ACS COLLEGE OF ENGINEERING

BENGALURU

Department of

ELECTRONICS AND COMMUNICATION ENGINEERING

COMPUTER NETWORKS LAB

15ECL68

VI SEMESTER [CBCS SCHEME]

2017 - 18

COMPILED, TESTED AND EDITED BY

Mr. Rahul Rai
Asst. Professor
ECE Department

SYLLABUS

Computer Networks Laboratory

B.E., VI Semester, EC

[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 Exam Hours: 03 CREDITS – 02

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
    I. Without error
    II. 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:

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

DOs and DON'Ts

• Conduct yourself in a responsible manner always in the laboratory. Don't talk aloud or crack jokes in lab.

• A lab coat should be worn during laboratory experiments. Dress properly during a laboratory activity.

• Observe good housekeeping practices. Replace the materials in proper place after work to keep the lab area tidy.

• Do not wander around the room, distract other students, startle other students or interfere with the laboratory experiments of others.

• Do not open any irrelevant internet sites on lab computer

• Do not use a flash drive on lab computers.

• Do not upload, delete or alter any software on the lab PC.

• Don't leave the lab without making proper shut down of the computer.

• Students should carry observation notes and record completed in all aspects.

• The Program and its theoretical result should be there in the observation before coming to the next lab.

• The Practical Result should be noted down into their observations and result must be shown to the Lecturer In-Charge for verification.

• Wiring of circuits and connecting of cables/hardware must be done with the power OFF. If you encounter a problem the first thing to do is power off all the hardware, logoff the station and reboot the PC. Sometimes the PC/hardware/software can go into unusual states requiring a cold restart.

• Ensure that all cabling and wiring is properly connected. As obvious as it may seem this does happen. Sometimes a problem can be as simple as a cable not being plugged in.

Guidelines For installation of NCTUns Network Simulator

Follow the following steps carefully, please do not skip any steps that have been mentioned below:

1. Install any Linux with kernel 2.6.9 (PCQ Linux 2004 is exception) Recommended → RED HAT LINUX ENTERPRISE EDITION

2. After installation Boot into Linux as root.

3. Copy the .tgz installation file of NCTUns that you got from college to the folder /bin/local Please don't change any folder name in this folder that is created after unzipping the above file. Don't even change the Case of the folder that is created

4. Now unzip the .tgz file by opening the terminal and changing the directory to /bin/local by the command :- [root@localhost ~] cd /bin/local

To unzip use the following command:- [root@localhost local] tar xvzf [the file name].tar

(Note there is no '-' before xvzf) This will create a folder called NCTUns in the directory /bin/local...

5. Now disable the Secure Linux option by running the following command :- [root@localhost local] vi /etc/selinux/config

When the file opens, there is a line similar to -- SELINUX=enforcing Change the "enforcing" to "disabled" (Note without quotes)

6. From the directory /bin/local change the current working directory to NCTUns by following command :- [root@localhost local] cd NCTUns

7. Now from here execute the installation shell script that will do the required compilations and settings for you:- [root@localhost local] ./install.sh

During this part it will ask for installation of tunnel files. Please type yes and Enter to continue

8. If the installation is successful, it will display the success message at the end. Now restart your computer. You will find a new entry in GRUB Menu "NCTUns kernel login". Boot into Linux using this entry.

9. Log in as root.

Now you have to modify any .bash_profile file

[root@localhost ~] vi .bash_profile

The Content of file should look like as follows.

```
# .bash_profile

# Get the aliases and function

if [ -t ~/.bashrc ]; then
 . ~/.bashrc fi

# User specified environment variables and startup programs

PATH=$PATH:$HOME/bin:/usr/local/nctuns/bin export
LD_LIBRARY_PATH=/usr/local/nctuns/lib export NCTUNSHOME=/usr/local/nctuns

export PATH
export USERNAME
```

If it's not like above, change it to look like above.

10. Now save this file and log off and then log on again.

11. Create another user account.

12. before using simulator, please execute the following command

[root@localhost ~] iptables -F

13. Run the simulator using three commands where each command should be executed in different window.

[root@localhost ~] dispatcher

[root@localhost ~] coordinator

[root@localhost ~] nctunsclient

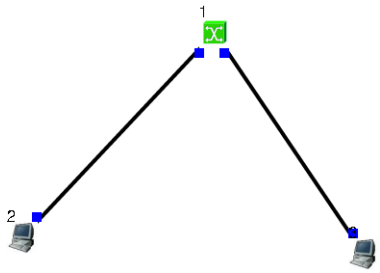
14. In the NCTUns window Settings → Dispatcher. Provide the username and password of the user account you created in step 11. Then Click OK.

PART– A

Experiment No: 1

Aim: 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.

Topology:



Sender:- stcp –p 2000 –l 1024 1.0.1.2

Receiver:- rtcp –p 2000 –l 1024

Parameters:- Drop Packets and Collision Packets.

Step1: Drawing topology

1. Select/click the HOST icon on the toolbar 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 toolbar and click the left mouse button on the editor, to place HUB1 on the editor.

