Sri Adichunchanagiri Shikshana Trust®

S.J.C. INSTITUTE OF TECHNOLOGY

DEPARTMENT OF ELECTRONICS & COMMUNICATION ENGINEERING, CHICKBALLAPUR – 562101

COMPUTER NETWORKS & PROTOCOLS LAB MANUAL

SUBJECT CODE: BEC702

SEMESTER: VII

2025-26

PREPARED BY:

RAVINDRA KUMAR M, Assistant Professor
SRIVANI E N, Assistant Professor
HARISH T L, Assistant Professor

VISION OF THE INSTITUTE

Preparing Competent Engineering and Management Professionals to Serve the Society

MISSION OF THE INSTITUTE

M1: Providing Students with a Sound Knowledge in Fundamentals of their branch of Study

M2: Promoting Excellence in Teaching, Training, Research and Consultancy.

M3: Exposing Students to Emerging Frontiers in various domains enabling Continuous Learning.

M4: Developing Entrepreneurial acumen to venture into Innovative areas.

M5: Imparting Value based Professional Education with a sense of Social Responsibility.

VISION OF THE DEPARTMENT

Transforming Individuals into Competent Electronics and Communication Engineers for Technological & Societal advancements.

MISSION OF THE DEPARTMENT

M1: Imparting fundamental knowledge in ECE for Academic and Professional Excellence.

M2: Empowering faculty and students through Research and Exploration in Emerging Technologies.

M3: Fostering Industry- Institute synergy to advance cutting-edge technological understanding and Entrepreneurial spirit.

M4: Equipping students with technical skills and ethical values to develop innovative solutions for societal needs.

M5: Nurturing Lifelong Learning through a supportive environment and continuous learning opportunities.

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

PROGRAM OUTCOMES

Engineering Graduates will be able to:

1. **Engineering Knowledge:** Apply knowledge of mathematics, natural science, computing, engineering fundamentals and an engineering specialization as specified in WK1 to WK4 respectively to develop to the solution of complex engineering problems.
2. **Problem Analysis:** Identify, formulate, review research literature and analyze complex engineering problems reaching substantiated conclusions with consideration for sustainable development. (WK1 to WK4)
3. **Design/Development of Solutions:** Design creative solutions for complex engineering problems and design/develop systems/components/processes to meet identified needs with consideration for the public health and safety, whole-life cost, net zero carbon, culture, society and environment as required. (WK5)
4. **Conduct Investigations of Complex Problems**: Conduct investigations of complex engineering problems using research-based knowledge including design of experiments, modelling, analysis & interpretation of data to provide valid conclusions. (WK8).
5. **Engineering Tool Usage:** Create, select and apply appropriate techniques, resources and modern engineering & IT tools, including prediction and modelling recognizing their limitations to solve complex engineering problems. (WK2 and WK6)
6. **The Engineer and The World:** Analyze and evaluate societal and environmental aspects while solving complex engineering problems for its impact on sustainability with reference to economy, health, safety, legal framework, culture and environment. (WK1, WK5, and WK7).
7. **Ethics:** Apply ethical principles and commit to professional ethics, human values, diversity and inclusion; adhere to national & international laws. (WK9)
8. **Individual and Collaborative Team work:** Function effectively as an individual, and as a member or leader in diverse/multi-disciplinary teams.
9. **Communication:** Communicate effectively and inclusively within the engineering
10. **Project Management and Finance:** Apply knowledge and understanding of engineering management principles and economic decision-making and apply these to one's own work, as a member and leader in a team, and to manage projects and in multidisciplinary environments.
11. **Life-Long Learning:** Recognize the need for, and have the preparation and ability for i) independent and life-long learning ii) adaptability to new and emerging technologies and iii) critical thinking in the broadest context of technological change. (WK8)

WK1: A systematic, theory-based understanding of the natural sciences applicable to the discipline and awareness of relevant social sciences.

WK2: Conceptually-based mathematics, numerical analysis, data analysis, statistics and formal aspects of computer and information science to support detailed analysis and modelling applicable to the discipline.

WK3: A systematic, theory-based formulation of engineering fundamentals required in the engineering discipline.

