COMPUTER NETWORKS LABORATORY B.E., VI Semester, Electronics & Communication Engineering [As per Choice Based Credit System (CBCS) scheme]

Subject Code	15ECL68	IA Marks	20
Number of Lecture Hours/Week	01Hr Tutorial (Instructions) + 02 Hours Laboratory = 03	Exam Marks	80
RBT Levels	L1, L2, L3	Exam Hours	03

CREDITS – 02

Course objectives: This course will enable students to:

- Choose suitable tools to model a network and understand the protocols at various OSI reference levels.
- Design a suitable network and simulate using a Network simulator tool.
- Simulate the networking concepts and protocols using C/C++ programming.
- Model the networks for different configurations and analyze the results.

Laboratory Experiments

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

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.
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.

Course outcomes: On the completion of this laboratory course, the students will be able to:

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

Conduct of Practical Examination:
- All laboratory experiments are to be included for practical examination.
  For examination one question from software and one question from hardware or only one hardware experiments based on the complexity to be set.
- Students are allowed to pick one experiment from the lot.
  Strictly follow the instructions as printed on the cover page of answer script for breakup of marks.
- Change of experiment is allowed only once and Marks allotted to the procedure part to be made zero.

NOTE:awk -f programme name.awk programme name.tr (To generate o/p from trace file)

Programme1:

//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.

<u>prog1.tcl</u>

```
set ns [new Simulator]
set nf [open prog1.nam w]
$ns namtrace-all $nf
set nd [open prog1.tr w]
$ns trace-all $nd

proc finish { } {
global ns nf nd
$ns flush-trace
close $nf
close $nd
exec nam prog1.nam &
exit 0
}

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

$ns duplex-link $n0 $n1 1Mb 10ms DropTail
$ns duplex-link $n1 $n2 512Kb 10ms DropTail
$ns queue-limit $n1 $n2 5

set udp0 [new Agent/UDP]
$ns attach-agent $n0 $udp0

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

set sink [new Agent/Null]
$ns attach-agent $n2 $sink
$ns connect $udp0 $sink
```

```
$ns at 0.2 "$cbr0 start"
$ns at 4.5 "$cbr0 stop"
$ns at 5.0 "finish"
$ns run
```

prog1.awk

```
BEGIN {
dcount = 0;
rcount = 0;
}
{
event = $1;
if(event == "d")
{
dcount++;
}
if(event == "r")
{
rcount++;
}
}
END {
printf("The no.of packets dropped  : %d\n ",dcount);
printf("The no.of packets recieved : %d\n ",rcount);
}
```

Programme2:

//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.

prog2.tcl

```
set ns [new Simulator]
set nf [open prog2.nam w]
$ns namtrace-all $nf
set nd [open prog2.tr w]
$ns trace-all $nd

proc finish {} {
global ns nf nd
$ns flush-trace
close $nf
exec nam prog2.nam &
exit 0
}

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

$n0 label TCP
$n1 label UDP
$n3 label NULL-TCPSINK

$ns duplex-link $n0 $n2 1Mb 10ms DropTail
$ns duplex-link $n1 $n2 1Mb 10ms DropTail
$ns duplex-link $n2 $n3 1Mb 10ms DropTail

set tcp0 [new Agent/TCP]
$ns attach-agent $n0 $tcp0
set sink0 [new Agent/TCPSink]
$ns attach-agent $n3 $sink0
$ns connect $tcp0 $sink0

set ftp0 [new Application/FTP]
$ftp0 attach-agent $tcp0

set udp0 [new Agent/UDP]
$ns attach-agent $n1 $udp0
set null0 [new Agent/Null]
$ns attach-agent $n3 $null0
```

```
$ns connect $udp0 $null0

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

$ns at 0.2 "$cbr0 start"
$ns at 0.1 "$ftp0 start"
$ns at 4.5 "$cbr0 stop"
$ns at 4.4 "$ftp0 stop"

$ns at 5.0 "finish"
$ns run
```

prog2.awk

```
BEGIN {
ctcp=0;
cudp=0;
}
{
pkt=$5;
if(pkt=="cbr") { cudp++;}
if(pkt=="tcp") { ctcp++;}
}
END {
printf("No of packets sent\nTcp : %d\nUdp : %d\n",ctcp,cudp);
}
```

Programme3:
Implement Ethernet LAN using n (6-10) nodes. Compare the throughput by changing the error rate and data rate.

```
set ns [new Simulator]
set nf [open prog5.nam w]
$ns namtrace-all $nf
set nd [open prog5.tr w]
$ns trace-all $nd

proc finish {} {
global ns nf nd
$ns flush-trace
close $nf
close $nd
exec nam prog5.nam &
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]

$ns make-lan "$n0 $n1 $n2 $n3 $n4 $n5 $n6" 0.2Mb 40ms LL Queue/DropTail
Mac/802_3

set tcp [new Agent/TCP]
$ns attach-agent $n0 $tcp
set sink [new Agent/TCPSink]
$ns attach-agent $n5 $sink
$ns connect $tcp $sink

set ftp [new Application/FTP]
$ftp attach-agent $tcp

$ns at 1.0 "$ftp start"
$ns at 5.0 "$ftp stop"
$ns at 5.5 "finish"
$ns run
```

