VSM's

SOMASHEKHAR R KOTHIWALE INSTITUTE OF TECHNOLOGY, NIPANI

DEPT. OF ELECTRONICS & COMMUNICATION ENGINEERING.

COMPUTER COMMUNICATION NETWORKS LABORATORY

(17ECL68)

LAB MANUAL

2019-20

Faculty Incharge

Prof. Prashant M. Ganji

Asst. Prof. Electronics & Communication Dept.

Syllabus

COMPUTER NETWORKS LAB B.E., VI Semester, Electronics & Communication Engineering [As per Choice Based Credit System (CBCS) Scheme]			
Course Code	**17ECL68**	**CIE Marks**	**40**
Number of Lecture Hours/Week	**01Hr Tutorial (Instructions) + 02 Hours Laboratory = 03**	**SEE Marks**	**60**
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: C or C++ Programming

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

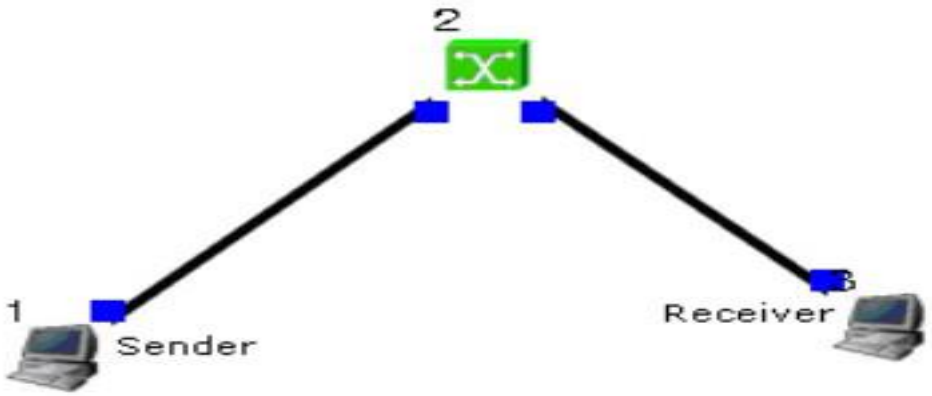
b. With error

5. Implementation of Stop and Wait Protocol and Sliding Window Protocol
6. Write a program for congestion control using leaky bucket algorithm.

PART – A

DEMO EXPERIMENT

TITLE-SIMULATE A THREE-NODE POINT-TO-POINT NETWORK WITH A DUPLEX LINK BETWEEN THEM. SET THE QUEUE SIZE AND VARY THE BANDWIDTH AND FIND THE NUMBER OF PACKETS DROPPED.



Procedure:

Step1: Drawing topology

1. Select/click the HOST icon on the tool bar and click the left mouse button on the editor, to place a HOST1 on the editor. Repeat the above procedure and place another host "HOST2" on the editor.
2. Select/click the HUB icon on the tool bar and click the left mouse button on the editor, to place HUB1 on the editor.
3. Click on the LINK icon on the tool bar and connect HOST1 to HUB1 and HUB1 to HOST2
4. Click on the "E" icon on the tool bar to save the current topology e.g: file1.tpl (Look for the ******.tpl extension.)

NOTE: Changes cannot / (should not) be done after selecting the "E" icon.

Step2: Configuration

1. Double click the left mouse button while cursor is on HOST1 to open the HOST window.
2. Select Add button on the HOST window to invoke the command window and provide the following command in the command text box.

stg –u 1024 100 1.0.1.2 for UDP

or

stcp -p 7000 -l 1024 1.0.1.2 for TCP

3. Click OK button on the command window to exit and once again click on the OK button on the HOST window to exit.
4. Double click the left mouse button while cursor is on HOST2 to open the HOST window.
5. Select Add button on the HOST window to invoke the command window and provide the following command in the command text box.

rtg –u –w log1 for UDP

or

rtcp -p 7000 -l 1024 for TCP

6. Click OK button on the command window to exit.
7. Click NODE EDITOR Button on the HOST window and select the MAC tab from the modal window that pops up.
8. Select LOG STATISTICS and select check boxes for Number of Drop Packet and Number of Collisions in the MAC window
9. Click OK button on the MAC window to exit and once again click on the OK button on the HOST window to exit.

Note: To set QUEUE size

1. Double click the left mouse button while cursor is on HOST2 to open the HOST window.
2. Click NODE EDITOR Button on the HOST window and select the FIFO tab from the modal window that pops up.
3. Change Queue size (Default 50).
4. Click OK button on the FIFO window to exit and once again click on the OK button on the HOST window to exit.

Step3: Simulate

1. Click "R" icon on the tool bar
2. Select Simulation in the menu bar and click/ select RUN in the drop down list to execute the simulation.
3. To start playback select "►" icon located at the bottom right corner of the editor.
4. To view results, Open Output throughput, Input Throughput and Drop log files from file1.results folder in separate word processor.

Caution: file1 is the hypothetical name given to this simulation.

Changing configurations

Change 1

1. Open the above file,
2. Do not change the topology or any other configuration,
3. Select E icon on the tool bar
4. Reduce the bandwidth at link by double clicking the left mouse button
5. Repeat Step3 (Simulate)

Change 3

1. Open the above file,
2. Remove HUB and replace it with SWITCH.
3. Do not change anything in the configuration
4. Repeat Step3(Simulate)

By setting the bandwidth as 10 Mbps on both the links and queue size as 50 we obtain the

following results: bandwidth to 10 Mbps in the destination link, we obtain the following results

Output throughput nl-pl = 1177

Input throughput n3-pl = 1177

Collision and drop = 0

By changing bandwidth to 9Mbps in the destination link, we obtain the following results:

Output throughput nl-pl =1177

Input throughput n3-pl = - 0

Collision and drop = 1100

Note: The results of the experiments vary from simulation to simulation.

By using SWITCH

Results: By setting the bandwidth as 10 Mbps on both the links and queue size as 50 we obtain the following results:

output throughput nl-pl= 1190

input throughput n3-pl = 1190

collision and drop = 0

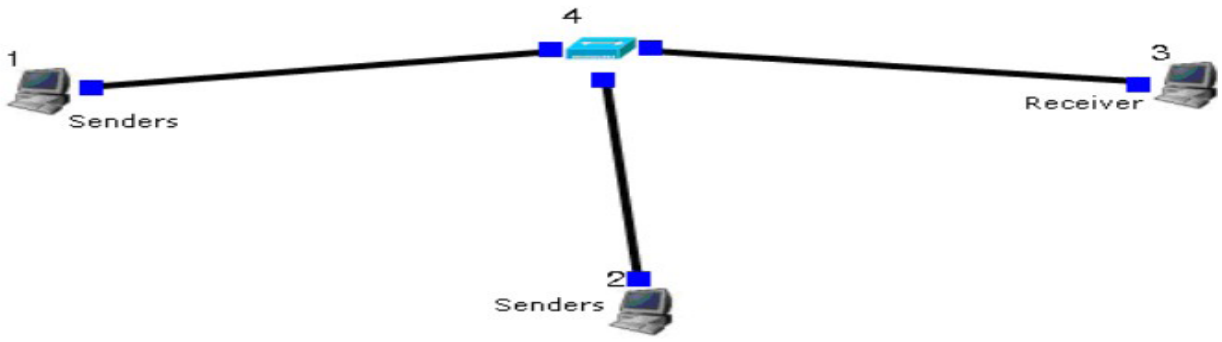
By changing bandwidth to 9Mbps in the destination link, we obtain the following results:

Output throughput nl-pl =1190

Input throughput n3-pl = varying

EXPERIMENT 2

TITLE- SIMULATES A FOUR NODE POINT-TO-POINT NETWORK WITH THE LINKS CONNECTED AS FOLLOWS: 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.



