RNS INSTITUTE OF TECHNOLOGY

(AICTE Approved, VTU Affiliated and NAAC 'A Grade' Accredited)
UG Program Accredited by NBA up to 30.06.2025

Electronics and Communication Engineering

COMPUTER NETWORKS LABORATORY MANUAL

VII SEMESTER (18ECL77)

RNS INSTITUTE OF TECHNOLOGY

Dr. VISHNUVARDHAN ROAD, CHANNASANDRA, BENGALURU -560 098

Department of Electronics & Communication Engineering

VISION of the College: Building RNSIT into a World - Class Institution

MISSION of the College: To impart high quality education in Engineering, Technology and Management with a difference, enabling students to excel in their career by

1. Attracting quality Students and preparing them with a strong foundation in fundamentals so as *to achieve distinctions in various walks of life* leading to outstanding contributions.
2. Imparting value based, need based, and choice based and skill based professional education to the aspiring youth and *carving them into disciplined, World class Professionals* with *social responsibility.*
3. Promoting excellence in Teaching, Research and Consultancy that galvanizes academic consciousness among Faculty and Students.
4. Exposing Students to emerging frontiers of knowledge in various domains and make them suitable for Industry, Entrepreneurship, Higher studies, and Research & Development.
5. Providing freedom of action and choice for all the Stake holders with better visibility.

VISION of the Department: Conquering technical frontiers in the field of Electronics and Communications

MISSION of the Department

1. To achieve and foster excellence in core Electronics and Communication engineering with focus on the hardware, simulation and design.

2. To pursue Research, development and consultancy to achieve self sustenance.
3. To create benchmark standards in Electronics and Communication engineering by active involvement of all stakeholders.

PROGRAM EDUCATIONAL OBJECTIVES (PEOs)

The graduates shall

PEO1: Demonstrate the ability to apply fundamental concepts of Electronics and Communication engineering subjects, mathematics and basic sciences to solve real world problems.

PEO2: Exhibit technical and analytical skills for the design and development of innovative electronic systems leading to research and product development and to become entrepreneurs.

PEO3: Be able to crack various competitive examinations and pursue higher education in India and abroad for a successful carrier.

PEO4: Work in multidisciplinary fields encouraging team work to come up with new technologies for the benefit of the society.

PEO5: Display the finest professional skills: positive attitude, leadership qualities, effective communication, self-education and ethics needed for a successful career and life-long learning.

PROGRAM OUTCOMES (POs)

Engineering Graduates will be able to:

PO1: Engineering knowledge: Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization for the solution of complex engineering problems

PO2: Problem analysis: Identify, formulate, research literature, and analyze complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences and engineering sciences.

PO3: Design/development of solutions: Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for public health and safety, and cultural, societal, and environmental considerations.

PO4: Conduct investigations of complex problems: Use research-based knowledge and research methods including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions.

PO5: Modern tool usage: Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools, including prediction and modeling to complex engineering activities, with an understanding of the limitations.

PO6: The engineer and society: Apply reasoning informed by the contextual knowledge to assess Societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to the professional engineering practice.

PO7: Environment and sustainability: Understand the impact of the professional engineering solutions in societal and environmental contexts, and demonstrate the knowledge of, and need for sustainable development.

PO8: Ethics: Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.

PO9: Individual and team work: Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.

PO10: Communication: Communicate effectively on complex engineering activities with the engineering community and with the society at large, such as, being able to comprehend and write effective reports and design documentation, make effective presentations, and give and receive clear instructions.

PO11: Project management and finance: Demonstrate knowledge and understanding of the engineering and management principles and apply these to one's own work, as a member and leader in a team, to manage projects and in multidisciplinary environments.

PO12: Life-long learning: Recognize the need for, and have the preparation and ability to engage in independent and life-long learning in the broadest context of technological change.

PROGRAM SPECIFIC OUTCOMES(PSOs)

The graduates of the department will be able to:

PSO1: Apply fundamental knowledge of Electronics, Communications, Signal processing, VLSI, Embedded and Control systems etc., in the analysis, design, and development of various types of real-time integrated electronic systems and to synthesize and interpret the experimental data leading to valid conclusions.

PSO2: Demonstrate competence in using Modern hardware languages and IT tools for the design and analysis of complex electronic systems as per industry standards along with analytical and managerial skills to arrive at appropriate solutions, either independently or in team.

RNS INSTITUTE OF TECHNOLOGY

Dr. VISHNUVARDHAN ROAD, CHANNASANDRA, BENGALURU -560 098

Department of Electronics & Communication Engineering

COMPUTER NETWORKS LABORATORY

Subject Code: 18ECL77 **Total Hours: 40**

Hours/Week: 4 hours **Exam Hours: 03**

Subject Code	18ECL77
CIE MARKS	40
SEE Marks	60

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.

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

CO1	Use the network simulator for learning and practice of networking algorithms.
CO2	Illustrate the operations of network protocols and algorithms using C programming
CO3	Simulate the network with different configurations to measure the performance parameters.
CO4	Implement the data link and routing protocols using C programming.
CO5	Design and simulate network elements with various protocols and standards

CO mapping to PO/PSOs

	PO1	PO2	PO3	PO4	PO5	PO6	PO7	PO8	PO9	PO10	PO11	PO12	**PSO1**	**PSO2**
CO1	2	2	1	1	2					1				3
CO2	1	2	2	1	1					1				3
CO3	1	1	2	1	2					1				3
CO4	2	2	1	1	2					1				3
CO5	2	2	1	1	3					1				3
Final_CO_one_row	1.6	1.8	1.4	1	2					1				**3**

COMPUTER NETWORKS Laboratory

Evaluation Rubrics

Subject Code: 18ECL77	CIE Marks:40
Hours/Week: 2 Hrs Tutorial +02 Hours Laboratory	Exam Hours: 03
Total Hours: 40	SEE Marks:60

Lab Write-up and Execution Rubrics (Max: 10 marks)

<u>For CBCS scheme</u>:

Total Lab Marks = Record(10M) + Observation (5M) + Viva(2M)+Unit test(4M+4M) +Final Lab internal test marks(20M) =40M

Scheme	Record(10M)	Unit Test(4+4)	Observation (5M)	Lab Internal Test (20M)
CBCS	**10 marks is allocated for:** 1. Working program 2. Algorithm/ flowchart 3. Theory 4. Example 5. Result <5 marks if any lacuna in the above	Full marks are given based on 1. Working program 2. Algorithm/ flowchart 3. Example Less than total marks if any lacuna in the above	This is based on the program to be executed in the lab and recording the results	**The test is conducted at the end of the semester, after the lab completion.** Test is conducted for 100 marks and reduced to 20 marks scale. The marks split up for 100 is as follows: Procedure-15 Viva- 15 Conduction-70 < 70 is given for Conduction, if the student fails to conduct the experiment successfully

COMPUTER NETWORKS LAB

Course Code : **18ECL76**	CIE Marks : 40	SEE Marks : 60
Lecture Hours/Week: 02 Hours Tutorial (Instructions) + 02 Hours Laboratory		
RBT Level : L1, L2, L3	Exam Hours : 03	
CREDITS – 02		

Course Learning Objectives: This course will enable students to:

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

Laboratory Experiments

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

1. Implement a point to point network with four nodes and duplex links between them. Analyze the network performance by setting the queue size and varying the bandwidth.
2. Implement a four node point to point network with links n0-n2, n1-n2 and n2-n3. Apply TCP agent between n0-n3 and UDP between n1-n3. Apply relevant applications over TCP and UDP agents changing the parameter and determine the number of packets sent by TCP/UDP.
3. Implement Ethernet LAN using n (6-10) nodes. Compare the throughput by changing the error rate and data rate.
4. Implement Ethernet LAN using n nodes and assign multiple traffic to the nodes and obtain congestion window for different sources/ destinations.
5. Implement ESS with transmission nodes in Wireless LAN and obtain the performance parameters.
6. Implementation of Link state routing algorithm.

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

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

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

1. Choose suitable tools to model a network.
2. Use the network simulator for learning and practice of networking algorithms.
3. Illustrate the operations of network protocols and algorithms using C programming.
4. Simulate the network with different configurations to measure the performance parameters.
5. Implement the data link and routing protocols using C programming.

Conduct of Practical Examination:

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

Part A:

NCTUns & NS3

The following experiments shall be conducted using either NCTUns/NS3 or any other simulators.

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 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.
3. Implement Ethernet LAN using n-nodes (6-10), change error rate and data rate and compare the throughput.
4. Simulate an Ethernet LAN using n nodes and assign multiple traffic nodes and plot congestion window for different source/destination.
5. Implement ESS with transmitting nodes in wire-less LAN by simulation and obtain the performance parameters.
6. Implementation of Link State routing algorithm.

NCTUns (Produced and maintained by Network and System Laboratory, Department of Computer Science, National Chiao Tung University, Taiwan)

Components and Architecture of NCTUns

NCTUns adopts a distributed architecture. It is a system comprising eight components.

1. The first component is the GUI program by which a user edits a network topology, configures the protocol modules used inside a network node, specifies mobile nodes, initial location and moving paths, plots performance graphs, plays back the animation of a packet transfer trace, etc.
2. The second component is the simulation engine program, which provides basic and useful simulation services (e.g.,event scheduling, timer management, and packet manipulation,etc.) to protocol modules. We call a machine on which a simulation engine program resides a "simulation server."
3. The third component is the set of various protocol modules, each of which implements a specific protocol or function (e.g., packet scheduling or buffer management).All protocol modules are C++ classes and are compiled and linked with the simulation engine program.
4. The fourth component is the simulation job dispatcher program that can simultaneously manage and use multiple simulation servers to increase the aggregate simulation throughput. It can be run on a separate machine or on a simulation server.
5. The fifth component is the coordinator program. On every simulation server, the "coordinator" program must be run up. The coordinator should be alive as long as the simulation server is alive. When a simulation server is powered on and brought up, the coordinator must be run up. It will register itself with the dispatcher to join in the dispatcher's simulation server farm. Later on, when the

status (idle or busy) of the simulation server changes, it will notify the dispatcher of the new status. This enables the dispatcher to choose an available simulation server from its simulation server farm to service a job.

When the coordinator receives a job from the dispatcher, it forks a simulation engine process to simulate the specified network and protocols. It may also fork several real-life application program processes specified in the job. These processes are used to generate traffic in the simulated network. When the simulation engine process is alive, the coordinator communicates with the dispatcher and the GUI program on behalf of the simulation engine process. For example, the simulation engine process needs to periodically send its current simulation clock to the GUI program. This is done by first sending the clock information to the coordinator and then asking the coordinator to forward this information to the GUI program. This enables the GUI user to know the progress of the simulation. During a simulation, the GUI user can also on-line set or get an object's value (e.g., to query or set a switch's current switch table). Message exchanges that occur between the simulation engine process and the GUI program are all relayed via the coordinator.

6 The sixth component is the kernel patches that need to be made to the kernel source code so that a simulation engine process can run on a UNIX machine correctly. Currently NCTUns 6.0 runs on Red-Hat's Fedora 11, which uses the Linux 2.6.28.9 kernel.

7 The seventh component is the various real-life user-level application programs. Due to the novel kernel-reentering simulation methodology, any real-life existing or to-be developed application program can be directly run up on a simulated network to generate realistic network traffic.

8 The eighth component is the various user-level daemons that are run up for the whole simulation case. For example, NCTUns provides RIP and OSPF routing daemons. By running these daemons, the routing entries needed for a simulated network can be constructed automatically. As another example, NCTUns provides and automatically runs up several emulation daemons when it is turned into an emulator.

Following steps are followed for simulation

Draw network topology

After the starting screen of NCTUns disappears, a user will be presented a working window where the required topology can be drawn.

Edit the properties of the network components

A network node (device) may have many parameters to set. For example, we may want to set the maximum queue length of a FIFO queue used inside a network interface. For another example, we may want to specify that some application programs (traffic generators) should be run up on some hosts or routers to generate network traffic.