3. Click on the LINK icon on the toolbar and connect HOST1 to HUB1 and HUB1 to HOST2

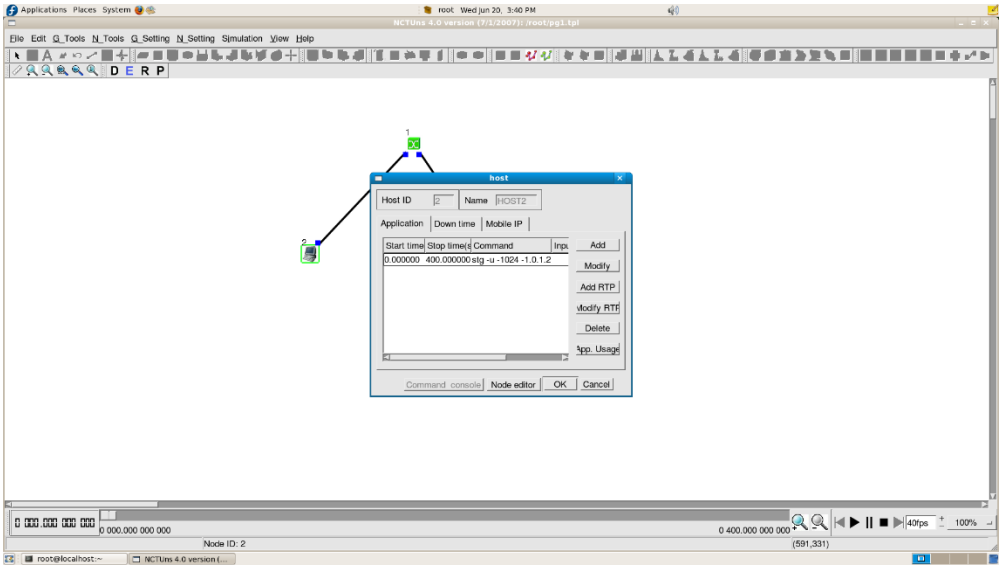
4. Click on the "E" icon on the toolbar to save the current topology e.g: file1.tpl (Look for the ******.tpl extension.)

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 textbox. stg –u 1024 100 1.0.1.2



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 textbox. rtg –u –w log1

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 checkboxes 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.

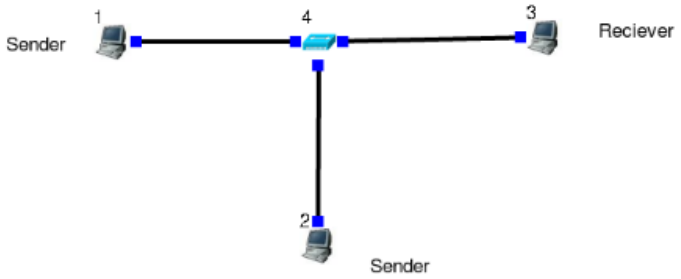
2. Click NODE EDITOR Button on the HOST window and select the FIFO tab from the modal window that pops up.

3. Change Queue size (Default 50).

Experiment No: 2

Aim: 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.

Topology:



Sender:- stcp –p 3000 –l 1024 1.0.1.3 stg –u 1024 1.0.1.3

Receiver:- rtcp –p 3000 –l 1024 rtg –u 3000

Parameters:- Throughput of incoming and outgoing Packets

Step1: Drawing topology

1. Select/click the HOST icon on the toolbar 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 toolbar 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 toolbar and connect HOST1 to HUB, HOST2 to HUB and HUB to HOST3

4. Click on the "E" icon on the toolbar 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 textbox. stcp –p 21 –l 1024 1.0.1.3

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 checkbox 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 textbox. stg –u 1024 100 1.0.1.3

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 LOG STATISTICS and select checkbox 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 textbox. rtcp –p 21 –l 1024

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

16. Also add the following command on HOST3 rtg –u –w log1

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 checkbox for input and output throughput in the MAC window

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

Step3: Simulate

i. Click “R” icon on the tool bar

ii. Select Simulation in the menu bar and click/ select RUN in the dropdown list to execute the simulation.

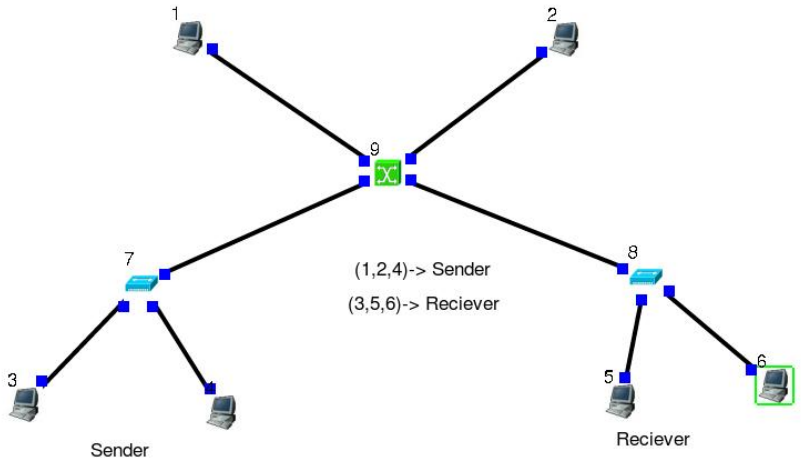
iii. To start playback select “►” icon located at the bottom right corner of the editor.

iv. To view results, Open new TERMINAL window, move to file2.results folder and open input and output throughput log files in separate TERMINAL window.

Experiment No: 3

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

Topology:



Sender:- stcp –p 2000 –l 1024 1.0.1.4

Receiver:- rtcp –p 2000 –l 1024

Double click on receiver link and change BER to 0.000001, Run Again.

Parameters:- Throughput of outgoing Packets

Step1: Drawing topology

1. Select/click the HOST icon on the toolbar and click the left mouse button on the editor, to place HOST1 on the editor. 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 toolbar 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 toolbar and connect HOST1, HOST2 and HOST3 to HUB1, HOST4, HOST5 and HOST6 to HUB2.

4. Select/click the SWITCH icon on the toolbar and click the left mouse button on the editor, to place SWITCH1 on the editor.

5. Click on the LINK icon on the toolbar and connect HUB1 to SWITCH1 and HUB2 to SWITCH1.

6. Click on the "E" icon on the toolbar 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 textbox. stcp –p 21 –l 1024 1.0.1.4

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, but use different commands stcp –p 21 –l 1024 1.0.1.5 at HOST2 stcp –p 21 –l 1024 1.0.1.6 at 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 21 –l 1024

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 checkbox 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, but use different commands rtcp –p 21 –l 1024 at HOST 5 rtcp –p 21 –l 1024 at 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 checkbox 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

i. Click "R" icon on the tool bar

ii. Select Simulation in the menu bar and click/ select RUN in the dropdown list to execute the simulation.

