C BYREGOWDA INSTITUTE OF TECHNOLOGY

DEPARTMENT OF ELECTRONICS AND COMMUNICATION ENGINEERING

COMPUTER NETWORKS LAB MANUAL

SUBJECT CODE: 18ECL76

SEMESTER: VII

VISION

To excel in Electronics and Communication Engineering by creating quality learning-centric facility to meet the growing needs of Industry and Society

MISSION

M-1 Create a conductive and integrated learning environment for sustained growth in Electronics and Communication Engineering and Technology.

M-2 Develop Professional skills and attitude through distinctive teaching and training methodologies among students and faculty.

M-3 Nurture, in particular rural students through interactions with Academia, Industry and Alumni, in turn create employment opportunities

Prepared By

Dr G K VENKATESH

Associate Professor

PROGRAM EDUCATIONAL OBJECTIVES (PEOs)

PEO1: Graduates of the Program will have Successful Technical and Professional Career in Engineering, Technology and Multidisciplinary Environments.

PEO2: Graduates of the Program will utilize their Knowledge, Technical and Communication Skills to Propose Optimal Solutions to Problems Related to Society in the Field of Electronics and Communication.

PEO3: Graduates of the Program will Exhibit Good Interpersonal Skills, Leadership Qualities and adapt themselves for Lifelong Learning

PROGRAMME SPECIFIC OUTCOMES (PSOs)

At the end of the program students will have

PSO1: Ability to Absorb and Apply Fundamental Knowledge of Core Electronics and Communication Engineering in the Analysis, Design and Development of Electronics Systems as well as to Interpret and Synthesize Experimental Data Leading to Valid Conclusions

PSO2: Ability to Solve Complex Electronics and Communication Engineering Problems, using latest Hardware and Software Tools, along with Analytical and Managerial Skills to arrive at appropriate Solutions, either Independently or in Team

COURSE OUTCOME

At the end of the course, the students will have the ability to:

CO1	Use the simulator for learning and practice of networking algorithms
CO2	Illustrate the operations of network protocols and algorithms using C Programming
CO3	Simulate the network with different configurations to measure the performance parameters
CO4	Implement the data link and routing protocols using C programming

INDEX

PROGRAM OUTCOMES

Engineering Graduates will be able to:

1. **Engineering knowledge**: Apply the knowledge of mathematics, science, engineering fundamentals and engineering specialization to the solution of complex engineering problems.

2. **Problem analysis**: Identify, formulate, review research literature and analyze complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences and engineering sciences.

3. **Design/development of solutions**: Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for the public health and safety and the cultural, societal and environmental considerations.

4. **Conduct investigations of complex problems**: Use research-based knowledge and research methods including design of experiments, analysis and interpretation of data and synthesis of the information to provide valid conclusions.

5. **Modern tool usage**: Create, select and apply appropriate techniques, resources and modern engineering & IT tools including prediction and modelling to complex engineering activities with an understanding of the limitations.

6. **The engineer and society**: Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to the professional engineering practice.

7. **Environment and sustainability**: Understand the impact of the professional engineering solutions in societal and environmental contexts and demonstrate the knowledge of and need for sustainable development.

8. **Ethics**: Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.

9. **Individual and team work**: Function effectively as an individual and as a member or leader in diverse teams and in multidisciplinary settings.

10. **Communication**: Communicate effectively on complex engineering activities with the engineering community and with society at large, such as, being able to comprehend and write effective reports and design documentation, make effective presentations and give & receive clear instructions.

11. **Project management and finance**: Demonstrate knowledge and understanding of the engineering and management principles and apply these to one’s own work, as a member and leader in a team, to manage projects and in multidisciplinary environments.

12. **Life-long learning**: Recognize the need for and have the preparation and ability to engage in independent and life-long learning in the broadest context of technological change.

