0]^data[2]^data[4];
    data[5]=data[0]^data[1]^data[4];
    data[3]=data[0]^data[1]^data[2];
    printf("The encoded data is \n");
    for(i=0;i<7;i++)

    printf("%d",data[i]);
```

```

printf("\n\nEnter received data bits one by one\n");
for(i=0;i<7;i++)
scanf("%d",&dataatrec[i]);
c1=dataatrec[6]^dataatrec[4]^dataatrec[2]^dataatrec[0];
c2=dataatrec[5]^dataatrec[4]^dataatrec[1]^dataatrec[0];
c3=dataatrec[3]^dataatrec[2]^dataatrec[1]^dataatrec[0];
c=c3*4+c2*2+c1;
if(c==0)
{
printf("\nNo error while transmission of data\n");
}
else
{
printf("\nError on position %d",c);
printf("\nData sent: ");
for(i=0;i<7;i++)

printf("%d",dataatrec[i]);
printf("\nCorrect message is \n");
//if erroneous bit is 0 we complement it else vice versa
if(dataatrec[7-c]==0)
dataatrec[7-c]=1;
else
dataatrec[7-c]=0;
for(i=0;i<7;i++)
{
printf("%d",dataatrec[i]);
}
}
}

```

## OUTPUT:

```
C:\Users\admin\Desktop\hammingcode.exe
Enter 4 bits of data one by one
1
0
1
0
The encoded data is
1010010

Enter received data bits one by one
1
0
1
0
1
1
1
0
Error on position 3
Data sent: 1010110
Correct message is
1010010
-----
Process exited after 19.51 seconds with return value 1
Press any key to continue . . .
```

## Experiment : 5

**Aim:** Algorithm to implement data set of characters the three CRC polynomials – CRC 12, CRC 16 and CRC CCIP.

### Description:

CRC method can detect a single burst of length  $n$ , since only one bit per column will be changed, a burst of length  $n+1$  will pass undetected, if the first bit is inverted, the last bit is inverted and all other bits are correct. If the block is badly garbled by a long burst or by multiple shorter burst, the probability that any of the  $n$  columns will have the correct parity that is 0.5. so the probability of a bad block being expected when it should not be  $2^{\text{power}(-n)}$ . This scheme sometimes is known as Cyclic Redundancy Code.

### Example:

#### CALCULATION FOR GENERATING A CRC

Message = 110101  
Polynomial = 101

$$\begin{array}{r} 11010100 \div 101 = 11101 \\ \underline{101} \phantom{000000} \\ 111 \phantom{000000} \\ \underline{101} \phantom{000000} \\ 100 \phantom{000000} \\ \underline{101} \phantom{000000} \\ 110 \phantom{000000} \\ \underline{101} \phantom{000000} \\ 110 \phantom{000000} \\ \underline{101} \phantom{000000} \\ 11 \phantom{000000} \end{array}$$

Quotient (has no function in CRC calculation)

← Remainder = CRC checksum

Message with CRC = 11010111

#### CHECKING A MESSAGE FOR A CRC ERROR

Message with CRC = 11010111  
Polynomial = 101

$$\begin{array}{r} 11010111 \div 101 = 11101 \\ \underline{101} \phantom{000000} \\ 111 \phantom{000000} \\ \underline{101} \phantom{000000} \\ 100 \phantom{000000} \\ \underline{101} \phantom{000000} \\ 111 \phantom{000000} \\ \underline{101} \phantom{000000} \\ 101 \phantom{000000} \\ \underline{101} \phantom{000000} \\ 00 \phantom{000000} \end{array}$$

Quotient

← Checksum is zero, therefore, no transmission error

## Program

```
#include <stdio.h>
#include <string.h>
#define N strlen(g)
char t[28],cs[28],g[28];
int a,e,c,b;
void xor()
{
    for(c=1;c<N;c++)
        cs[c]=((cs[c]==g[c])?'0':'1');
}
void crc()
{
    for(e=0;e<N;e++)
        cs[e]=t[e];
    do
    {
        if(cs[0]=='1')
            xor();
        for(c=0;c<N-1;c++)
            cs[c]=cs[c+1];
        cs[c]=t[e++];
    } while(e<=a+N-1);
}

int main()
{
    int flag=0;
    do
    {
        printf("\n1.crc12\n2.crc16\ncrc ccit\n4.exit\n\nEnter your option.");
        scanf("%d",&b);
        switch(b)
        {
            case 1:strcpy(g,"1100000001111");
                    break;
            case 2:strcpy(g,"11000000000000101");
                    break;
            case 3:strcpy(g,"10001000000100001");
                    break;
            case 4:return 0;
        }
        printf("\nEnter data:");
        scanf("%s",t);
        //printf("\nN value is %d",N);
        printf("\n-----\n");
        printf("\nGenerating polynomial:%s",g);
        a=strlen(t);
        for(e=a;e<a+N-1;e++)
            t[e]='0';
        printf("\n-----\n");
    }
}
```

```

printf("modified data is:%s",t);
printf("\n-----\n");
crc();
printf("checksum is:%s",cs);
for(e=a;e<a+N-1;e++)
    t[e]=cs[e-a];
printf("\n-----\n");
printf("\n final codeword is : %s",t);
printf("\n-----\n");
printf("\ntest error detection 0(yes) 1(no)?:"");
scanf("%d",&e);
if(e==0)
{
    do
    {
        printf("\nEnter the position where error is to be inserted:");
        scanf("%d",&e);
    } while(e==0||e>a+N-1);
    t[e-1]=(t[e-1]=='0')?'1':'0';
    printf("\n-----\n");
    printf("\nErroneous data:%s\n",t);
}
    crc();
    for(e=0;(e<N-1)&&(cs[e]!='1');e++);
    if(e<N-1)
        printf("Error detected\n\n");
    else
        printf("\nNo error detected \n\n");
    printf("\n-----");

} while(flag!=1);
}

```

## OUTPUT:

- 1.crc12
- 2.crc16
- 3.crc ccit
- 4.exit

Enter your option.1

enter data:1100110011100011

-----

generating polynomial:1100000001111

-----

mod-ified data is:11001100111000110000000000000011000000001111

-----

checksum is:1101110110001

-----



-----

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

no error detected

-----

1.crc12

2.crc16

3.crc ccit

#### 4.exit

Enter your option.2

enter data:11001100111000

-----