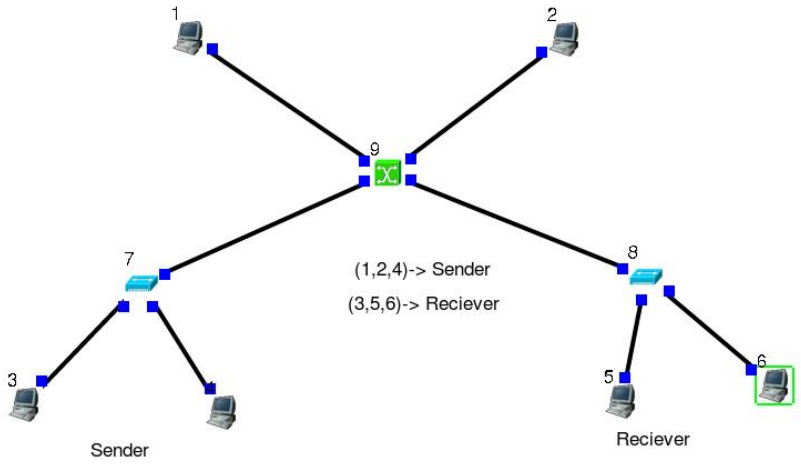
iii. To start playback select "►" icon located at the bottom right

iv. To view results, open new TERMINAL window, move to file5.results folder and open output throughput log files in separate TERMINAL window.

Experiment No: 4

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

Topology:



Sender:- stcp –p 2000 –l 1024 1.0.1.4

Receiver:- rtcp –p 2000 –l 1024

Parameters:- Receiver side Collision Packets and Drop Packets

Step1: Drawing topology

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

Repeat the above procedure and place 3 other hosts “HOST2”, “HOST3”, “HOST4”, “HOST5”, and “HOST6”on the editor.

2. Select/click the HUB icon on the toolbar 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 toolbar and connect HOST1, HOST2 and HOST3 to HUB1, HOST4, HOST5 and HOST6 to HUB2.

4. Select/click the SWITCH icon on the toolbar and click the left mouse button on the editor, to place SWITCH1 the editor.

5. Click on the LINK icon on the toolbar and connect HUB1 to SWITCH1 and HUB2 to SWITCH1.

6. Click on the "E" icon on the toolbar to save the current topology e.g: file7.tpl (Look for the ******.tpl extension.) NOTE: Changes cannot / (should not) be done after selecting the "E" icon.

Step 2: 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 textbox. stcp –p 21 –l 1024 1.0.1.4

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, but use different commands stcp –p 23 –l 1024 1.0.1.5 at HOST2 stcp –p 25 –l 1024 1.0.1.6 at 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 21 –l 1024

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 checkbox for Number of drop and collisions packets 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, but use different commands rtcp –p 23 –l 1024 at HOST5 rtcp –p 25 –l 1024 at 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 MAC tab from the modal window that pops up.

14. Select LOG STATISTICS and select checkbox for Number of drop and collisions packets in the MAC window

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

16. Also select the drop and collisions at HOST6.

Step3: Simulate

i. Click “R” icon on the tool bar

ii. Select Simulation in the menu bar and click/ select RUN in the dropdown list to execute the simulation.

iii. To start playback select “►” icon located at the bottom right corner of the editor.

iv. To plot congestion window select Tools in the menu bar and select PLOT GRAPH in the drop down list.

v. In the Graph window, select File->OPEN, move to file7.results folder and the drop and collision log file.

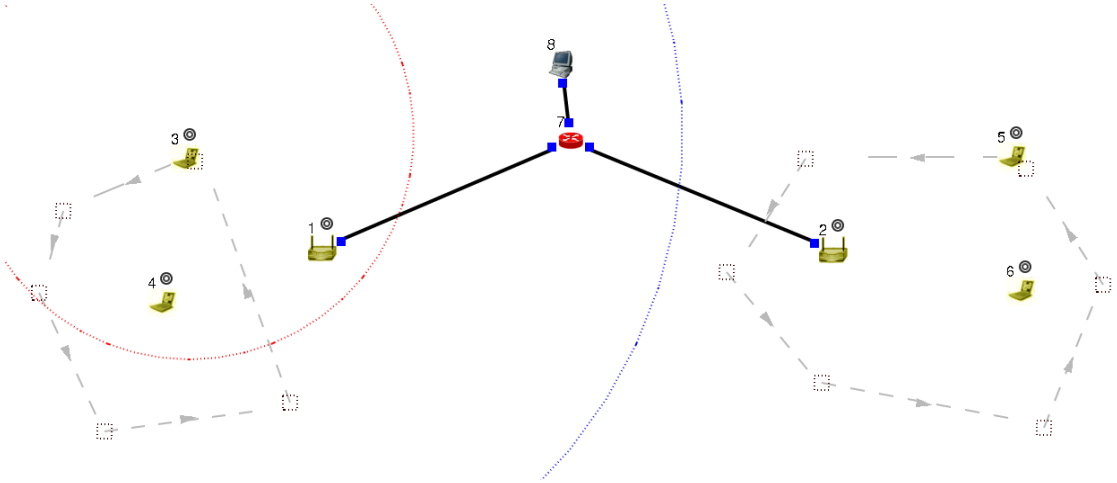
vi. To open another Graph window, Select File->New tab on the drop down list to open up to a maximum of 6 windows

vii. To view results, Open up new TERMINAL window, move to file7.results folder and open input and output throughput log files in separate TERMINAL window.

Experiment No: 5

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

Topology:



Click on “access point”. Goto wireless interface and tick on “show transmission range and then click OK.