COURSE SYLLABUS

PART-A: Simulation experiments using NS2/ NS3/ OPNET/ NCTUNS/ NetSim/ QualNet/ Packet Tracer or any other equivalent tool

1. Implement a point to pint network with four nodes and duplex links between them. Analyze the network performance by setting the queue size and varying the bandwidth.
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.
3. Implement Ethernet LAN using n (6-10) nodes. Compare the throughput by changing the error rate and data rate.
4. Implement Ethernet LAN using n nodes and assign multiple traffic to the nodes and obtain congestion window for different sources/ destinations.
5. Implement ESS with transmission nodes in Wireless LAN and obtain the performance parameters.
6. Implementation of Link state routing algorithm

PART-B: Implement the following in C/C++

1. Write a program for a HLDC frame to perform the following.
   i) Bit stuffing
   ii) Character stuffing.
2. Write a program for distance vector algorithm to find suitable path for transmission.
3. Implement Dijkstra's algorithm to compute the shortest routing path.
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
5. Implementation of Stop and Wait Protocol and Sliding Window Protocol
6. Write a program for congestion control using leaky bucket algorithm.

LABORATORY RUBRICS

Sl. No.	**DESCRIPTION**	**MARKS**
1.	CONTINUOUS EVALUATION **a.** Observation write up & punctuality **b.** Conduction of experiment and output **c.** Viva voce **d.** Record write up	25.0 5.0 8.0 4.0 8.0
2.	INTERNAL TEST	15.0

PART-A: Simulation Experiments using NS2

1. Implement a point to pint 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 f [open lab1.tr w]
$ns trace-all $f

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

proc finish {} {
      global f nf ns
      $ns flush-trace
      close $f
      close $nf
      exec nam lab1.nam &
      exit 0
}

set n0 [$ns node]
set n1 [$ns node]
set n2 [$ns node]
set n3 [$ns node]
```

$ns duplex-link $n0 $n1 0.3Mb 10ms 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

```
$ns queue-limit $n0 $n1 20
$ns queue-limit $n1 $n2 20
$ns queue-limit $n2 $n3 20

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

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

$ns connect $udp0 $null0

$ns at 0.1 "$cbr0 start"
```

```
$ns at 4.5 "$cbr0 stop"
$ns at 5.0 "finish"
$ns run
```

Steps for execution:

- *Open **gedit** editor and type program. Program name should have the extension " **.tcl** "*

  [root@localhost ~]# gedit lab1.tcl
- *Save the program and quit.*
- *Run the simulation program*

  [root@localhost~]# ns lab1.tcl
- *Here **"ns"** indicates network simulator. We get the topology shown in the network animator. Now press the play button in the simulation window and the simulation will begins.*
- *To calculate the network performance. Execute the following command.*

  For calculating number of received packets

  [root@localhost~]#grep ^r lab1.tr | grep "cbr" | awk '{s+=$6}END{print s}'

  For calculating total time

  [root@localhost~]#grep ^r lab1.tr | grep "cbr" | awk '{s+=$2}END{print s}'

  Network performace = (Packet received/ Total Time) (bps)

- *Write the value of network performance in observation sheet. Repeat the above step by changing the bandwidth to [***0.3Mb, 0.4Mb, 0.5Mb, 0.7Mb***]to the following line of the program.*
- **$ns duplex-link $n0 $n1 0.7Mb 10ms DropTail #vary bandwidth 0.3, 0.4, 0.5 0.7**
- **$ns duplex-link $n1 $n2 0.7Mb 20ms DropTail #vary bandwidth 0.3, 0.4, 0.5 0.7**
- **$ns duplex-link $n2 $n3 0.7Mb 20ms DropTail #vary bandwidth 0.3, 0.4, 0.5 0.7**

Sl. No.	**Bandwidth**	**Network performance**
1.	**0.3**	
2.	**0.4**	
3.	**0.5**	
4.	**0.7**	

- *Plot a graph with x- axis with bandwidth and y-axis with network performance of UDP protocol.*

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 &
     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
```

