1. Implement a point to point network with four nodes and duplex links between them. Analyze the network performance by setting the queue size and varying the bandwidth.

```
set ns [new Simulator]

set tf [open lab1.tr w]
$ns trace-all $tf

set nf [open lab1.nam w]
$ns namtrace-all $nf

#finish procedure
proc finish {} {
     global tf nf ns
     $ns flush-trace
     close $tf
     close $nf
     exec nam lab1.nam &
     exec awk -f throughput.awk lab1.tr &
     exit 0
}
# create nodes
set n0 [$ns node]
set n1 [$ns node]
set n2 [$ns node]
set n3 [$ns node]

#setup the links
$ns duplex-link $n0 $n1 0.3Mb 20ms DropTail
#vary bandwidth 0.3, 0.4, 0.5, 0.7
$ns duplex-link $n1 $n2 0.3Mb 20ms DropTail
#vary bandwidth 0.3, 0.4, 0.5, 0.7
$ns duplex-link $n2 $n3 0.3Mb 20ms DropTail
#vary bandwidth 0.3, 0.4, 0.5, 0.7

# setup queue
$ns queue-limit $n0 $n1 50
$ns queue-limit $n1 $n2 20
$ns queue-limit $n2 $n3 20

# Setup traffic type (UDP)
# setup source
set udp0 [new Agent/UDP]
$ns attach-agent $n0 $udp0

set cbr0 [new Application/Traffic/CBR]
$cbr0 attach-agent $udp0
$cbr0 set packetSize_ 500
$cbr0 set interval_ 0.005


#setup null (destination)
```

```
set null0 [new Agent/Null]
$ns attach-agent $n3 $null0
$ns connect $udp0 $null0

# set timings
$ns at 0.1 "$cbr0 start"
$ns at 4.5 "$cbr0 stop"
$ns at 5.0 "finish"
$ns run
```

AWK Script: File name --> throughput.awk

```
BEGIN{
stime=0
ftime=0
flag=0
fsize=0
throughput=0
latency=0
}
{
if($1=="r" && $4==3)
     {
     fsize+=$6
     if(flag==0)
          {
          stime=$2
          flag=1
          }
     ftime=$2
     }
}
END{
latency=ftime-stime
throughput = (fsize*8)/latency
printf("\n Throughput : %f bps", throughput)
}
```

Output:

```
Throughput : 300835.633647 bps
```

2. Implement a four node point to point network with links n0-n2, n1-n2 and n2-n3. Apply TCP agent between n0-n3 and UDP between n1-n3. Apply relevant applications over TCP and UDP agents changing the parameter and determine the number of packets sent by TCP/UDP.

```
set ns [new Simulator]

set f [open lab2.tr w]
$ns trace-all $f

set nf [open lab2.nam w]
$ns namtrace-all $nf

$ns color 1 "Blue"
$ns color 2 "Red"

proc finish {} {
     global ns f nf
     $ns flush-trace
     close $f
     close $nf
     exec nam lab2.nam &
     exec awk -f tcp_packet.awk lab2.tr &
     exec awk -f udp_packet.awk lab2.tr &
     exit 0
}
set n0 [$ns node]
set n1 [$ns node]
set n2 [$ns node]
set n3 [$ns node]

$ns duplex-link $n0 $n2 2Mb 10ms DropTail
$ns duplex-link $n1 $n2 2Mb 10ms DropTail
$ns duplex-link $n2 $n3 2.75Mb 20ms DropTail
$ns queue-limit $n2 $n3 50

set tcp0 [new Agent/TCP]
$ns attach-agent $n0 $tcp0
$tcp0 set class_ 1
set ftp0 [new Application/FTP]
$ftp0 attach-agent $tcp0

set sink [new Agent/TCPSink]
$ns attach-agent $n3 $sink
$ns connect $tcp0 $sink

set udp0 [new Agent/UDP]
$ns attach-agent $n1 $udp0
$udp0 set class_ 2
set cbr0 [new Application/Traffic/CBR]
$cbr0 attach-agent $udp0
$cbr0 set packetSize_ 1000
```

```
$cbr0 set interval_ 0.005

set null0 [new Agent/Null]
$ns attach-agent $n3 $null0
$ns connect $udp0 $null0

$ns at 0.1 "$cbr0 start"
$ns at 1.0 "$ftp0 start"
$ns at 4.0 "$ftp0 stop"
$ns at 4.5 "$cbr0 stop"
$ns at 5.0 "finish"
$ns run
```