Double click on Router -> Node Editor and then Left stack -> throughput of Incoming packets Right stack -> throughput of Outgoing packets

Select mobile hosts and access points then click on. Tools -> WLAN mobile nodes-> WLAN Generate infrastructure. Subnet ID: Port number of router (2) Gateway ID: IP address of router

Mobile Host 1 ttcp –t –u –s –p 3000 1.0.1.1

Mobile Host 1 ttcp –t –u –s –p 3001 1.0.1.1

(Receiver) ttcp –r –u –s –p 3000

ttcp –r –u –s –p 3001

Run and then play to plot the graph.

Step1: Drawing topology

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

2. Select/click the ROUTER icon on the toolbar 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 toolbar and click the left mouse button on the editor, to place ACCESS POINT 1 on the editor. Repeat this procedure and place ACCESS POINT 2 on the editor.

4. Select/click the MOBILE NODE (infrastructure mode) icon on the toolbar and click the left mouse button on the editor, to place MOBILE NODE 1 on the editor. Repeat this procedure and place MOBILE NODE 2, MOBILE NODE3 and MOBILE NODE 4 on the editor.

5. Click on the LINK icon on the toolbar and connect ACCESS POINT1 to ROUTER1 and ACCESS POINT2 to ROUTER1

6. Click on the "Create a moving path" icon on the toolbar and draw moving path across MOBILE NODE 1 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).

7. To create Subnet: - Select wireless subnet icon in the toolbar 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 toolbar 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.

Step 2: 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 textbox.

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 checkbox 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 textbox. ttcp –t –u –s –p 8001 1.0.2.2 (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 checkbox 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 NODE 2:- ttcp –t –u –s –p 8002 1.0.2.2

MOBILE NODE3:- 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 ROUTER1 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 checkbox 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 checkbox 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 checkbox for Output throughput in the MAC window. Click OK button on the MAC window to exit.

Step3: Simulate

i. Click “R” icon on the tool bar

ii. Select Simulation in the menu bar and click/select RUN in the dropdown list to execute the simulation.

iii. To start playback select “►” icon located at the bottom right corner of the editor.

iv. MOBILE NODE’s start moving across the paths already drawn.

PART– B

Experiment No: 1

Stuffing & Destuffing of Bits and Characters

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

i) Bit stuffing ii) Character stuffing.

Theory: In digital communication, bit patterns are reserved for control signals. However, if these bit patterns appear in the data, they should not be mistaken for control signals. Hence, stuffing is done in the data link layer. **Stuffing** is the process of adding redundant information or increasing the amount of data transmitted to accommodate the control signals.

The task of the data link layer is to convert the raw bit stream offered by the physical layer into a stream of frames for use by the network layer, and similarly to convert the packets given by the network layer into frames for transmission. It is the function of the data link layer to detect and correct errors in transmission.

The predominant 2 methods used for stuffing are – BYTE STUFFING and BIT STUFFING.

The problem of resynchronization after an error is removed, by having each frame start and end with special bytes called FRAME DELIMITERS.

In **byte stuffing**, we assume an ASCII set, with the frame delimiters being STX (ASCII 03, start text), and ETX (ASCII 02, end text). In case ETX or STX happens to be present in the data, it should be treated as data, and not end of text, or start of text. To ensure this, we use another byte DLE (ASCII 16, escape byte). The sender's data link layer inserts this DLE before each STX or ETX that occurs in the data, and the receiver's data link layer "de-stuffs" the escape byte DLE from the data. An escape byte in the data is stuffed with another escape byte. Thus, a framing flag byte can be distinguished from one in the data by the absence or presence of an escape byte before it.

The **disadvantage** of byte stuffing is that it is associated with the use of 8 bit characters, while all character codes don't use 8-bit characters – eg : UNICODE uses 16-bit characters.

Bit stuffing is used when a bit oriented communication system is used. Each frame begins and ends with a special bit pattern "01111110". Whenever the sender's data link layer encounters 5 consecutive '1's in the data, it automatically stuffs a '0' bit into the outgoing bit stream. Thus, the flag bit pattern is never detected in the data. The receiver's data link layer automatically de-stuffs the '0' bit appearing after 5 consecutive '1's.

HDLC Frame (High Level Data Link Control)

The high level data link control protocol is a bit oriented protocol which uses bit stuffing for data transparency. The HDLC frame format is shown below:

8	8	8	>0	16	8
0111 1110	Address	Control	Data	Checksum	0111 1110

The *Address* field is used to identify one of the terminals in a multi-terminal system. The *Control* field is used for sequence numbers, acknowledgements, and other purposes. The *Data* field holds the information. The *Checksum* field is a cyclic redundancy code using the CRC-CCITT polynomial as the generator.

Algorithm for Bit Stuffing

1. Input data sequence.
2. Add start of frame to output sequence.
3. Initialise counter to 0.
4. For every bit in input
    a. Append bit to output sequence.
    b. Is bit = 1?
        i. If YES,
            1. Increment counter.
            2. If counter = 5, append 0 to output sequence, and reset counter.
        ii. ELSE

1. Reset counter.

5. Add stop of frame bits to output sequence.

Algorithm for Character Stuffing

1. Input data sequence.
2. Add start of frame to output sequence (DLE STX).
3. For every character in input
    a. Append character to output sequence.
    b. Is character = DLE?

    If YES,

    Add DLE to output sequence.
4. Add stop of frame to output sequence (DLE ETX).