WK4: Engineering specialist knowledge that provides theoretical frameworks and bodies of knowledge for the accepted practice areas in the engineering discipline; much is at the forefront of the discipline.

WK5: Knowledge, including efficient resource use, environmental impacts, whole-life cost, reuse of resources, net zero carbon, and similar concepts, that supports engineering design and operations in a practice area

WK6: Knowledge of engineering practice (technology) in the practice areas in the engineering discipline.

WK7: Knowledge of the role of engineering in society and identified issues in engineering practice in the discipline, such as the professional responsibility of an engineer to public safety and sustainable development.

WK8: Engagement with selected knowledge in the current research literature of the discipline, awareness of the power of critical thinking and creative approaches to evaluate emerging issues.

WK9: Ethics, inclusive behavior and conduct. Knowledge of professional ethics, responsibilities, and norms of engineering practice. Awareness of the need for diversity by reason of ethnicity, gender, age, physical ability etc. with mutual understanding and respect, and of inclusive attitudes.

INDEX

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.

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

CO-PO MAPPING

	PO1	PO2	PO3	PO4	PO5	PO6	PO7	PO8	PO9	PO10	PO11
CO1	2		3	1	3				1		
CO2	2	3	3	1	3				1	1	1
CO3	2		3	3	3				1	1	
CO4	2		3	3	3				1	1	
AVERAGE	2	3	3	3	3				1	1	1

CO-PSO MAPPING

CO	PSO1	PSO2
CO1	3	2
CO2	3	2
CO3	3	1
CO4	3	3
AVERAGE	3	2

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 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 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
```

Steps for execution:

- *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
```

<u>Steps for execution:</u>

- *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++

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

1(i). Bit stuffing: Bit stuffing is a process of inserting an extra bit as 0, once the frame sequence encountered 5 consecutive 1's.

Program:

```
 #include<stdio.h>
#include<conio.h>
#include<string.h>
#define max 1000
void main()
{
char
in[max]={'\0'},out[max]={'\0'},des[max]={'\0'},flag[9]=
"01111110";
int inlen,i, ,j, k, outlen, len, count=0;
clrscr()
printf("Enter the data to be bit stuffed:\n");
gets(in);
inlen = strlen(in);
strcpy(out, flag);
for(i=0, j=8; i<inlen; i++)
{
     if(in[i] = ='1')
          count++;

     else
          count=0;

     out[j++] = in[i];

     if(count = =5)
     {
          out[j++]='0';
          count=0;
     }
}
out[j] = '\0';
strcat(out,flag);
printf("The bit stuffed frame is:\n%s",out);
getch( );
     }
```

1(ii). Character stuffing : Character Stuffing is process in which DLESTX and DLEETX are used to denote start and end of character data with some constraints imposed on repetition of characters.

Program:

```
#include<stdio.h>
#include<conio.h>
#include<string.h>
void main()

{
char in[100]={‘\0’},
out[100]={‘\0’}, f1[8]=”DLE STX”;
char f2[8]=”DLE ETX”, des[100];

int i, j, inlen, outlen, k;
clrscr();
printf(“Enter the data to be character stuffed\n”);
gets(in);

printf(“The input data is :%s\n”, in);
inlen = strlen(in);
printf(“The data length is :%d\n”, inlen);
strcpy(out,f1);

for(i=0,j=7; i<inlen; i++)
{
     if(in[i] = =’D’ && in[i+1] = =’L’ && in[i+2] = =’E’)
     {
          strcat(out,”DLEDLE”);
          i = i+2;
          j = j+5;
     }
out[j++] = in[i];
}

out[j]=’\0’;
strcat(out,f2);

printf(“The character stuffed
data is :%s\n”, out); getch(
);
  }
```

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

Distance vector routing algorithms operate by having each router maintain a table (i.e., vector) giving the best known distance to each destination and which line to get there. These tables are updated by exchanging information with the neighbours.

Program:

```
#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;
}
for(i=0;i<nodes;i++)
{
printf("\n\n For router %d\n",i);
for(j=0;j<nodes;j++)
printf("\t\nnode %d via %d Distance
%d",j,rt[i].from[j],rt[i].dist[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 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");
}
```

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

Dijkstra algorithm is also called single source shortest path algorithm. The algorithm maintains a list visited[ ] of vertices, whose shortest distance from the source is calculated.