prog3.awk

```
BEGIN {
```

```
sSize=0;
startTime = 5.0;
stopTime = 0.1;
Tput = 0;
}
{
event = $1;
time = $2;
from = $3;
to = $4;
pkt = $5;
size = $6;
fid = $7;
src = $8;
dst = $9;
seqn = $10;
pid = $11;
if (event == "+") {
if(time < startTime) {
startTime = time;
}
}
if (event == "r") {
if(time > stopTime) {
stopTime = time;
}
sSize+=size;
}
Tput = (sSize/(stopTime-startTime))*(8/1000);
printf("%f\t%.2f\n",time,Tput);
}
END {
}
```

```
set ns [new Simulator]
set tf [open lab5.tr w]
$ns trace-all $tf
set nf [open lab5.nam w]
$ns namtrace-all $nf
$ns color 0 blue
set n0 [$ns node]
$n0 color "red"
set n1 [$ns node]
$n1 color "red"
```

```
set n2 [$ns node]
 $n2 color "red"
set n3 [$ns node] $n3
color "red"
set n4 [$ns node]
$n4 color "magenta"
set n5 [$ns node]
$n5 color "magenta"
set n6 [$ns node]
$n6 color "magenta"
set n7 [$ns node]
$n7 color "magenta"

$ns make-lan "$n0 $n1 $n2 $n3" 100Mb 300ms LL Queue/ DropTail Mac/802_3
$ns make-lan "$n4 $n5 $n6 $n7" 100Mb 300ms LL Queue/ DropTail Mac/802_3
$ns duplex-link $n3 $n4 100Mb 300ms DropTail
$ns duplex-link-op $n3 $n4 color "green"
# set error rate. Here ErrorModel is a class and it is single word and space should
not be given between Error and Model
# lossmodel is a command and it is single word. Space should not be given between
loss and model
set err [new ErrorModel]

$ns lossmodel $err $n3 $n4

$err set rate_ 0.1

# error rate should be changed for each output like 0.1,0.3,0.5…. */
set udp [new Agent/UDP]

$ns attach-agent $n1 $udp

set cbr [new Application/Traffic/CBR]

$cbr attach-agent $udp

$cbr set fid_ 0

$cbr set packetSize_ 1000

$cbr set interval_ 0.0001

set null [new Agent/Null]

$ns attach-agent $n7 $null

$ns connect $udp $null

proc finish { } {
```

```
global ns nf tf

$ns flush-trace

close $nf

close $tf

exec nam lab5.nam &

exit 0 }

$ns at 0.1 "$cbr start"

$ns at 3.0 "finish"

$ns run
```

AWK file: *(Open a new editor using “vi command” and write awk file and save with “.awk” extension)*

```
BEGIN{
pkt=0;
time=0;
}
{
if($1= ="r" && $3= ="9" && $4= ="7"){
pkt = pkt + $6;
time =$2;
}
}
END {
printf("throughput:%fMbps",(( pkt / time) * (8 / 1000000)));
```

Programme4:
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 nf [open prog7.nam w]
$ns namtrace-all $nf
set nd [open prog7.tr w]
$ns trace-all $nd

$ns color 1 Blue
$ns color 2 Red

proc finish { } {
global ns nf nd
$ns flush-trace
close $nf
close $nd
exec nam prog7.nam &
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 [$ns node]
set n7 [$ns node]
set n8 [$ns node]

$n7 shape box
$n7 color Blue
$n8 shape hexagon
$n8 color Red

$ns duplex-link $n1 $n0 2Mb 10ms DropTail
$ns duplex-link $n2 $n0 2Mb 10ms DropTail
$ns duplex-link $n0 $n3 1Mb 20ms DropTail
```

```
$ns make-lan "$n3 $n4 $n5 $n6 $n7 $n8" 512Kb 40ms LL Queue/DropTail Mac/802_3

$ns duplex-link-op $n1 $n0 orient right-down
$ns duplex-link-op $n2 $n0 orient right-up
$ns duplex-link-op $n0 $n3 orient right

$ns queue-limit $n0 $n3 20

set tcp1 [new Agent/TCP/Vegas]
$ns attach-agent $n1 $tcp1
set sink1 [new Agent/TCPSink]
$ns attach-agent $n7 $sink1
$ns connect $tcp1 $sink1
$tcp1 set class_ 1
$tcp1 set packetsize_ 55

set ftp1 [new Application/FTP]
$ftp1 attach-agent $tcp1

set tfile [open cwnd.tr w]
$tcp1 attach $tfile
$tcp1 trace cwnd_

set tcp2 [new Agent/TCP/Reno]
$ns attach-agent $n2 $tcp2
set sink2 [new Agent/TCPSink]
$ns attach-agent $n8 $sink2
$ns connect $tcp2 $sink2
$tcp2 set class_ 2
$tcp2 set packetSize_ 55

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

set tfile2 [open cwnd2.tr w]
$tcp2 attach $tfile2
$tcp2 trace cwnd_

$ns at 0.5 "$ftp1 start"
$ns at 1.0 "$ftp2 start"
$ns at 5.0 "$ftp2 stop"
$ns at 5.0 "$ftp1 stop"

$ns at 5.5 "finish"
$ns run
```

prog4.awk

```
BEGIN {
}
{
if($6=="cwnd_") {
printf("%f\t%f\n",$1,$7);
}
}
END {
}
```

Programme5:
Implement ESS with transmission nodes in Wireless LAN and obtain the performance parameters.