Run the simulation

When the mode is switched to the "Run Simulation" mode, the GUI will export many simulation files that collectively describe the simulation case. These simulation files will be transferred to the (either remote or local) simulation server for it to execute the simulation. These files are stored in the "mainFileName.sim" directory, where mainFileName is the name of this simulation case.

Packet Animation Player

By using the packet animation player, a packet transfer trace logged during a simulation can be replayed at a specified speed. Both wired and wireless networks are supported. This capability is very useful because it helps a researcher visually see and debug the behavior of a network protocol. It is very useful for educational purposes because students can see how a protocol behaves.

Play back animation features

- A link is painted in yellow color if there is any packet flowing on it.
- A packet is depicted by a segment with an arrow (for brevity, it will be called an “arrow” in the following description).
- A collided packet is depicted by an arrow with a cross on it.
- During a packet transfer, if a link is painted in red, it means that this link is an intermediate link for this packet. In contrast, if this link is painted in yellow, it means that one end of this link is the real source or destination node of the packet.
- The arrow length is determined by the packet’s length. Therefore, a user can expect that the arrow length is proportional to the packet length. In fact, a packet’s segment length on a particular link is determined by the transmission time of that packet on that link relative to the signal propagation delay of the link.

GETTING STARTED

Setting up the environment

A user using the NCTUns in single machine mode, needs to do the following steps before he/she starts the GUI program:

1. Start up the dispatcher on terminal 1.
2. Start up the coordinator on terminal 2.
3. Start up the nctunsclient on terminal 3.

4. Enter command ./run and press tab and press enter.

After the above steps are followed, the starting screen of NCTUns disappears and the user is presented with the working window.

<u>Commands used:</u>

TCP command for sender	UDP command for receiver
stcp –p 2000 –l 1024 1.0.1.2	stg –u 1024 100 1.0.1.2
rtcp –p 2000 –l 1024	rtg –u –w log1

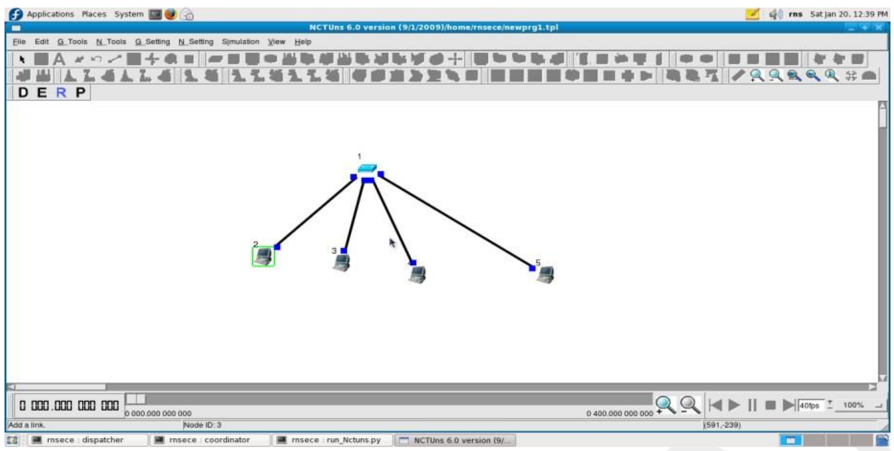
where,

p = port no, 1024 = length of packet size, 1.0.1.2 – receiver ip address for TCP

u = user datagram, 100 = time in seconds for UDP

<u>**PART A programs**</u>

1. Implement a point to point network with four nodes and duplex links between them. Analyze the network performance by setting queue size and varying the bandwidth .



STEPS:

Step1: Create a topology as shown in the below figure.

Step 2: Select the hub icon on the toolbar and drag it onto the working window.

Step 3: Select the host icon on the toolbar and drag it onto the working window. Repeat this for another host icon.

Step 4: Select the link icon on the toolbar and drag it on the screen from host (node 2) to the hub ,host(node 3) to hub, node 4 to hub and again from host(node 5) to the hub.

here node1 is sender and other nodes are receiver where bandwidth and queue size varied to compare network performance.

This leads to the creation of the 4-node point-to-point network topology. Save this topology as a **.tpl file.**

Step 5: Double-click on host(node 2) -sender, a host dialog box will open up.

Click on the add and type the following Command

stcp –p 3000 –l 1024 1.0.1.2 click ok.

Here, 1.0.1.2 is IP address of the host 2 (Node 3), and 3000 is the port no.

Click on **Node editor** and you can see the different layers- interface, ARP, FIFO, MAC, TCPDUMP, Physical layers.

Select MAC and then select full-duplex and in log Statistics, select Throughput of incoming packets for receiver and Throughput of outgoing packets for sender.

select FIFO to change queue size, and select PHY to change bandwidth (AT RECEIVER NODE)

Step 6: Double-click on host (node 3),host (node 4),host(node5) and follow the same step as above with only change in command according to the following syntax:

rtcp –p 3000 –l 1024 and click OK.

here node 2 is considered as sender, node 3 and node 4 as receiver.

Step 7: Click on the **E button** (Edit Property) present on the toolbar in order to save the changes made to the topology. Now click on the **R button** (Run Simulation).

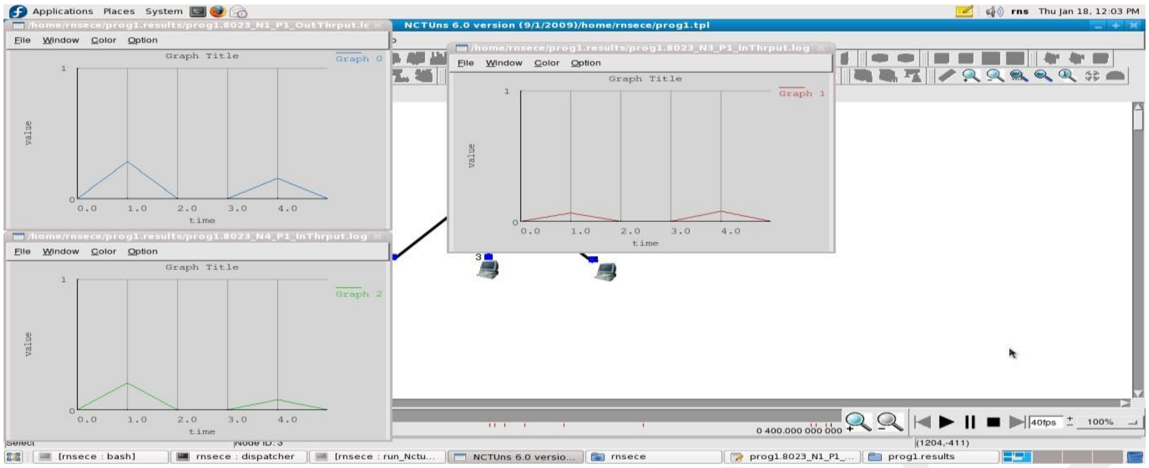
By doing so a user can run/pause/continue/stop/abort/disconnect/reconnect/submit a simulation. No settings in the simulation can be changed in this mode.

Step 8: Now go to **Menu->Simulation->Run**. Executing this command will submit the current Simulation job to one available simulation server managed by the dispatcher. When the simulation server is executing, the user will see the time knot at the bottom of the screen move.The time knot reflects the current virtual time (progress) of the simulation case.

Step 9: After the simulation is completed, click on the play button and mean while plot the graphs of the drop packets and through put input and through put output. These log files are created in **filename.results** folder.

Step 10: Now click on the link (connecting the hub and host2) and change the bandwidth say, 9 Mbps, and run the simulation and compare the two results.

Graph sheets:



EXAMPLE AND RESULTS OF EXPT 1

By using HUB:

Commands used: stcp –p 3000 –l 1024 1.0.1.2

rtcp –p 3000 –l 1024

By setting the bandwidth as 10 Mbps on both the links and queue size as 50, we obtain the following results:(node2)

Output throughput (host 2) = 426kbs

Input throughput (host 3) = 206kbs

By changing bandwidth to 2Mbps in the destination link and queue size as 10 , we obtain the following results:(node3)

Output throughput (host 2) =426kbs

Input throughput (host 4) = 192kbs

By changing bandwidth to 1Mbps in the destination link and queue size as 5 , we obtain the following results:(node4)

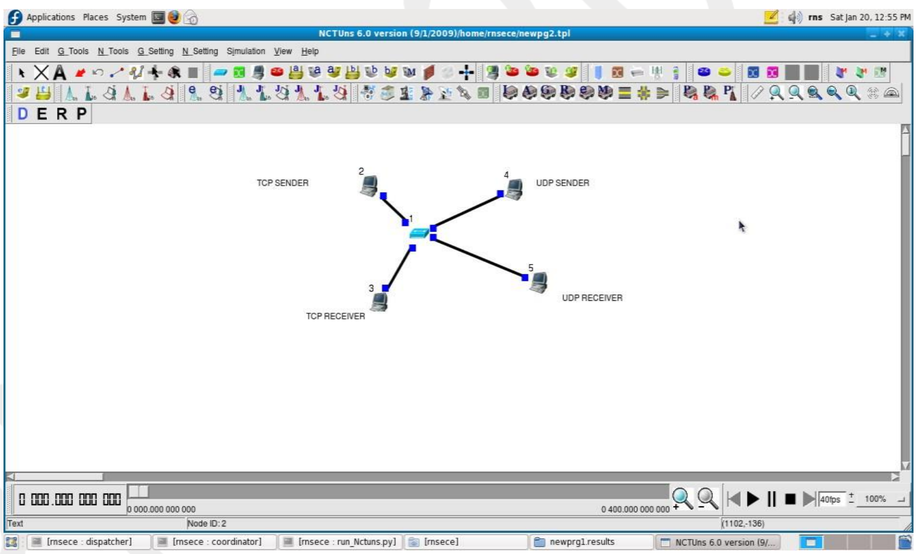
Output throughput (host 2) =426kbs

Input throughput (host 4) = 270kbs

2. Implement a four node point to point network with links n0-n1,n1-n2 and n2-n3.apply TCP agent between n0-n3 and UDP between n1-n3.apply relevant application over TCP and UDP agents changing the parameter and determine the number of packets sent by TCP/UDP

STEPS:

Step1: Create a topology as shown in the below figure.



Step 2: Select 4 Nodes (2,3,4,5) and connect them using a hub as shown

Step 3: Go to mode edit and save the topology.

Step 4a: Double click on host (Node 2) and goto **node editor,** and click on MAC 8023 and put a check on the Throughput of Outgoing Packets. Click ok. Then click on **ADD** and type the following command.

stcp –p 3000 –l 1024 1.0.1.3

and click OK. Here 1.0.1.3 is the IP address of the Host 3 (Receiver) and 3000 is the port no.

Step 4b: Double-click on host (node 4), and follow the same step as above with only change in command according to the following syntax:

stg –u 1024 100 1.0.1.3

and click OK. Here, 1.0.1.3 is Receiver IP and 100 is the bandwidth. **This forms the UDP connection**.