(a) Simulation of Bit Stuffing & De-stuffing :

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

void main()
{
  /* Array is already initialised to "01111110" which is the flag byte. Hence, a counter 'i' has to point
  to the eighth location in the same array. */
    char ch, array[50]="01111110",recd_array[50];
  int counter=0,i=8,j,k;
  clrscr();
    /* Only the data portion of the frame is inputted */
  printf("\nEnter the original data stream for bit stuffing:\n");
  while((ch=getche())!='\r')
  {
        if(ch=='1')
              ++counter;
        else
              counter=0;
        array[i++]=ch;
        if(counter==5)   /* If 5 ones are encountered, append a zero */
        {
           array[i++]='0';
           counter=0;
        }
  }
  strcat(array,"01111110");       /* Appending the flag byte at the end of the screen */
  /* i now has the value of the flag byte length + data stream length */
  printf("\nThe stuffed data stream is: \n");
  for(j=0;j<i+8;j++)
```

```
            printf( "%c",array[j]);


    /* Destuffing */
    counter=0;
    printf("\nThe destuffed data stream is:\n");
    for (j=8, k=0;j<i+8;j++)
    {
        if(array[j]=='1')
            ++counter;
        else
            counter=0;
        recd_array[k++]=array[j];
        if(counter==6)   /* End if six ones are encountered */
            break;
        else if(counter==5 && array[j+1]=='0')    /*If five ones appear, delete the following zero */
        {
            ++j;
            counter=0;
        }
    }
    for(j=0; j<=k-strlen("01111110"); j++)
        printf("%c", recd_array[j]);        /* Printing the final destuffed array */
    getch();
}
```

Output:

Enter the original data stream for bit stuffing:
011011111111110010

The stuffed data stream is:

011111100110111110111110001001111110

The destuffed data stream is:

011011111111110010

(b) Simulation of Character Stuffing and De-stuffing :

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

#define DLE 16
#define STX 2
#define ETX 3


void main()

{

    char ch;

    char arr[100]={DLE,STX};        /* Initialise the array to have */

    int i=2,j,c;                    /* the frame delimiter bytes initially */

    clrscr();

    /* Inputting the data stream alone */

    printf("\nEnter the data stream (CTRL+B->STX,CTRL+C->ETX,CTRL+P->DLE) :\n");

    while((ch = getch())!='\r')

    {

            if(ch==DLE) /* Checking for DLE */

            {

                arr[i++]=DLE;
                 printf("DLE");
             }

            else if (ch==STX) printf("STX");
            else if (ch==ETX) printf("ETX");
            else printf("%c",ch);
            arr[i++]=ch;

    }

    arr[i++]=DLE;
    arr[i++]=ETX;
    printf("\nThe stuffed stream is\n");
    for(j=0; j<i ;j++)

    {
```

```
        if(arr[j]== DLE) printf("DLE");
        else if(arr[j]==STX) printf("STX");
        else if(arr[j]==ETX) printf("ETX");
        else printf("%c",arr[j]);
    }


    /* Destuffing of character stream */

    printf("\nThe destuffed data is \n");
    for(j=2;j<i-2;j++)
    {

        if(arr[j]==DLE)    /* Checking for DLE */

        {
                printf("DLE");
                j++;
        }
        else if (arr[j]==STX) printf("STX");
        else if(arr[j]==ETX) printf("ETX");
        else printf(" %c",arr[j]);
    }

    getch();

}
```

Output:

Enter the data stream (CTRL+B->STX,CTRL+C->ETX,CTRL+P->DLE) :

A DLE B

The stuffed data stream is

DLE STX A DLE DLE B DLE ETX

The destuffed stream is

A DLE B

Experiment No: 2

Distance Vector Algorithm (Bellman-Ford)

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

Theory: The concept of a distance vector is the rationale for the name distance-vector routing. A least-cost tree is a combination of least-cost paths from the root of the tree to all destinations. These paths are graphically glued together to form the tree. Distance-vector routing unglues these paths and creates a distance vector, a one-dimensional array to represent the tree.

A distance vector does not give the path to the destinations as the least-cost tree does; it gives only the least costs to the destinations. Each node in an internet, when it is booted, creates a very rudimentary distance vector with the minimum information the node can obtain from its neighborhood. The node sends some greeting messages out of its interfaces and discovers the identity of the immediate neighbors and the distance between itself and each neighbor. It then makes a simple distance vector by inserting the discovered distances in the corresponding cells and leaves the value of other cells as infinity.

To improve these vectors, the nodes in the internet need to help each other by exchanging information. After each node has created its vector, it sends a copy of the vector to all its immediate neighbours. After a node receives a distance vector from a neighbour, it updates its distance vector using the Bellman-Ford equation (second case). However, we need to understand that we need to update, not only one least cost, but N of them in which N is the number of the nodes in the internet.

Distance Vector Routing in this program is implemented using Bellman Ford Algorithm

```
#include<stdio.h>
struct node
{
   unsigned dist[20];
   unsigned from[20];
}rt[10];
int main()
{
   int costmat[20][20];
   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];//initialise the distance equal to cost matrix
         rt[i].from[j]=j;
      }
   }
      do
      {
         count=0;
         for(i=0;i<nodes;i++)//We choose arbitary vertex k and we calculate the direct
distance from the node i to k using the cost matrix
         //and add the distance from k to node j
         for(j=0;j<nodes;j++)
         for(k=0;k<nodes;k++)
            if(rt[i].dist[j]>costmat[i][k]+rt[k].dist[j])
            {//We calculate the minimum distance
               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<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");
  // getch();
}
```

Output:-

```
Enter the number of nodes :
3
Enter the cost matrix :
0 2 7
2 0 1
7 1 0
For router 1
node 1 via 1 Distance 0
node 2 via 2 Distance 2
node 3 via 3 Distance 3
For router 2
node 1 via 1 Distance 2
node 2 via 2 Distance 0
node 3 via 3 Distance 1
For router 3
node 1 via 1 Distance 3
node 2 via 2 Distance 1
node 3 via 3 Distance 0
```

Experiment No: 3

Shortest Path Routing using Dijkstra's Algorithm

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

Theory: The main function of the network layer is routing packets from the source machine to the destination machine. The routing algorithm is that part of the network layer software that is responsible for deciding which output line an incoming packet should be transmitted on. These algorithms can be grouped into: NON-ADAPTIVE & ADAPTIVE.

Non-adaptive algorithms don't base their routing decisions on measurements or estimates of the current traffic and topology, but instead compute the route in advance, off-line and download them to the routers when the network is booted. Adaptive algorithms change their routing decisions to reflect changes in topology and traffic.