Procedure:

Step1: Drawing topology

1. Select/click the HOST icon on the tool bar and click the left mouse button on the editor, to place a host on the editor. Repeat the above procedure and place two other hosts "HOST2" and "HOST3" on the editor.
2. Select/click the HUB (or SWITCH) icon on the tool bar and click the left mouse button on the editor, to
   place a HUB (or SWITCH) on the editor.
3. Click on the LINK icon on the tool bar and connect HOST1 to HUB, HOST2 to HUB and HUB to HOST3
4. Click on the "E" icon on the tool bar to save the current topology e.g: file2.tpl (Look for the ******.tpl
   extension.)

NOTE: Changes cannot / (should not) be done after selecting the "E" icon.

Step2: Configuration

1. Double click the left mouse button while cursor is on HOST1 to open the HOST window.
2. Select Add button on the HOST window to invoke the command window and provide the following
   command in the command text box.
   stcp –p 7000 –l 1024 1.0.1.3 (for TCP start time 0.0 and end time 20.0)
3. Click OK button on the command window to exit
4. Click NODE EDITOR Button on the HOST window and select the MAC tab from the modal window that
   pops up.
5. Select LOG STATISTICS and select check box for output throughput in the MAC window
6. Click OK button on the MAC window to exit and once again click on the OK button on the HOST window to exit
7. Double click the left mouse button while cursor is on HOST2 to open the HOST window.
8. Select Add button on the HOST window to invoke the command window and provide the following

command in the command text box.

stg –u 1024 100 1.0.1.3 (for UDP start time 21.0 and end time 40.0)

9. Click OK button on the command window to exit

10. Click NODE EDITOR Button on the HOST window and select the MAC tab from the modal window that pops up.

11. Select L

OG STATISTICS and select check box for output throughput in the MAC window

12. Click OK button on the MAC window to exit and once again click on the OK button on the HOST window to exit.

13. Double click the left mouse button while cursor is on HOST3 to open the HOST window.

14. Select Add button on the HOST window to invoke the command window and provide the following

command in the command text box.

rtcp –p 7000 –l 1024 (for TCP start time 0.0 and end time 20.0)

15. Click OK button on the command window to exit.

16. Also add the following common on HOST3

rtg –u –w log1 (for UDP start time 21.0 and end time 40.0)

17. Click NODE EDITOR Button on the HOST window and select the MAC tab from the modal window that pops up.

18. Select LOG STATISTICS and select check box for input and output throughput in the MAC window

19. Click OK button on the MAC window and once again click on the OK button on the HOST window to exit.

Step3: Simulate

1. Click "R" icon on the tool bar

2. Select Simulation in the menu bar and click/ select RUN in the drop down list to execute the simulation.

3. To start playback select "►" icon located at the bottom right corner of the editor.

4. To view results, Open input and output throughput log files from file2.results folder in separate word processor.

Caution: file2 is the hypothetical name given to this simulation

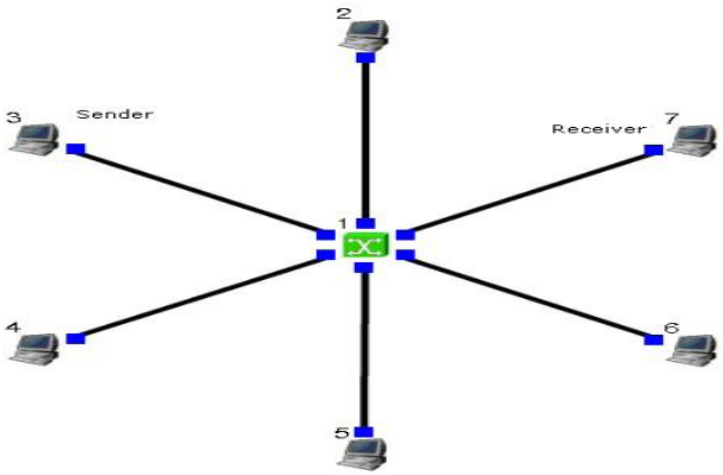
RESULTS ANALYSIS :

With Bandwidth 10 MBPS and BER 0.000000000

Time	Packets		
	N1_Output Throughput (TCP)	N2_Output Throughput (UDP)	N3_Input Throughput (TCP+UDP)
1			
2			
3			
4			
5			

EXPERIMENT -3

TITLE-SIMULATE THE TRANSMISSION OF PING MESSAGES OVER A NETWORK TOPOLOGY CONSISTING OF 6 NODES AND FIND THE NUMBER OF PACKETS DROPPED DUE TO CONGESTION.



Procedure:

Step1: Drawing topology

1. Select/click the SUBNET icon on the tool bar and click the left mouse button on the editor, to place a
   SUBNET on the editor.
2. A pop up window appears requesting the number of nodes and radius for the subnet, Set number of
   nodes=6; Set radius of subnet >150
3. Click on the "E" icon on the tool bar to save the current topology e.g: file4.tpl (Look for the ******.tpl
   extension.)

NOTE: Changes cannot / (should not) be done after selecting the "E" icon.

Step2: Configuration

1. Double click the left mouse button while cursor is on a HOST to open the HOST window.
2. Click NODE EDITOR Button on the HOST window and select the INTERFACE tab (1st tab) from the modal window that pops up.
3. Determine the IP address of the selected host.
4. Click OK button on the INTERFACE window to exit and once again click on the OK button on the HOST window to exit.
5. Repeat the above step for 2 other HOSTS
6. Also click NODE EDITOR Button on the HOST window and select the MAC tab from the modal window
   that pops up.
7. Select LOG STATISTICS and select check box for drop and collision log statistics in the MAC window
8. Click OK button on the MAC window to exit and once again click on the OK button on the HOST window to exit.
9. Repeat steps 6 to 9 for the other hosts selected at step 5.
10. Select G _ setting from the menu bar and select Simulation from the drop down list Set

simulation

time>600sec

Step3: Simulate

1. Click “R” icon on the tool bar
2. Select Simulation in the menu bar and click/ select RUN in the drop down list to execute the simulation.
3. During simulation, double click the mouse button on a HOST, the HOST window pops up, select / click on command console button located at the bottom.
4. A terminal window appears, type ping IP address of a host in the subnet at the command prompt.

Note: The no: of drop packets are obtained only when the traffic in more in network.