Step 5: Double-click on host (node 3), and follow the same step as above and check on Throughput of Incoming Packets. And type the following commands:

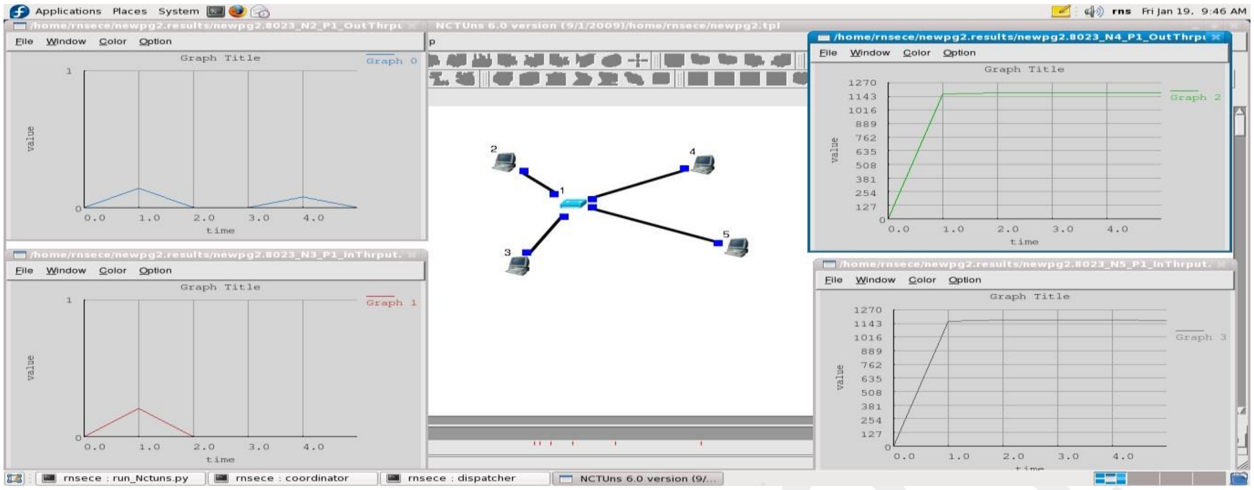
rtcp –p 3000 –l 1024 click ok (for TCP)

rtg –u –w log1 (node5) click ok (for UDP),

Here, w is bandwidth and log1 is the name of the file.

Step 6: Click **R Button** and then go to **Menu->Simulation->Run.**

Step 7: After the simulation is completed, click on the play button and mean while plot the graphs of the Throughput Input and Throughput Output. These log files are created in **filename.results** folder.



EXAMPLE AND RESULTS OF EXPT 2

Commands used:

stcp -p 3000 -l 1024 1.0.1.3 (for TCP)

stg -u 1024 100 1.0.1.3 (for UDP)

rtcp -p 3000 -l 1024 (for TCP)

rtg -u -w log1 (for UDP)

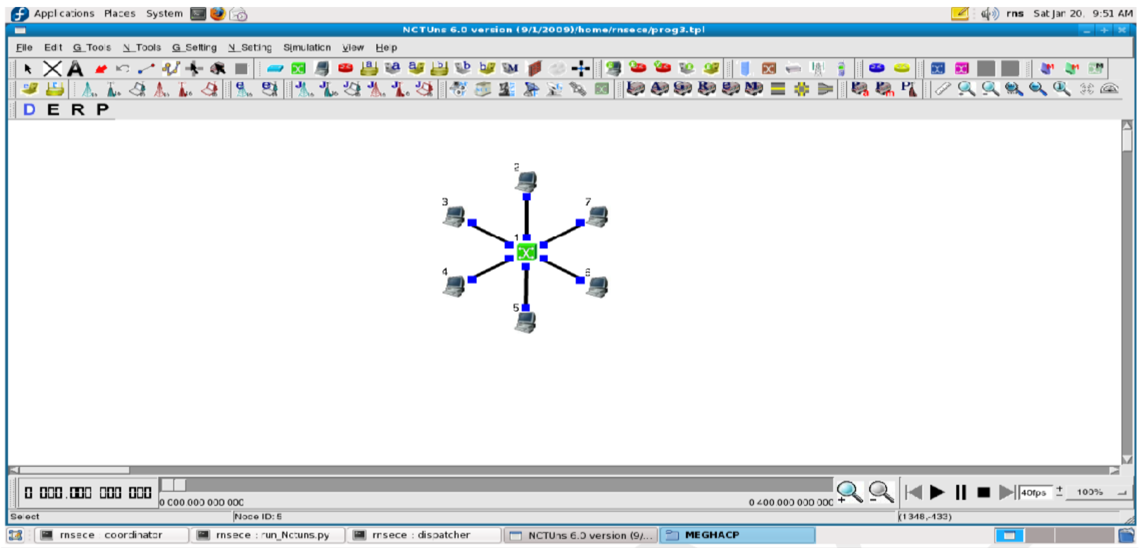
Results: By setting the bandwidth as 100 Mbps on the TCP link and queue size as 50, we obtain the following results:

Average no: of TCP packets sent = varying (142)

Average no: of UDP packets sent = 1164

Note: The result varies based on the bandwidth

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



Step1: Create a topology as shown in the above figure.

Step 2: Select host subnet option in the pop up window enter nodes = 6 and radius = 100 or select 6 Nodes (2-7) and connect them using a hub(1).

Step 3: Go to edit mode and save the topology.

Step 4: Let us say, Node 3, 2 and node 5 are source and destination respectively.

Double click on host (Node 3 and 2) and goto **node editor**, and click on MAC 8023 and put a check on the **Throughput of Outgoing Packets**. Click ok. Then click on **ADD** and type the following command.

stcp –p 3000 –l 1024 1.0.1.4

Here 1.0.1.6 is the IP address of the Host (Receiver) and 3000 is the port no.**Step 5**: Double-click on host/receiver (node 5), and follow the same steps as above and check on **Throughput of Incoming Packets**. And type the following commands:

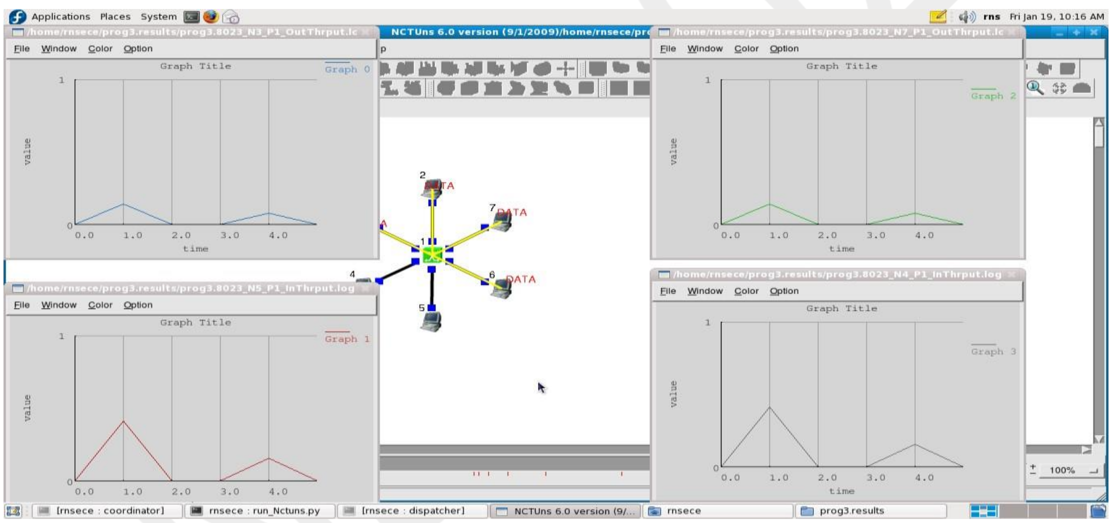
rtcp –p 3000 –l 1024

Step 6: Click on **R Button** and then goto **Menu->Simulation->Run.**

Step 7: After the simulation is completed, click on the play button and mean while plot the graphs of the T**hroughput Input and Throughput Output**. These log files are created in **filename.results** folder

Step 8: Change error rate (Bit error rate) and data rate (Bandwidth) in Physical layer or on the point to point link connecting any one of sender and receiver and then run the simulation again and compare the T**hroughput Input and Throughput Output** of 1st and 2nd reading.

Graphs Sheet:



Result:

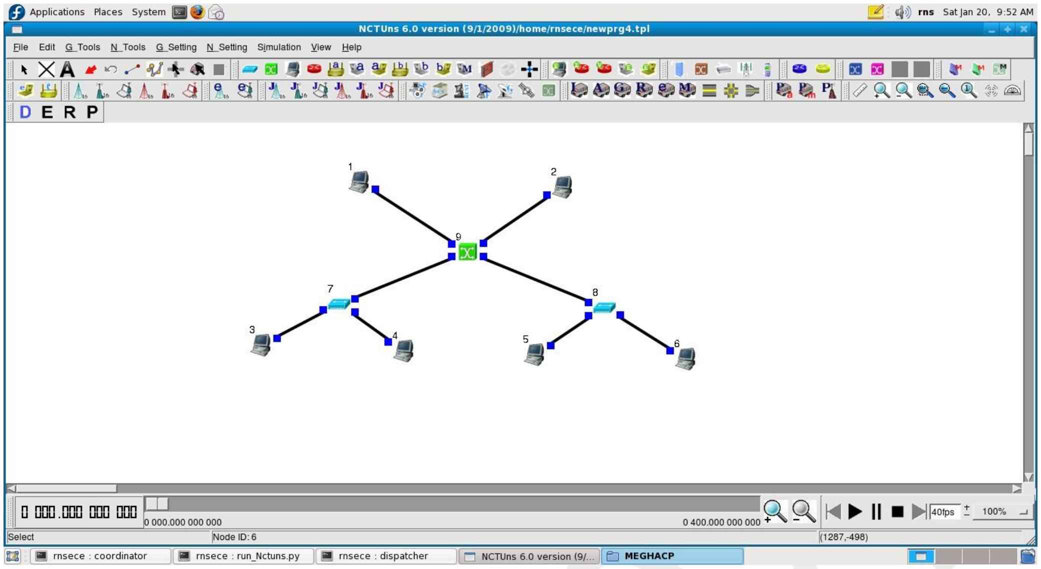
Node 3, 2 are senders: Outthroughput = 142 kbps

Node 5 is receiver : Inthroughput = 206 kbps.

Result varies based on the change in bit error rate and bandwidth.

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

Topology



Step1: Create a topology as shown in the below figure

Step 2: Go to **edit mode** and save the topology. IP addresses will be generated for all hosts.

Step 3: Node1 and Node2 are both source and destination. Node 3 is the receiver of node 2, node 4 is the sender to node 2.Node 5 is the receiver of node 1, node 6 is the sender to node1. Double click on host (Node 1) and go to **node editor**, and click on MAC 8023 and put a check on the **Collision and Packets Dropped**. Click ok. Then click on **ADD** and type the following command.

stcp –p 3000 –l 1024 1.0.1.5

Here 1.0.1.5 is the IP address of the Host 5 (Receiver) and 3000 is the port no.

Repeat the same step as above for node 2.

Step 4: Double-click on host (node 5), and follow the same step as above And type the following commands:

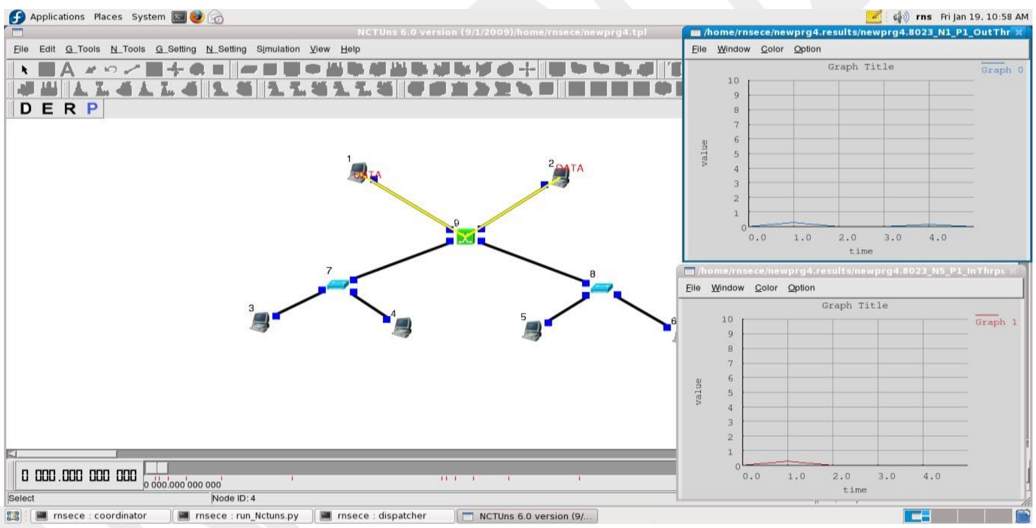
rtcp –p 3000 –l 1024

Repeat the same step as above for node 2 and node 3. While node 4 and node 6 acts as senders to nodes 1 and 2 respectively thereby create a multiple traffic.