```
set ns [new Simulator]


set tf [open lab4.tr w]
$ns trace-all $tf

set topo [new Topography]
$topo load_flatgrid 1000 1000

set nf [open lab4.nam w]

$ns namtrace-all-wireless $nf 1000 1000


$ns node-config -adhocRouting DSDV \
                -llType LL \
                -macType Mac/802_11 \
                -ifqType Queue/DropTail \
                -ifqLen 50 \
                -phyType Phy/WirelessPhy \
                -channelType Channel/WirelessChannel \
                -propType Propagation/TwoRayGround \
                -antType Antenna/OmniAntenna \
                -topoInstance $topo \
                -agentTrace ON \
                -routerTrace ON
create-god 3
set n0 [$ns
node]
set n1 [$ns
node]
set n2 [$ns
node]
$n0 label "tcp0"
$n1 label "sink1/tcp1"
$n2 label "sink2"

$n0 set X_ 50
$n0 set Y_ 50
$n0 set Z_ 0
$n1 set X_ 100
```

```
$n1 set Y_ 100
$n1 set Z_ 0
$n2 set X_ 600
$n2 set Y_ 600
$n2 set Z_ 0

$ns at 0.1 "$n0 setdest 50 50 15"
$ns at 0.1 "$n1 setdest 100 100 25"
$ns at 0.1 "$n2 setdest 600 600 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 $n1 $tcp1

set ftp1 [new Application/FTP]
$ftp1 attach-agent $tcp1

set sink2 [new Agent/TCPSink]
$ns attach-agent $n2 $sink2
$ns connect $tcp1 $sink2

$ns at 5 "$ftp0 start"
$ns at 5 "$ftp1 start"

$ns at 100 "$n1 setdest 550 550 15"
$ns at 190 "$n1 setdest 70
70 15"

proc finish { } {
global ns nf tf
$ns flush-trace
                exec  nam
        lab4.nam        &
        close $tf
        exit 0
}
$ns at 250 "finish"
$ns run
```

AWK file

```
        BEGIN{
        count1=0
        count2=0
        pack1=0
        pack2=0
        time1=0
        time2=0
}
{
        if($1= ="r"&& $3= ="_1_" && $4= ="AGT")
        {
                count1++
                pack1=pack1+$8
                time1=$2
        }
        if($1= ="r" && $3= ="_2_" && $4= ="AGT")
        {
                count2++
                pack2=pack2+$8

                time2=$2
        }
}
END{
printf("The Throughput from n0 to n1: %f Mbps \n”, ((count1*pack1*8)/
(time1*1000000)));
printf("The Throughput from n1 to n2: %f Mbps", ((count2*pack2*8)/
(time2*1000000)));
}
```

Programme6:
Implementation of Link state routing algorithm

```
set ns [new Simulator]

set nf [open out.nam w]

$ns namtrace-all $nf

set tr [open out.tr w]

$ns trace-all $tr

proc finish {} {

        global nf ns tr

        $ns flush-trace

        close $tr

        exec nam out.nam &

        exit 0

        }

set n0 [$ns node]

set n1 [$ns node]

set n2 [$ns node]

set n3 [$ns node]

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

$ns duplex-link $n1 $n3 10Mb 10ms DropTail

$ns duplex-link $n2 $n1 10Mb 10ms DropTail

$ns duplex-link-op $n0 $n1 orient right-down

$ns duplex-link-op $n1 $n3 orient right

$ns duplex-link-op $n2 $n1 orient right-up
```

```
set tcp [new Agent/TCP]

$ns attach-agent $n0 $tcp

set ftp [new Application/FTP]

$ftp attach-agent $tcp

set sink [new Agent/TCPSink]

$ns attach-agent $n3 $sink

set udp [new Agent/UDP]

$ns attach-agent $n2 $udp

set cbr [new Application/Traffic/CBR]

$cbr attach-agent $udp

set null [new Agent/Null]

$ns attach-agent $n3 $null

$ns connect $tcp $sink

$ns connect $udp $null

$ns rtmodel-at 1.0 down $n1 $n3

$ns rtmodel-at 2.0 up $n1 $n3

$ns rtproto LS

$ns at 0.0 "$ftp start"

$ns at 0.0 "$cbr start"

$ns at 5.0 "finish"

$ns run
```

PART-B

Program to implement Character Stuffing

Security and Error detection are the most prominent features that are to be provided by any application which transfers data from one end to the other end. One of such a mechanism in tracking errors which may add up to the original data during transfer is known as Stuffing. It is of two types namely Bit Stuffing and the other Character Stuffing. Coming to the Bit Stuffing, a flag(ex01111110) is appended within the original data while transfer of it. The following program describes how it is stuffed at the sender end and de-stuffed at the reciever end

C Programme