<u>Steps for execution:</u>

- *Open **gedit** editor and type program. Program name should have the extension " **.tcl** "*

  [root@localhost ~]# gedit lab2.tcl

- *Save the program and quit.*
- *Run the simulation program*

  [root@localhost~]# ns lab2.tcl

- *Here **"ns"** indicates network simulator. We get the topology shown in the network animator. Now press the play button in the simulation window and the simulation will begins.*
- *To calculate the number of packets sent by TCP. Execute the following command.*

  [root@localhost~]#grep ^r lab2.tr | grep "tcp" -c

- *To calculate the number of packets sent by UDP. Execute the following command.*

  [root@localhost~]#grep ^r lab2.tr | grep "cbr" –c

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 trf [open lab3.tr w]
$ns trace-all $trf
set naf [open lab3.nam w]
$ns namtrace-all $naf

proc finish { } {
global nf ns tf
exec nam lab3.nam &
close $naf
close $trf
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
$udp0 set class_ 1
```

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

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

Steps for execution:

- *Open **gedit** editor and type program. Program name should have the extension " **.tcl** "*
  [root@localhost ~]# gedit lab3.tcl
- *Save the program and quit.*
- *Run the simulation program*
  [root@localhost~]# ns lab3.tcl
- *Here **"ns"** indicates network simulator. We get the topology shown in the network animator. Now press the play button in the simulation window and the simulation will begins.*
- *To calculate the throughput. Execute the following command.*
  For calculating number of received packets
  [root@localhost~]#grep ^r lab3.tr | grep "2 6" | awk '{s+=$6}END{print s}'
  For calculating total time
  [root@localhost~]#grep ^r lab3.tr | grep "2 6" | awk '{s+=$2}END{print s}'

Throughput = (Packet received/ Total Time) (bps)

- *Write the value of throughput in observation sheet. Repeat the above step by changing the error rate to the following line of the program.*

$err set rate_ 0.7 #vary error rate 0.1, 0.4, 0.5 and 0.7

Sl. No.	**Error rate**	**Throughput**
1.	**0.1**	
2.	**0.4**	
3.	**0.5**	
4.	**0.7**	

- *Plot a graph with x- axis with Error rate and y-axis with Throughput.*

4.Implement Ethernet LAN using n nodes and assign multiple traffic to the nodes and obtain congestion window for different sources/ destinations.

```
set ns [new Simulator]
set f [open lab4.tr w]
$ns trace-all $f

set nf [open lab4.nam w]
$ns namtrace-all $nf

proc finish {} {
        global ns f nf outFile1 outFile2
        $ns flush-trace
         close $f
        close $nf
        exec nam lab4.nam &
        exec xgraph Congestion1.xg Congestion2.xg -geometry
400x400 &
        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]

$n0 label "Src1"
$n4 label "Dst1"
$n1 label "Src2"
$n5 label "Dst2"

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


set tcp1 [new Agent/TCP]
$ns attach-agent $n0 $tcp1
set ftp1 [new Application/FTP]
$ftp1 attach-agent $tcp1
set sink1 [new Agent/TCPSink]
$ns attach-agent $n4 $sink1
$ftp1 set maxPkts_ 1000
$ns connect $tcp1 $sink1

set tcp2 [new Agent/TCP/Reno]
```

```
$ns attach-agent $n1 $tcp2
set ftp2 [new Application/FTP]
$ftp2 attach-agent $tcp2
set sink2 [new Agent/TCPSink]
$ns attach-agent $n5 $sink2
$ftp2 set maxPkts_ 1000
$ns connect $tcp2 $sink2

set outFile1 [open Congestion1.xg w]
set outFile2 [open Congestion2.xg w]

proc findWindowSize {tcpSource outFile} {
         global ns
         set now [$ns now]
         set cWindSize [$tcpSource set cwnd_]
         puts $outFile "$now $cWindSize"
         $ns at [expr $now + 0.1] "findWindowSize $tcpSource
$outFile"
}