```
/*generating polynomial:1 10000000000000101
```

-----

[illegible]

---

checksum is:1111111110110000

-----

final codeword is : 1100110011100011111111101100000000000000101

---

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

no error detected

1.crc12

2.crc16

3.crc ccit

## 4.exit

Enter your option.3

enter data:11001100111000

---

generating polynomial:10001000000100001

-----

mod-ified data is:110011001110000000000000000000001000000100001

---

checksum is:11100111100111010

---

final codeword is : 110011001110001110011110011101001000000100001

---

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

Enter the position where error is to be inserted:3

---

erroneous data:1110110011100011110011110011101001000000100001

error detected

---

1.crc12

2.crc16

3.crc ccit

## 4.exit

Enter your option.4

## Experiment : 6

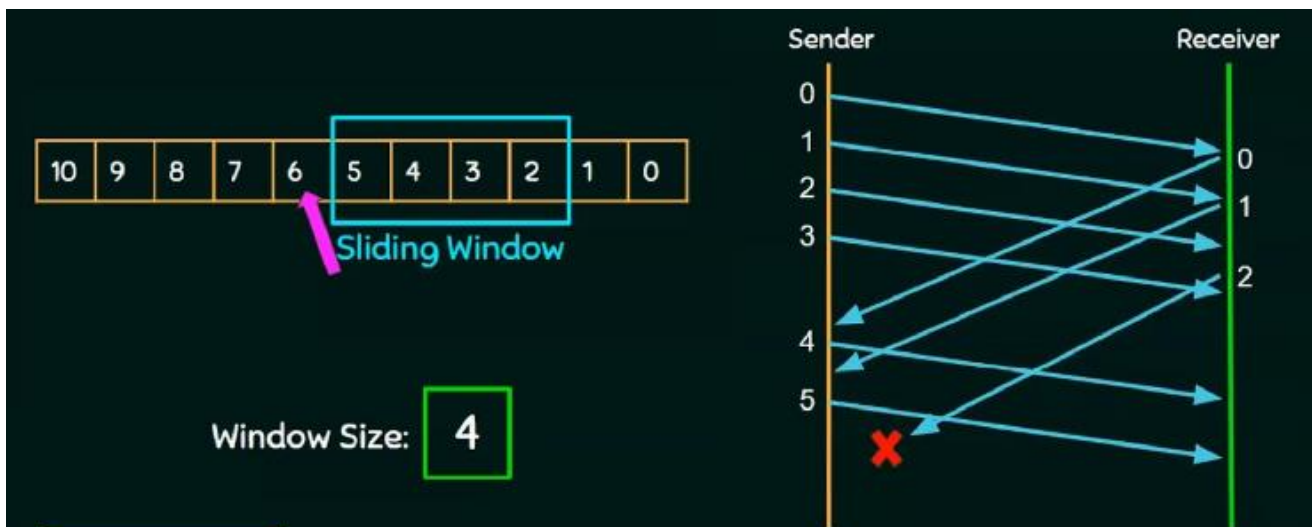
**Aim:** Algorithm to implement Sliding window protocol for Goback N

### Description:

Go-Back-N protocol, also called Go-Back-N Automatic Repeat reQuest, is a data link layer protocol that uses a sliding window method for reliable and sequential delivery of data frames. It is a case of sliding window protocol having to send window size of N and receiving window size of 1.

Go – Back – N ARQ provides for sending multiple frames before receiving the acknowledgment for the first frame. The frames are sequentially numbered and a finite number of frames. The maximum number of frames that can be sent depends upon the size of the sending window. If the acknowledgment of a frame is not received within an agreed upon time period, all frames starting from that frame are retransmitted.

### Example:



### Program:

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

int n,r;

struct frame
{
    char ack;
    int data;
```

```

}frm[10];

int sender(void);
void recvack(void);
void resend_gb(void);

int main()
{
    int c;
    sender();
    recvack();
    resend_gb();
    printf("\n All Frames sent Successfully\n");
}

int sender()
{
    int i;
    printf("\n Enter no. of Frames to be sent: ");
    scanf("%d",&n);
    for(i=1;i<=n;i++)
    {
        printf("\n Enter data for Frames [%d] ", i);
        scanf("%d",&frm[i].data);
        frm[i].ack='y';
    }
    return 0;
}

void recvack()
{

```

```

int i;

//rand();

r = rand()%n;
frm[r].ack='n';
for(i=1;i<=n;i++)
{
    if(frm[i].ack=='n')
        printf("\n The frame Number %d is not Recieved",r);
}
}

void resend_gb()
{
    int i;
    printf("\n Resending Frame %d ", r);
    for(i=r;i<=n;i++)
    {
        sleep(2);
        frm[i].ack='y';
        printf("\n The Recieved Frame is %d",frm[i].data);
    }
}

```

C:\Users\admin\Downloads\go\_back\_n.exe

Enter no. of Frames to be sent: 4

Enter data for Frames [1] 1

Enter data for Frames [2] 2

Enter data for Frames [3] 3

Enter data for Frames [4] 4

The frame Number 1 is not Recieved

Resending Frame 1

The Recieved Frame is 1

The Recieved Frame is 2

The Recieved Frame is 3

The Recieved Frame is 4

All Frames sent Successfully

-----  
Process exited after 23.25 seconds with return value 0  
Press any key to continue . . .

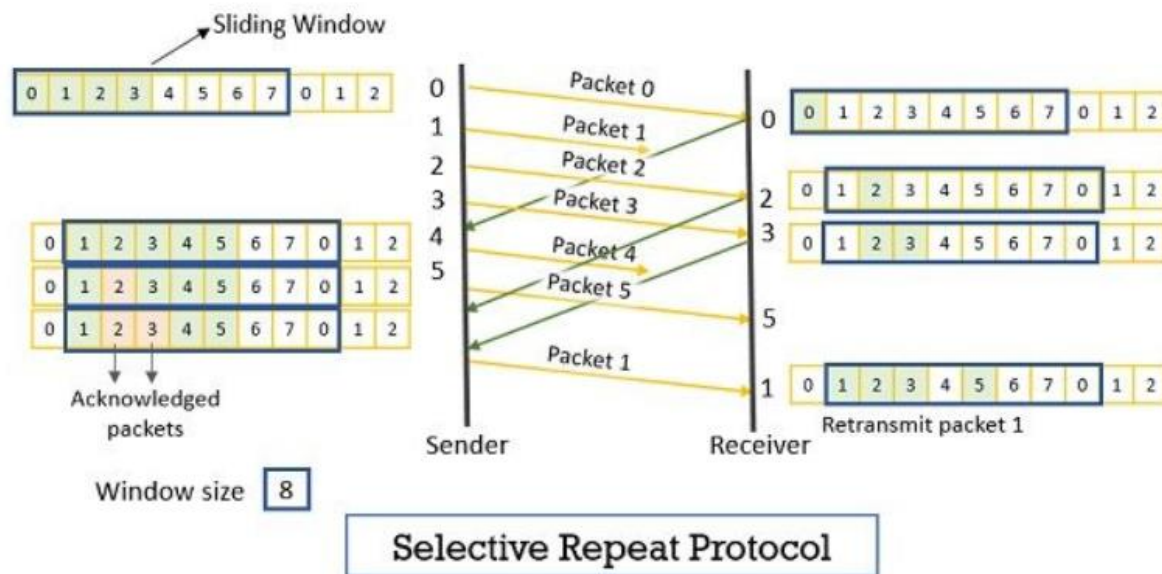
## Experiment : 7

**Aim:** Algorithm to implement Sliding window protocol for Selective repeat.

### Description:

Selective repeat protocol is a sliding window protocol that uses the concept of pipelining where multiple packets can be sent while the sender is waiting for the acknowledgement for the first sent packet. The selective repeat protocol manages error and flows control between the sender and receiver.

### Example:



### Program:

```
#include <stdio.h>
#include<stdlib.h>
#include<math.h>
#include<unistd.h>

int n,r;
struct frame
{
    char ack;
    int data;
}frm[10];

int sender(void);
void recvack(void);
void resend_sr(void);

int main()
{
    sender();
    recvack();
```

```

    resend_sr();
    printf("\n All Frames sent Successfully\n");
}

int sender()
{
    int i;
    printf("\n Enter no. of Frames to be sent: ");
    scanf("%d",&n);
    for(i=1;i<=n;i++)
    {
        printf("\n Enter data for Frames [%d] ", i);
        scanf("%d",&frm[i].data);
        frm[i].ack='y';
    }
    return 0;
}

void recvack()
{
    int i;
    rand();
    r = rand()%n;
    frm[r].ack='n';
    for(i=1;i<=n;i++)
    {
        if(frm[i].ack=='n')
            printf("\n The frame Number %d is not Recieved",r);
    }
}

void resend_sr()
{
    printf("\n Resending Frame %d ", r);
    sleep(2);
    frm[r].ack='y';
    printf("\n The Recieved Frame is %d",frm[r].data);
}

```

C:\Users\admin\Downloads>Selective\_Repeat.exe

Enter no. of Frames to be sent: 5

Enter data for Frames [1] 1

Enter data for Frames [2] 2

Enter data for Frames [3] 3

Enter data for Frames [4] 4

Enter data for Frames [5] 5

The frame Number 2 is not Recieved

Resending Frame 2

The Recieved Frame is 2

All Frames sent Successfully

-----

Process exited after 6.581 seconds with return value 0

Press any key to continue . . .