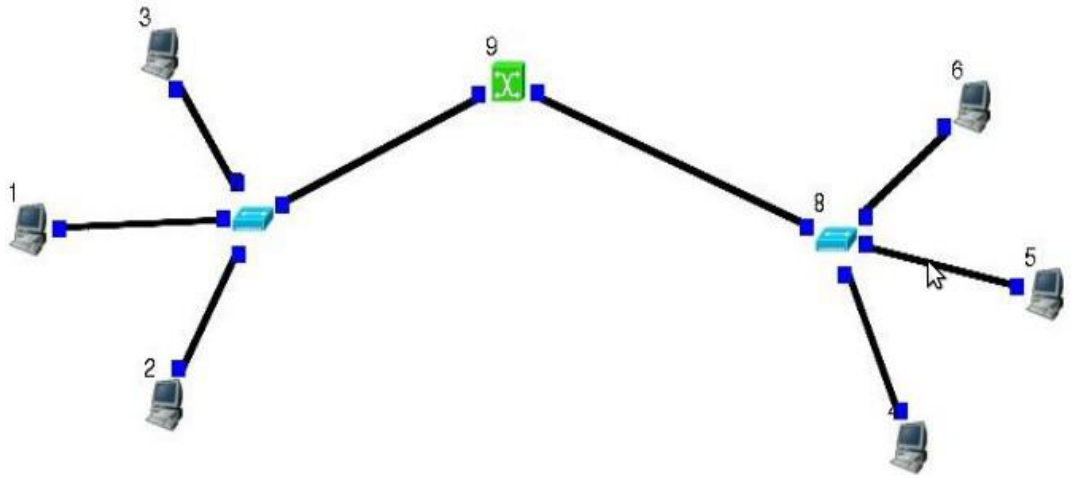
Results Analysis:

WITH BANDWIDTH 10 MBPS AND BER 0.000000000

Time	No. of Packets		
	N3_Output Throughput	N7_Collision	N7_Drop
1			
2			
3			
4			
5			

EXPERIMENT 4

Simulate an Ethernet LAN using N nodes (6-10), change error rate and data rate and compare throughput.



Step1: Drawing topology

1. Select/click the HOST icon on the tool bar and click the left mouse button on the editor, to place HOST1 on the editor.

i. Repeat the above procedure and place 5 other hosts “HOST2”, “HOST3”, “HOST4”, “HOST5”, and “HOST6”on the editor.

2. Select/click the HUB icon on the tool bar and click the left mouse button on the editor, to place HUB1 on the editor. Repeat the above procedure and place another host “HUB2” on the editor

3. Click on the LINK icon on the tool bar and connect HOST1, HOST2 and HOST3 to HUB1, HOST4, HOST5 and HOST6 to HUB2.

4. Select/click the SWITCH icon on the tool bar and click the left mouse button on the editor, to place SWITCH1 on the editor.5. Click on the LINK icon on the tool bar and connect HUB1 to SWITCH1 and HUB2 to SWITCH1.

6. Click on the “E” icon on the tool bar to save the current topology e.g: file5.tpl (Look for the *******.tpl extension.)

NOTE: Changes cannot / (should not) be done after selecting the “E” icon.

Step2: Configuration

1. Double click the left mouse button while cursor is on HOST1 to open the HOST window.

2. Select Add button on the HOST window to invoke the command window and provide the following command in the command text box.

stcp[-p port] [-1 writesizej host IP addr

3. Click OK button on the command window to exit and once again click on the OK button on the HOST window to exit.

4. Repeat this step at HOST 2 and HOST3,

5. Double click the left mouse button while cursor is on HOST4 to open the HOST window.

6. Select Add button on the HOST window to invoke the command window and provide the

following
command in the command textbox.

rtcp [-p port] [-l read size]

7. Click OK button on the command window to exit.
8. Click NODE EDITOR Button on the HOST window and select the MAC tab from the modal window that
pops up.
9. Select LOG STATISTICS and select check box for output throughput in the MAC window
10. Click OK button on the MAC window to exit and once again click on the OK button on the HOST window to exit.
11. Repeat this step at HOST 5 and HOST6,
12. Double click the left mouse button while cursor is on HOST5 to open the HOST window.
13. Click NODE EDITOR Button on the HOST5 window and select the PHYSICAL tab from the modal window that pops up.
14. Change Bit Error Rate
15. Click OK button on the PHYSICAL window to exit and once again click on the OK button to return to the HOST window
16. Click NODE EDITOR Button on the HOST window and select the MAC tab from the modal window that pops up.
17. Select LOG STATISTICS and select check box for output throughput in the MAC window
18. Click OK button on the MAC window to exit and once again click on the OK button on the HOST window to exit.
19. Repeat this step HOST6, Change Bandwidth this time while undoing the change in Bit Error Rate, also select the output throughput at HOST6.

Step3: Simulate

1. Click “R” icon on the tool bar
2. Select Simulation in the menu bar and click/ select RUN in the drop down list to execute the simulation.
3. To start playback select “►” icon located at the bottom right corner of the editor.
4. To view results, Open input and output throughput log files from file5.results folder in separate word
processor.

Caution: file5 is the hypothetical name we gave to this simulation

Results:

WITH BANDWIDTH 10 MBPS AND BER 0.000000000

Time	No. of Packets					
	Output Throughput			Output Throughput		
	Node_N1	Node_N2	Node_N3	Node_N4	Node_N5	Node_N6
1						
2						
3						
4						
5						

WITH BANDWIDTH 8 MBPS AND BER 0.0000000000

Time	No. of Packets					
	Output Throughput			Output Throughput		
	Node_N1	Node_N2	Node_N3	Node_N4	Node_N5	Node_N6
1						
2						
3						
4						
5						

WITH BANDWIDTH 10 MBPS AND BER 0.0000005000

Time	No. of Packets					
	Output Throughput			Output Throughput		
	Node_N1	Node_N2	Node_N3	Node_N4	Node_N5	Node_N6
1						
2						
3						
4						
5						

EXPERMENT 5

TITLE-SIMULATE AN ETHERNET LAN USING N NODES AND SET MULTIPLE TRAFFIC NODES AND PLOT CONGESTION WINDOWS FOR DIFFERENT SOURCES/ DESTINATION.

PROCEDURE - PROGRAMME - ACTIVITY :

Step1: Drawing topology

1. connect one set of hosts with a hub and anther set of hosts also through a hub connect these two hubs through a switch. This forms an Ethernet LAN.
2. Setup multiple traffic connection between the hosts on one hub and hosts on another hub using the following command.

   Stcp [-p port] [-l writesize] hostIPaddr

   Rtcp[-p port] [-l readsize]
3. Setup the collision log at the destination hosts in the MAC layer as described in the earlier experiments.

4. To plot the congestion window go to Menu → Tools → Plot Graph → File-open → Filename. Results→ filename.coll.log.
5. View the results in the filename Results.