AWK Script: File name --> tcp_packet.awk

```
BEGIN{
fsize=0
}
{
if($1=="r" && $4==3 && $5=="tcp")
     {
     fsize+=1
     }
}
END{
printf("\n No. Of TCP packets : %f", fsize)
}
```

AWK Script: File name --> udp_packet.awk

```
BEGIN{
fsize=0
}
{
if($1=="r" && $4==3 && $5=="cbr")
     {
     fsize+=1
     }
}
END{
printf("\n No. Of UDP packets : %f", fsize)
}
```

Output:

```
No. Of TCP packets : 418.000000
No. Of UDP packets : 881.000000
```

3. Implement Ethernet LAN using n (6-10) nodes. Compare the throughput by changing the error rate and data rate.

```
set ns [new Simulator]
set tf [open lab3.tr w]
$ns trace-all $tf
set nf [open lab3.nam w]
$ns namtrace-all $nf

proc finish { } {
     global nf ns tf
     close $nf
     close $tf
     exec nam lab3.nam &
     exec awk -f lab3thr.awk lab3.tr &
     exit 0
}
set n0 [$ns node]
set n1 [$ns node]
set n2 [$ns node]
set n3 [$ns node]
set n4 [$ns node]
set n5 [$ns node]
set n6 [$ns node]

$n1 label "Source"
$n2 label "Error Node"
$n5 label "Destination"

$ns make-lan "$n0 $n1 $n2 $n3" 10Mb 10ms LL Queue/DropTail Mac/802_3
$ns make-lan "$n4 $n5 $n6" 10Mb 10ms LL Queue/DropTail Mac/802_3
$ns duplex-link $n2 $n6 30Mb 100ms DropTail

set udp0 [new Agent/UDP]
$ns attach-agent $n1 $udp0
set cbr0 [ new Application/Traffic/CBR]
$cbr0 attach-agent $udp0

set null5 [new Agent/Null]
$ns attach-agent $n5 $null5
$ns connect $udp0 $null5
$cbr0 set packetSize_ 100
$cbr0 set interval_ 0.001

set err [new ErrorModel]
$ns lossmodel $err $n2 $n6
$err set rate_ 0.1
#vary error rate 0.1, 0.4, 0.5 and 0.7

$ns at 0.1 "$cbr0 start"
$ns at 6.0 "finish"
$ns run
```

AWK Script: File name --> lab3thr.awk

```
BEGIN{
stime=0
ftime=0
flag=0
fsize=0
throughput=0
latency=0
}
{
if($1=="r" && $4==5 && $5=="cbr")
     {
     fsize+=$6
     if(flag==0)
     {
     stime=$2
     flag=1
     }
ftime=$2
}
}
END{
latency=ftime-stime
throughput = (fsize*8)/latency
printf("\n throughput : %f bps", throughput)
}
```

Output:

```
Throughput : 719694.338312 bps
```

4. Implementation of Link state routing algorithm

```
set ns [new Simulator]
$ns rtproto LS

set nf [open lsr.nam w]
$ns namtrace-all $nf

proc finish { } {
     global nf ns
     $ns flush-trace
     close $nf
     exec nam lsr.nam &
     exit 0
}

set node1 [$ns node]
set node2 [$ns node]
set node3 [$ns node]
set node4 [$ns node]
set node5 [$ns node]
set node6 [$ns node]
set node7 [$ns node]

$node1 label "node 1"
$node2 label "node 2"
$node3 label "node 3"
$node4 label "node 4"
$node5 label "node 5"
$node6 label "node 6"
$node7 label "node 7"
$node1 label-color blue
$node2 label-color red
$node3 label-color red
$node4 label-color blue
$node5 label-color blue
$node6 label-color blue
$node7 label-color blue

$ns duplex-link $node1 $node2 1.5Mb 10ms DropTail
$ns duplex-link $node2 $node3 1.5Mb 10ms DropTail
$ns duplex-link $node3 $node4 1.5Mb 10ms DropTail
$ns duplex-link $node4 $node5 1.5Mb 10ms DropTail
$ns duplex-link $node5 $node6 1.5Mb 10ms DropTail
$ns duplex-link $node6 $node7 1.5Mb 10ms DropTail
$ns duplex-link $node7 $node1 1.5Mb 10ms DropTail

set tcp2 [new Agent/TCP]
$ns attach-agent $node1 $tcp2
set sink2 [new Agent/TCPSink]
$ns attach-agent $node4 $sink2
$ns connect $tcp2 $sink2
```

```
set ftp2 [new Application/FTP]
$ftp2 attach-agent $tcp2

$ns at 0.5 "$ftp2 start"
$ns rtmodel-at 1.0 down $node2 $node3
$ns rtmodel-at 2.0 up $node2 $node3
$ns at 3.0 "$ftp2 start"
$ns at 4.0 "$ftp2 stop"
$ns at 5.0 "finish"
$ns run
```