Step 5: Click on **R Button** and then goto **Menu->Simulation->Run.**

Step 6: After the simulation is completed, click on the play button and mean while plot the graphs of the **Collision and packets Dropped**. These log files are created in **filename.results** folder.

Graphs Sheet:



<u>Results:</u>

Node 1 sender to node 5 : Outthroughput = 434 kbps

Node 1 receiver from node 6 : Inthroughput = 270 kbps

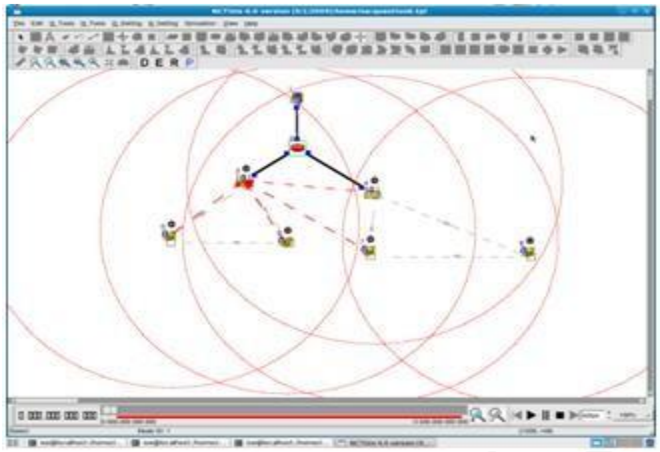
Node 5 receiver from node 1 : Inthroughput = 256 kbps

Node 6 sender to node 1 : outthroughput = 330 kbps

Node 5 collision packets = 2

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

Topology



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 to create Subnet

7. 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.

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.

ttcp –r –u –s –p 8001 (ttcp = test transmission control protocol)

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 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 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 NODE 3:- ttcp –t–u –s –p 8003 1.0.2.2

MOBILE NODE 4:- 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 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

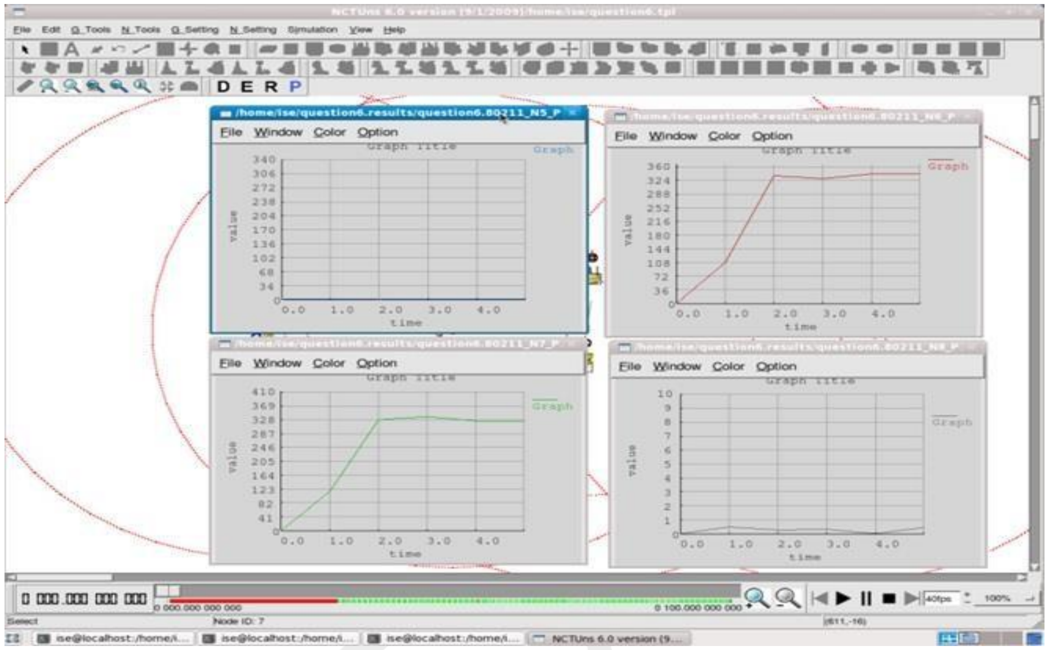
1. Click “R” icon on the tool bar

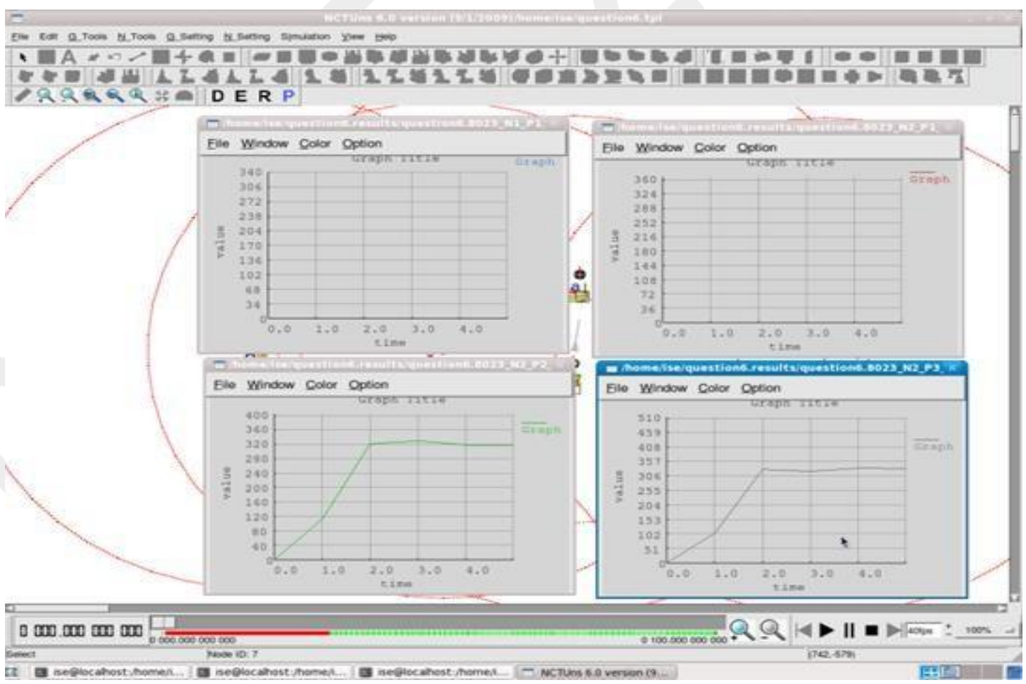
2. Select Simulation in the menu bar and click/ select RUN in the dropdown list to execute the simulation.

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.

Graphs Sheet:





NS3

Program 1:-

Implement a point – to – point network four nodes and with duplex links between them. Analyse the network performance by setting the queue size, vary the bandwidth.

Network topology

```
   10.1.1.0     10.1.2.0          10.1.3.0
n0________n1_______n2_______n3
Point-to-point
```

In this program we have created 4 point to point nodes n0, n1, n2, n3. Node n0 has IP address 10.1.1.1 and n4 has 10.1.3.2. Node n1 has 2 interfaces (10.1.1.2 and 10.1.2.1). Node 2 has 2 interfaces (10.1.2.2 and 10.1.3.1). UDP echo client-server application is used to generate the traffic at source node n0. Packets move from n0 to n3 via n1 & n2. Details of the flow (Number of packets sent, received and dropped) can be verified by using trace metrics (ccn1.tr file).

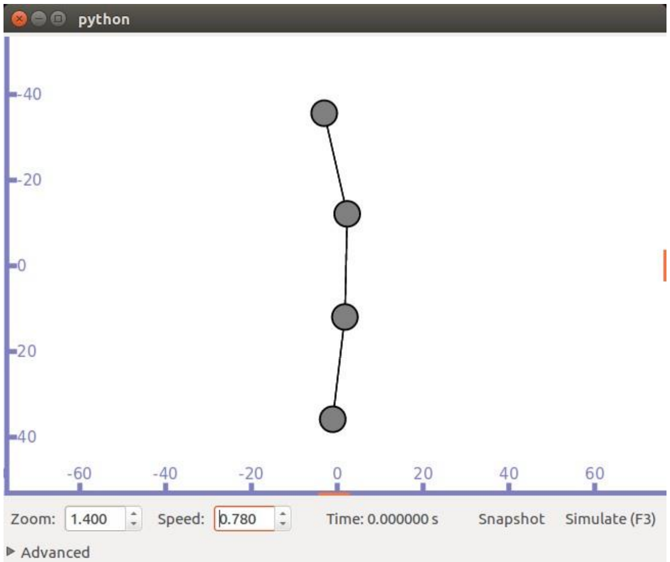
Algorithm

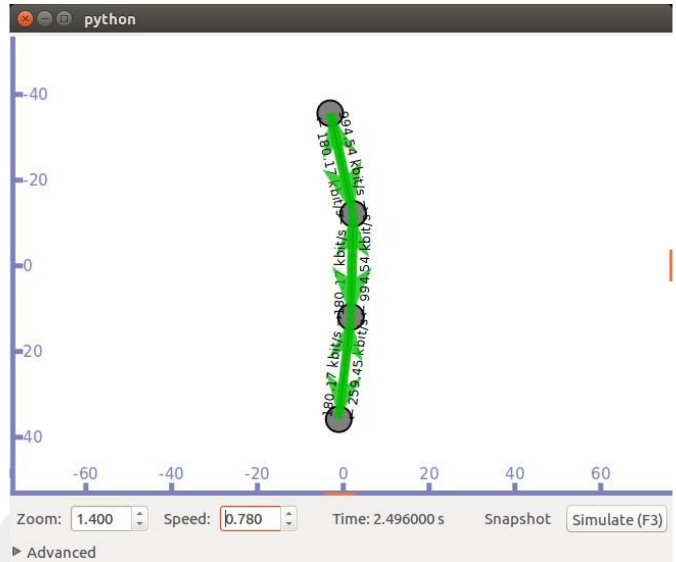
1. Start
2. Create a network of 3 nodes.
3. Add the internet stack to all nodes.
4. Create point to point channel between the nodes and set the device attribute like data rate and channel attribute like delay.
5. Assign the IP address to all nodes. (Note node 1 will have two IP address 10.1.1.2 for subnet1 and 10.1.2.1 to the subnet2)
6. Model the channel for an appropriate BER.
7. Install onoff application on both source (node 0) and sink (node2)
8. Simulate the application for 10 seconds.
9. Stop the simulation process by releasing the resources.
10. End

Program

```
#include "ns3/core-module.h"
#include "ns3/network-module.h"
#include "ns3/internet-module.h"
#include "ns3/point-to-point-module.h"
#include "ns3/applications-module.h"
#include "ns3/traffic-control-module.h"
using namespace ns3;
NS_LOG_COMPONENT_DEFINE ("FirstScriptExample");
int main (int argc, char *argv[])
{
  CommandLine cmd;
  cmd.Parse (argc, argv);

  NodeContainer nodes;                    // to create 3 nodes
 nodes.Create (4);
 InternetStackHelper stack;               // installing into stack
  stack.Install (nodes);

  PointToPointHelper pointToPoint;        // properties of channel
  pointToPoint.SetDeviceAttribute ("DataRate", StringValue ("5Mbps"));
  pointToPoint.SetChannelAttribute ("Delay", StringValue ("4ms"));

pointToPoint.SetQueue("ns3::DropTailQueue","MaxPackets",UintegerValue(10));

  Ipv4AddressHelper address;       // assigning IP address between node 0 to node1
  address.SetBase ("10.1.1.0", "255.255.255.0");

  NetDeviceContainer devices; // installing p2p link
  devices = pointToPoint.Install (nodes.Get(0),nodes.Get(1));
  Ipv4InterfaceContainer interfaces = address.Assign (devices);


  devices = pointToPoint.Install (nodes.Get(1),nodes.Get(2));  // installing p2p link
                                                               // node1 to node2
 address.SetBase ("10.1.2.0", "255.255.255.0");   // assigning IP address b/w
                                                  // node1 to node2
```