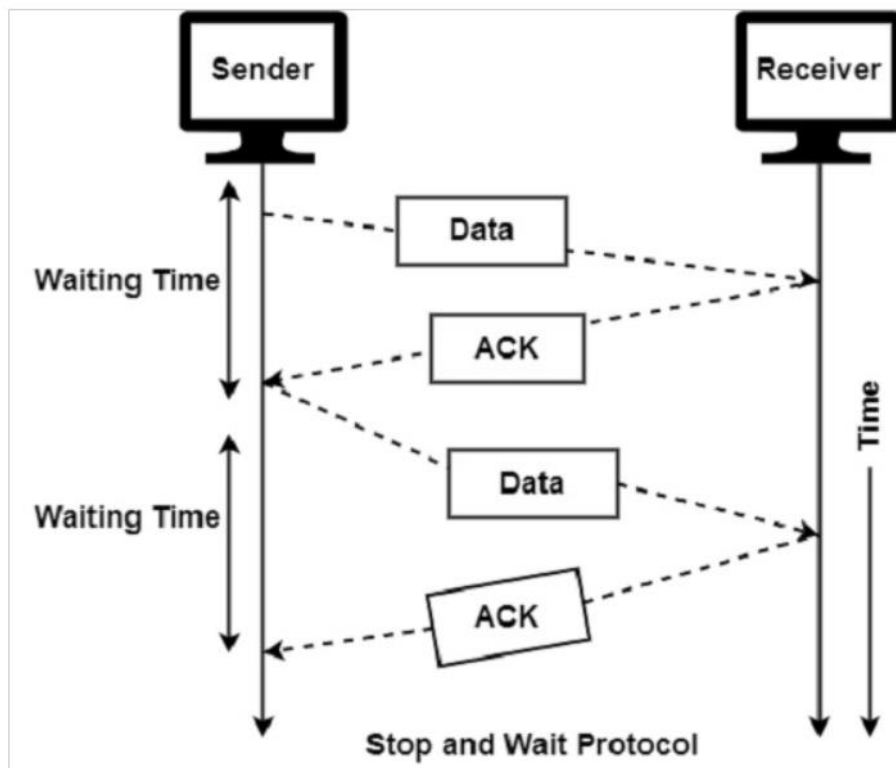
## Experiment : 8

**Aim:** Algorithm to implement Stop and Wait Protocol

**Description:**

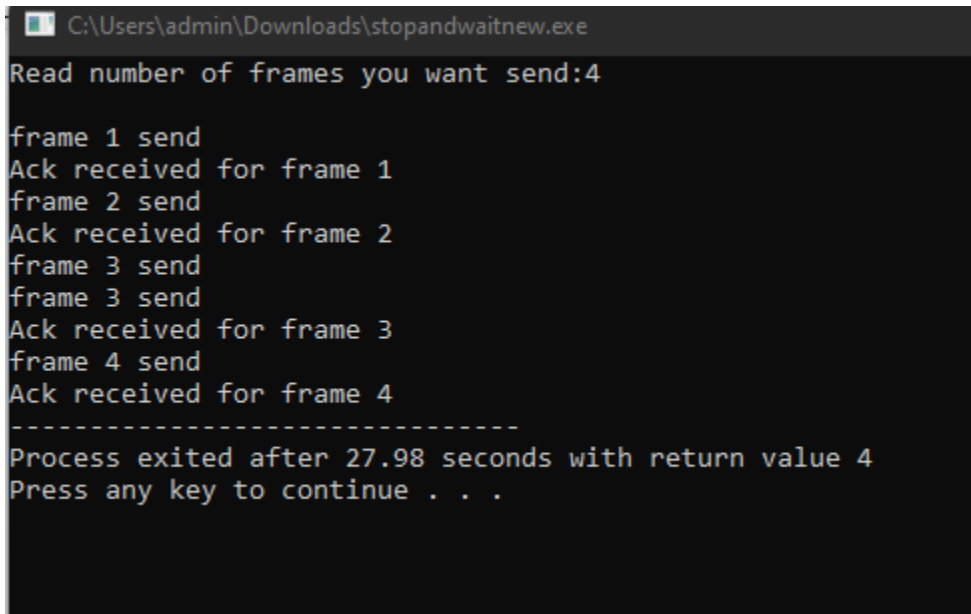
It is the simplest flow control method. In this, the sender will transmit one frame at a time to the receiver. The sender will stop and wait for the acknowledgement from the receiver. This time (i.e. the time joining message transmitting and acknowledgement receiving) is the sender's waiting time, and the sender is idle during this time.

When the sender gets the acknowledgement (ACK), it will send the next data packet to the receiver and wait for the disclosure again, and this process will continue as long as the sender has the data to send. While sending the data from the sender to the receiver, the data flow needs to be controlled. If the sender is transmitting the data at a rate higher than the receiver can receive and process it, the data will get lost.



### Program:

```
int main()
{
    int n,i,wait;
    printf("Read number of frames you want send:");
    scanf("%d",&n);
    for(i=1;i<=n;i++)
    {
        printf("\nframe %d send",i);
        wait=rand()%8;
        sleep(5);
        if(wait>5)
        {
            i=i-1;
            continue;
        }
        else
            printf("\nAck received for frame %d",i);
    }
}
```



```
C:\Users\admin\Downloads\stopandwaitnew.exe
Read number of frames you want send:4

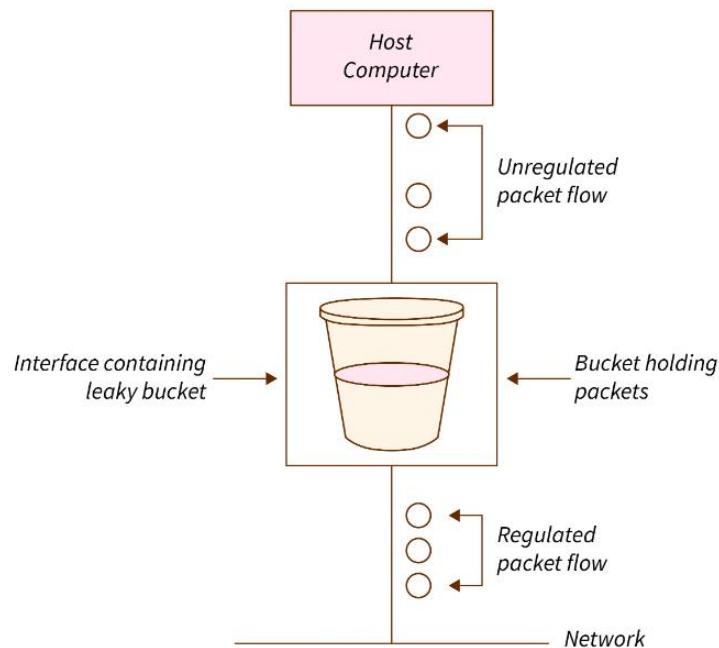
frame 1 send
Ack received for frame 1
frame 2 send
Ack received for frame 2
frame 3 send
frame 3 send
Ack received for frame 3
frame 4 send
Ack received for frame 4
-----
Process exited after 27.98 seconds with return value 4
Press any key to continue . . .
```

## Experiment : 9

**Aim:** Algorithm to implement congestion control using leaky bucket algorithm

### Description:

The leaky bucket algorithm is a method of congestion control where multiple packets are stored temporarily. These packets are sent to the network at a constant rate that is decided between the sender and the network. This algorithm is used to implement congestion control through traffic shaping in data networks.