Program:

```
#include<stdio.h>
#include<conio.h>
#define INFINITY 99
#define MAX 10
#define startnode 0
void dijkstra(int cost[MAX][MAX],int n);
int main()
{
  int cost[MAX][MAX],i,j,n,u;
  printf("Enter no. of vertices:");
  scanf("%d",&n);
  printf("\nEnter the cost matrix:\n");
  for(i=0;i<n;i++)
    for(j=0;j<n;j++)
      scanf("%d",&cost[i][j]);
  dijkstra(cost,n);
  return 0;
}

void dijkstra(int cost[MAX][MAX],int n)
{
  int distance[MAX],pred[MAX];
  int visited[MAX],count, mindistance, nextnode, 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);
         }
    }
```

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<conio.h>
#include<string.h>

unsigned int xor2div(char *i, char *a, int mode)
{
  unsigned int j,  k;
  char g[81]={"100011"};
  strcpy(a, i);
  if(mode)
  strcat(a,"00000");
  for(j=0;j<strlen(i);j++)
  if(*(a+j) = ='1')

  for(k=0;k<strlen(g);k++)
  {
   if (((*(a+j+k = ='0') && (g[k] = ='0')) || ((*(a+j+k =
   ='1') && (g[k] = ='1')))

          *(a+j+k)='0';
          else
          *(a+j+k)='1';
  }
 for (j=0;j<strlen(a);j++)
 {
          if(a[j]=='1')
          return(1);
 }
return(0);
}

void main()
{
 char i[81]={'\0'}, a[81]= {'\0'}, r[81]= {'\0'};
 clrscr();
 printf("\n Enter the data in binary: \t");
 scanf("%s",&i)
 xor2div(i,a,1);
 printf(" \n CRC-CCITT code is : \n %s%s ", I,
 a+strlen(i));

 printf("\n Enter the received code in binary: \n);
 scanf("%s", &r);
if (strlen(a) = = strlen(r))
{
 if(!xor2div(r,a,0))
```

```
 printf(" \n The received data is error free ");
 else
 printf("\n The received data is error ");
}
else
printf(" \n Wrong input, run the program with correct
input");
getch( );
}
```

5. Implementation of Stop and Wait Protocol and Sliding Window Protocol

5(a). Stop and Wait protocol

Program:

```
#include <stdio.h>
#include <stdlib.h>
#define RTT 4

#define TIMEOUT 4
#define TOT_FRAMES 7
enum {NO,YES} ACK;
int main()
{
int wait_time,i=1;
ACK=YES;
for(;i<=TOT_FRAMES;)
{
if (ACK==YES && i!=1)
{
printf("\nSENDER: ACK for Frame %d Received.\n",i-1);
}
printf("\nSENDER: Frame %d sent, Waiting
for ACK...\n",i); ACK=NO;
wait_time= rand() % 4+1;
if (wait_time==TIMEOUT)
{
printf("SENDER: ACK not received for Frame
%d=>TIMEOUT Resending Frame...",i);
}
else
{
sleep(RTT);
printf("\nRECEIVER: Frame %d received, ACK sent\n",i);
printf("--------------------------------------------");
ACK=YES;
i++;
}
}
return 0;
}
```

5(b). Sliding window protocol program

Program:

```
#include <stdio.h>
#include <stdlib.h>
#define RTT 5
int main()
{
   int window_size,i,f,frames[50];
        printf("Enter window size: ");
   scanf("%d",&window_size);
   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("\nAfter sending %d frames at each stage
   sender waits for ACK ",window_size);
   printf("\nSending frames in the following
   manner....\n\n");
   for(i=1;i<=f;i++)
   {
   if(i%window_size!=0)
   {
   printf(" %d",frames[i]);
   }
   else
   {
   printf(" %d\n",frames[i]);
   printf("SENDER: waiting for ACK...\n\n");
   sleep(RTT/2);
   printf("RECEIVER: Frames Received, ACK Sent\n");
   printf("-------------------------------------------\n");
   sleep(RTT/2);
   printf("SENDER:ACK received, sending next frames\n");
   }
   }
   if(f%window_size!=0)
   {
   printf("\nSENDER: waiting for ACK...\n");
   sleep(RTT/2);
    printf("\nRECEIVER:Frames Received, ACK Sent\n");
   printf("--------------------------------------------------
   \n");
   sleep(RTT/2);
   printf("SENDER:ACK received.");
   }
   return 0;
}
```

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

In Leaky bucket, 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.

Program:

```
#include<stdio.h>
#define bucketsize 1000
#define n 5
void bucketoutput(int *bucket,int op)
{
if(*bucket > 0 && *bucket > op)
{
*bucket= *bucket-op;
printf("\n%d-outputed remaining is %d",op,*bucket);
}
else if(*bucket > 0)
{
printf("\nRemaining data output = %d",*bucket);
*bucket=0;
}
}
int main()
    {
    int op,newpack,oldpack=0,wt,i,j,bucket=0;
    printf("enter output rate");
    scanf("%d",&op);
    for(i=1;i<=n;i++)
    {
    newpack=rand()%500;
    printf("\n\n new packet size = %d",newpack);
    newpack=oldpack+newpack;
    wt=rand()%5;
    if(newpack<bucketsize)
      bucket=new
            ack;
    else
    {
    printf("\n%d = the newpacket and old pack is
    greater than bucketsize reject",newpack);
    bucket=oldpack;
    }
    printf("\nThe data in bucket = %d",bucket);
    printf("\n the next packet will arrive after = %d
    sec",wt);

    for(j=0;j<wt;j++)
```

```
{
bucketoutput(&bucket,op);

sleep(1);

}
oldpack=bucket;
}
while(bucket>0)
bucketoutput(&bucket,op);
return 0;
```

NETWORK SIMULATOR 2 (NS2)

(https://www.tutorialsweb.com/ns2)

1. What is NS2