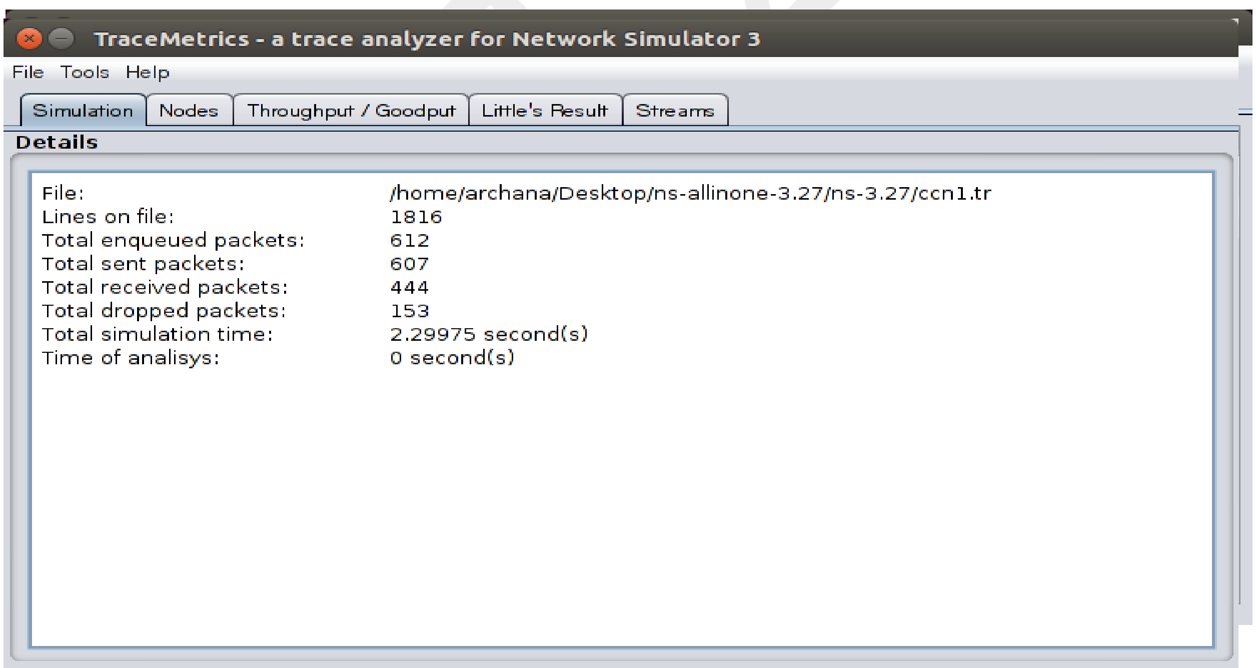
```
interfaces = address.Assign (devices);


devices = pointToPoint.Install (nodes.Get(2),nodes.Get(3));  // installing p2p link
                                                              //node2 to node3
address.SetBase ("10.1.3.0", "255.255.255.0"); // assigning IP address b/w node2
                                               //to node3
interfaces = address.Assign (devices);

Ptr<RateErrorModel> em = CreateObject<RateErrorModel>();
em –›SetAttribute("ErrorRate",DoubleValue(0.001));

devices.Get (0) –›SetAttribute("ReceiveErrorModel",PointerValue(em));devices.Get(1)–
›SetAttribute("ReceiveErrorModel",PointerValue(em));

Ipv4GlobalRoutingHelper::PopulateRoutingTables();

UdpEchoServerHelper echoServer (9);
ApplicationContainer serverApps = echoServer.Install (nodes.Get(3));
serverApps.Start (Seconds (1.0));
serverApps.Stop (Seconds (10.0));

UdpEchoClientHelper echoClient (interfaces.GetAddress (1), 9);
echoClient.SetAttribute ("MaxPackets", UintegerValue (500));
echoClient.SetAttribute ("Interval", TimeValue (Seconds (0.0)));
echoClient.SetAttribute ("PacketSize", UintegerValue (1024));

ApplicationContainer clientApps = echoClient.Install (nodes.Get (0));
clientApps.Start (Seconds (2.0));
clientApps.Stop (Seconds (10.0));

AsciiTraceHelper ascii;
pointToPoint.EnableAsciiAll(ascii.CreateFileStream("ccn1.tr"));

Simulator::Run ();
Simulator::Destroy ();
return 0;
}
./waf –run scratch/ccn1 --run
```



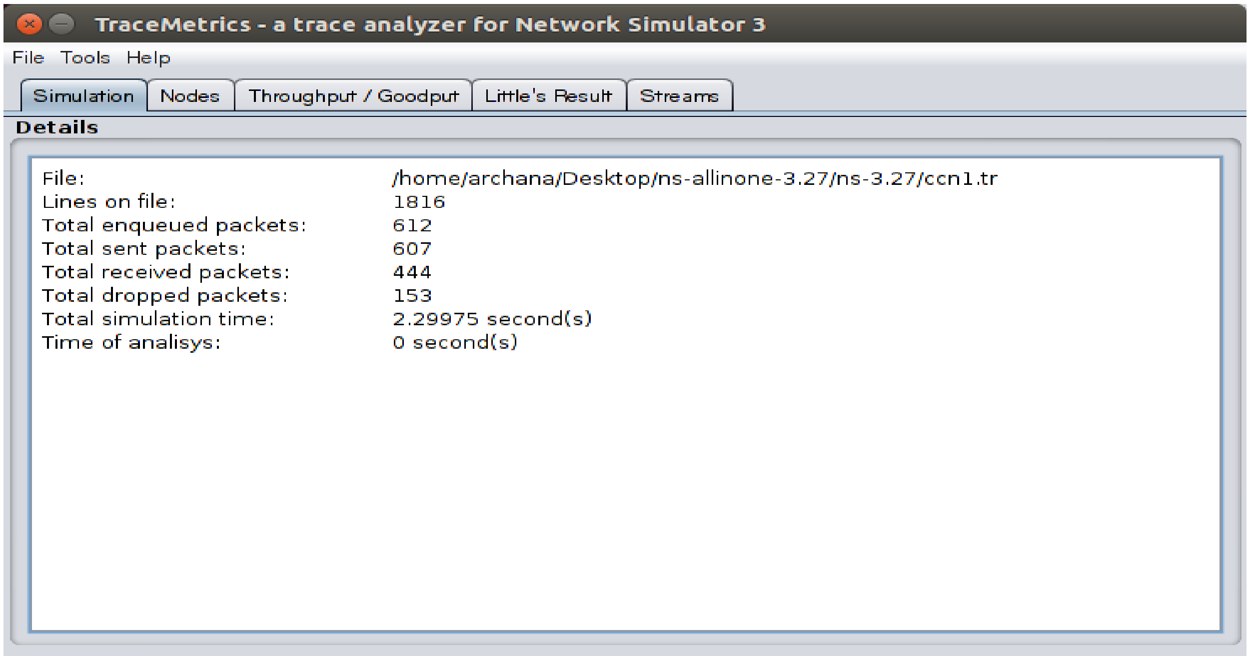


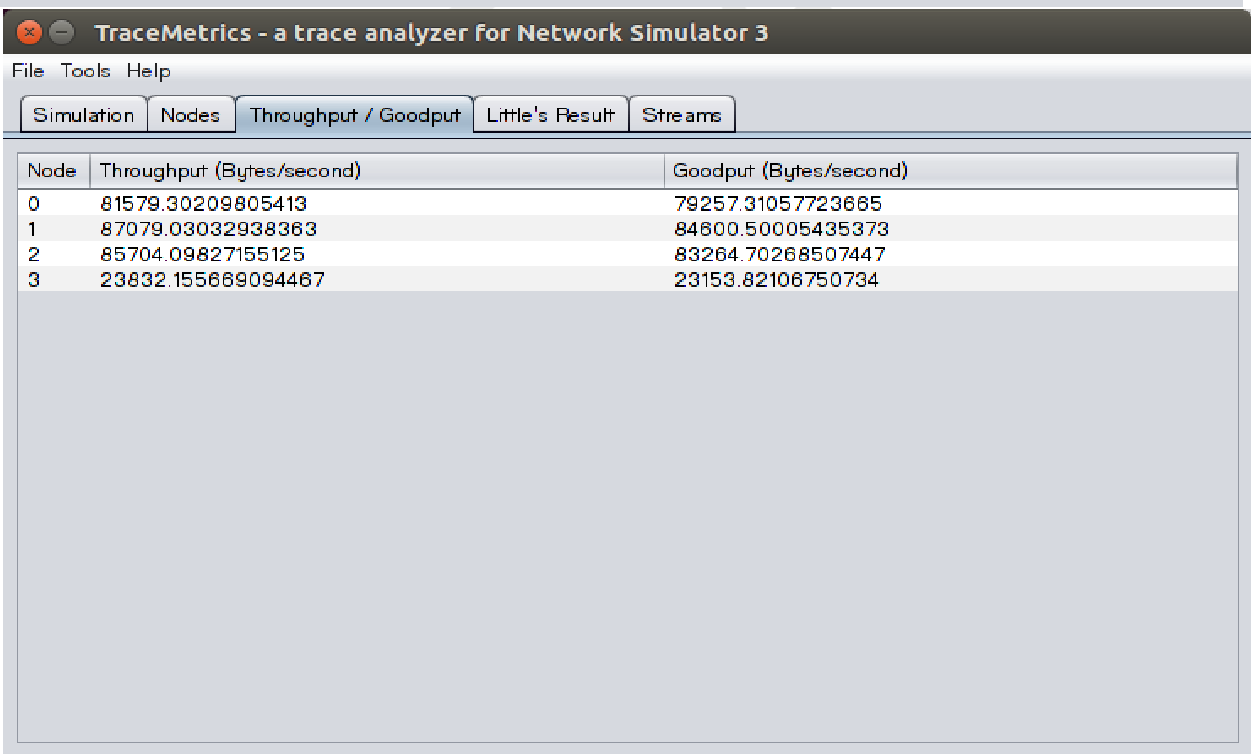
Desktop/tracemetrics-1.3.0 $ java -jar tracemetrics.jar

ASCII trace packet dropped and throughput at Data Rate=5Mbps



ASCII trace packet dropped and throughput at Data Rate=1Mbps.





Program 2:-

Implement a four node point to point network with links n0-n2, n1-n2 and n2-n3. Apply TCP agent between n0-n3and UDP between n1-n3. Apply appropriate TCP and UDP applications and determine the number of packets sent.

Network topology

```
    n0
      \ 5 Mb/s, 2ms (TCP)
       \
        \        1.5Mb/s, 10ms
        n2__________________n3
       /
      / 5 Mb/s, 2ms (UDP)
     /
    n1
```

Algorithm:-

1. Start
2. Create a network of 4 nodes such that N0 communicates N3 through N2 and runs a TCP application. Whereas N1 communicates N3 through N2 and runs UDP application.