```
#include<stdio.h>
#include<string.h>
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;
        clrscr();
        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);
        getch();
```

}

Output:-

```
Enter characters to be stuffed: goodday
Enter a character that represents starting delimiter: d
Enter a character that represents ending delimiter: g
After stuffing:  dggooddddayg.
```

/*Bitstuffing and destuffing*/

```
#include<stdio.h>
#include<stdlib.h>
#define MAXSIZE 100

int main()
{
 char *p,*q;
 char temp;
 char in[MAXSIZE];
 char stuff[MAXSIZE];
 char destuff[MAXSIZE];

 int count=0;

 printf("enter the input character string (0‘s & 1‘s
only):\n");
 scanf("%s",in);

 p=in;
 q=stuff;

 while(*p!='\0')
 {
 if(*p=='0')
 {
 *q=*p;
 q++;
 p++;
 }
 else
 {
 while(*p=='1' && count!=5)
 {
 count++;
 *q=*p;
 q++;
 p++;
  }

 if(count==5)
 {
 *q='0';
 q++;
  }
 count=0;
 }
 }
 *q='\0';
 printf("\nthe stuffed character string is");
 printf("\n%s",stuff);

 p=stuff;
 q=destuff;
 while(*p!='\0')
 {
 if(*p=='0')
 {
 *q=*p;
 q++;
 p++;
 }
 else
 {
 while(*p=='1' && count!=5)
{
 count++;
 *q=*p;
   q++;
  p++;
 }
 if(count==5)
 {
  p++;
  }
 count=0;
 }
 }
 *q='\0';
 printf("\nthe destuffed character string is");
 printf("\n%s\n",destuff);
return 0;
}
```

Output:

```
enter the input character string (0‘s & 1‘s only):
1 0 1 0 1 1 1 1 1 1

the stuffed character string is
1 0 1 0 1 1 1 1 1 0 1

the destuffed character string is

1 0 1 0 1 1 1 1 1 1
```

Cyclic Redundancy Check(CRC)

A **cyclic redundancy check** (CRC) is an error-detecting code commonly used in digital networks and storage devices to detect accidental changes to raw data. Blocks of data entering these systems get a short *check value* attached, based on the remainder of a polynomial division of their contents; on retrieval the calculation is repeated, and corrective action can be taken against presumed data corruption if the check values do not match.

How to Compute CRC Cyclic Redundancy Check ?

To compute an *n*-bit binary CRC, line the bits representing the input in a row, and position the (*n*+1)-bit pattern representing the CRC's divisor (called a "polynomial") underneath the left-hand end of the row.

Start with the message to be encoded:

```
11010011101100
```

This is first padded with zeroes corresponding to the bit length *n* of the CRC. Here is the first calculation for computing a 3-bit CRC:

```
11010011101100 000 <--- input right padded by 3 bits
1011               <--- divisor (4 bits) = x³+x+1
------------------
01100011101100 000 <--- result
```

If the input bit above the leftmost divisor bit is 0, do nothing. If the input bit above the leftmost divisor bit is 1, the divisor is XORed into the input (in other words, the input bit above each 1-bit in the divisor is toggled). The divisor is then shifted one bit to the right, and the process is repeated until the divisor reaches the right-hand end of the input row. Here is the entire calculation:

```
11010011101100 000 <--- input right padded by 3 bits
1011               <--- divisor
01100011101100 000 <--- result
 1011              <--- divisor ...
00111011101100 000
  1011
00010111101100 000
   1011
00000001101100 000
       1011
00000000110100 000
```

```
        1011
00000000011000 000
         1011
00000000001110 000
          1011
00000000000101 000
           101 1
-----------------
00000000000000 100 <---remainder (3 bits)
```

Since the leftmost divisor bit zeroed every input bit it touched, when this process ends the only bits in the input row that can be nonzero are the n bits at the right-hand end of the row. These *n* bits are the remainder of the division step, and will also be the value of the CRC function (unless the chosen CRC specification calls for some postprocessing).

The validity of a received message can easily be verified by performing the above calculation again, this time with the check value added instead of zeroes. The remainder should equal zero if there are no detectable errors.

```
11010011101100 100 <--- input with check value
1011               <--- divisor
01100011101100 100 <--- result
 1011              <--- divisor ...
00111011101100 100

......