1. Write a program for a HDLC frame to perform the following.
a) Bit stuffing

```
#include<stdio.h>
int main()
{
int a[20];
int i,j,k,n,c=0,pos=0;
printf("\n Enter the number of bits");
scanf("%d",&n);
printf("\n Enter the bits");
for(i=0;i<n;i++)
scanf("%d",&a[i]);
for(i=0;i<n;i++)
     {
     if(a[i]==1)
          {
          c++;
          if(c==5)
               {
               pos=i+1;
               c=0;
               for(j=n;j>=pos;j--)
                    {
                    k=j+1;
                    a[k]=a[j];
                    }
               a[pos]=0;
               n=n+1;
               }
          }
     else
     c=0;
     }
printf("\n DATA AFTER STUFFING \n");
for(i=0;i<n;i++)
     {
     printf("%d",a[i]);
     }
}
```

Output:

```
Enter the number of bits10

Enter the bits1
1
1
1
1
1
1
1
1
1

DATA AFTER STUFFING
111110111110
```

b) Character stuffing

```
#include<stdio.h>
#include<string.h>
int main()
{
char a[30], fs[50] = " ", t[3], sd, ed, x[3], s[3], d[3], y[3];
int i, j, p = 0, q = 0;

printf("Enter characters to be stuffed:");
scanf("%s", a);
printf("\nEnter a character that represents starting delimiter:");
scanf(" %c", &sd);
printf("\nEnter a character that represents ending delimiter:");
scanf(" %c", &ed);

x[0] = s[0] = s[1] = sd;
x[1] = s[2] = '\0';
y[0] = d[0] = d[1] = ed;
d[2] = y[1] = '\0';
strcat(fs, x);
for(i = 0; i < strlen(a); i++)
     {
     t[0] = a[i];
     t[1] = '\0';
     if(t[0] == sd)
          strcat(fs, s);
     else if(t[0] == ed)
          strcat(fs, d);
     else
          strcat(fs, t);
     }
strcat(fs, y);
printf("\n After stuffing:%s", fs);
}
```

Output:

```
Enter characters to be stuffed:goodday

Enter a character that represents starting delimiter:g

Enter a character that represents ending delimiter:d

After stuffing: gggooddddayd
```

2. Write a program for distance vector algorithm to find suitable path for transmission.

```
#include<stdio.h>
struct node
{
unsigned dist[20];
unsigned from[20];
}rt[10];

int main()
{
int dmat[20][20];
int n,i,j,k,count=0;
printf("\nEnter the number of nodes : ");
scanf("%d",&n);
printf("\nEnter the cost matrix (999 for no link):\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("\n\nState value for router %d is\n",i+1);
     for(j=0;j<n;j++)
          {
          printf("\t\nnode %d via %d Distance%d",j+1,rt[i].from[j]
          +1,rt[i].dist[j]);
          }
     }
printf("\n\n");
}
```

Output:

```
Enter the number of nodes : 4

Enter the cost matrix (999 for no link):
 0   2 999  1
 2   0  5   2
999  5  0   6
 1   2  6   0


State value for router 1 is

node 1 via 1 Distance 0
node 2 via 2 Distance 2
node 3 via 2 Distance 7
node 4 via 4 Distance 1

State value for router 2 is

node 1 via 1 Distance 2
node 2 via 2 Distance 0
node 3 via 3 Distance 5
node 4 via 4 Distance 2

State value for router 3 is

node 1 via 2 Distance 7
node 2 via 2 Distance 5
node 3 via 3 Distance 0
node 4 via 4 Distance 6

State value for router 4 is

node 1 via 1 Distance 1
node 2 via 2 Distance 2
node 3 via 3 Distance 6
node 4 via 4 Distance 0
```