In the shortest path determination, a graph of the subnet is built with each node of the graph representing a router and each arc a communication link. To choose a route between a given pair of routers, the algorithm just finds the shortest path between them on the graph. The labels on the arcs are computed as a function of distance, bandwidth, average traffic, communication cost, mean queue length, measured delay and other factors.

The algorithm used here is the DIJKSTRA algorithm. Each node is labelled with its distance from the source node along the best-known path. Initially, no paths are known, so all nodes are labelled with infinity. A label may either be tentative or permanent. Initially, all labels are tentative. As the algorithm proceeds, labels on the nodes may change, reflecting better paths. When it is discovered that a label represents the shortest possible path from the source to that node, it is made permanent the and never changed thereafter.

Initially, the source node is made permanent, and all the nodes adjacent to it are relabelled with their distance from it. Next, all the tentatively labelled nodes in the graph are examined and the one with the smallest label is made permanent. This becomes the next working node.

The process continues till the destination node is reached. The shortest path is found by searching backwards from the destination.

Examples of **non-adaptive** (static) algorithms include the **Dijkstra** algorithm.

Examples of **adaptive** (dynamic) algorithms include the **Distance Vector Routing**, and the **Link State Routing** algorithms.

Apart from routing, the network layer performs the tasks of **congestion control** and **billing.**

Algorithm:

1. Input graph data.
2. Make all nodes TENTATIVE.
3. Input source, destination.
4. Make the source node a working node.
5. Make the working node PERMANENT.
6. Check all the tentative nodes, which are connected to working node. Update weight if required.
7. Find the tentative node with the smallest weight. Make this the working node.
8. If working node is the destination node, goto step 9, else goto step 5.
9. Trace back from destination to source.

```
#include<stdio.h>
#include<conio.h>
#include<ctype.h>


#define NUM_OF_NODES 10
#define PERMANENT 1
#define TENTATIVE 0


struct node

{

        unsigned int weight ; /* weight of node => -1 is max value */
        int prev;                       /* value of previous node */
        int state;              /* state of node */
};



void main()

{

 int table[NUM_OF_NODES][NUM_OF_NODES] =

        { /*      A B C D E F G H I J */
          /*A*/   {0,1,0,0,0,4,0,0,0,0},
          /*B*/   {1,0,4,0,0,0,0,1,0,0},
          /*C*/   {0,4,0,3,2,0,0,0,3,0},
          /*D*/   {0,0,3,0,1,0,0,0,0,0},
          /*E*/   {0,0,2,1,0,3,0,0,0,1},
          /*F*/   {4,0,0,0,3,0,1,0,0,0},
          /*G*/   {0,0,0,0,0,1,0,2,0,2},
          /*H*/   {0,1,0,0,0,0,2,0,1,0},
          /*I*/   {0,0,3,0,0,0,0,1,0,2},
          /*J*/   {0,0,0,0,1,0,2,0,2,0}
        /*          A B C D E F G H I J */
        };
        /* Interpret as A is connected to B at a weight of 1 as table[A][B] = table[B][A] = 1 */

 int src,dest,i,working_node;

 /* src is source node, dest is destination node */

 unsigned int min;
 struct node nodes[NUM_OF_NODES];


 /* Initialise all nodes as tentative and having weight of -1 or maximum */

 for(i=0;i<NUM_OF_NODES;i++)

 {
```

```
            nodes[i].weight = -1;
            nodes[i].state = TENTATIVE;
  }

  printf("\nEnter Source: ");
src = toupper(getche()) - 'A';      /* convert src character to uppercase, then subtract 'A' from it to get
                                    src index */
  working_node =src;

  /* Source is the working node initially and has prev = -1 and weight = 0  */

  nodes[src].prev = -1;
  nodes [src].weight = 0;
  printf ("\nEnter Destination: ");
  dest =toupper (getche()) - 'A';


do

  {

          /* make working node permanent */
            nodes[working_node].state = PERMANENT;
            for (i=0;i<NUM_OF_NODES; i++)
            {
                    if (table[working_node][i] !=0 && nodes[i].state == TENTATIVE)
                    {
                            /* If connection exists and node i is tentative */
                            if (nodes[i].weight > table[working_node][i] + nodes[working_node].weight)
                            {
                                    /* If lesser weight is achieved with this node as previous node */
                                nodes[i].weight = table[working_node][i] + nodes[working_node].weight;
                                    nodes[i].prev = working_node;
                                    /* update weight and previous node */
                            }
                     }

          }

            /* Find minimum weighted tentative node */
            min = -1;
            for (i=0;i<NUM_OF_NODES ;i++)
            {
                    if(nodes[i].state == TENTATIVE  && nodes[i].weight <min)
                    {
                            min = nodes[i].weight;
                            working_node =i;
                    }
            }
  } while(working_node != dest);
```

```
 /* Print shortest path by traversing back through node::prev */
 printf("\nShortest Path got --> \n%c",dest+65);
 do
 {
        working_node = nodes[working_node].prev;
        printf ("<-%c ",working_node +65);
  } while (nodes[working_node].prev !=-1);
 printf("\nAt a total weight of: % d",nodes[dest].weight);
 getch();
}
```

Output:

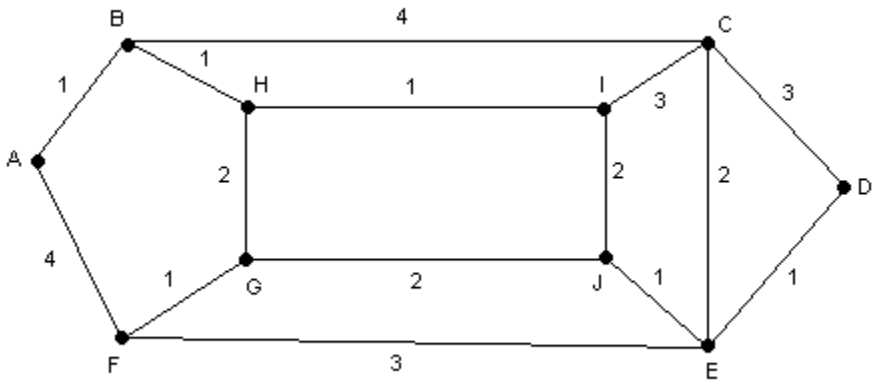
Enter source: A

Enter destination: D

Shortest path got ->

D<-E<-J<-I<-H<-B<-A

At a total weight of: 7



Experiment No: 4

Polynomial Code Checksum

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: Error correcting codes are used widely on the error prone wireless links. However, error detection codes are more efficient for copper wire or fiber, where the error rate is much lower.