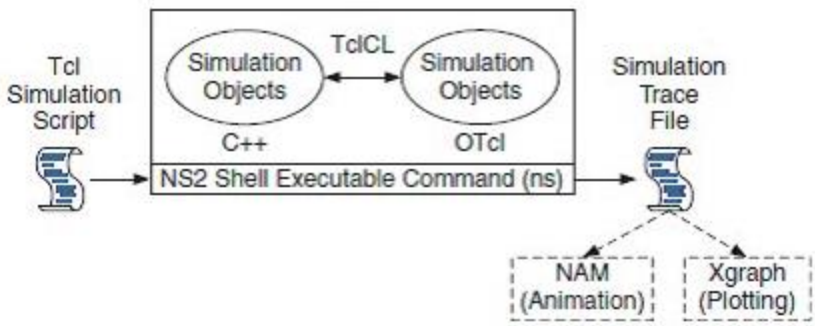
NS2 stands for Network Simulator Version 2. It is an open-source event-driven simulator designed specifically for research in computer communication networks.

2. Features of NS2

- It is a discrete event simulator for networking research.
- It provides substantial support to simulate bunch of protocols like TCP, FTP, UDP, https and DSR.
- It simulates wired and wireless network.
- It is primarily Unix based.
- Uses TCL as its scripting language.
- Otcl: Object oriented support
- Tclcl: C++ and otcl linkage
- Discrete event scheduler

3. Basic Architecture

NS2 consists of two key languages: C++ and Object-oriented Tool Command Language (OTcl). While the C++ defines the internal mechanism (i.e., a backend) of the simulation objects, the OTcl sets up simulation by assembling and configuring the objects as well as scheduling discrete events. The C++ and the OTcl are linked together using TclCL



Basic architecture of NS.

4. Why two language? (TCL and C++)

NS2 uses OTcl to create and configure a network, and uses C++ to run simulation. All C++ codes need to be compiled and linked to create an executable file.