RESULTS:

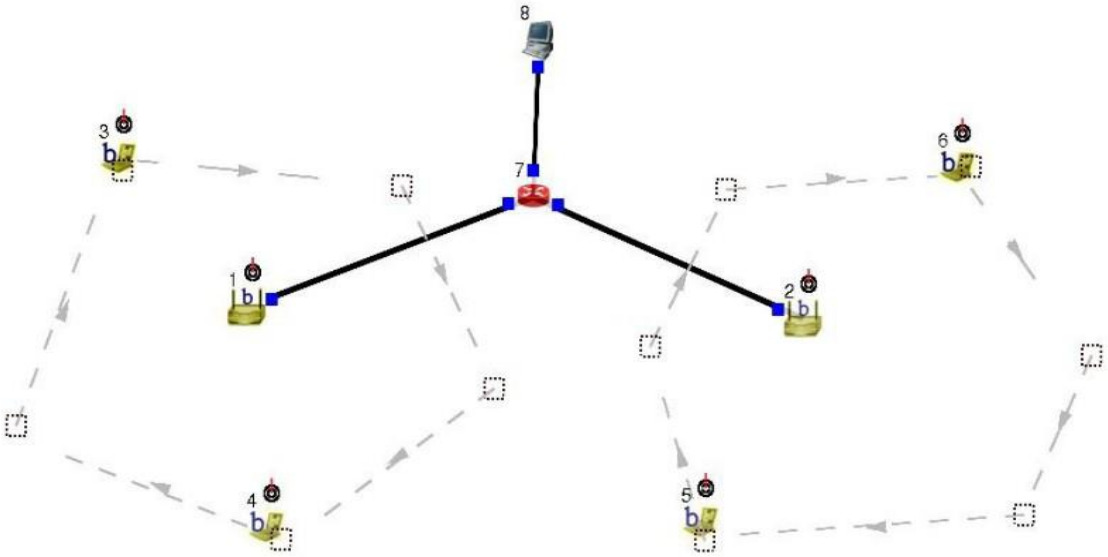
RESULTS:

Drops at node 5: 324 - 7560

Drops at node4: 274 - 930

EXPERIMENT 6

TITLE-SIMULATE SIMPLE ESS AND WITH TRANSMITTING NODES IN WIRELESS LAN BY SIMULATION AND DETERMINE THE PERFORMANCE WITH RESPECT TO TRANSMISSION OF PACKETS.



PROCEDURE:

Step1: Drawing topology

1. Select/click the HOST icon on the tool bar and click the left mouse button on the editor, to place HOST1 on the editor.
2. Select/click the ROUTER icon on the tool bar and click the left mouse button on the editor, to place ROUTER1 on the editor.
3. Select/click the WIRELESS ACCESS POINT(802.11b) icon on the tool bar and click the left mouse button on the editor, to place ACCESS POINT 1 on the editor.
4. Repeat this procedure and place ACCESS POINT 2 on the editor. Select/click the MOBILE NODE (infrastructure mode) icon on the tool bar and click the left mouse button on the editor, to place MOBIL NODE 4 on the editor.
5. Click on the LINK icon on the tool bar and connect ACCESS POINT1 to ROUTER1 and ACCESS POINT2 to ROUTER1
6. Click on the "Create a moving path" icon on the tool bar and draw moving path across MOBILE NODE1 and 2, Repeat for MOBILE NODE 3and 4 (Accept the default speed value 10 and close the window, Click the right mouse button to terminate the path).

To create Subnet

7. Select wireless subnet icon in the tool bar now select MOBILE NODE1 MOBILE NODE2 and ACCESS POINT1 by clicking on left mouse button, and clicking right mouse button will create a subnet.

8. Repeat the above step for MOBILE NODE3, MOBILE NODE4 and ACCESS POINT2.
9. Click on the "E" icon on the tool bar to save the current topology e.g: file8.tpl (Look for the ******.tpl
extension.)
NOTE: Changes cannot / (should not) be done after selecting the "E" icon.

Step2: Configuration

1. Double click the left mouse button while cursor is on HOST1 to open the HOST window.
2. Select Add button on the HOST window to invoke the command window and provide the following
command in the command text box.
ttcp –r –u –s –p 8001
3. Click OK button on the command window to exit

4. Repeat this step and add the following commands at HOST1
ttcp –r –u –s –p 8002
ttcp –r –u –s –p 8003
ttcp –r –u –s –p 8004
5. Click NODE EDITOR Button on the HOST1 window and select the MAC tab from the modal window that pops up.
6. Select LOG STATISTICS and select check box for Input throughput in the MAC window
7. Click OK button on the MAC window to exit and once again click on the OK button on the HOST window to exit.
8. Double click the left mouse button while cursor is on MOBILE NODE 1 to open the MOBILE NODE
window.
9. Select Application tab and select Add button to invoke the command window and provide the following
command in the command text box.
ttcp –t –u –s –p 80011.0.2.2 (host's ip address)
10. Click NODE EDITOR Button on the MOBILE NODE1 window and select the MAC tab from the nodal
window that pops up.
11. Select LOG STATISTICS and select check box for Output throughput in the MAC window
12. Click OK button on the MAC window to exit and once again click on the OK button on the MOBILE NODE1 window to exit.
13. Repeat the above steps (step 8 to step12) for the MOBILE NODE2,3 and 4 and add the following
Commands at MOBILE NODE2:- ttcp –t –u –s –p 8002 1.0.2.2
MOBILE NODE 3:- ttcp –t–u –s –p 8003 1.0.2.2
MOBILE NODE4:- ttcp –t –u –s –p 8004 1.0.2.2
14. Double click the left mouse button while cursor is on ROTER1 to open the ROUTER window.
15. Click NODE EDITOR Button on the ROUTER1 window and you can see three stacks. two stacks for two ACCESS POINTS and another stack for HOST1 which is connected to the ROUTER1.
16. Select the MAC tab of ACCESS POINT1 and Select LOG STATISTICS and select check box for Input

throughput in the MAC window. Click OK button on the MAC window to exit.

17. Select the MAC tab of ACCESS POINT2 and Select LOG STATISTICS and select check box for Input
throughput in the MAC window. Click OK button on the MAC window to exit.

18. Select the MAC tab of HOST1 and Select LOG STATISTICS and select check box for Output throughput in the MAC window. Click OK button on the MAC window to exit.

Step3: Simulate

1. Click "R" icon on the tool bar
2. Select Simulation in the menu bar and click/ select RUN in the drop down list to execute the simulation.
3. To start playback select "►" icon located at the bottom right corner of the editor.
4. MOBILE NODE's start moving across the paths already drawn.

Caution: file8 is the hypothetical name given to this simulation,

RESULTS & ANALYSIS:

WITH BANDWIDTH 10 MBPS AND BER 0.000000000

Time	No. of Packets	
	Node_N1_Output_Throughput	Mobile_Node_Input_Throughput
1		
2		
3		
4		
5		

PART-B

Program 1:-

Write a program for a HLDC frame to perform the following:

(i) Bit stuffing

Aim:- Write a program to perform bit stuffing in C language and execute the same and display the result.

Theory: The new technique allows data frames to contain an arbitrary number if bits and allows character codes with an arbitrary no of bits per character. Each frame begins and ends with special bit pattern, 01111110, called a flag byte. Whenever the sender's data link layer encounters five consecutive ones in the data, it automatically stuffs a 0 bit into the outgoing bit stream. This bit stuffing is analogous to character stuffing, in which a DLE is stuffed into the outgoing character stream before DLE in the data

Algorithm:-

Step 1: Read frame length n
Step 2: Repeat step (3 to 4) until i<n (Read values in to the input frame (0'sand 1's) i.e.
Step 3: Initialize i =0;
Step 4: read a[i] and increment i.
Step 5: Initialize i=0, j=0, count =0.
Step 6: repeat step (7 to 22) until i<n.
Step 7: If a[i] == 1 then.
Step 8: b[j] = a[i]
Step 9: Repeat step (10 to 18) until (a[k] =1 and k<n and count <5).
Step 10: Initialize k=i+1;
Step 11: Increment j and b[j] = a[k];
Step 12: Increment count.
Step 13: if count =5 then.
Step 14: increment j.
Step 15: b[j] =0.
Step 16: end if .
Step 17: i=k.
Step 18: Increment k.
Step 19: else
Step 20: b[j] = a[i]
Step 21: end if
Step 22: Increment I and j.
Step 23: Print the frame after bit stuffing.
Step 24: Repeat step (25 to 26) until i< j.
Step 25: Print b[i].
Step 26: Increment i.
End.