3. Add the internet stack to all nodes.
4. Create point to point channel between the nodes and set the device attribute like data rate and channel attribute like delay.
5. Assign the IP address to all nodes. (Note node 2 will have three IP address 10.1.1.2 for subnet1, 10.1.2.1 to the subnet2 and 10.1.3.1 for subnet 3)
6. Model the channel for an appropriate BER.
7. Install on off application on both source (node 0) and sink (node3) for TCP communication and Echo-client-server application on client (node1) and (node2)
8. Simulate both the application for 10 seconds.
9. Stop the simulation process by releasing the resources.
10. End

Program:-

```
#include "ns3/core-module.h"

#include "ns3/network-module.h"
#include "ns3/internet-module.h"
#include "ns3/point-to-point-module.h"
#include "ns3/applications-module.h"
#include "ns3/traffic-control-module.h"
using namespace ns3;
NS_LOG_COMPONENT_DEFINE ("Lab-Program-1");
int main (int argc, char *argv[])
{

std::string socketType= "ns3::TcpSocketFactory";;
CommandLine cmd;

cmd.Parse (argc, argv);
Time::SetResolution (Time::NS);
LogComponentEnable ("UdpEchoClientApplication", LOG_LEVEL_INFO);
LogComponentEnable ("UdpEchoServerApplication", LOG_LEVEL_INFO);

NodeContainer nodes;
nodes.Create (4);   //4 point-to-point nodes are created

InternetStackHelper stack;
stack.Install (nodes);

//TCP-IP layer functionality configured on all nodes
//Bandwidth and delay set for the point-to-point channel. Vary these parameters
//to see the variation in number of packets sent/received/dropped.

PointToPointHelper p2p1;
p2p1.SetDeviceAttribute ("DataRate", StringValue ("5Mbps"));
p2p1.SetChannelAttribute ("Delay", StringValue ("1ms"));

//Set the base address for the first network (nodes n0 and n1)

Ipv4AddressHelper address;
address.SetBase ("10.1.1.0", "255.255.255.0");
NetDeviceContainer devices;
```

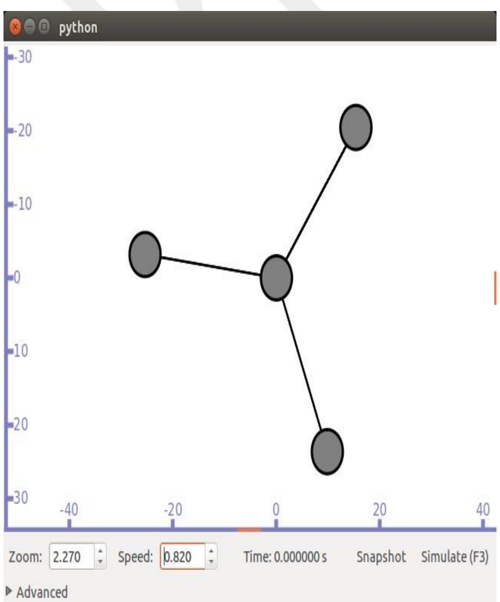
```
devices = p2p1.Install (nodes.Get (0), nodes.Get (2));
Ipv4InterfaceContainer interfaces = address.Assign (devices);

p2p1.SetDeviceAttribute ("DataRate", StringValue ("5Mbps"));
p2p1.SetChannelAttribute ("Delay", StringValue ("1ms"));

address.SetBase("10.1.2.0", "255.255.255.0");
devices = p2p1.Install (nodes.Get (2), nodes.Get (3));


interfaces = address.Assign (devices);

//UDP between n1 and n3
p2p1.SetDeviceAttribute ("DataRate", StringValue ("15Mbps"));
p2p1.SetChannelAttribute ("Delay", StringValue ("10ms"));

address.SetBase ("10.1.3.0", "255.255.255.0");
devices=p2p1.Install(nodes.Get(1),nodes.Get(2));


interfaces = address.Assign (devices);

p2p1.SetDeviceAttribute ("DataRate", StringValue ("15Mbps"));
p2p1.SetChannelAttribute ("Delay", StringValue ("10ms"));

devices = p2p1.Install (nodes.Get (2), nodes.Get (3));
address.SetBase("10.1.4.0", "255.255.255.0");

interfaces = address.Assign (devices);

  UdpEchoServerHelper echoServer (9);


ApplicationContainer serverApps = echoServer.Install (nodes.Get (3));
  serverApps.Start (Seconds (0.0));
  serverApps.Stop (Seconds (30.0));

  UdpEchoClientHelper echoClient (interfaces.GetAddress (1), 9);
  echoClient.SetAttribute ("MaxPackets", UintegerValue (1));
  echoClient.SetAttribute ("Interval", TimeValue (Seconds (2.0)));
  echoClient.SetAttribute ("PacketSize", UintegerValue (1024));
```

```
  ApplicationContainer clientApps = echoClient.Install (nodes.Get (1));
  clientApps.Start (Seconds (1.0));
  clientApps.Stop (Seconds (30.0));

//RateErrorModel allows us to introduce errors into a Channel at a given rate.
//Vary the error rate value to see the variation in number of packets dropped.

Ptr<RateErrorModel>em = CreateObject<RateErrorModel> ();
em ->SetAttribute ("ErrorRate", DoubleValue (0.00001));
devices.Get (1) ->SetAttribute ("ReceiveErrorModel", PointerValue (em));

//create routing table at all nodes
Ipv4GlobalRoutingHelper::PopulateRoutingTables ();
uint32_t payloadSize = 1448;

OnOffHelper onoff (socketType, Ipv4Address::GetAny ());

//Generate traffic by using On Off application.

onoff.SetAttribute ("OnTime",
StringValue("ns3::ConstantRandomVariable[Constant=1]"));
onoff.SetAttribute ("OffTime",
StringValue("ns3::ConstantRandomVariable[Constant=0]"));
onoff.SetAttribute ("PacketSize", UintegerValue (payloadSize));
onoff.SetAttribute ("DataRate", StringValue ("50Mbps")); //bit/s

uint16_t port = 7;
//Install receiver (for packet sink) on node 2
Address localAddress1 (InetSocketAddress (Ipv4Address::GetAny (), port));
PacketSinkHelper packetSinkHelper1 (socketType, localAddress1);
ApplicationContainer sinkApp1 = packetSinkHelper1.Install (nodes.Get (3));

sinkApp1.Start (Seconds (0.0));
sinkApp1.Stop (Seconds (10));

//Install sender app on node 0
ApplicationContainer apps;
AddressValue remoteAddress (InetSocketAddress (interfaces.GetAddress (1),
port));
onoff.SetAttribute ("Remote", remoteAddress);
```

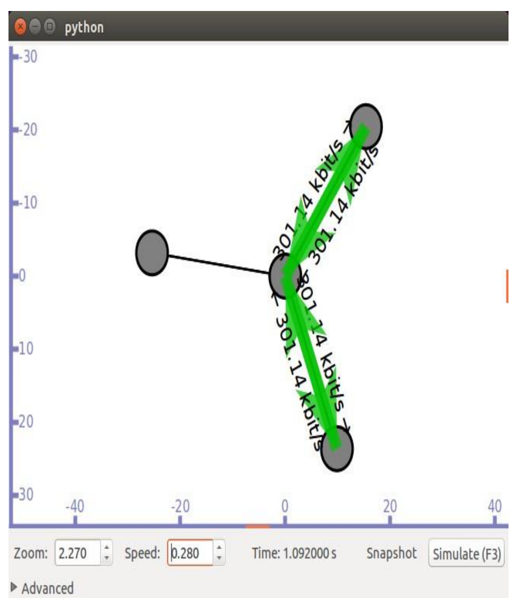
```
apps.Add (onoff.Install (nodes.Get (0)));

apps.Start (Seconds (4.0));
apps.Stop (Seconds (10));

Simulator::Stop (Seconds (30));

AsciiTraceHelper ascii;

p2p1.EnableAsciiAll (ascii.CreateFileStream ("lab1.tr"));
//Run the simulator

Simulator::Run ();
Simulator::Destroy ();
return 0;
}
```


<u>OUTPUT:-</u>

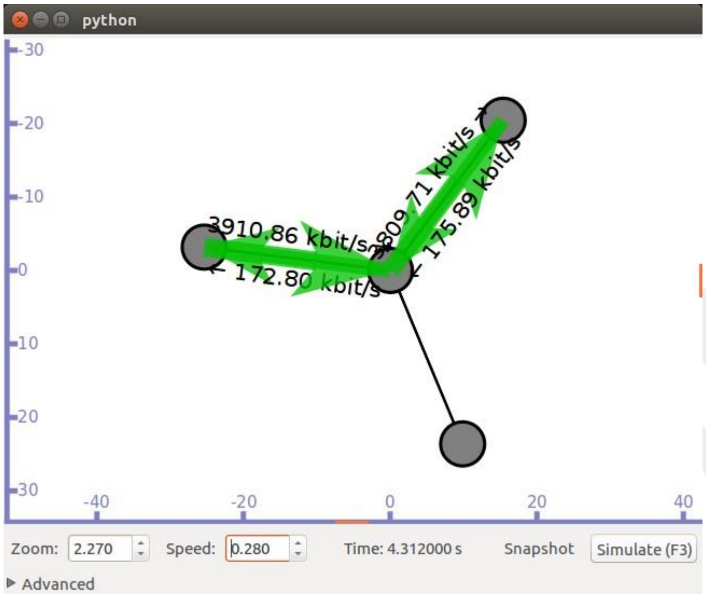
TCP and UDP network



UDP transmission between n3 to n2



TCP transmission between n0 to n2



Part B:

C Programs

The following experiments shall be conducted using 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

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's and 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 & 1's only): "); //enter the length of frame
        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) //check if the data is '1'
                 {
                         b[j]=a[i];
                        for(k=i+1;a[k]==1 && k<n && count<5;k++)
                        {
                                j++;
                                b[j]=a[k];
                                count++; //increment and count the number of 1's
                                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"); //append 01111110 at the begining of the data
        for(i=0;i<j;i++)
        printf("%d",b[i]);
        printf("01111110"); //append 01111110 at the end of the data
        //getch();
}
```

Result: -

```
nandini@nandini-HP-G62-Notebook-PC:~/Desktop/Cpgm$ ./a.out
Enter Frame length:12
Enter input frame (0's & 1's only):1
1
0
0
1
0
1
1
1
1
1
1
After stuffing the frame is:01111110110010111110101111110
```

(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"); //enter the character
	scanf("%s",&a);
	n=strlen(a);   //length of string
	b[0]='d'; // append dlestx at the begin of string
	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') //if entered character is dle, then output will be dledle//
		{
			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:

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.

Distance Vector Algorithm

1. A router transmits its distance vector to each of its neighbors in a routing packet.
2. Each router receives and saves the most recently received distance vector from each of its neighbors.
3. A router recalculates its distance vector when:
   - It receives a distance vector from a neighbor containing different information than before.
   - It discovers that a link to a neighbor has gone down.

The DV calculation is based on minimizing the cost to each destination

Dx(y) = Estimate of least cost from x to y
C(x,v) = Node x knows cost to each neighbor v
Dx = [Dx(y): y ∈ N] = Node x maintains distance vector
Node x also maintains its neighbors' distance vectors
– For each neighbor v, x maintains Dv = [Dv(y): y ∈ N]

Note –

- From time-to-time, each node sends its own distance vector estimate to neighbors.
- When a node x receives new DV estimate from any neighbor v, it saves v’s distance vector and it updates its own DV using B-F equation:
- Dx(y) = min { C(x,v) + Dv(y)} for each node y ∈ N

Program: -

```
/*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];

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++)	/*for loop construct matrix */
		{
```