### Program:

```
#include<stdio.h>
#include<unistd.h>
#define bucketSize 512
void bktInput(int packetSize,int output)
{
    if(packetSize>bucketSize)
        printf("\nBucket overflow");
    else
    {
        sleep(2);
        while(packetSize>output)
        {
            printf("\n\t%d bytes outputted.",output);
            packetSize=packetSize-output;
            sleep(2);
        }
        if(packetSize>0)
```

```

        printf("\n\t\tLast %d bytes sent\t",packetSize);
        printf("\n\t\tBucket output successful");
    }
}
int main()
{
    int op,i,pktSize,time;
    //randomize();
    printf("Enter output rate : ");
    scanf("%d",&op);
    for(i=1;i<=5;i++)
    {
        time=rand()% 10+1;
        printf("\nwaiting... %d second(s)",time);
        sleep(time);
        //printf("\nwaited\n");
        pktSize=rand()% 1000+1;
        printf("\nPacket no %d,\tPacket size = %d",i,pktSize);
        bktInput(pktSize,op);
    }
    return 0;
}

```

C:\Users\admin\Downloads\leakybuck.exe

Enter output rate : 100

waiting... 2 second(s)

Packet no 1, Packet size = 468

100 bytes outputted.

100 bytes outputted.

100 bytes outputted.

100 bytes outputted.

Last 68 bytes sent

Bucket output successful

waiting... 5 second(s)

Packet no 2, Packet size = 501

100 bytes outputted.

100 bytes outputted.

100 bytes outputted.

100 bytes outputted.

100 bytes outputted.

Last 1 bytes sent

Bucket output successful

waiting... 10 second(s)

Packet no 3, Packet size = 725

Bucket overflow

waiting... 9 second(s)

Packet no 4, Packet size = 359

100 bytes outputted.

100 bytes outputted.

100 bytes outputted.

Last 59 bytes sent

Bucket output successful

waiting... 3 second(s)

Packet no 5, Packet size = 465

100 bytes outputted.

100 bytes outputted.

100 bytes outputted.

100 bytes outputted.

Last 65 bytes sent

Bucket output successful

-----  
Process exited after 71.71 seconds with return value 0

Press any key to continue . . .

## Experiment : 10

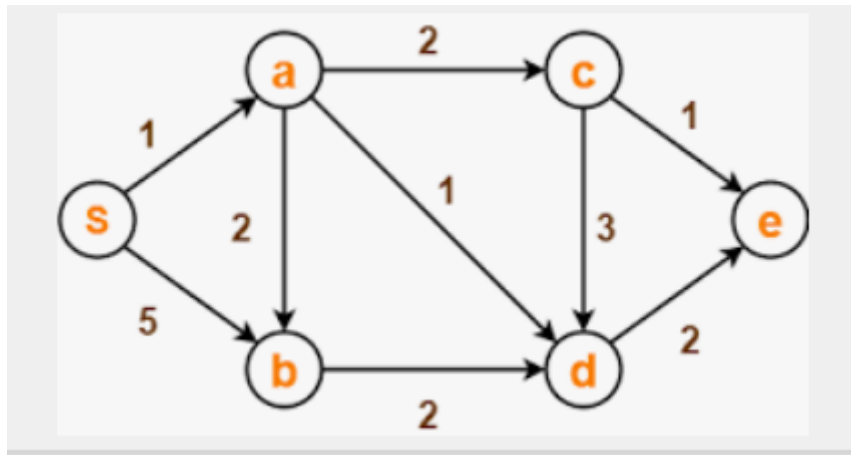
**Aim:** Algorithm to implement Dijkstra's algorithm to compute the Shortest path through a graph

### Description:

The Dijkstra's algorithm finds the shortest path from a particular node, called the source node to every other node in a connected graph. It produces a shortest path tree with the source node as the root. It is profoundly used in computer networks to generate optimal routes with the aim of minimizing routing costs.

Input – A graph representing the network; and a source node, s

Output – A shortest path tree, spt[], with s as the root node.



### Program:

```
#include<stdio.h>
#include<conio.h>
void main()
{
    int path[5][5],i,j,min,a[5][5],p,st=1,ed=5,stp,edp,t[5],index;
    printf("enter the cost matrix\n");
    for(i=0;i<5;i++)
        for(j=0;j<5;j++)
            scanf("%d",&a[i][j]);
    printf("enter the paths\n");
    scanf("%d",&p);
    printf("enter possible paths\n");
    for(i=0;i<p;i++)
        for(j=0;j<5;j++)
            scanf("%d",&path[i][j]);
    for(i=0;i<p;i++)
    {
        t[i]=0;stp=st;
        for(j=0;j<5;j++)
        {
```

```

    edp=path[i][j+1];

    t[i]=t[i]+a[stp][edp];
    if(edp==ed)
        break;
    else
        stp=edp;
    }
}
min=t[st];
index=st;
for(i=0;i<p;i++)
{
    if(min>t[i])
    {
        min=t[i];
        index=i;
    }
}
printf("minimum cost %d",min);
printf("\n minimum cost path ");
for(i=0;i<5;i++)
{
    printf("--> %d",path[index][i]);
    if(path[index][i]==ed)
        break;
}
getch();
}

```

Output:

```

enter the cost matrix
0 1 4 2 0
1 0 3 7 0
4 3 0 5 0
2 7 5 0 6
0 0 0 6 0
enter the paths
4
enter possible paths
1 2 3 4 5
1 2 4 5 0
1 3 4 5 0
1 4 5 0 0
minimum cost 32766
    minimum cost path --> 1--> 4--> 5
...Program finished with exit code 255
Press ENTER to exit console.

```

## Experiment : 11

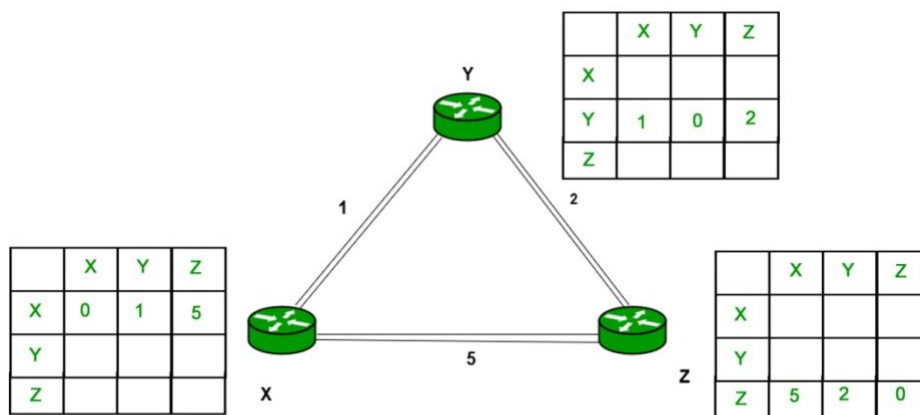
**Aim:** Algorithm to implement Distance vector routing algorithm by obtaining routing table at each node

### Description:

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).

### Example:

Consider 3-routers X, Y and Z as shown in figure. Each router have their routing table. Every routing table will contain distance to the destination nodes.



Consider router X, X will share its routing table to neighbors and neighbors will share their routing table to it to X and distance from node X to destination will be calculated using Bellman-Ford equation.

$$D_x(y) = \min \{ C(x,v) + D_v(y) \} \text{ for each node } y \in N$$

As we can see that distance will be less going from X to Z when Y is intermediate node(hop) so it will be updated in routing table X.