Program: - // BIT Stuffing program.

```
#include<stdio.h>
#include<string.h>
```

```
void main()
{
  int a[20],b[30],i,j,k,count,n;
printf("Enter Frame length :");
//gets(n);
scanf("%d",&n);
printf("Enter input frame (0's &Â 1's only): ");
for(i=0;i<n;i++)
scanf("%d",&a[i]);
     i=0;
count=1;
j=0;
while(i<n)
   {          //count=0;
      if(a[i]==1)
       {
          b[j]=a[i];
          for(k=i+1;a[k]==1 && k<n && count<5;k++)
        {
j++;
        b[j]=a[k];
              count++;
if(count==5)
         {
        j++;
              b[j]=0;
         }
              i=k;
            }
       }
    else
         {
      b[j]=a[i];
         }
     i++;
    j++;
count=1;
}
printf("After stuffing the frame is:");
printf("01111110");
for(i=0;i<j;i++)
printf("%d",b[i]);
printf("01111110");
}
```

Result: - The output is displayed on the screen after stuffing bits into the input.

(ii) Character stuffing.

Aim: Write a program to perform character stuffing in C language and execute the same and display the result.

Theory: The framing method gets around the problem of resynchronization after an error by having each frame start with the ASCII character sequence DLE STX and the sequence DLE ETX. If the destination ever losses the track of the frame boundaries all it has to do is look for DLE STX or DLE ETX characters to figure it out. The data link layer on the receiving end removes the DLE before the data are given to the network layer. This technique is called character stuffing.

Algorithm:

Begin
Step 1: Initialize i and j as 0.
Step 2: Declare n and pos as integer and a[20], b[50], ch as character.
Step 3: Read the string a.
Step 4: Find the length of the string n, i.e., n-strlen(a).
Step 5: Read the position, pos.
Step 6: if pos > n then.
Step 7: Print invalid position and read again the position, pos.
Step 8: End if.
Step 9: Read the character, ch.
Step 10: Initialize the array b, b[0…5] as 'd', 'l', 'e', 's' ,'t' ,'x' respectively.
Step 11: j=6;
Step 12: Repeat step[(13to22) until i<n.
Step 13: if i==pos-1 then
Step 14: initialize b array, b[j],b[j+1]…b[j+6] as 'd', 'l', 'e' ,'ch, 'd', 'l', 'e' respectively.
Step 15: Increment j by 7, i.e., j=j+7.
Step 16: end if
Step 17: if a[i]=='d' and a[i+1]=='l' and a[i+2]=='e' then
Step 18: Initialize array b, b[13…15]='d', 'l', 'e' respectively.
Step 19: Increment j by 3, i.e., j=j+3.
Step 20: end if.
Step 21: b[j]=a[i]
Step 22: Increment I and j.
Step 23: Initialize b array, b[j],b[j+1]…b[j+6] as 'd', 'l', 'e' ,'e', 't', 'x','\0'respectively.
Step 24: Print frame after stuffing.
Step 25: Print b.
End.

Program: **//PROGRAM FOR CHARACTER STUFFING.**

```
#include<stdio.h>
#include<string.h>
void main()
{
int i=0,j=0,n,pos;
char a[20],b[50],ch;
printf("enter string\n");
scanf("%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("\nframe after stuffing:\n");
printf("%s",b);
}
```

Result: The output is displayed on the screen after stuffing characters into the input.

Program 2

Aim: Write a program for Distance Vector Algorithm to find suitable path for transmission.

Theory:

Routing algorithm is a part of network layer software which is responsible for deciding which output line an incoming packet should be transmitted on. If the subnet uses datagram internally, this decision must be made anew for every arriving data packet since the best route may have changed since last time. If the subnet uses virtual circuits internally, routing decisions are made only when a new established route is being set up. The latter case is sometimes called session routing, because a rout remains in force for an entire user session (e.g., login session at a terminal or a file).

Routing algorithms can be grouped into two major classes: adaptive and non-adaptive. non-adaptive algorithms do not base their routing decisions on measurement or estimates of current traffic and topology. Instead, the choice of route to use to get from I to J (for all I and J) is compute in advance, offline, and downloaded to the routers when the network ids booted. This procedure is sometime called static routing.

Adaptive algorithms, in contrast, change their routing decisions to reflect changes in the topology, and usually the traffic as well. Adaptive algorithms differ in where they get information (e.g., locally, from adjacent routers, or from all routers), when they change the routes (e.g., every ΔT sec, when the load changes, or when the topology changes), and what metric is used for optimization (e.g., distance, number of hops, or estimated transit time).

Two algorithms in particular, distance vector routing and link state routing are the most popular. 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 neighbors.

In distance vector routing, each router maintains a routing table indexed by, and containing one entry for, each router in subnet. This entry contains two parts: the preferred outgoing line to use for that destination, and an estimate of the time or distance to that destination. The metric used might be number of hops, time delay in milliseconds, total number of packets queued along the path, or something similar.

The router is assumed to know the “distance” to each of its neighbor. If the metric is hops, the distance is just one hop. If the metric is queue length, the router simply examines each queue. If the metric is delay, the router can measure it directly with special ECHO packets hat the receiver just time stamps and sends back as fast as possible.

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");
}
```

Result: -

```
 ./a.out  dist.c

Enter the number of nodes 3
Enter the cost matrix
0 3 5
3 0 2
5 2 0
```

Program 3

Aim: Implement Dijkstra's algorithm to compute the shortest path.

Theory:

In order to transfer packets from a source host to the destination host, the network layer must determine the path or route that the packets are to follow. This is the job of the network layer routing protocol. As the heart of any routing protocol is the routing algorithm that determines the path for a packet from source router to destination router. Given a set of router, with links connecting the routers, a routing algorithm finds a good path from source router to destination router.

Dijkstra's method of computing the shortest path is a static routing algorithm. It involves building a graph of the subnet, with each node of the graph representing a router and each arc representing a communication line or a link. To find a route between a pair of routers, the algorithm just finds the shortest path between them on the graph.

Dijkstra's algorithm finds the solution for the shortest path problems only when all the edge-weights are non-negative on a weighted, directed graph. In Dijkstra's algorithm the metric used for calculation is distance. Each node is labeled with its distance from the source node along the best known path. Initially, no paths are known, so all nodes are labeled with infinity. As the algorithm proceeds and path are found, the labels may change, reflecting better paths. A label may either be tentative or permanent. Initially all nodes are tentative and once it is discovered that the shortest possible path to a node is got it is made permanent and never be changed.

Dijkstra's Algorithm

1. Enter cost matrix C[ ][ ]. C[i][j] is the cost of going from vertex i to vertex j. If there is no edge between vertices i and j then C[i][j] is infinity.
2. Array visited[ ] is initialized to zero.