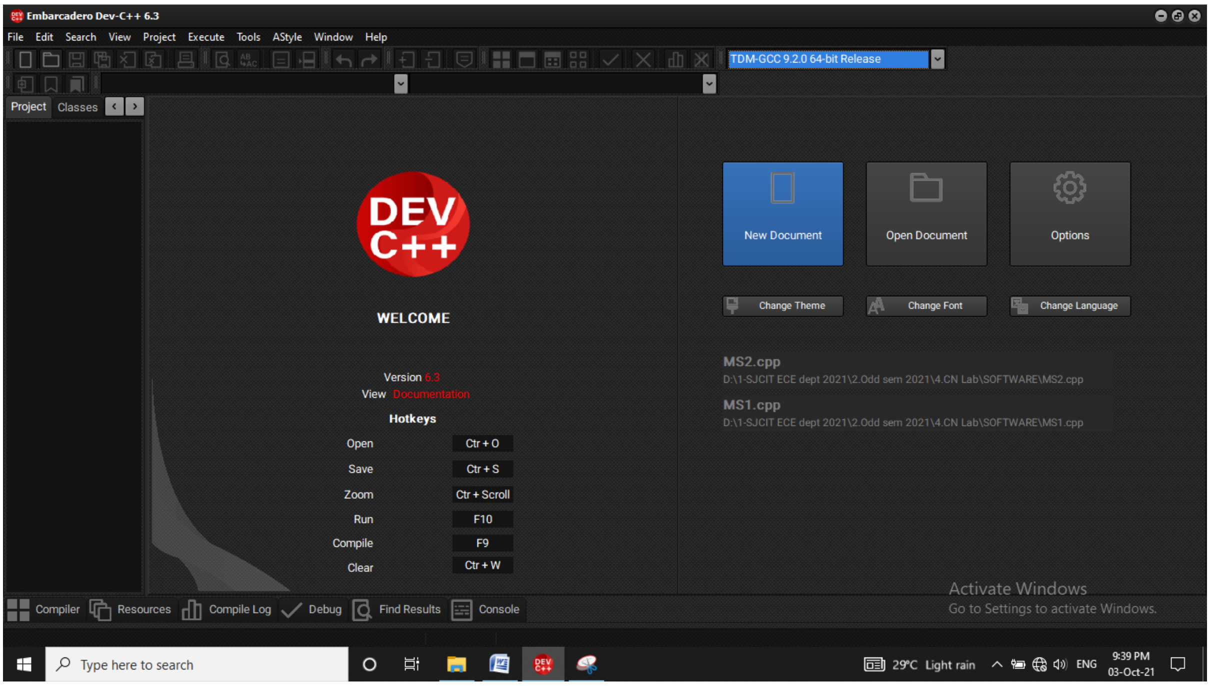
DEV C++ EDITOR

Step 1: Opening DEV C++

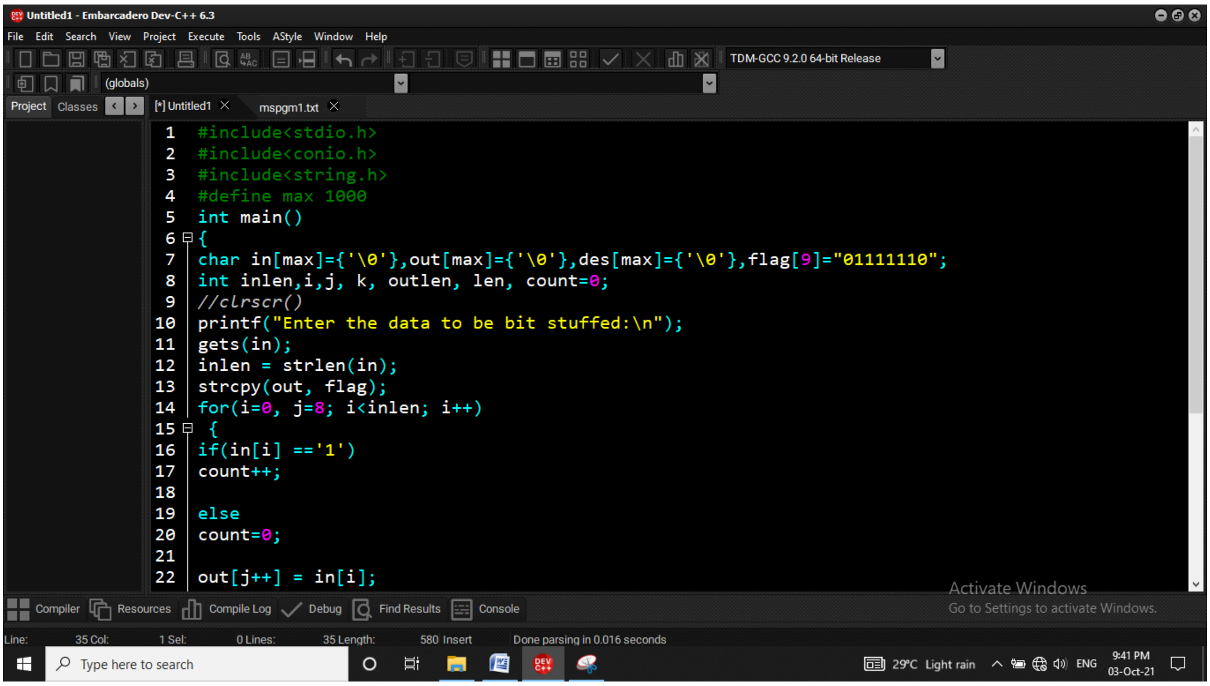
Double click on the Dev C++ icon available on the desktop to open the editor.

Step 2: Coding new c/c++ Program

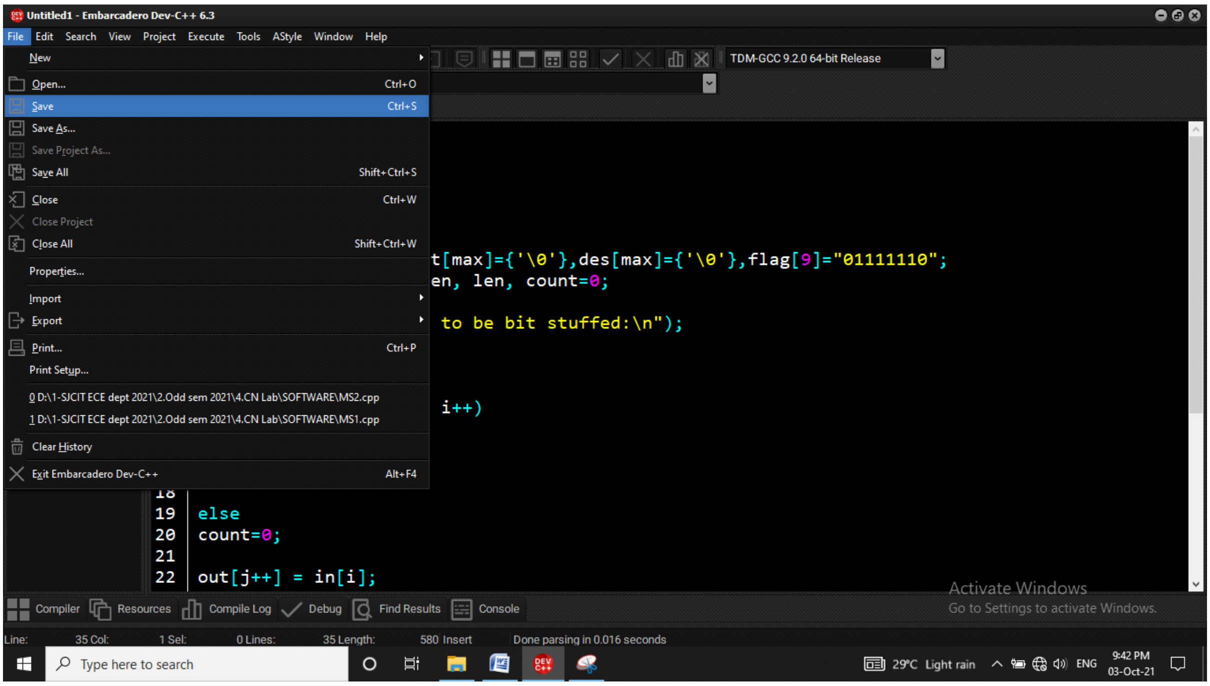
Click on the "New Document" option on the welcome screen for coding new program.



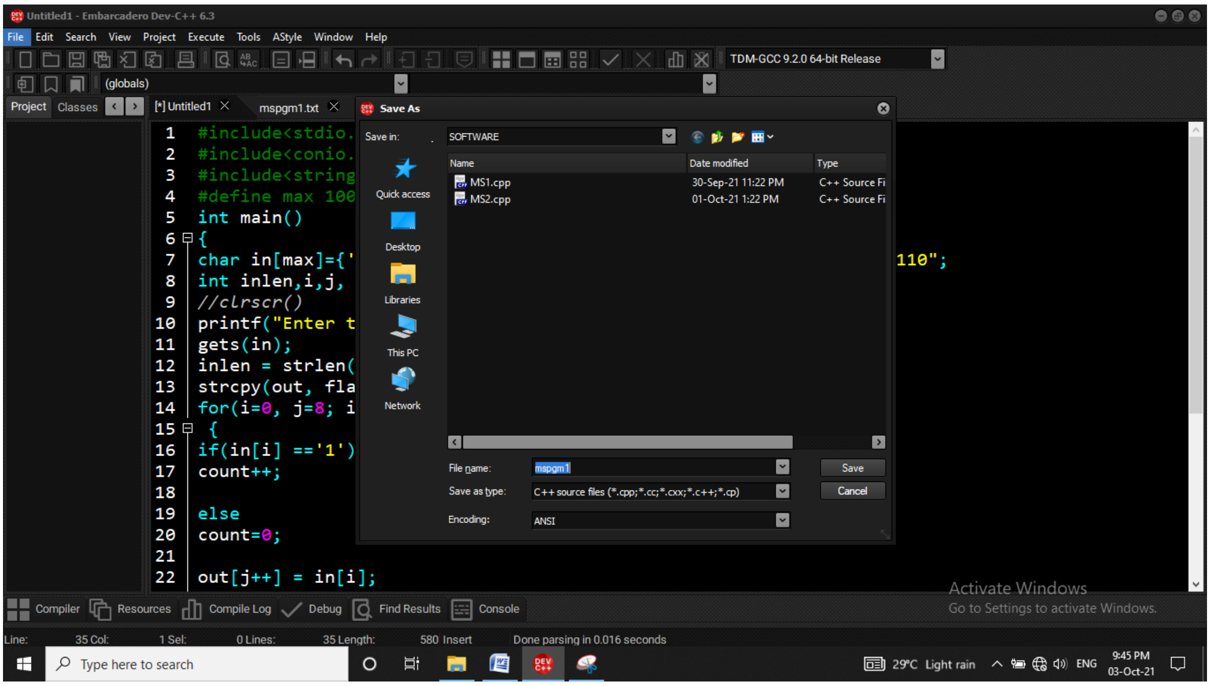
Step 3: Code the C program



Step 4: Save the Program by selecting File→Save



Step 5: Provide the Name of program like "mspgm1" and click save.

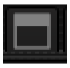


Step 6: Press F9 (or) icon  **to compile the errors.**

Step 7: Eliminating Errors

If any Errors are present it will listed under Compiler option. Once program is free from errors below message will be displayed.

- Errors: 0
- Warnings: 0

Step 8: Press F10 (or)  **to Run the program.**

QUESTION BANK

PART-A	
A.1.	**Using NS2,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.**
A.2.	**Using NS2,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.**
A.3.	**Using NS2,Implement Ethernet LAN using n (6-10) nodes. Compare the throughput by changing the error rate and data rate**
A.4.	**Using NS2,Implement Ethernet LAN using n nodes and assign multiple traffic to the nodes and obtain congestion window for different sources/ destinations.**
A.5.	**Using NS2,Implement ESS with transmission nodes in Wireless LAN and obtain the performance parameters.**
A.6.	**Using NS2,Implementation of Link state routing algorithm.**

PART-B	
B.1.	**Write a C/C++ program for a HLDC frame to perform the following (i)Bit stuffing (ii) Character stuffing**
B.2.	**Write a C/C++ program for distance vector algorithm to find suitable path for transmission.**
B.3.	**Implement Dijkstra’s algorithm to compute the shortest routing path using C/C++**
B.4.	**For the given data, use CRC-CCITT polynomial to obtain CRC code. Verify the C/C++ program for the cases a.Without error b. With error**
B.5.	**Implement using C/C++ for Stop and Wait Protocol and Sliding Window Protocol**
B.6.	**Write a C/C++ program for congestion control using leaky bucket algorithm**