3. Implement Dijkstra's algorithm to compute the shortest routing path.

```
#include<stdio.h>
#define INFINITY 9999
#define MAX 10
void dijkstra(int G[MAX][MAX],int n,int startnode);

int main()
{
int G[MAX][MAX],i,j,n,u;
printf("Enter no. of vertices:");
scanf("%d",&n);
printf("\nEnter the adjacency matrix:\n");
for(i=0;i<n;i++)
     for(j=0;j<n;j++)
          scanf("%d",&G[i][j]);
     printf("\nEnter the starting node:");
     scanf("%d",&u);
     dijkstra(G,n,u);
return 0;
}

void dijkstra(int G[MAX][MAX],int n,int startnode)
{
int cost[MAX][MAX],distance[MAX],pred[MAX];
int visited[MAX],count,mindistance,nextnode,i,j;
for(i=0;i<n;i++)
     for(j=0;j<n;j++)
          if(G[i][j]==0)
               cost[i][j]=INFINITY;
          else
               cost[i][j]=G[i][j];
     for(i=0;i<n;i++)
          {
          distance[i]=cost[startnode][i];
          pred[i]=startnode;
          visited[i]=0;
          }
distance[startnode]=0;
visited[startnode]=1;
count=1;
while(count<n-1)
     {
     mindistance=INFINITY;
     for(i=0;i<n;i++)
          if(distance[i]<mindistance&&!visited[i])
               {
               mindistance=distance[i];
               nextnode=i;
               }
          visited[nextnode]=1;
     for(i=0;i<n;i++)
```

```
          if(!visited[i])
               if(mindistance+cost[nextnode][i]<distance[i])
                    {
                    distance[i]=mindistance+cost[nextnode][i];
                    pred[i]=nextnode;
                    }
               count++;
     }
for(i=0;i<n;i++)
     if(i!=startnode)
          {
          printf("\nDistance of node%d=%d",i,distance[i]);
          printf("\nPath=%d",i);
          j=i;
          do
               {
               j=pred[j];
               printf("<-%d",j);
               }while(j!=startnode);
          }
}
```

Output:

```
Enter no. of vertices:5

Enter the adjacency matrix:
0 10 0 30 100
10 0 50 0 0
0 50 0 20 10
30 0 20 0 60
100 0 10 60 0

Enter the starting node:0

Distance of node1=10
Path=1<-0
Distance of node2=50
Path=2<-3<-0
Distance of node3=30
Path=3<-0
Distance of node4=60
```

4. For the given data, use CRC-CCITT polynomial to obtain CRC code. Verify the program for the cases
a. Without error
b. With error

```
#include<stdio.h>
#include<string.h>
#include<stdlib.h>
#define N strlen(g)
char t[128],cs[128],g[]="1000100000010001";
int a,e,c;
void xor(){
     for(c=1;c<N;c++)
     cs[c]=((cs[c]==g[c])?'0':'1');
     }
void crc(){
     for(e=0;e<N;e++)
     cs[e]=t[e];
     do{
          if(cs[0]=='1')
          xor();
          for(c=0;c<N-1;c++)
               cs[c]=cs[c+1];
               cs[c]=t[e++];
     }while(e<=a+N-1);
}

int main(){
printf("\nEnter 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 postion where error is to be inserted:");
     scanf("%d",&e);
     t[e]=(t[e]=='0')?'1':'0';
     printf("Erroneous 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");
```

```
return 0;
}
```

Output:

```
Enter poly:1011101

Generating Polynomial is: 1000100000010001
Modified t[u] is: 10111010000000000000
Checksum is: 100010111011000
Final Codeword is: 10111011000010111011000
Test Error detection 0 (Yes) 1 (No) ? : 0
Enter postion where error is to be inserted:3
Erroneous Data: 10101011000010111011000
Error Detected

Enter poly: 1011101

Generating Polynomial is: 1000100000010001
Modified t[u] is: 10111010000000000000
Checksum is: 100010111011000
Final Codeword is: 10111011000010111011000
Test Error detection 0 (Yes) 1 (No) ? : 1
No Error Detected
```

5a. Implementation of Stop and Wait Protocol

```
#include<stdio.h>
#include<stdlib.h>
#include <unistd.h>
#define w 1
int main()
{
int i,f,frames[50];
printf("\nEnter number of frames to transmit:");
scanf("%d",&f);
printf("\nEnter %d frames: ",f);
for(i=1;i<=f;i++)
     scanf("%d",&frames[i]);
printf("\n With stop and wait protocol the frames will be sent in
the following manner (assuming no corruption of frames)\n\n");

printf("After sending %d frames at each stage sender waits for
acknowledgement sent by the receiver\n\n",w);

printf("if error occur negative acknowledge is detected same frame
is resend back\n");

for(i=1;i<=f;i++)
     {
     if((random()%2)==1)
          {
          if(i%w==0)
               {
               printf("%d\n",frames[i]);
               printf("Acknowledgement of above frames sent is
               received by sender\n\n");
               }
          else
               printf("%d ",frames[i]);
          }
     else
          {
          sleep(3);
          printf("negative acknowledge resend %d frame\n",i);
          i=i-1;
          sleep(1);
          }
     }
return 0;
}
```