```
                    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++)                    /*i-source node */
            for(j=0;j<nodes;j++)                /*j-destination node */
                    if(i!=j)
                    for(k=0;k<nodes;k++)        /*k-intermediate node*/
                        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];/*update cost*/
                        rt[i].from[j]=rt[i].from[k];        /*update route*/
                        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\n node%d via %d Distance %d ",j+1,rt[i].from[j]+1,rt[i].dist[j]);
        }
        printf("\n\n");
        }
```

<u>Output:</u>

Enter the number of nodes 3

Enter the cost matrix

0 3 5

3 0 2

5 2 0

State value for router 0 is

Via 0 distance 0

Via 1 distance 3

Via 2 Distance 5

State value for router 1 is

Via 0 distance 3

Via 1 distance 0

Via 2 Distance 2

State value for router 2 is

Via 0 distance 5

Via 1 distance 2

Via 2 Distance 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.

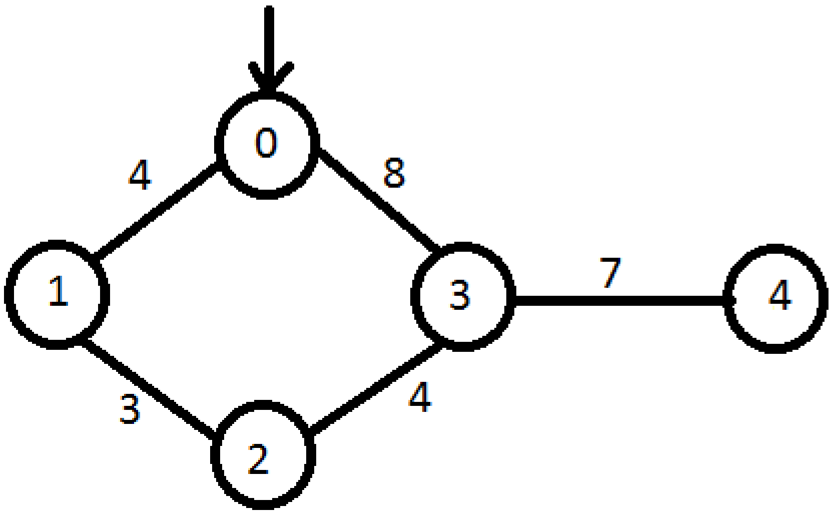
– Recalculate the shortest distance of remaining vertices from the source.

– 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

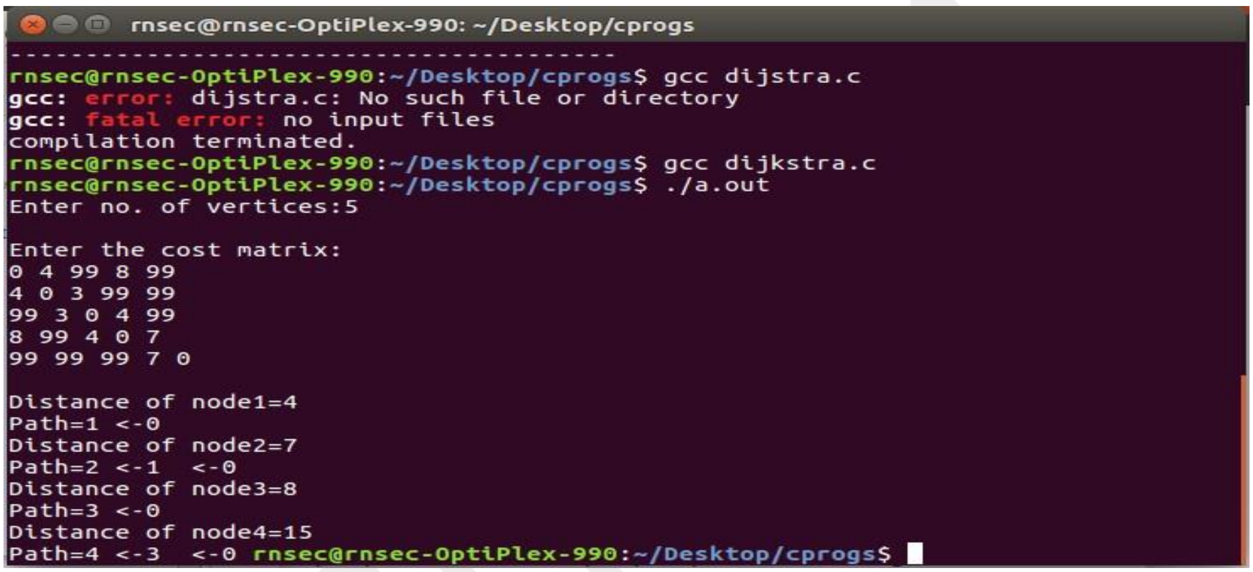
<u>Program: -</u>

```
#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') // if first bit is 1 do xor, othwise directly go for shifting
            xor();
        for(j=0;j<N-1;j++)
            cs[j]=cs[j+1]; //shifting
        cs[j]=t[i++];   //droping next bit for division
    }while(i<=a+N-1);

int main()
{
    printf("\nMessage to be send is: ");
    scanf("%s",t);
    printf("\n______________________________");
    printf("\nGeneratng polynomial : %s",g);
    a=strlen(t);
   // printf("a=%d",a); a=4
   //printf("N=%d",N); N=17
    for(i=a;i<a+N-1;i++)
        t[i]='0';
    printf("\n______________________________");
    printf("\nAfter appending zero's to message : %s",t);
    printf("\n______________________________");
    crc();
    printf("\nChecksum is : %s",cs);
    for(i=a;i<a+N-1;i++) //data+checksum
        t[i]=cs[i-a];
    printf("\n______________________________");
    printf("\nFinal codeword (message+checksum) to be transmitted is : %s",t);
```

```
    printf("\n______________________________");
    printf("\nEnter received message ");
    scanf("%s",t);
    crc();
    printf("\nRemainder : %s",cs);
    for(i=0;(i<N-1) && (cs[i]!='1');i++);  //if 1 is encounter than the for loop will terminate
        if(i<N-1)
            printf("\n\nError detected\n\n");
        else
            printf("\n\nNo error detected\n\n");
            printf("\n______________________________\n");
        return 0;
}
```

Output without Error

```
rnsec@rnsec-OptiPlex-990: ~/Desktop/cprogs
rnsec@rnsec-OptiPlex-990:~$ cd Desktop
rnsec@rnsec-OptiPlex-990:~/Desktop$ cd cprogs
rnsec@rnsec-OptiPlex-990:~/Desktop/cprogs$ gcc crc.c
rnsec@rnsec-OptiPlex-990:~/Desktop/cprogs$ ./a.out

Message to be transmitted is: 1101

----------------------------------------
Generatng polynomial : 10001000000100001
----------------------------------------
After appending zero's to message : 11010000000000000000
----------------------------------------
Checksum is : 1101000110101101
----------------------------------------
Final codeword (message+checksum) to be transmitted is : 11011101000110101101
----------------------------------------
Enter received message 11011101000110101101

Remainder : 0000000000000000
No error detected


----------------------------------------
rnsec@rnsec-OptiPlex-990:~/Desktop/cprogs$
```

Output with error

```
rnsec@rnsec-OptiPlex-990: ~/Desktop/cprogs

----------------------------------------
rnsec@rnsec-OptiPlex-990:~/Desktop/cprogs$ ./a.out

Message to be transmitted is: 1101

----------------------------------------
Generatng polynomial : 10001000000100001
----------------------------------------
After appending zero's to message : 11010000000000000000
----------------------------------------
Checksum is : 1101000110101101
----------------------------------------
Final codeword (message+checksum) to be transmitted is : 11011101000110101101
----------------------------------------
Enter received message 10111101000110101101

Remainder : 0110000011000110
Error detected


----------------------------------------
rnsec@rnsec-OptiPlex-990:~/Desktop/cprogs$
```

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.

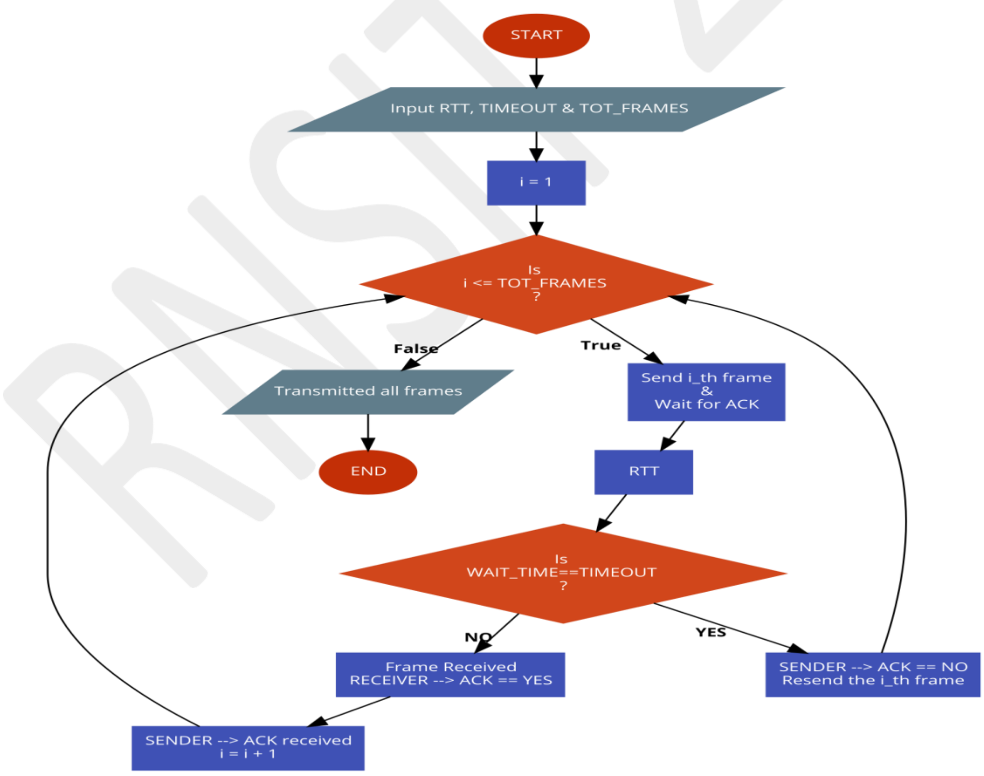
Design

Figure 11.8 illustrates the mechanism. Comparing this figure with Figure 11.6, we can see the traffic on the forward channel (from sender to receiver) and the reverse channel. At any time, there is either one data frame on the forward channel or one ACK frame on the reverse channel. We therefore need a half-duplex link.

Example 11.2

Figure 11.9 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.

Flow chart :



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;
}
```

Output

```
SENDER: Frame 1 sent, Waiting for ACK...

RECEIVER: Frame 1 received,ACK sent
----------------------------------------------
SENDER: ACK for Frame 1 Received.

SENDER: Frame 2 sent, Waiting for ACK...

RECEIVER: Frame 2 received,ACK sent
----------------------------------------------
SENDER: ACK for Frame 2 Received.

SENDER: Frame 3 sent, Waiting for ACK...
SENDER: ACK not received for Frame 3=>TIMEOUT Resending Frame...
SENDER: Frame 3 sent, Waiting for ACK...

RECEIVER: Frame 3 received,ACK sent
----------------------------------------------
SENDER: ACK for Frame 3 Received.

SENDER: Frame 4 sent, Waiting for ACK...
SENDER: ACK not received for Frame 4=>TIMEOUT Resending Frame...
SENDER: Frame 4 sent, Waiting for ACK...
SENDER: ACK not received for Frame 4=>TIMEOUT Resending Frame...
SENDER: Frame 4 sent, Waiting for ACK...

RECEIVER: Frame 4 received,ACK sent
----------------------------------------------
SENDER: ACK for Frame 4 Received.

SENDER: Frame 5 sent, Waiting for ACK...

RECEIVER: Frame 5 received,ACK sent
----------------------------------------------
SENDER: ACK for Frame 5 Received.

SENDER: Frame 6 sent, Waiting for ACK...

RECEIVER: Frame 6 received,ACK sent
----------------------------------------------
SENDER: ACK for Frame 6 Received.

SENDER: Frame 7 sent, Waiting for ACK...
SENDER: ACK not received for Frame 7=>TIMEOUT Resending Frame...
SENDER: Frame 7 sent, Waiting for ACK...
```

RECEIVER: Frame 7 received,ACK sent

(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 2^m - 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.

