BET's
BASAVAKALYAN ENGINEERING COLLEGE,
BASAVAKALYAN

Department
Of
Computer Science & Engineering

NETWORK'S LABORATORY
(10CS L77)

VII Semester

Prepared by:-

Mr.Viresha Sugoor
Asst. Professor

SNote: Student is required to solve one problem from PART-A and one problem from PART-B. Both the parts have equal weightage.

PART A – Simulation Exercises

The following experiments shall be conducted using either NS228/OPNET or any other simulators.

1. **Simulate a three nodes point-to-point network with duplex links between them. Set the queue size vary the bandwidth and find the number of packets dropped.**
2. **Simulate a four node point-to-point network, and connect the links as follows: n0-n2, n1-n2 and n2-n3. Apply TCP agent between n0-n3 and UDP n1-n3. Apply relevant applications over TCP and UDP agents changing the parameter and determine the number of packets sent by TCP/UDP.**
3. **Simulate the transmission of ping messages over a network topology consisting of 6 nodes and find the number of packets dropped due to congestion.**
4. **Simulate an Ethernet LAN using N-nodes(6-10), change error rate and data rate and compare the throughput.**
5. **Simulate an Ethernet LAN using N nodes and set multiple traffic nodes and plot congestion window for different source/destination.**
6. **Simulate simple ESS and with transmitting nodes in wire-less LAN by simulation and determine the performance with respect to transmission of packets.**

PART B

The following experiments shall be conducted using C/C++.

1. **Write a program for error detecting code using CRC-CCITT (16-bits).**
2. **Write a program for distance vector algorithm to find suitable path for transmission.**
3. **Using TCP/IP sockets, write a client-server program to make client sending the file name and the server to send back the contents of the requested file if present.**
4. **Implement the above program using as message queues or FIFOs as IPC channels.**
5. **Write a program for simple RSA algorithm to encrypt and decrypt the data.**
6. **Write a program for congestion control using Leaky bucket algorithm.**

INTRODUCTION

Network simulation is an important tool in developing, testing and evaluating network protocols. Simulation can be used without the target physical hardware, making it economical and practical for almost any scale of network topology and setup. It is possible to simulate a link of any bandwidth and delay, even if such a link is currently impossible in the real world. With simulation, it is possible to set each simulated node to use any desired software. This means that meaning deploying software is not an issue. Results are also easier to obtain and analyze, because extracting information from important points in the simulated network is as done by simply parsing the generated trace files.

Simulation is only of use if the results are accurate, an inaccurate simulator is not useful at all. Most network simulators use abstractions of network protocols, rather than the real thing, making their results less convincing. S.Y. Wang reports that the simulator OPNET uses a simplified finite state machine to model complex TCP protocol processing. [19] NS-2 uses a model based on BSD TCP, it is implemented as a set of classes using inheritance. Neither uses protocol code that is used in real world networking.