The polynomial code (or CRC – Cyclic Redundancy Check), based upon treating bit strings as representations of polynomials with coefficients of 0 and 1 only, is widely used for this purpose. A k-bit frame is regarded as the coefficient list for a polynomial with k terms, ranging from x^{k-1} to x^0. This polynomial has a degree of k-1. Polynomial arithmetic is done using modulo-2 rules.

The sender and the receiver must agree upon a common generator polynomial, G(x), in advance. To compute the **checksum** for some frame with m bits, corresponding to the polynomial M(x), the frame must be longer than the generator polynomial. The checksum is then appended to the end of the frame, so that the transmitted checksummed frame polynomial is divisible by G(x). If there is a remainder in the received checksummed frame, then there has been a transmission error.

Algorithm:

1. Let r be the degree of G(x). Append r zero bits to the low order end of the frame so it now contains m + r bits and corresponds to the polynomial $x^r M(x)$.
2. Divide the bit string corresponding to G(x) into the bit string corresponding to $x^r M(x)$, using modulo 2 division.
3. Subtract the remainder (which is always r or fewer bits) from the bit string corresponding to $x^r M(x)$ using modulo 2 subtraction. The result is the checksummed frame to be transmitted.

Note that a polynomial code with r check bits will detect all burst errors of length <= r.

The CRC-CCITT polynomial is $G(x) = x^{16} + x^{12} + x^5 + 1$.

Error control and detection are functions of the data link layer.

Computation of the Polynomial Code Checksum for CRC-CCITT:

```
#include<stdio.h>
#include<conio.h>
#define DEGREE 16

int mod2add (int, int);
int getnext (int*, int);
int result[30];
void calc_crc(int length)

{
        int ccitt[]={1,0,0,0,1,0,0,0,0,0,0,1,0,0,0,0,1};
        int i=0,pos=0,newpos;
        while(pos<length-DEGREE)
        {
                /*Peforming mod-2 division for degree number of bits*/
                for (i=pos;i<pos+DEGREE+1;++i)
                        result[i]=mod2add(result[i],ccitt[i-pos]);
                newpos=getnext(result,pos);
        /* skipping mod-2 division if lesser than DEGREE bits are available for division */
                if(newpos>pos+1)
                        pos=newpos-1;
                ++pos;
        }
}
int getnext(int array[],int pos)
{
        int i =pos;
        /* Checking if bits are 0s and skipping */
        while(array[i]==0)
          ++i;
        return i;
}
int mod2add(int x,int y)
{
        return (x==y?0:1);
}
```

```
void main()
{
        int array[30],ch,length,i=0;
        clrscr();
        /*Input data */
        printf ("Enter the data stream: ");
        while((ch=getche())!='\r')
                array[i++]=ch-'0';
        length=i;

        /*Appending zeroes*/
        for(i=0;i<DEGREE;++i)
                array[i+length]=0;
        length+=DEGREE;

        /*Duplicating data input for use by functions*/
        for (i=0;i<length;++i)
                result [i]=array[i];
        calc_crc(length);                       /*Calculation of CRC*/
        printf("\nThe transmitted frame is:");
        for (i=0;i<length-DEGREE;++i)           /*Printing the data*/
                printf("%d",array[i]);
        for (i=length-DEGREE;i<length;++i)      /*Printing the checksum*/
                printf("%d",result[i]);
        printf("\nEnter the stream for which CRC has to be checked");
        i=0;

        /*Inputting the stream to be checked*/
        while((ch=getche())!='\r')
                array[i++]=ch-'0';
        length=i;

        /*Duplicating the array*/
        for (i=0;i<length;i++)
                result [i]=array [i];
        calc_crc(length);                       /*Calculation of CRC*/
        printf("\nChecksum: ");
        for(i=length-DEGREE;i<length;i++)       /*Printing the checksum*/
                printf("%d",result[i]);
        getch();
}
```

Output:

Enter the data stream: 10011
The transmitted frame is: 100110010001001010010
Enter the stream for which CRC has to be checked: 100110010001001010010
Checksum: 0000000000000000

Experiment No: 5

Stop and Wait / Sliding Window Protocols

Aim: Write a C- program to implementation of Stop and Wait Protocol and Sliding Window Protocol.