```
for(i=0;i<n;i++)
        visited[i]=0;
```

3. If the vertex 0 is the source vertex then visited [0] is marked as 1.
4. Create the distance matrix, by storing the cost of vertices from vertex 0 to n-1 from the source vertex 0.

```
for(i=1;i<n;i++)
        distance[i]=cost[0][i];
```

Initially, distance of source vertex is taken as 0. i.e. distance[0]=0;

5. for(i=1;i<n;i++)

– Choose a vertex w, such that distance[w] is minimum and visited[w] is 0. Mark visited[w] as 1.

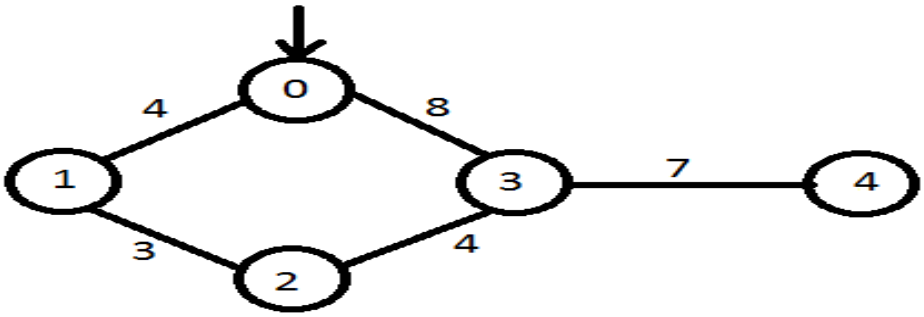
– Recalculate the shortest distance of remaining vertices from the source.

– Only, the vertices not marked as 1 in array visited[ ] should be considered for recalculation of distance. i.e. for each vertex

6. Stop the algorithm if, when all the nodes has been marked visited.

Below is an example which further illustrates the Dijkstra's algorithm mentioned.

Consider a weighted graph as shown:



Here 0, 1, 2, 3 and 4 which are inside the circle are nodes of the graph, and the number between them are the distances of the graph. Now using Dijkstra's algorithm we can find the shortest path between initial node and the remaining vertices. For this, the cost matrix of the graph above is,

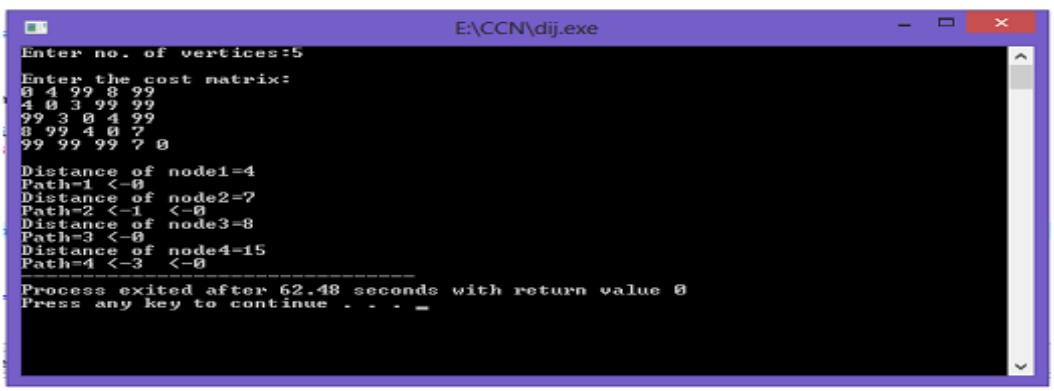
n	0	1	2	3	4
0	0	4	INFINITY	8	INFINITY
1	4	0	3	INFINITY	INFINITY
2	INFINITY	3	0	4	INFINITY
3	8	INFINITY	4	0	7
4	INFINITY	INFINITY	INFINITY	7	0

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);
        }
}
```

Output:



Program 4

Aim: For the given data, use CRC-CCITT polynomial to obtain CRC code. Verify the program for the cases.
i) Without error. ii) With error.

Theory:
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.
Example: 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 WITH ZERO BITS
1011                    DIVISOR (4 BITS)
--------------------
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. 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 post processing).

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.

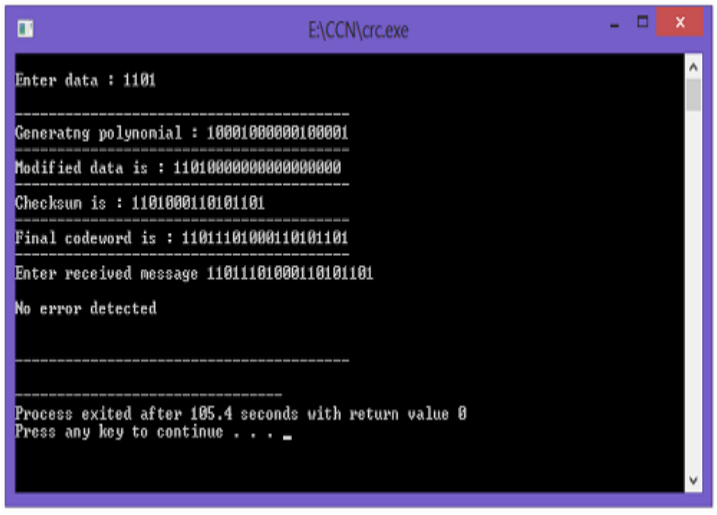
```
11010011101100 100 <--- input with check value
1011               <--- divisor
01100011101100 100 <--- result
 1011              <--- divisor ...
00111011101100 100


......


00000000001110 100
          1011
00000000000101 100
           101 1
------------------
                0 <--- remainder
```

Program:

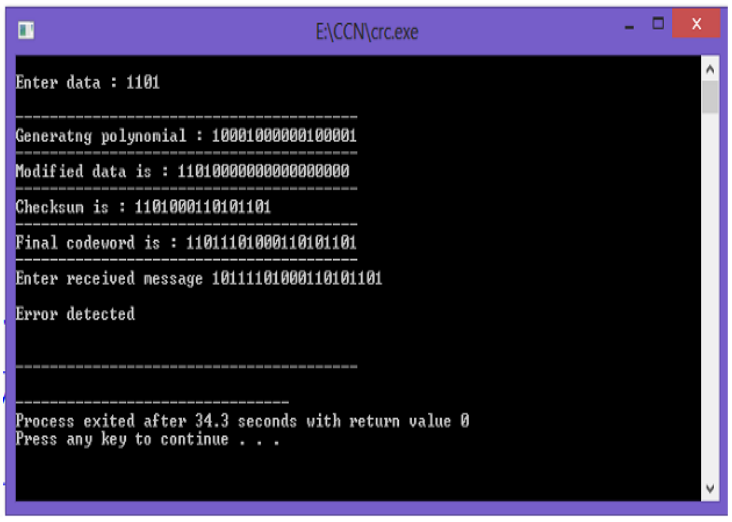
```
//crc can detect all single bit error, double bit, odd bits of error and burst error
#include<stdio.h>
#include<string.h>
#define N strlen(g)

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 without error



Output with error



Program 5

Aim: Implement Stop and Wait Protocol and Sliding Window Protocol in a C program and execute the same and display the result.

(i) Stop and Wait Protocol

Theory:

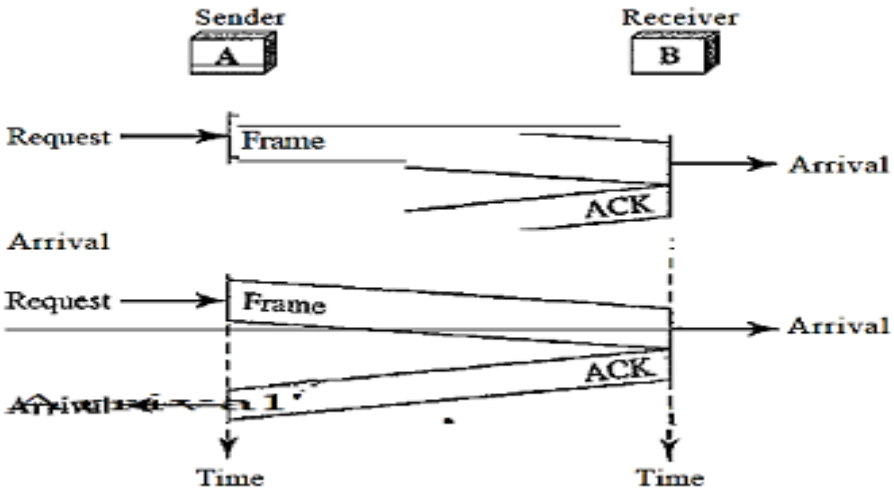
If data frames arrive at the receiver site faster than they can be processed, the frames must be stored until their use. Normally, the receiver does not have enough storage space, especially if it is receiving data from many sources.

This may result in either the discarding of frames or denial of service. To prevent the receiver from becoming overwhelmed with frames, we somehow need to tell the sender to slow down. There must be feedback from the receiver to the sender.

The protocol we discuss now is called the Stop-and-Wait Protocol because the sender sends one frame, stops until it receives confirmation from the receiver (okay to go ahead), and then sends the next frame. We still have unidirectional communication for data frames, but auxiliary ACK frames (simple tokens of acknowledgment) travel from the other direction. We add flow control to our previous protocol.

Example:

Figure shows an example of communication using this protocol. It is still very simple. The sender sends one frame and waits for feedback from the receiver.



When the ACK arrives, the sender sends the next frame. Note that sending two frames in the protocol involves the sender in four events and the receiver in two events.

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)
```

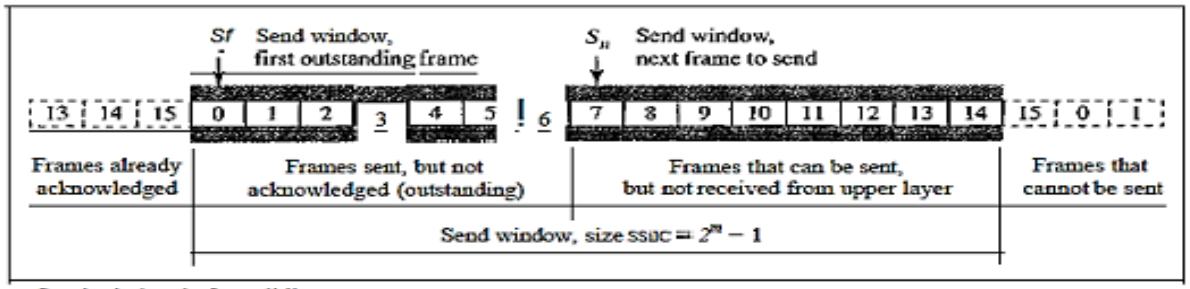
```
{
printf("\nSENDER: ACK for Frame %d Received.\n",i-1);
}
printf("\nSENDER: Frame %d sent, Waiting for ACK...\n",i);
ACK=NO;
wait_time= rand() % 4+1;
if (wait_time==TIMEOUT)
{
printf("SENDER: ACK not received for Frame %d=>TIMEOUT Resending Frame...",i);
}
else
{
sleep(RTT);
printf("\nRECEIVER: Frame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

(ii) Sliding Window Protocol :

Theory:

Sliding Window:

In this protocol (and the next), the sliding window is an abstract concept that defines the range of sequence numbers that is the concern of the sender and receiver. In other words, the sender and receiver need to deal with only part of the possible sequence numbers. The range which is the concern of the sender is called the send sliding window; the range that is the concern of the receiver is called the receive sliding window. We discuss both here. The send window is an imaginary box covering the sequence numbers of the data frames which can be in transit. In each window position, some of these sequence numbers define the frames that have been sent; others define those that can be sent. The maximum size of the window is 2m-1 for reasons that we discuss later. In this chapter, we let the size be fixed and set to the maximum value, but we will see in future chapters that some protocols may have a variable window size.



a. Send window before sliding

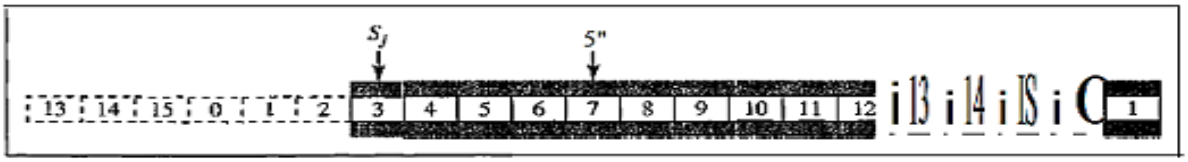
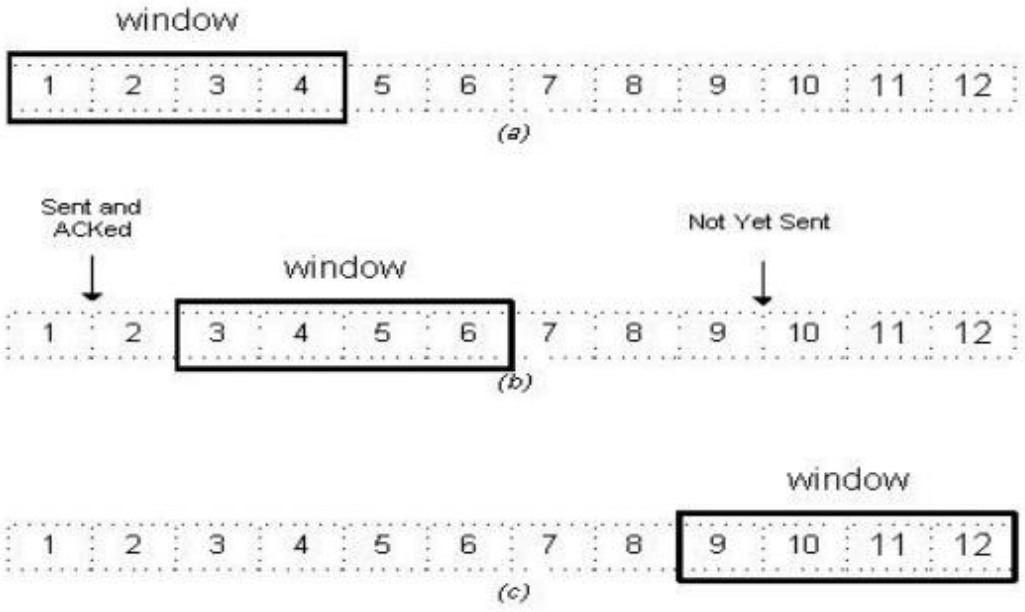


Figure shows a sliding window of size 15 (m =4). The window at any time divides the possible sequence numbers into four regions.

The first region, from the far left to the left wall of the window, defines the sequence numbers belonging to frames that are already acknowledged. The sender does not worry about these frames and keeps no copies of them. The second region, colored in Figure defines the range of sequence numbers belonging to the frames that are sent and have an unknown status. The sender needs to wait to find out if these frames have been received or were lost. We call these outstanding frames.

The third range, white in the figure, defines the range of sequence numbers for frames that can be sent; however, the corresponding data packets have not yet been received from the network layer. Finally, the fourth region defines sequence numbers that cannot be used until the window slides.



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 e.g.:- if ack of frame 3 is received it understands that frame 1 and frame 2 are received properly.

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;
  }
```

Result:

Both the protocols are implemented and executed and the result is displayed on the screen.