00000000001110 100
          1011
00000000000101 100
           101 1
------------------
```

c programme

```
  #include<stdio.h>
int a[100],b[100],i,j,len,k,count=0;
//Generator Polynomial:g(x)=x^16+x^12+x^5+1
int gp[]={1,0,0,0,1,0,0,0,0,0,0,1,0,0,0,0,1,};
int main()
{
void div();
//system("clear");
printf("\nEnter the length of Data Frame :");
scanf("%d",&len);
printf("\nEnter the Message :");
for(i=0;i<len;i++)
scanf("%d",&a[i]);
//Append r(16) degree Zeros to Msg bits
for(i=0;i<16;i++)
a[len++]=0;
//Xr.M(x) (ie. Msg+16 Zeros)
for(i=0;i<len;i++)
b[i]=a[i];
//No of times to be divided ie. Msg Length
k=len-16;
div();
for(i=0;i<len;i++)
b[i]=b[i]^a[i]; //MOD 2 Substraction
printf("\nData to be transmitted : ");
for(i=0;i<len;i++)
printf("%2d",b[i]);
printf("\n\nEnter the Reveived Data \n: ");
for(i=0;i<len;i++)
scanf("%d",&a[i]);
div();
for(i=0;i<len;i++)
if(a[i]!=0)
{printf("\nERROR in Recived Data");
return 0;
}
printf("\nData Recived is ERROR FREE");
}
void div()
{
for(i=0;i<k;i++)
{
if(a[i]==gp[0])
{
for(j=i;j<17+i;j++)
a[j]=a[j]^gp[count++];
}
count=0;
}
}
```

Distance Vector Routing using C

In computer communication theory relating to packet-switched networks, is one of the two major classes of routing protocols, the other major class being the link-state protocol. A distance-vector routing protocol uses the Bellman-Ford algorithm to calculate paths.

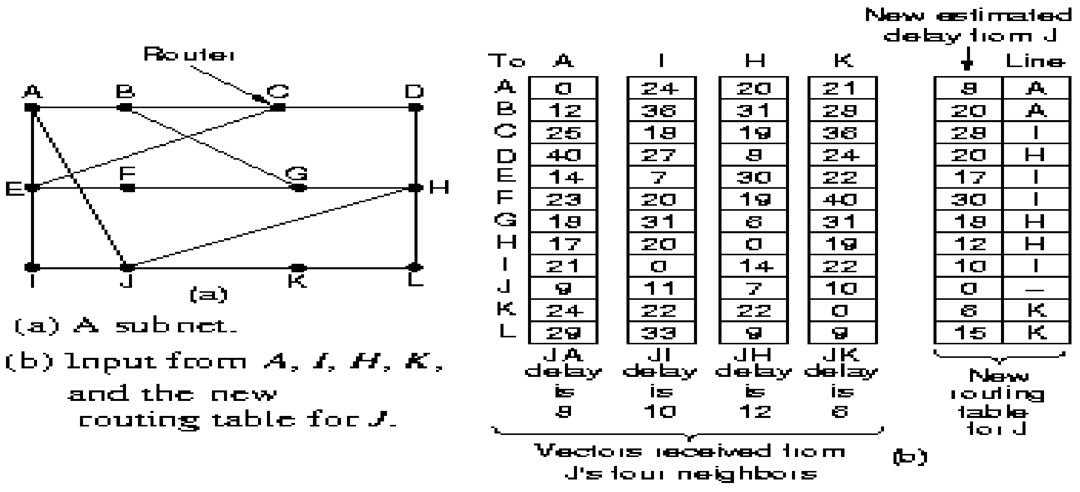
A distance-vector routing protocol requires that a router informs its neighbors of topology changes periodically and, in some cases, when a change is detected in the topology of a network. Compared to link-state protocols, which require a router to inform all the nodes in a network of topology changes, distance-vector routing protocols have less computational complexity and message overhead.

Distance Vector means that Routers are advertised as vector of distance and direction. 'Direction' is represented by next hop address and exit interface, whereas 'Distance' uses metrics such as hop count.

Routers using distance vector protocol do not have knowledge of the entire path to a destination. Instead DV uses two methods:

1. Direction in which or interface to which a packet should be forwarded.
2. Distance from its destination.

Examples of distance-vector routing protocols include Routing Information Protocol Version 1 & 2, RIPv1 and RIPv2 and IGRP. EGP and BGP are not pure distance-vector routing protocols because a distance-vector protocol calculates routes based only on link costs whereas in BGP, for example, the local route preference value takes priority over the link cost.



To	A	I	H	K	New estimated delay from J	Line
A	0	24	20	21	8	A
B	12	36	31	28	20	A
C	25	18	19	36	28	I
D	40	27	8	24	20	H
E	14	7	30	22	17	I
F	23	20	19	40	30	I
G	18	31	6	31	18	H
H	17	20	0	19	12	H
I	21	0	14	22	10	I
J	9	11	7	10	0	—
K	24	22	22	0	6	K
L	29	33	9	9	15	K
	JA delay is 8	JI delay is 10	JH delay is 12	JK delay is 6		

(a) A subnet.
(b) Input from *A*, *I*, *H*, *K*, and the new routing table for *J*.