C-program to implement Stop-and-wait protocol

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

int main()
{

   int i,j,noframes,x,x1=10;
   for(i=0;i<200;i++)
      rand();
   noframes=rand()/200;
   i=1;
   j=1;
   noframes=noframes/8;
   printf("\n Number of frames is %d", noframes);
   getch();
   while(noframes>0)
   {
      printf("\n Sending frame %d",i);
      srand(x1++);
      x=rand()%10;
      if(x%2==0)
      {
         printf("\nsending frame %d",i);
         srand(x1++);
         x=rand()%10;
      }
      printf("\n Ack for frame %d ",j);
      noframes-=1;
      i++;
      j++;
      }
      printf("\n End of Stop and wait protocol");
   }
```

Output:

```
Number of frames is 8
 Sending frame 1
 Ack for frame 1
 Sending frame 2
sending frame 2
 Ack for frame 2
 Sending frame 3
 Ack for frame 3
 Sending frame 4
sending frame 4
 Ack for frame 4
 Sending frame 5
sending frame 5
 Ack for frame 5
 Sending frame 6
 Ack for frame 6
 Sending frame 7
sending frame 7
 Ack for frame 7
 Sending frame 8
 Ack for frame 8
 End of Stop and wait protocol
```

C- program to implement Sliding Window Protocol

```
#include<stdio.h>
int main()
{
        int windowsize,sent=0,ack,i;
        printf("enter window size\n");
        scanf("%d",&windowsize);
        while(1)
        {
                for( i = 0; i < windowsize; i++)
                        {
                                printf("Frame %d has been transmitted.\n",sent);
                                sent++;
                                if(sent == windowsize)
                                        break;
                        }
                        printf("\nPlease enter the last Acknowledgement received.\n");
```

```
                        scanf("%d",&ack);

                        if(ack == windowsize)
                                break;
                        else
                                sent = ack;
        }
return 0;
}
```

Output:

```
enter window size
8
Frame 0 has been transmitted.
Frame 1 has been transmitted.
Frame 2 has been transmitted.
Frame 3 has been transmitted.
Frame 4 has been transmitted.
Frame 5 has been transmitted.
Frame 6 has been transmitted.
Frame 7 has been transmitted.

Please enter the last Acknowledgement received.
7
Frame 7 has been transmitted.

Please enter the last Acknowledgement received.
8
```

Experiment No: 6

Leaky Bucket Algorithm

AIM: Write a program for congestion control using Leaky bucket algorithm.

Theory: 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.

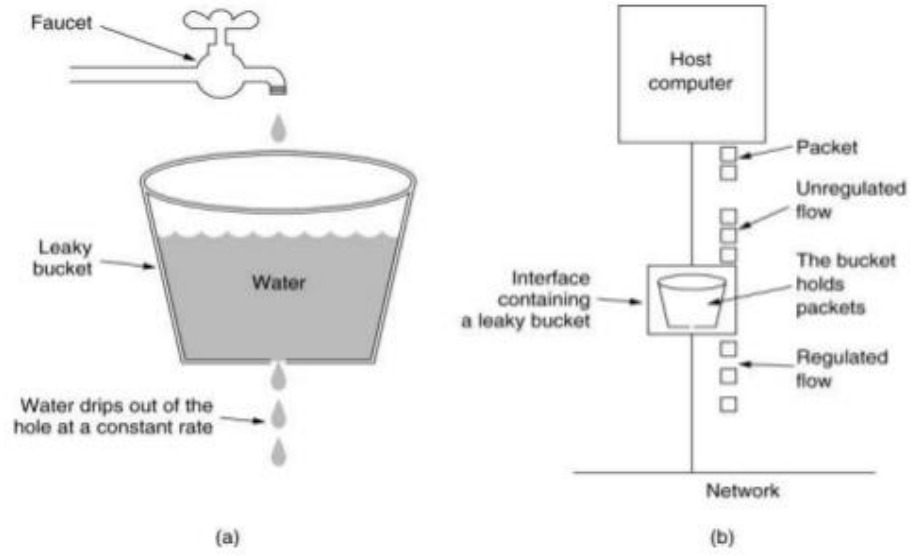
In contrast, closed loop solutions are based on the concept of a feedback loop if there is any congestion. Closed loop algorithms are also divided into two sub categories: explicit feedback and implicit feedback. In explicit feedback algorithms, packets are sent back from the point of congestion to warn 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.



(a) A leaky bucket with water. (b) a leaky bucket with packets.

C- program to implement Leaky Bucket Interface

```
#include<stdio.h>
#define MAX 25
#define min(x,y) ((x)<(y)? (x):(y))

int main()
{
   int cap, oprt,nsec,cont,i=0,inp[MAX],ch;

   printf("Leaky Bucket Algorithm \n");
   printf("Enter the bucket size:\n");
   scanf("%d",&cap);                  //Capacity of the Bucket
   printf("Enter the accepted rate:\n");
   scanf("%d",&oprt);                 //Rate at which Bucket Outputs Packets
   do
   {
      printf("Enter the number of packets entering at %d seconds: ",i+1);
      scanf("%d",&inp[i]);            //How many packets enter?
```

```
    i++;
    printf("Enter 1 to insert packets or 0 to quit\n");
    scanf("%d",&ch);                //To continue press 1.
  }while(ch);                       // As long as choice is 1, loop back.
  nsec=i;
  printf("\n (seconds):(packets received):(packets sent):(packets left in bucket)\n");
  cont=0;
  for(i=0;i<nsec;i++)
  {
    cont+=inp[i];
    if(cont>cap)
      cont=cap;
    printf("(%d):",(i+1));
    printf("\t\t(%d):",inp[i]);
    printf("\t\t(%d):",min(cont,oprt));
    cont=cont-min(cont,oprt);
    printf("\t\t(%d)\n",cont);
  }
  for(;cont!=0;i++)
  {

    if(cont>cap)
      cont=cap;
    printf("(%d):",(i+1));
    printf("\t\t (0):");
    printf("\t\t(%d):",min(cont,oprt));
    cont=cont-min(cont,oprt);
    printf("\t\t(%d)\n",cont);
  }
  return(0);
}
```

Output:

```
Leaky Bucket Algorithm
Enter the bucket size:
20
Enter the accepted rate:
5
Enter the number of packets entering at 1 seconds: 25
Enter 1 to insert packets or 0 to quit
1
Enter the number of packets entering at 2 seconds: 5
```

Enter 1 to insert packets or 0 to quit
0

(seconds):(packets received):(packets sent):(packets left in bucket)

(1):	(25):	(5):	(15)
(2):	(5):	(5):	(15)
(3):	(0):	(5):	(10)
(4):	(0):	(5):	(5)
(5):	(0):	(5):	(0)

END OF DOCUMENT