Figure 11.12 *Send window for Go-Back-NARQ*

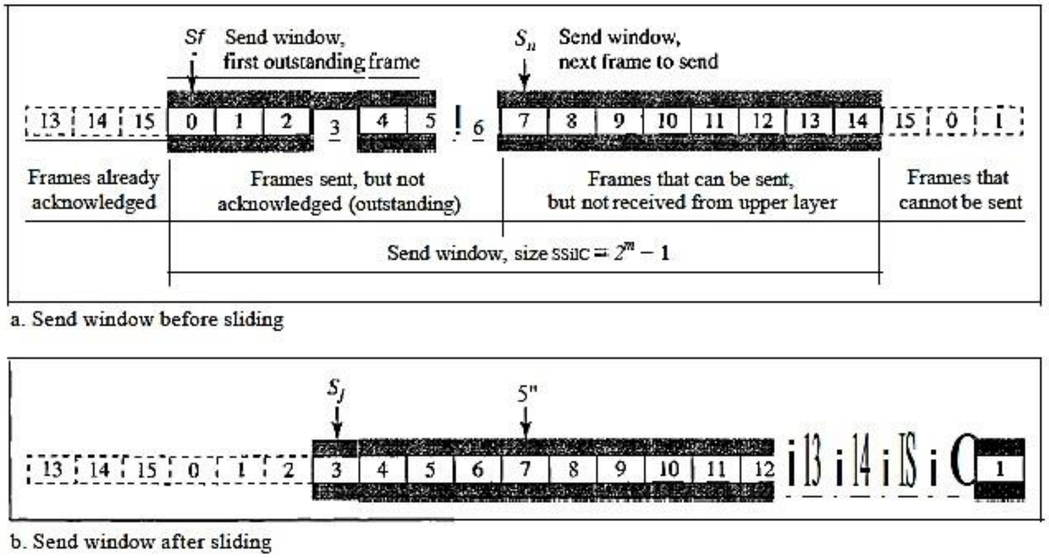


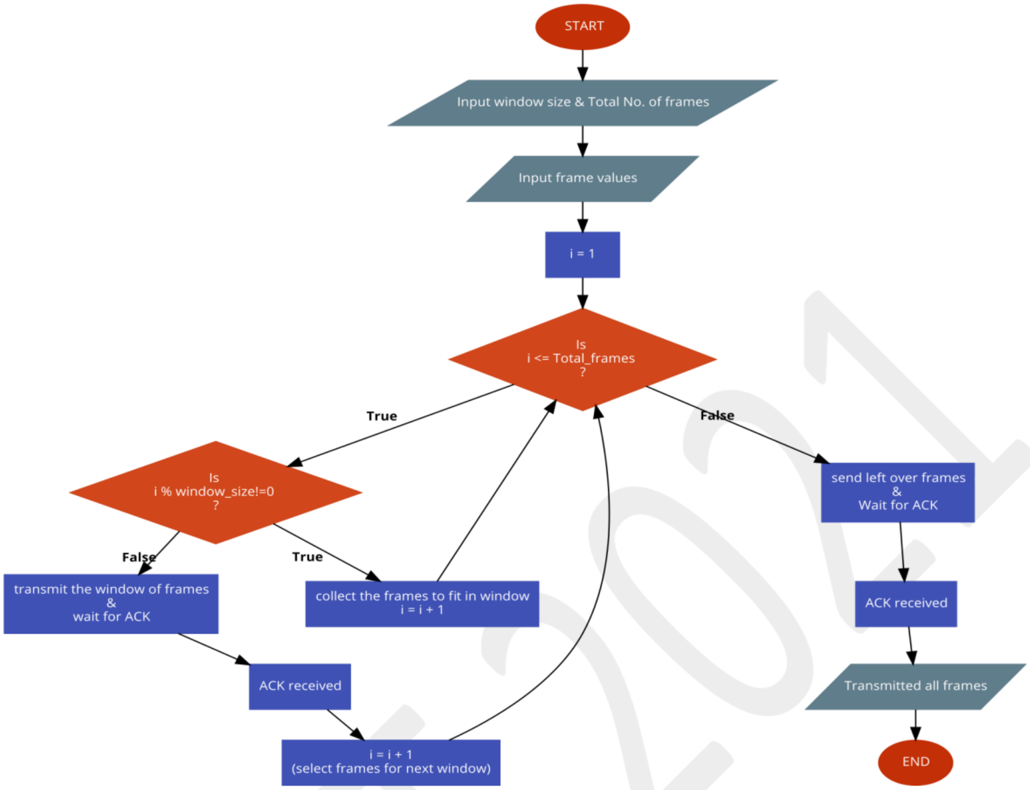
Figure 11.12 shows a sliding window of size 15 *(m* =4). The window at any time divides the possible sequence numbers into four regions.

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 11.12a, 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.

Flow chart:



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;
}
```

Output :

```
Enter window size: 2

Enter number of frames to transmit: 5

Enter 5 frames: 1       2       3       4       5

After sending 2 frames at each stage sender waits for ACK
Sending frames in the following manner....

  1 2
SENDER:waiting for ACK...

RECEIVER:Frames Received, ACK Sent
--------------------------------------------
SENDER:ACK received, sending next frames
  3 4
SENDER:waiting for ACK...

RECEIVER:Frames Received, ACK Sent
--------------------------------------------
SENDER:ACK received, sending next frames
  5
SENDER:waiting for ACK...

RECEIVER:Frames Received, ACK Sent
--------------------------------------------
SENDER:ACK received.
--------------------------------------------
```

Program 6:-

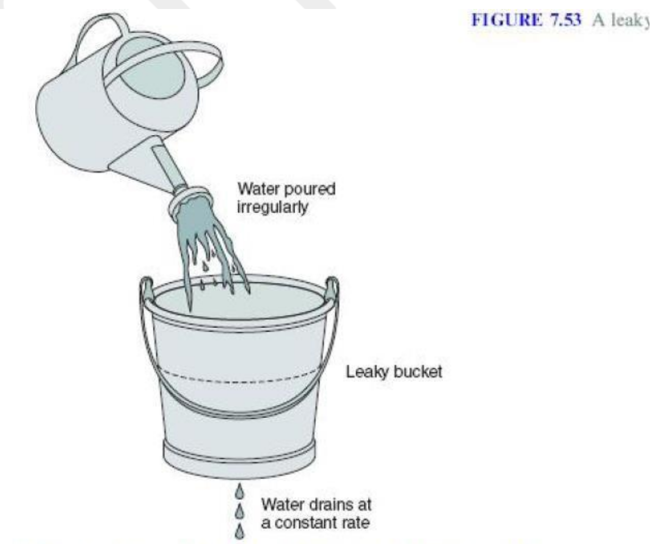
Aim: - Write a program for congestion control using leaky bucket algorithm in C language and execute the same and display the result.

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



Let X = bucket content at last conforming packet arrival
Let ta be last conforming packet arrival time = depletion in bucket

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.

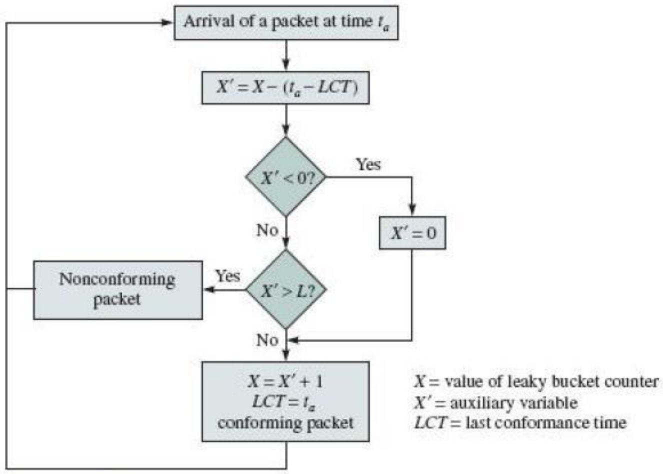
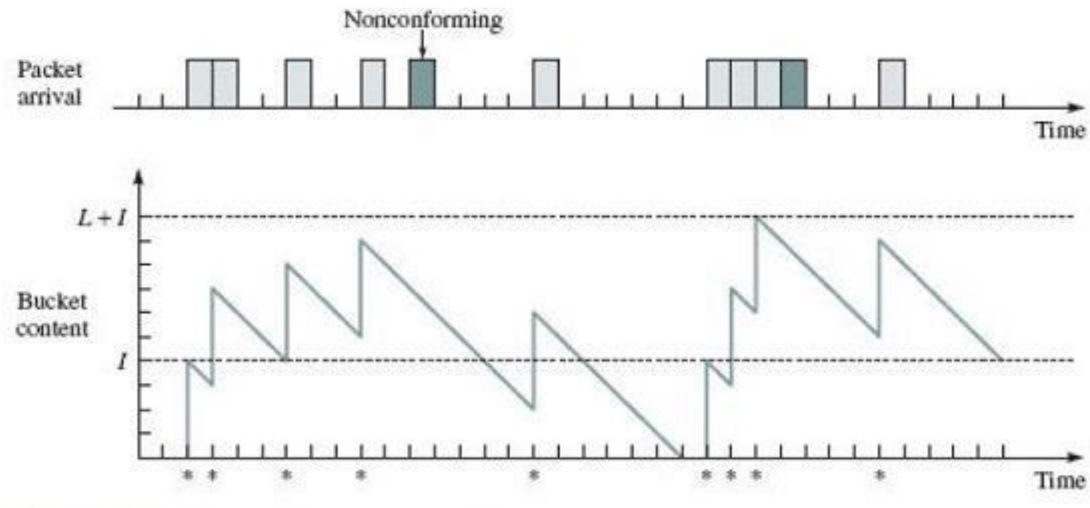


FIGURE 7.54 Leaky bucket algorithm used for policing

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 kth 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.

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.



Behavior of leaky bucket

Program:

```
    //leacky bucket program
#include<stdio.h>
#define bucketsize 1000
#define n 5 // user defined function to output the contents of bucket at a constant rate
void  bucketoutput(int *bucket,int op)
{
if(*bucket > 0 && *bucket > op)  // if the no. of bytes in thebucket >output rate
    {
        *bucket= *bucket-op; //no. of bytes in bucket – output rate
        printf("\n%d-outputed remaining is %d",op,*bucket);
    }
    else if(*bucket > 0) // if the bucket is not empty
```

```
	{
		printf("\n remaining data output = %d",*bucket);
		*bucket=0;
	}
}

int main()
{
	int op,newpack,oldpack=0,wt,i,j,bucket=0; // op – ouput rate, wt- waiting
time, bucket- no.of bytes in the bucket at any point of time
	printf("enter output rate"); // input the output rate
	scanf("%d",&op);
	for(i=1;i<=n;i++)
	{
		newpack=rand()%500; //new packet with random size is generated
		printf("\n\n new packet size = %d",newpack);
		newpack=oldpack+newpack;
		wt=rand()%5; // random waiting time is generated
		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);
                // calling output rate function with wait time
                for(j=0;j<wt;j++)
                {
                        bucketoutput(&bucket,op);
                        sleep(1);
                }
                oldpack=bucket;
        }
        while(bucket>0)
                bucketoutput(&bucket,op);
        return 0;
}
```

<u>Output</u>

```
enter output rate 500
new packet size = 383
the data in bucket = 383
the next packet will arrive after = 1 sec
remaining data output = 383
new packet size = 277
the data in bucket = 277
the next packet will arrive after = 0 sec
new packet size = 293
```

the data in bucket = 570

the next packet will arrive after = 0 sec

new packet size = 386

the data in bucket = 956

the next packet will arrive after = 2 sec

500-outputed remaining is 456

remaining data output = 456

new packet size = 149

the data in bucket = 149

the next packet will arrive after = 1 sec

remaining data output = 149