```
#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 :\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");
}
```

Example output

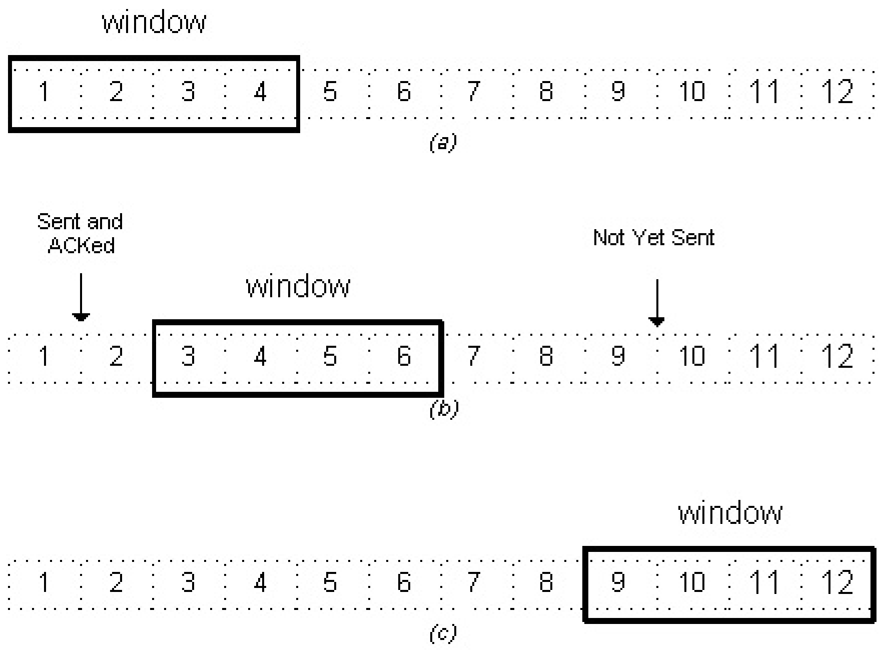
```
Enter the number of nodes : 4

Enter the cost matrix :
0 3 5 99
3 0 99 1
5 4 0 2
99 1 2 0

State value for router 1 is
node 1 via 1 Distance0
node 2 via 2 Distance3
node 3 via 3 Distance5
node 4 via 2 Distance4

State value for router 2 is

node 1 via 1 Distance3
node 2 via 2 Distance0
node 3 via 4 Distance3
node 4 via 4 Distance1

State value for router 3 is

node 1 via 1 Distance5
node 2 via 4 Distance3
node 3 via 3 Distance0
node 4 via 4 Distance2

State value for router 4 is

node 1 via 2 Distance4
node 2 via 2 Distance1
node 3 via 3 Distance2
node 4 via 4 Distance0
```

Sliding Window Protocol

In computer networks sliding window protocol is a method to transmit data on a network. Sliding window protocol is applied on the Data Link Layer of OSI model. At data link layer, data is in the form of frames. In Networking, Window simply means a buffer which has data frames that needs to be transmitted.

Both sender and receiver agrees on some window size. If window size=w then after sending w frames sender waits for the acknowledgement (ack) of the first frame.

As soon as sender receives the acknowledgement of a frame it is replaced by the next frames to be transmitted by the sender. If receiver sends a collective or cumulative acknowledgement to sender then it understands that more than one frames are properly received, for eg:- if ack of frame 3 is received it understands that frame 1 and frame 2 are received properly.



In sliding window protocol the receiver has to have some memory to compensate any loss in transmission or if the frames are received unordered.

Efficiency of Sliding Window Protocol

$\eta = (W*t_x)/(t_x+2t_p)$

W = Window Size

t_x = Transmission time

t_p = Propagation delay

Sliding window works in full duplex mode

It is of two types:-

1. Selective Repeat: Sender transmits only that frame which is erroneous or is lost.

2. Go back n: Sender transmits all frames present in the window that occurs after the error bit including error bit also.

/*Sliding Window Protocol*/

```
#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: 5

Enter 5 frames: 12 5 89 4 6

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

12 5 89
Acknowledgement of above frames sent is received by sender

4 6
Acknowledgement of above frames sent is received by sender
```

Write a program for congestion control using Leaky bucket algorithm.

Description

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

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

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

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

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

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

Algorithm:

1. Start
2. Set the bucket size or the buffer size.
3. Set the output rate.

4. Transmit the packets such that there is no overflow.
5. Repeat the process of transmission until all packets are transmitted. (Reject packets where its size is greater than the bucket size)
6. Stop

```
 #include<stdio.h>

 #include<stdlib.h>

#include<unistd.h>

#define NOF_PACKETS 10

int rand(int a)
{
   int rn = (random() % 10) % a;
   return  rn == 0 ? 1 : rn;
}

int main()
{
   int packet_sz[NOF_PACKETS], i, clk, b_size, o_rate, p_sz_rm=0, p_sz, p_time, op;
   for(i = 0; i<NOF_PACKETS; ++i)
      packet_sz[i] = rand(6) * 10;
   for(i = 0; i<NOF_PACKETS; ++i)
      printf("\npacket[%d]:%d bytes\t", i, packet_sz[i]);
   printf("\nEnter the Output rate:");
   scanf("%d", &o_rate);
   printf("Enter the Bucket Size:");
   scanf("%d", &b_size);
   for(i = 0; i<NOF_PACKETS; ++i)
   {
      if( (packet_sz[i] + p_sz_rm) > b_size)
         if(packet_sz[i] > b_size)/*compare the packet siz with bucket size*/
            printf("\n\nIncoming packet size (%dbytes) is Greater than bucket capacity
(%dbytes)-PACKET REJECTED", packet_sz[i], b_size);
         else
            printf("\n\nBucket capacity exceeded-PACKETS REJECTED!!");
      else
      {
         p_sz_rm += packet_sz[i];
         printf("\n\nIncoming Packet size: %d", packet_sz[i]);
         printf("\nBytes remaining to Transmit: %d", p_sz_rm);
```