$ns at 0.0 "findWindowSize $tcp1 $outFile1"
$ns at 0.1 "findWindowSize $tcp2 $outFile2"
$ns at 0.3 "$ftp1 start"
$ns at 0.5 "$ftp2 start"
$ns at 50.0 "$ftp1 stop"
$ns at 50.0 "$ftp2 stop"
$ns at 50.0 "finish"
$ns run
```

<u>Steps for execution:</u>

- *Open **gedit** editor and type program. Program name should have the extension " **.tcl** "*

  [root@localhost ~]# gedit lab4.tcl
- *Save the program and quit.*
- *Run the simulation program*

  [root@localhost~]# ns lab4.tcl
- *Here **"ns"** indicates network simulator. We get the topology shown in the network animator. Now press the play button in the simulation window and the simulation will begins.*
- *The xgraph automatically calculates and plot the two graph of Congestion window with TCP1 and TCP2.*

5.Implement ESS with transmission nodes in Wireless LAN and obtain the performance parameters.

```
set ns [new Simulator]
set tf [open lab5.tr w]
$ns trace-all $tf
set topo [new Topography]
$topo load_flatgrid 1300 1300
set nf [open lab5.nam w]
$ns namtrace-all-wireless $nf 1300 1300

$ns node-config -adhocRouting DSDV \
                -llType LL \
                -macType Mac/802_11 \
                -ifqType Queue/DropTail/PriQueue\
                -channelType  Channel/WirelessChannel \
                -propType Propagation/TwoRayGround \
                -antType Antenna/OmniAntenna \
                -ifqLen 50 \
                -phyType Phy/WirelessPhy \
                -topoInstance $topo \
                -agentTrace ON \
                -routerTrace ON

create-god 3
set n0 [$ns node]
set n1 [$ns node]
set n2 [$ns node]
$n0 label "ESS"
$n1 label "mob1"
$n2 label "mob2"
$n0 set X_ 10
$n0 set Y_ 600
$n0 set Z_ 0
$n1 set X_ 80
$n1 set Y_ 600
$n1 set Z_ 0
$n2 set X_ 1200
$n2 set Y_ 600
$n2 set Z_ 0
$ns at 0.1 "$n0 setdest 10 600 15"
$ns at 0.1 "$n1 setdest 80 600 25"
$ns at 0.1 "$n2 setdest 1200 600 25"

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

```
$ftp0 attach-agent $tcp0
set sink1 [new Agent/TCPSink]
$ns attach-agent $n1 $sink1
$ns connect $tcp0 $sink1
set tcp1 [new Agent/TCP]
$ns attach-agent $n0 $tcp1
set ftp1 [new Application/FTP]
$ftp1 attach-agent $tcp1
set sink2 [new Agent/TCPSink]
$ns attach-agent $n2 $sink2
$ns connect $tcp1 $sink2

$ns at 2 "$ftp0 start"
$ns at 15 "$ftp1 start"
$ns at 3 "$n1 setdest 1000 600 250"
$ns at 3 "$n2 setdest 80 600 250"

proc finish { } {
        global ns nf tf
        $ns flush-trace
        exec nam lab5.nam &
        close $tf
        exit 0
}
$ns at 20 "finish"
$ns run
```

<u>Steps for execution:</u>

- *Open* **gedit** *editor and type program. Program name should have the extension "* **.tcl** *"*

  [root@localhost ~]# gedit lab5.tcl
- *Save the program and quit.*
- *Run the simulation program*

  [root@localhost~]# ns lab5.tcl
- *Here **"ns"** indicates network simulator. We get the topology shown in the network animator. Now press the play button in the simulation window and the simulation will begins.*
- *To calculate the throughput. Execute the following command.*

  For calculating number of received packets

  [root@localhost~]#grep ^r lab5.tr | grep "AGT" |grep "tcp" | awk '{s+=$8}END{print s}'