### Program:

```
#include<stdio.h>
#include<conio.h>
struct node
{
    unsigned dist[20];
    unsigned from[20];
}rt[10];
void main()
{
    int dmat[20][20];
    int n,i,j,k,count=0;
    printf("\nEnter the number of nodes : ");
```



```

scanf("%d",&n);

printf("Enter the cost matrix :\n");
for(i=0;i<n;i++)
{
for(j=0;j<n;j++)
{
scanf("%d",&dmat[i][j]);
dmat[i][i]=0;
rt[i].dist[j]=dmat[i][j];
rt[i].from[j]=j;
}
}
do
{
count=0;
for(i=0;i<n;i++)
{
for(j=0;j<n;j++)
{
for(k=0;k<n;k++)
{
if(rt[i].dist[j]>dmat[i][k]+rt[k].dist[j])
{
rt[i].dist[j]=rt[i].dist[k]+rt[k].dist[j];
rt[i].from[j]=k;
count++;
}
}
}
}
}while(count!=0);
for(i=0;i<n;i++)
{
printf("\nState value for router %d is \n",i+1);
for(j=0;j<n;j++)
{
printf("\nnode %d via %d Distance%d",j+1,rt[i].from[j]+1,rt[i].dist[j]);
}
}
printf("\n");
}

```

Output:

```
Enter the number of nodes : 3
```

```
Enter the cost matrix :
```

```
0 2 4
```

```
2 0 5
```

```
4 5 0
```

```
State value for router 1 is
```

```
node 1 via 1 Distance0
```

```
node 2 via 2 Distance2
```

```
node 3 via 3 Distance4
```

```
State value for router 2 is
```

```
node 1 via 1 Distance2
```

```
node 2 via 2 Distance0
```

```
node 3 via 3 Distance5
```

```
State value for router 3 is
```

```
node 1 via 1 Distance4
```

```
node 2 via 2 Distance5
```

```
node 3 via 3 Distance0
```

```
...Program finished with exit code 10
```

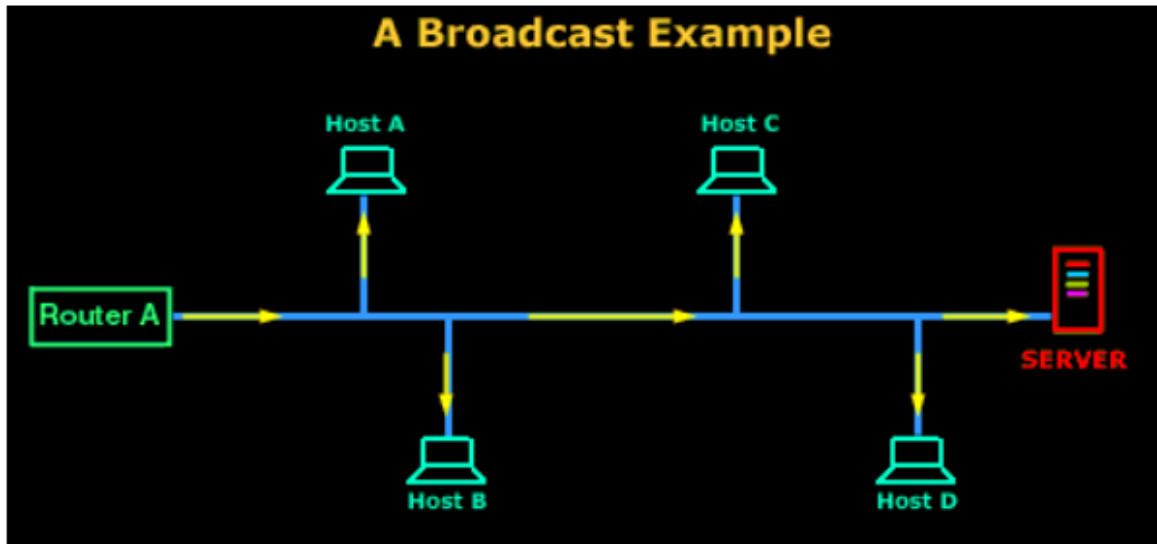
```
Press ENTER to exit console. 
```

## Experiment : 12

**Aim:** Algorithm to implement Broadcast tree by taking subnet of hosts

### Description:

This technique is widely used because it is simple and easy to understand. The idea of this algorithm is to build a graph of the subnet with each node of the graph representing a router and each arc of the graph representing a communication line. To choose a route between a given pair of routers the algorithm just finds the broadcast between them on the graph.



### Program:

```
#include<stdio.h>
#include<conio.h>
struct node
{
    unsigned dist[20];
    unsigned from[20];
}rt[10];
void main()
{
    int dmat[20][20];
    int n,i,j,k,count=0;
    printf("\nEnter the number of nodes : ");
    scanf("%d",&n);
    printf("Enter the cost matrix :\n");
    for(i=0;i<n;i++)
    {
        for(j=0;j<n;j++)
        {
            scanf("%d",&dmat[i][j]);
            dmat[i][i]=0;
            rt[i].dist[j]=dmat[i][j];
        }
    }
}
```

```

    rt[i].from[j]=j;
    }

}
do
{
    count=0;
    for(i=0;i<n;i++)
    {
        for(j=0;j<n;j++)
        {
            for(k=0;k<n;k++)
            {
                if(rt[i].dist[j]>dmat[i][k]+rt[k].dist[j])
                {
                    rt[i].dist[j]=rt[i].dist[k]+rt[k].dist[j];
                    rt[i].from[j]=k;
                    count++;
                }
            }
        }
    } while(count!=0);
    for(i=0;i<n;i++)
    {
        printf("\nState value for router %d is \n",i+1);
        for(j=0;j<n;j++)
        {
            printf("\nnode %d via %d Distance%d",j+1,rt[i].from[j]+1,rt[i].dist[j]);
        }
    }
    printf("\n");
}

```

Output:

```
Enter the number of nodes : 3
```

```
Enter the cost matrix :
```

```
0 2 4
```

```
2 0 5
```

```
4 5 0
```

```
State value for router 1 is
```

```
node 1 via 1 Distance0
```

```
node 2 via 2 Distance2
```

```
node 3 via 3 Distance4
```

```
State value for router 2 is
```

```
node 1 via 1 Distance2
```

```
node 2 via 2 Distance0
```

```
node 3 via 3 Distance5
```

```
State value for router 3 is
```

```
node 1 via 1 Distance4
```

```
node 2 via 2 Distance5
```

```
node 3 via 3 Distance0
```

```
...Program finished with exit code 10
```

```
Press ENTER to exit console. 
```

## Augmented Experiments

### Experiment : 13

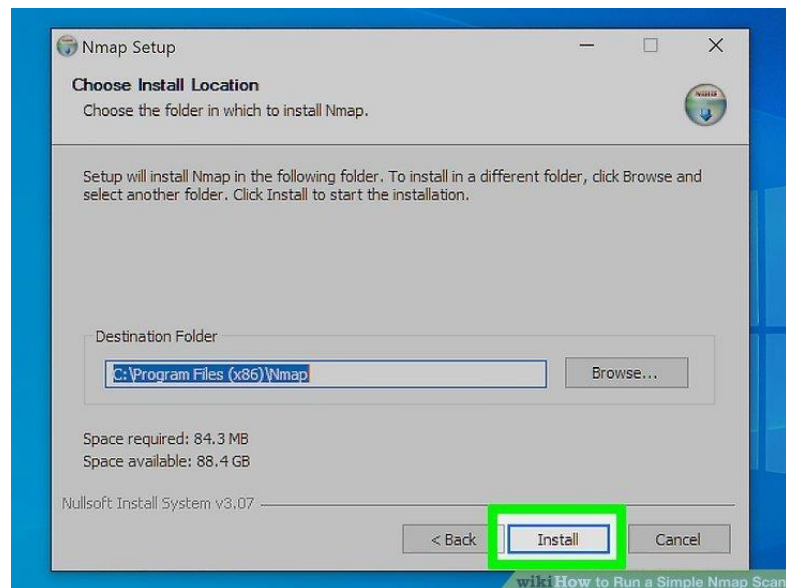
#### Aim: How to run Nmap scan

#### Description:

Ensuring that your router is protected from unwanted intruders is one of the foundations of a secure network. One of the basic tools for this job is Nmap, or Network Mapper. This program will scan a target and report which ports are open and which are closed, among other things. Security specialists use this program to test the security of a network. To learn how to use it yourself, see