```
            p_time = rand(4) * 10;
            printf("\nTime left for transmission: %d units", p_time);
            for(clk = 10; clk <= p_time; clk += 10)
            {
               sleep(1);
               if(p_sz_rm)
               {
                  if(p_sz_rm <= o_rate)/*packet size remaining comparing with output rate*/
                     op = p_sz_rm, p_sz_rm = 0;
                  else
                     op = o_rate, p_sz_rm -= o_rate;
                  printf("\nPacket of size %d Transmitted", op);
                  printf("----Bytes Remaining to Transmit: %d", p_sz_rm);
               }
               else
               {
                  printf("\nTime left for transmission: %d units", p_time-clk);
                  printf("\nNo packets to transmit!!");
               }
            }
         }
      }
   }
```

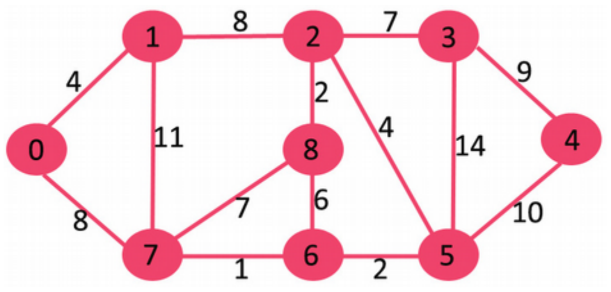
Dijkstra's algorithm

Dijkstra's algorithm is an algorithm for finding the shortest paths between nodes in a graph, which may represent, for example, road networks. It was conceived by computer scientist Edsger W. Dijkstra in 1956 and published three years later.

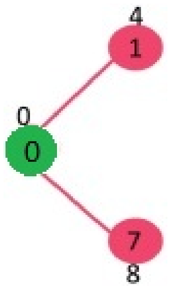
Dijkstra’s algorithm is very similar to Prim’s algorithm for minimum spanning tree. Like Prim’s MST, we generate a *SPT (shortest path tree)* with given source as root. We maintain two sets, one set contains vertices included in shortest path tree, other set includes vertices not yet included in shortest path tree. At every step of the algorithm, we find a vertex which is in the other set (set of not yet included) and has minimum distance from source.

Below are the detailed steps used in Dijkstra's algorithm to find the shortest path from a single source vertex to all other vertices in the given graph.
Algorithm
1) Create a set *sptSet* (shortest path tree set) that keeps track of vertices included in shortest path tree, i.e., whose minimum distance from source is calculated and finalized. Initially, this set is empty.
2) Assign a distance value to all vertices in the input graph. Initialize all distance values as INFINITE. Assign distance value as 0 for the source vertex so that it is picked first.
3) While *sptSet* doesn't include all vertices
….**a)** Pick a vertex u which is not there in *sptSet*and has minimum distance value.
….**b)** Include u to *sptSet*.
….**c)** Update distance value of all adjacent vertices of u. To update the distance values, iterate through all adjacent vertices. For every adjacent vertex v, if sum of distance value of u (from source) and weight of edge u-v, is less than the distance value of v, then update the distance value of v.

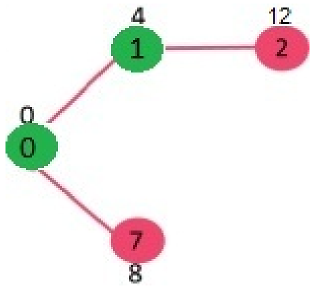
Let us understand with the following example:



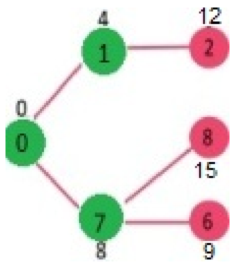
The set *sptSet*is initially empty and distances assigned to vertices are {0, INF, INF, INF, INF, INF, INF, INF} where INF indicates infinite. Now pick the vertex with minimum distance value. The vertex 0 is picked, include it in *sptSet*. So *sptSet* becomes {0}. After including 0 to *sptSet*, update distance values of its adjacent vertices. Adjacent vertices of 0 are 1 and 7. The distance values of 1 and 7 are updated as 4 and 8. Following subgraph shows vertices and their distance values, only the vertices with finite distance values are shown. The vertices included in SPT are shown in green color.



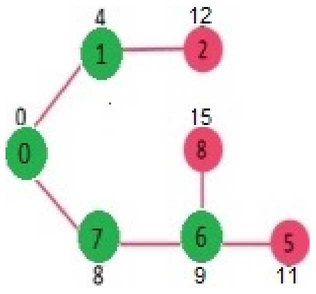
Pick the vertex with minimum distance value and not already included in SPT (not in sptSET). The vertex 1 is picked and added to sptSet. So sptSet now becomes {0, 1}. Update the distance values of adjacent vertices of 1. The distance value of vertex 2 becomes 12.



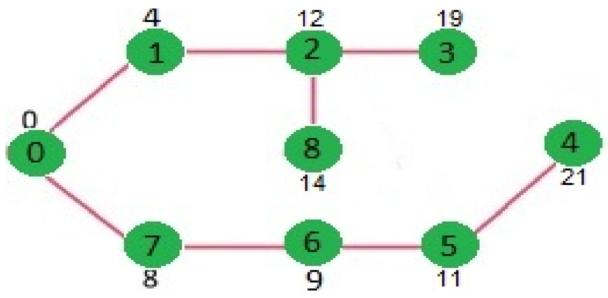
Pick the vertex with minimum distance value and not already included in SPT (not in sptSET). Vertex 7 is picked. So sptSet now becomes {0, 1, 7}. Update the distance values of adjacent vertices of 7. The distance value of vertex 6 and 8 becomes finite (15 and 9 respectively).



Pick the vertex with minimum distance value and not already included in SPT (not in sptSET). Vertex 6 is picked. So sptSet now becomes {0, 1, 7, 6}. Update the distance values of adjacent vertices of 6. The distance value of vertex 5 and 8 are updated.



We repeat the above steps until *sptSet* doesn’t include all vertices of given graph. Finally, we get the following Shortest Path Tree (SPT).