Output:

```
Enter number of frames to transmit:6

Enter 6 frames: 1
3
5
2
4
6

With stop and wait protocol the frames will be sent in the following
manner (assuming no corruption of frames)

After sending 1 frames at each stage sender waits for
acknowledgement sent by the receiver

if error occur negative acknowledge is detected same frame    is
resend back
1
Acknowledgement of above frames sent is received by sender

negative acknowledge resend 2 frame
3
Acknowledgement of above frames sent is received by sender

5
Acknowledgement of above frames sent is received by sender

2
Acknowledgement of above frames sent is received by sender

4
Acknowledgement of above frames sent is received by sender

negative acknowledge resend 6 frame
negative acknowledge resend 6 frame
6
Acknowledgement of above frames sent is received by sender
```

5b. Implementation of Sliding window protocol program

```
#include<stdio.h>
int main()
{
int w,i,f,frames[50];
printf("Enter window size: ");
scanf("%d",&w);
printf("\nEnter number of frames to transmit:");
scanf("%d",&f);
printf("\nEnter %d frames: ",f);
for(i=1;i<=f;i++)
     scanf("%d",&frames[i]);

printf("\nWith sliding window protocol the frames will be sent in
the following manner (assuming no corruption of frames)\n\n");

printf("After sending %d frames at each stage sender waits for
acknowledgement sent by the receiver\n\n",w);

for(i=1;i<=f;i++)
     {
     if(i%w==0)
          {
          printf("%d\n",frames[i]);
          printf("Acknowledgement of above frames sent is received
          by  sender\n\n");
          }
     else
          printf("%d ",frames[i]);
     }
if(f%w!=0)
     printf("\nAcknowledgement of above frames sent is received by
     sender\n");
return 0;
}
```

Output:

```
Enter window size: 3

Enter number of frames to transmit:10

Enter 10 frames: 13
17
19
23
29
31
37
43
47
53

With sliding window protocol the frames will be sent in the
following manner (assuming no corruption of frames)

After sending 3 frames at each stage sender waits for
acknowledgement sent by the receiver

13 17 19
Acknowledgement of above frames sent is received by  sender

23 29 31
Acknowledgement of above frames sent is received by  sender

37 43 47
Acknowledgement of above frames sent is received by  sender

53
Acknowledgement of above frames sent is received by sender
```

6. Write a program for congestion control using leaky bucket algorithm

```
#include<stdio.h>
#include<stdlib.h>
int bucket_size;

void bucket_input ( int pkt_sz, int op_rt )
     {
     if( pkt_sz > bucket_size )
          printf(" \n\nBucket overflow\n ");
     else
          {
          sleep(1);
          while ( pkt_sz > op_rt )
               {
               printf(" \n %d bytes outputted ", op_rt );
               pkt_sz-= op_rt;
               sleep(1);
               }
          if ( pkt_sz > 0 )
               printf(" \nLast %d bytes sent\n", pkt_sz );
          printf(" \n Bucket output successful \n" );
          }
     }

int main()
{
int i, op_rate, packet_size;
printf("\n Enter Bucket Size: " );
scanf( "%d", &bucket_size );
printf(" \n Enter output rate: " );
scanf( "%d", &op_rate );
for( i=1; i<=5; i++ )
     {
     sleep(1);
     packet_size = random()%1000;
     printf(" \n Packet number [%d] \t Packet size = %d ", i,
     packet_size );
     bucket_input( packet_size, op_rate );
     }
return 0;
}
```

Output:

```
Enter Bucket Size: 500

Enter output rate: 80

Packet number [1]    Packet size = 383
80 bytes outputted
80 bytes outputted
80 bytes outputted
80 bytes outputted
Last 63 bytes sent

Bucket output successful

Packet number [2]    Packet size = 886

Bucket overflow

Packet number [3]    Packet size = 777

Bucket overflow

Packet number [4]    Packet size = 915

Bucket overflow

Packet number [5]    Packet size = 793

Bucket overflow
```