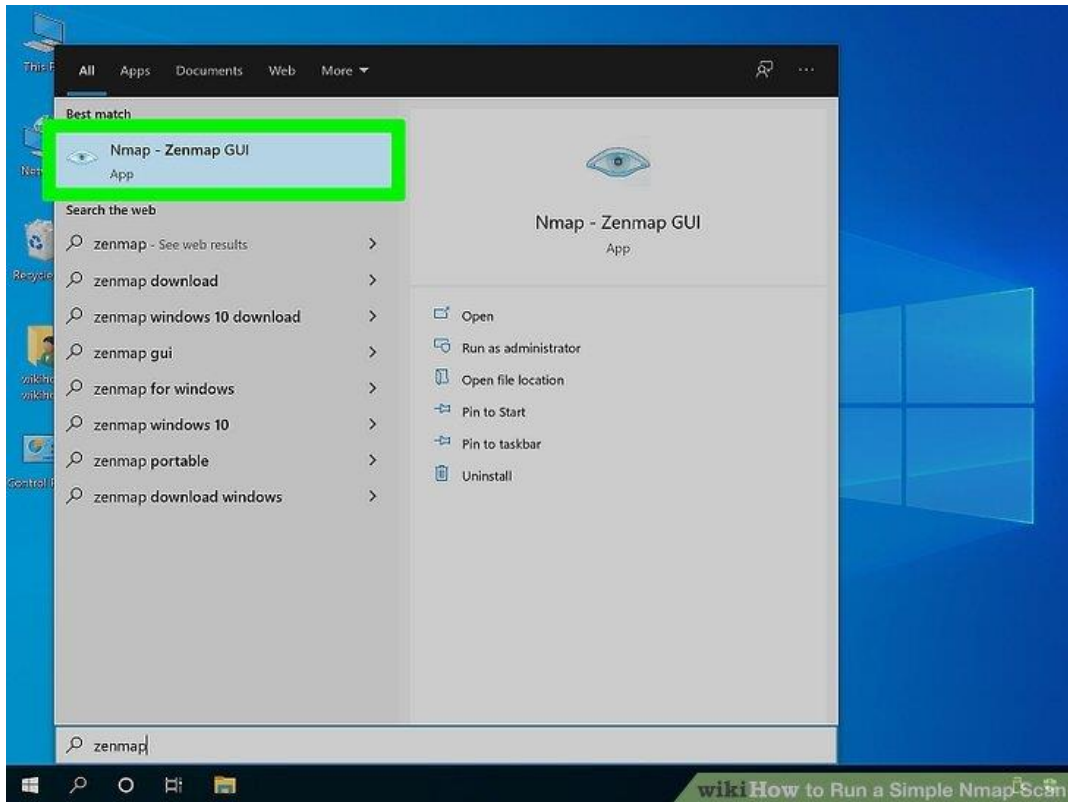
#### Step 1 below.

1. Download the Nmap installer. This can be found for free from the developer's website. It is highly recommended that you download directly from the developer to avoid any potential viruses or fake files. Downloading the Nmap installer includes Zenmap, the graphical interface for Nmap which makes it easy for newcomers to perform scans without having to learn command lines.

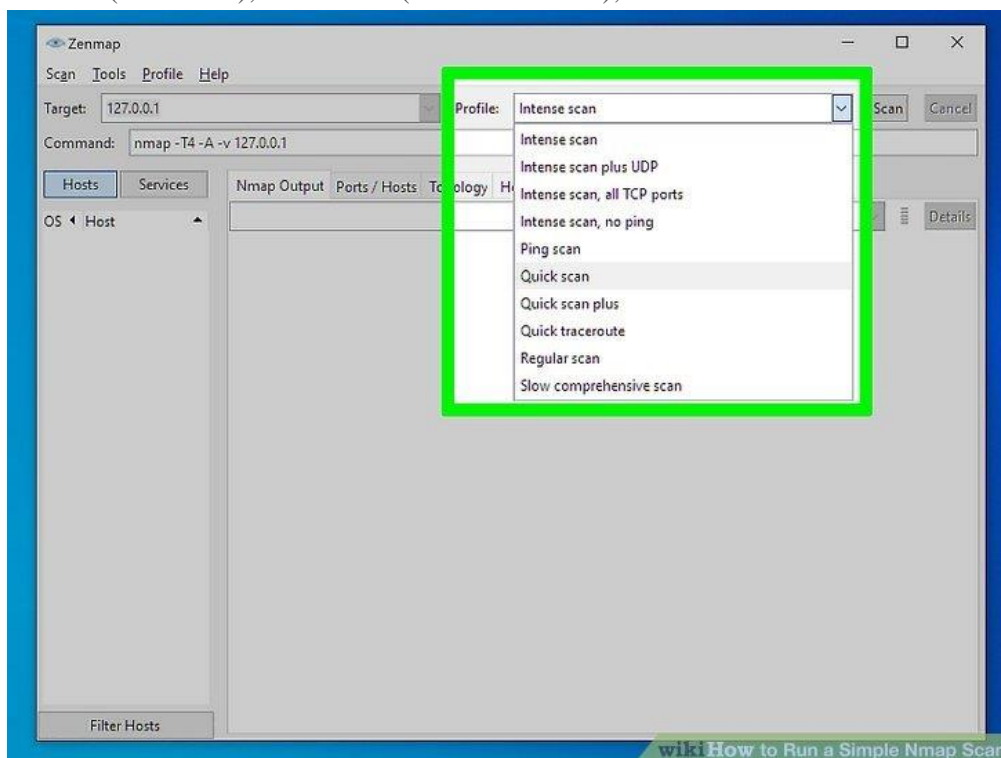


2. **Install Nmap.** Run the installer once it is finished downloading. You will be asked which components you would like to install. In order to get the full benefit of Nmap, keep all of these checked. Nmap will

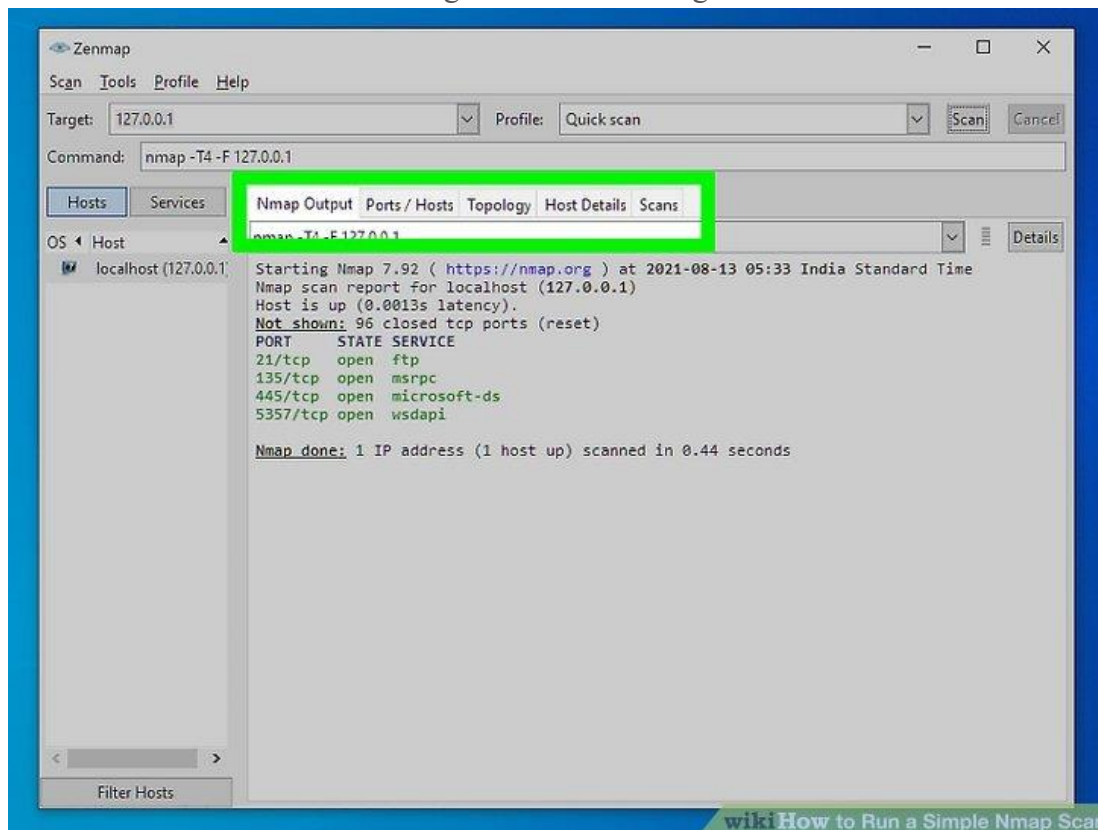
not install any adware or spyware.