```
/* C Program to find Shortest Distances or Path using Dijkstra's algorithm */
#include<stdio.h>
#define MAX 100
#define TEMP 0
#define PERM 1
#define infinity 9999
#define NIL -1
void findPath(int s, int v );
void Dijkstra( int s);
int min_temp( );
void create_graph();
int n; /* Denotes number of vertices in the graph */
int adj[MAX][MAX];
int predecessor[MAX]; /*predecessor of each vertex in shortest path*/
int pathLength[MAX];
int status[MAX];
int main()
{
int s,v;
create_graph();
printf("\nEnter source vertex : ");
scanf("%d",&s);
Dijkstra(s);
while(1)
```

```
{
printf("\nEnter destination vertex(-1 to quit): ");
scanf("%d",&v);
if(v == -1)
break;
if(v < 0 || v >= n )
printf("\nThis vertex does not exist\n");
else if(v == s)
printf("\nSource and destination vertices are same\n");
else if( pathLength[v] == infinity )
printf("\nThere is no path from source to destination vertex\n");
else
findPath(s,v);
}
return 0;
}/*End of main()*/
void Dijkstra( int s)
{
int i,current;
/* Make all vertices temporary */
for(i=0; i<n; i++)
{
predecessor[i] =NIL;
pathLength[i] = infinity;
status[i] = TEMP;
}
/* Make pathLength of source vertex equal to 0 */
pathLength[s] = 0;
while(1)
{
/*Search for temporary vertex with minimum pathLength
and make it current vertex*/
current = min_temp( );
if( current == NIL )
return;
status[current] = PERM;
for(i=0; i<n; i++)
{
/*Checks for adjacent temporary vertices */
if ( adj[current][i] !=0 && status[i] == TEMP )
if( pathLength[current] + adj[current][i] < pathLength[i] )
{
predecessor[i] = current;/*Relabel*/
pathLength[i] = pathLength[current] + adj[current][i];
}
}
}
}/*End of Dijkstra( )*/
/*Returns the temporary vertex with minimum value of pathLength
Returns NIL if no temporary vertex left or
all temporary vertices left have pathLength infinity*/
int min_temp( )
{
int i;
int min = infinity;
int k = NIL;
for(i=0;i<n;i++)
{
if(status[i] == TEMP && pathLength[i] < min)
{
min = pathLength[i];
k = i;
}
}
return k;
}/*End of min_temp( )*/
void findPath(int s, int v )
{
int i,u;
int path[MAX]; /*stores the shortest path*/
int shortdist = 0; /*length of shortest path*/
int count = 0; /*number of vertices in the shortest path*/
/*Store the full path in the array path*/
while( v != s )
{
count++;
path[count] = v;
u = predecessor[v];
shortdist += adj[u][v];
v = u;
}
count++;
path[count]=s;
printf("\nShortest Path is : ");
for(i=count; i>=1; i--)
printf("%d",path[i]);
printf("\nShortest distance is : %d\n", shortdist);
}/*End of findPath()*/
```

```
void create_graph()
{
int i,max_edges,origin,destin, wt;
printf("\nEnter number of vertices : ");
scanf("%d",&n);
max_edges = n*(n-1);
for(i=1;i<=max_edges;i++)
{
printf("\nEnter edge %d( -1 -1 to quit ) : ",i);
scanf("%d %d",&origin,&destin);
if( (origin == -1) && (destin == -1) )
break;
printf("\nEnter weight for this edge : ");
scanf("%d",&wt);
if( origin >= n || destin >= n || origin<0 || destin<0)
{
printf("\nInvalid edge!\n");
i--;
}
else
adj[origin][destin] = wt;
}}
```