Program 6

Aim: Write a program for congestion control using leaky bucket algorithm in C language and execute the same and display the result.

Theory:

Policing

1. Network monitors traffic flows continuously to ensure they meet their traffic contract.
2. The process of monitoring and enforcing the traffic flow is called policing.
3. When a packet violates the contract, network can discard or tag the packet giving it lower priority
4. If congestion occurs, tagged packets are discarded first
5. Leaky Bucket Algorithm is the most commonly used policing mechanism

(i) Bucket has specified leak rate for average contracted rate
(ii) Bucket has specified depth to accommodate variations in arrival rate
(iii) Arriving packet is conforming if it does not result in overflow

Leaky Bucket algorithm can be used to police arrival rate of a packet stream

Leaky Bucket Algorithm

1. The above figure shows the leaky bucket algorithm that can be used to police the traffic flow.
2. At the arrival of the first packet, the content of the bucket is set to zero and the last conforming time (LCT) is set to the arrival time of the first packet.
3. The depth of the bucket is L+I, where l depends on the traffic burstiness.
4. At the arrival of the kth packet, the auxiliary variable X' records the difference between the bucket content at the arrival of the last conforming packet and the inter-arrival time between the last conforming packet and the k^{th} packet.
5. If the auxiliary variable is greater than L, the packet is considered as nonconforming, otherwise the packet is conforming. The bucket content and the arrival time of the packet are then updated.

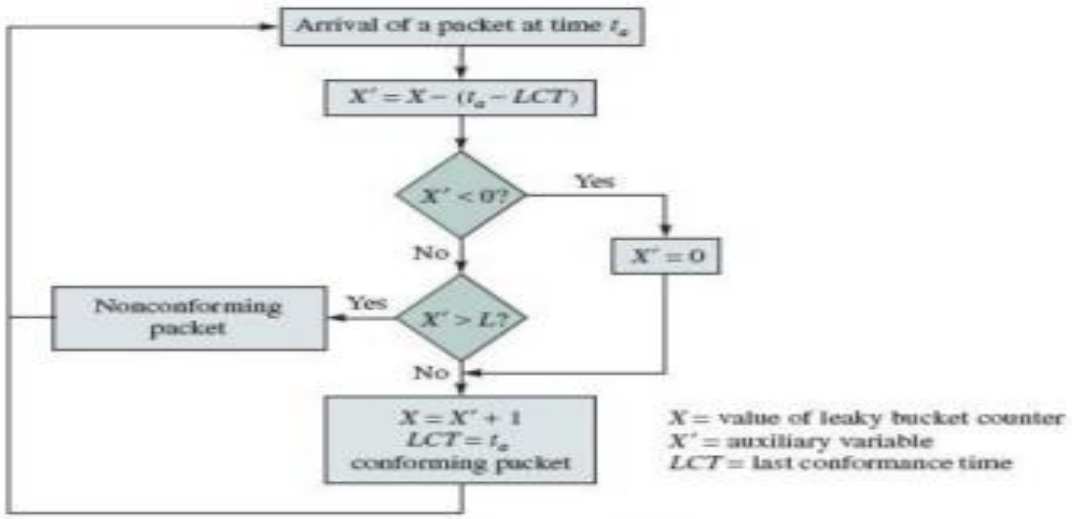


FIGURE 7.54 Leaky bucket algorithm used for policing

Leaky Bucket Example: - The operation of the leaky bucket algorithm is illustrated in the below figure.

1. Here the value I is four packet times, and the value of L is 6 packet times.
2. The arrival of the first packet increases the bucket content by four (packet times).
3. At the second arrival the content has decreased to three, but four more are added to the

bucket resulting in total of seven.
4. The fifth packet is declared as nonconforming since it would increase the content to 11, which would exceed L+I (10).
5. Packets 7, 8, 9 and 10 arrive back to back after the bucket becomes empty. Packets 7, 8 and 9 are conforming, and the last one is nonconforming.
6. Non-conforming packets not allowed into bucket & hence not included in calculations.

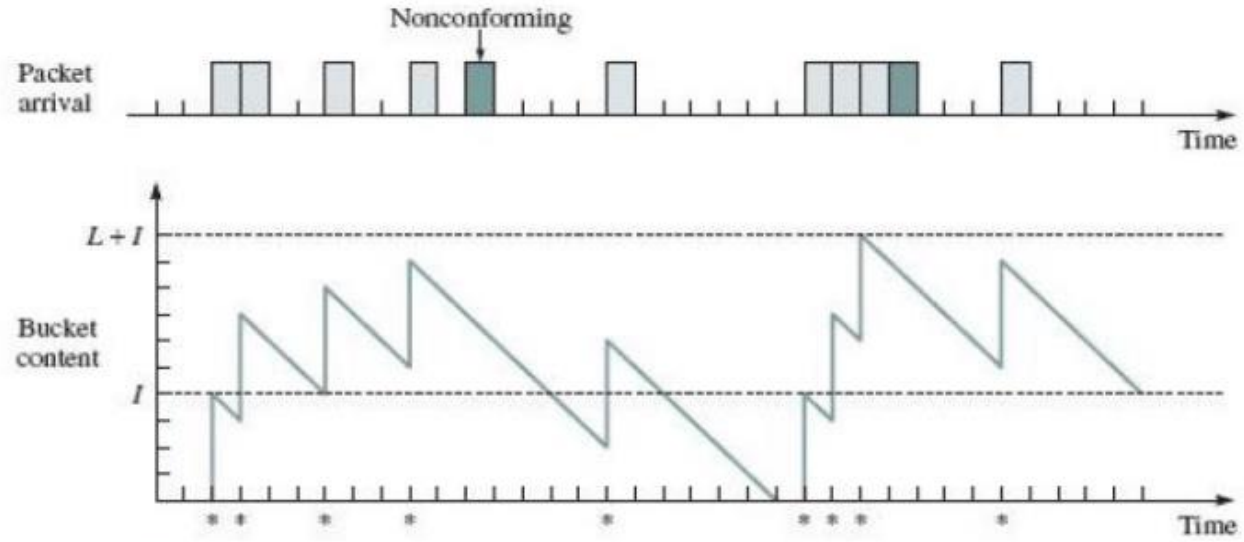


FIGURE 7.55 Behavior of leaky bucket

Program: //leaky bucket 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=newpack;
```

```
    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;
}
```

Result: Thus the program is written and executed in C language for congestion control using leaky bucket algorithm.