3. **Run the “Nmap – Zenmap” GUI program.** If you left your settings at default during installation, you should be able to see an icon for it on your desktop. If not, look in your Start menu. Opening Zenmap will start the program.
4. **Enter in the target for your scan.** The Zenmap program makes scanning a fairly simple process. The first step to running a scan is choosing your target. You can enter a domain (example.com), an IP address (127.0.0.1), a network (192.168.1.0/24), or a combination of those.



5. **Choose your Profile.** Profiles are preset groupings of modifiers that change what is scanned. The profiles allow you to quickly select different types of scans without having to type in the modifiers on the command line. Choose the profile that best fits your needs
6. **Click Scan to start scanning.** The active results of the scan will be displayed in the Nmap Output tab. The time the scan takes will depend on the scan profile you chose, the physical distance to the target, and the target's network configuration.



7. **Read your results.** Once the scan is finished, you'll see the message "Nmap done" at the bottom of the Nmap Output tab. You can now check your results, depending on the type of scan you performed. All of the results will be listed in the main Nmap Output tab, but you can use the other tabs to get a better look at specific data.[2]
- **Ports/Hosts** - This tab will show the results of your port scan, including the services for those ports.
  - **Topology** - This shows the traceroute for the scan you performed. You can see how many hops your data goes through to reach the target.
  - **Host Details** - This shows a summary of your target learned through scans, such as the number of ports, IP addresses, hostnames, operating systems, and more.



- **Scans** - This tab stores the commands of your previously-run scans. This allows you to quickly re-scan with a specific set of parameters.

```
root@kali:~# nslookup scanme.nmap.org
Server:          192.168.29.2
Address:         192.168.29.2#53

Non-authoritative answer:
Name:   scanme.nmap.org
Address: 45.33.32.156
Name:   scanme.nmap.org
Address: 2600:3c01::f03c:91ff:fe18:bb2f

root@kali:~# nslookup 45.33.32.156
156.32.33.45.in-addr.arpa      name = scanme.nmap.org.

Authoritative answers can be found from:

root@kali:~#
```

## Experiment : 14

**Aim:** Operating System Detection using Nmap

### Description:

One of Nmap's best-known features is remote OS detection using TCP/IP stack fingerprinting. Nmap sends a series of TCP and UDP packets to the remote host and examines practically every bit in the responses. After performing dozens of tests such as TCP ISN sampling, TCP options support and ordering, IP ID sampling, and the initial window size check, Nmap compares the results to its nmap-os-db database of more than 2,600 known OS fingerprints and prints out the OS details if there is a match. Each fingerprint includes a freeform textual description of the OS, and a classification which provides the vendor name (e.g. Sun), underlying OS (e.g. Solaris), OS generation (e.g. 10), and device type (general purpose, router, switch, game console, etc). Most fingerprints also have a Common Platform Enumeration (CPE) representation, like `cpe:/o:linux:linux_kernel:2.6`.

If Nmap is unable to guess the OS of a machine, and conditions are good (e.g. at least one open port and one closed port were found), Nmap will provide a URL you can use to submit the fingerprint if you know (for sure) the OS running on the machine. By doing this you contribute to the pool of operating systems known to Nmap and thus it will be more accurate for everyone.

OS detection enables some other tests which make use of information that is gathered during the process anyway. One of these is TCP Sequence Predictability Classification. This measures approximately how hard it is to establish a forged TCP connection against the remote host. It is useful for exploiting source-IP based trust relationships (rlogin, firewall filters, etc) or for hiding the source of an attack. This sort of spoofing is rarely performed any more, but many machines are still vulnerable to it.

Enables OS detection, as discussed above. Alternatively, you can use `-A` to enable OS detection along with other things.

`--osscan-limit` (Limit OS detection to promising targets)

OS detection is far more effective if at least one open and one closed TCP port are found. Set this option and Nmap will not even try OS detection against hosts that do not meet this criteria. This can save substantial time, particularly on `-Pn` scans against many hosts. It only matters when OS detection is requested with `-O` or `-A`.

`--osscan-guess; --fuzzy` (Guess OS detection results)

When Nmap is unable to detect a perfect OS match, it sometimes offers up near-matches as possibilities. The match has to be very close for Nmap to do this by default. Either of these (equivalent) options make Nmap guess more aggressively. Nmap will still tell you when an imperfect match is printed and display its confidence level (percentage) for each guess.

`--max-os-tries` (Set the maximum number of OS detection tries against a target)

When Nmap performs OS detection against a target and fails to find a perfect match, it usually repeats the attempt. By default, Nmap tries five times if conditions are favorable for OS fingerprint submission,

and twice when conditions aren't so good. Specifying a lower --max-os-tries value (such as 1) speeds Nmap up, though you miss out on retries which could potentially identify the OS. Alternatively, a high value may be set to allow even more retries when conditions are favorable. This is rarely done, except to generate better fingerprints for submission and integration into the Nmap OS database.

**Program:**

# **nmap -O -v scanme.nmap.org**

Starting Nmap ( <https://nmap.org> )

Nmap scan report for scanme.nmap.org (74.207.244.221)

Not shown: 994 closed ports

PORT	STATE	SERVICE
------	-------	---------

22/tcp	open	ssh
--------	------	-----

80/tcp	open	http
--------	------	------

646/tcp	filtered	ldp
---------	----------	-----

1720/tcp	filtered	H.323/Q.931
----------	----------	-------------

9929/tcp	open	nping-echo
----------	------	------------

31337/tcp	open	Elite
-----------	------	-------

Device type: general purpose

Running: Linux 2.6.X

OS CPE: cpe:/o:linux:linux\_kernel:2.6.39

OS details: Linux 2.6.39

Uptime guess: 1.674 days (since Fri Sep 9 12:03:04 2011)

Network Distance: 10 hops

TCP Sequence Prediction: Difficulty=205 (Good luck!)

IP ID Sequence Generation: All zeros

Read data files from: /usr/local/bin/../../share/nmap

Nmap done: 1 IP address (1 host up) scanned in 5.58 seconds

Raw packets sent: 1063 (47.432KB) | Rcvd: 1031 (41.664KB)

## Experiment : 15

**Aim: Do the following using NS2 Simulator i. NS2 Simulator-Introduction ii. Simulate to Find the Number of Packets Dropped iii. Simulate to Find the Number of Packets Dropped by TCP/UDP**

### **Description:**

The tool used for much research-level network simulations is ns, for network simulator and originally developed at the Information Sciences Institute. The ns simulator grew out of the REAL simulator developed by Srinivasan Keshav [SK88]; later development work was done by the Network Research Group at the Lawrence Berkeley National Laboratory.

We will describe in this chapter the ns-2 simulator, hosted at [www.isi.edu/nsnam/ns](http://www.isi.edu/nsnam/ns). There is now also an ns-3 simulator, available at [www.nsnam.org](http://www.nsnam.org). Because ns-3 is not backwards-compatible with ns-2 and the programming interface has changed considerably, we take the position that ns-3 is an entirely different package, though one likely someday to supercede ns-2 entirely. While there is a short introduction to ns-3 in this book (32 The ns-3 Network Simulator), its use is arguably quite a bit more complicated for beginners, and the particular simulation examples presented below are well-suited to ns-2. While ns-3 supports more complex and realistic modeling, and is the tool of choice for serious research, this added complexity comes at a price in terms of configuration and programming. The standard ns-2 tracefile format is also quite easy to work with using informal scripting.