  For calculating total time

  [root@localhost~]#grep ^r lab5.tr | grep "AGT" |grep "tcp" | awk '{s+=$2}END{print s}'

Throughput = (Packet received/ Total Time) (bps)

6.Implementation of Link state routing algorithm.

```
set ns [new Simulator]

$ns rtproto LS

set nf [open lab6.nam w]
$ns namtrace-all $nf

proc finish {} {
        global ns nf
        $ns flush-trace
        close $nf
        exec nam lab6.nam &
        exit 0
}

for {set i 0} {$i < 7} {incr i} {
        set n($i) [$ns node]
}


for {set i 0} {$i < 7} {incr i} {
        $ns duplex-link $n($i) $n([expr ($i+1)%7]) 1Mb 10ms
DropTail
}

set udp0 [new Agent/UDP]
$ns attach-agent $n(0) $udp0

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

set null0 [new Agent/Null]
$ns attach-agent $n(3) $null0

$ns connect $udp0 $null0

$ns at 0.5 "$cbr0 start"
$ns rtmodel-at 1.0 down $n(1) $n(2)
$ns rtmodel-at 2.0 up $n(1) $n(2)
$ns at 4.5 "$cbr0 stop"
$ns at 5.0 "finish"

$ns run
```

Steps for execution:

- *Open **gedit** editor and type program. Program name should have the extension " **.tcl** "*

  [root@localhost ~]# gedit lab6.tcl
- *Save the program and quit.*
- *Run the simulation program*

  [root@localhost~]# ns lab6.tcl
- *Here **"ns"** indicates network simulator. We get the topology shown in the network animator. Now press the play button in the simulation window and the simulation will begins.*
- *Explain link state routing algorithm using animation. How link state break and rerouting take place.*

PART-B: Implement in C/C++

Steps for execution:
Step 1. Open Terminal
Step 2. type gedit <filename.c>
example: lab1.c
Step 3. Type the program
Step 4. Save the program
Step5. Close the file
Step 6. Type gcc <filename.c>
Example gcc lab1.c
Step 7. type ./a.out

1. Write a program for a HLDC flame to perform the following.

i) Bit stuffing
ii) Character stuffing.

1a. BIT STUFFING PROGRAM

```
#include<stdio.h>
int main()
{
int a[15];
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 bits: 12

Enter the bits: 1 0 1 1 1 1 1 1 1 0 1 1

DATA AFTER STUFFING


1011111011011
```

1b. CHARACTER STUFFING PROGRAM

```
#include<stdio.h>
#include<string.h>
void main()
{
int i=0,j=0,n,pos;
char a[20],b[50],ch;
printf("Enter the Characters: ");
scanf("%s",&a);
printf("\nOrginal Data:%s",a);
n=strlen(a);
b[0]='d';
b[1]='l';
b[2]='e';
b[3]='s';
b[4]='t';
b[5]='x';
j=6;
while(i<n)
{
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("\nTransmitted Data: %s\n",b);
printf("Received Data:%s",a);
}
```

Output-1:

```
Enter the Characters: abcdefabcd
Orginal Data: abcdefabcd
Transmitted Data: dlestxabcdefabcddleetx
```

```
Received Data: abcdefabcd

Output-2:
Enter the Characters: abcdabcde
Orginal Data: abcdabcde
Transmitted Data: dlestxabcdabcdedleetx
Received Data: abcdabcde
```

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];
void main()
{
int costmat[20][20],source,desti;
int nodes,i,j,k,count=0;
printf("\nEnter the number of nodes : ");
scanf("%d",&nodes);
//Enter the nodes
printf("\nEnter the cost matrix :\n");
for(i=0;i<nodes;i++)
for(j=0;j<nodes;j++)
{
scanf("%d",&costmat[i][j]);
costmat[i][i]=0;
rt[i].dist[j]=costmat[i][j];
rt[i].from[j]=j;
}
do
{
count=0;
for(i=0;i<nodes;i++)
for(j=0;j<nodes;j++)
if(i!=j)
for(k=0;k<nodes;k++)
if(rt[i].dist[j]>rt[i].dist[k]+rt[k].dist[j])
{
rt[i].dist[j]=rt[i].dist[k]+rt[k].dist[j];
rt[i].from[j]=rt[i].from[k];
count++;
}
}
while(count!=0);
for(i=0;i<nodes;i++)
```