### **Code:**

```
# basic1.tcl simulation: A---R---B

#Create a simulator object

set ns [new Simulator]

#Open the nam file basic1.nam and the variable-trace file basic1.tr

set namfile [open basic1.nam w]

$ns namtrace-all $namfile

set tracefile [open basic1.tr w]

$ns trace-all $tracefile

#Define a 'finish' procedure

proc finish { } {

    global ns namfile tracefile

    $ns flush-trace

    close $namfile

    close $tracefile

    exit 0
```

```

}

#Create the network nodes
set A [$ns node]
set R [$ns node]
set B [$ns node]

#Create a duplex link between the nodes
$ns duplex-link $A $R 10Mb 10ms DropTail
$ns duplex-link $R $B 800Kb 50ms DropTail

# The queue size at $R is to be 7, including the packet being sent
$ns queue-limit $R $B 7

# some hints for nam
# color packets of flow 0 red
$ns color 0 Red

$ns duplex-link-op $A $R orient right
$ns duplex-link-op $R $B orient right
$ns duplex-link-op $R $B queuePos 0.5

# Create a TCP sending agent and attach it to A
set tcp0 [new Agent/TCP/Reno]

# We make our one-and-only flow be flow 0
$tcp0 set class_ 0
$tcp0 set window_ 100
$tcp0 set packetSize_ 960
$ns attach-agent $A $tcp0

# Let's trace some variables
$tcp0 attach $tracefile
$tcp0 tracevar cwnd_
$tcp0 tracevar ssthresh_
$tcp0 tracevar ack_
$tcp0 tracevar maxseq_

#Create a TCP receive agent (a traffic sink) and attach it to B

```

```
set end0 [new Agent/TCPSink]
$ns attach-agent $B $end0

#Connect the traffic source with the traffic sink
$ns connect $tcp0 $end0
#Schedule the connection data flow; start sending data at T=0, stop at T=10.0
set myftp [new Application/FTP]
$myftp attach-agent $tcp0
$ns at 0.0 "$myftp start"
$ns at 10.0 "finish"
#Run the simulation
$ns run
```

## Experiment : 16

**Aim:** Do the following using NS2 Simulator i.Simulate to Find the Number of Packets Dropped due to Congestion ii.Simulate to Compare Data Rate& Throughput.

### Description:

In packet networks the problem of congestion can be explained as when too many packets try to access the same router buffer as a result of which some amount of packets being dropped. One more definition of congestion is that when user sends data into the network at a rate higher than allowed by the networking resources. Congestion can occur when there are many clients

### Congestion Prevention

It is the preliminary step which focuses to keep the network neat and clean

### Congestion Control

It is the post step in which congestion has already degraded the performance of the network. In the worst case, one might need to stop the network, which is obviously not easy in normal circumstances

### Packet Loss Rate

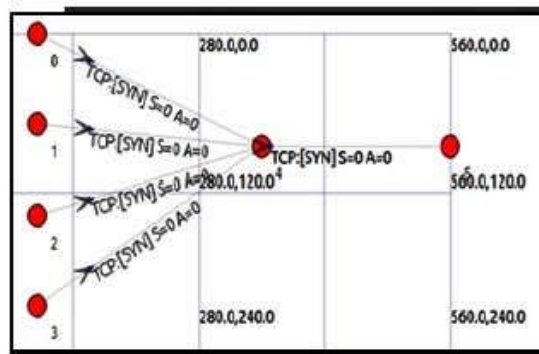
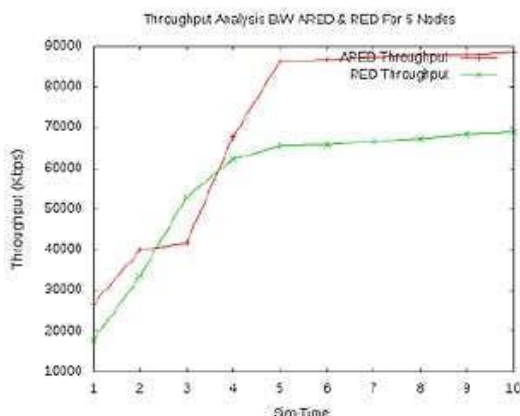
This can be defined as the number of packets lost during the transmission in per unit time.

### Packet Delivery Ratio (PDR)

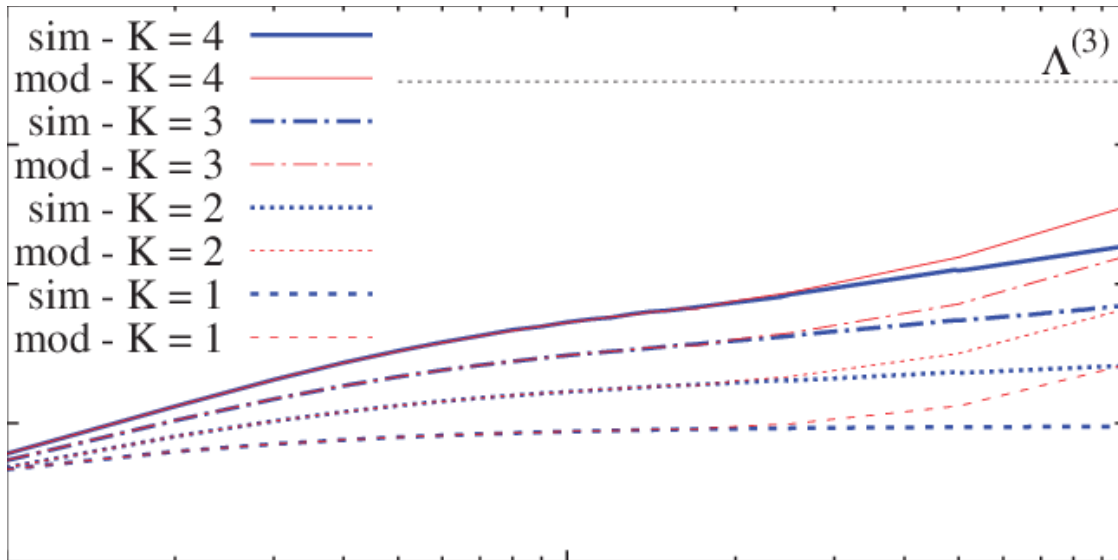
The ratio of packets that are successfully delivered to a destination compared to the number of packets that have been sent out by the sender

### Packet Loss Ratio

It is measured as a percentage of packets lost with respect to packets sent. The various packet parameters have been shown below. It shows that ARED has less packet drop rate as compared to RED. 16375 packets were received out of 16481 packets delivered from all connections were lost during simulation time when RED was used, which resulted in 106 number of packets dropped. 16512 packets received out of 16578 during the simulation when the router buffer operates by using ARED algorithm. This resulted in 66 dropped packets.



ii. Throughput comparison (model vs simulation) in the reference system, with  $BAP = \infty$ ,  $D = 0$ , as function of BSTA, for different number of stations



### Single-sender Throughput Experiments

According to the theoretical analysis in [19.7 TCP and Bottleneck Link Utilization](#), a queue size of close to zero should yield about a 75% bottleneck utilization, a queue size such that the mean cwnd equals the transit capacity should yield about 87.5%, and a queue size equal to the transit capacity should yield close to 100%. We now test this.

We first increase the per-link propagation times in the basic1.tcl simulation above to 50 and 100 ms:

```
$ns duplex-link $A $R 10Mb 50ms DropTail
$ns duplex-link $R $B 800Kb 100ms DropTail
```