```
{
```

```
printf("\n\n State Value For Router %d\n",i+1);
for(j=0;j<nodes;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 :
0 2 999 1
2 0 5 2
999 5 0 6
1 2 6 0

State Value For Router 1
node1 via 1 Distance 0
node2 via 2 Distance 2
node3 via 2 Distance 7
node4 via 4 Distance 1

State Value For Router 2
node1 via 1 Distance 2
node2 via 2 Distance 0
node3 via 3 Distance 5
node4 via 4 Distance 2

State Value For Router 3
node1 via 2 Distance 7
node2 via 2 Distance 5
node3 via 3 Distance 0
node4 via 4 Distance 6

State Value For Router 4
node1 via 1 Distance 1
node2 via 2 Distance 2
node3 via 3 Distance 6
node4 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;
//pred[] stores the predecessor of each node
//count gives the number of nodes seen so far
//create the cost matrix
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];
//initialize pred[],distance[] and visited[]
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;
//nextnode gives the node at minimum distance
for(i=0;i<n;i++)
if(distance[i]<mindistance&&!visited[i])
{
mindistance=distance[i];
nextnode=i;
}
//check if a better path exists through nextnode
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++;
}
//print the path and distance of each node
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
Path=4<-2<-3<-0
```

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>
#define N strlen(g)

char t[28],cs[28],g[]="10001000000100001";
int a,i,j;
void xor()
{
for(j = 1;j < N; j++)
cs[j] = (( cs[j] == g[j])?'0':'1');
}
void crc()
{
for(i=0;i<N;i++)
cs[i]=t[i];
do
{
if(cs[0]=='1')
xor();
for(j=0;j<N-1;j++)
cs[j]=cs[j+1];
cs[j]=t[i++];
}
while(i<=a+N-1);
}
int main()
{
printf("\nEnter data : ");
scanf("%s",t);
printf("\n----------------------------------------");
printf("\nGeneratng polynomial : %s",g);
a=strlen(t);
for(i=a;i<a+N-1;i++)
t[i]='0';
printf("\n----------------------------------------");
printf("\nModified data is : %s",t);
```

```
printf("\n----------------------------------------");
crc();
printf("\nChecksum is : %s",cs);
for(i=a;i<a+N-1;i++) t[i]=cs[i-a];
printf("\n----------------------------------------");
printf("\nFinal codeword is : %s",t);
printf("\n----------------------------------------");
printf("\nEnter received message ");
scanf("%s",t);
crc();
for(i=0;(i<N-1) && (cs[i]!='1');i++);
if(i<N-1)
printf("\nError detected\n\n");
else
printf("\nNo error detected\n\n");
printf("\n----------------------------------------\n");
return 0;
}
```

```
Output-1:
Enter data : 1011101
----------------------------------------
Generatng polynomial : 10001000000100001
----------------------------------------
Modified data is : 10111010000000000000000
----------------------------------------
Checksum is : 1000101101011000
----------------------------------------
Final codeword is : 10111011000101101011000
----------------------------------------
Enter received message 10111001100010110101000
Error detected
----------------------------------------

Output-2:
Enter data : 1011101
----------------------------------------
Generatng polynomial : 10001000000100001
----------------------------------------
Modified data is : 10111010000000000000000
----------------------------------------
Checksum is : 1000101101011000
----------------------------------------
Final codeword is : 10111011000101101011000
----------------------------------------
Enter received message 10111011000101101011000
```