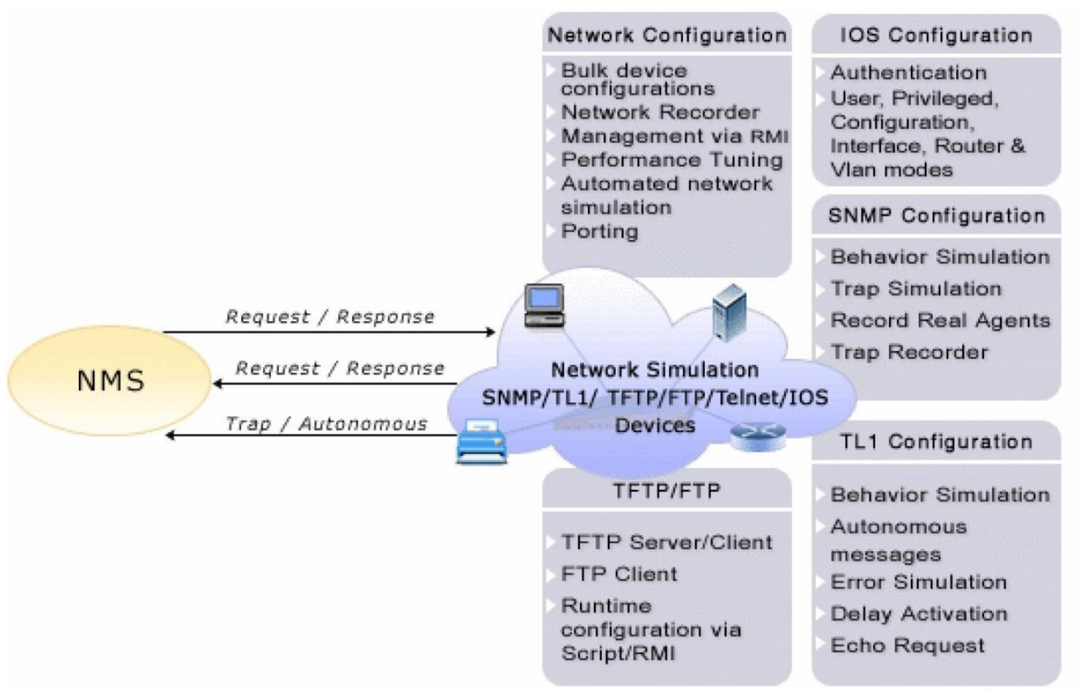
Improving Network Simulation:

Wang states that "Simulation results are not as convincing as those produced by real hardware and software equipment." This statement is followed by an explanation of the fact that most existing network simulators can only simulate real life network protocol implementations with limited detail, which can lead to incorrect results. Another paper includes a similar statement, "running the actual TCP code is preferred to running an abstract specification of the protocol." Brakmo and Peterson go on to discuss how the BSD implementations of TCP are quite important with respect to timers. Simulators often use more accurate round trip time measurements than those used in the BSD implementation, making results differ.

Using real world network stacks in a simulator should make results more accurate, but it is not clear how such stacks should be integrated with a simulator. The network simulator NCTUns shows how it is possible to use the network stack of the simulators machine.

Network Simulation Experience

The Network Simulator offers a simplified and complete network simulation experience. The following diagram depicts this functionality offered by the Network Simulator.



The Network Simulator can design and simulate a network with SNMP, TL1, TFTF, FTP, Telnet and IOS devices, in four simple steps:

1. Add devices to the Device tree: Add devices with the required configuration to the device tree in the Network Designer. Preconfigured devices are also bundled with the toolkit.

2. Create the Network: Create and add bulk devices to the network, at one shot.

3. Configure the Network devices: Configure the devices in the network, if required.

4. **Start the Network: Start the network or start individual agents in the network. The MIB Browser and TL1 Craft Interface test tools, can be used as the manager tools for testing.**

Network Emulation

Network emulation refers to actual network traffic passing through some software which might do some analysis or perhaps modify the traffic in some way. The Emulation Network in the WAND group is used for testing and evaluation of networking software and hardware. The scale is limited; it is made up 24 emulation machines and one central controlling computer. Setup of such a network is time consuming and expensive: in addition to the aforementioned 25 computers, a Cisco 2950 switch and a Cyclades 32 port terminal server are included in the network. Each emulation machine also has a 4 port network interface controller. The controlling machine includes special capture cards (known as DAG [6] cards) to allow easier capture and processing of network traffic. This network has no easy way of adding latency and bandwidth bottlenecks, which means creating adverse conditions on the network is difficult. It is possible to use Dummynet to

add latency, but this is a lot of work. There is a project to solve this issue; a non blocking crossbar Ethernet switch is being created for the network, but the cost involved is large. Other network emulation done in the WAND group include validating the WAND simulator. This was done by setting up a physical network with FreeBSD machines using Dummynet to add latency. Dummynet is one example of network emulation software, NIST Net is another, it claims to "allow controlled, reproducible experiments with network performance sensitive/adaptive applications and control protocols in a simple laboratory setting".

NS-2 also provides some network emulation functionality, it is able to capture packets from the live network and drop, delay, re-order, or duplicate them. Emulation, especially in the case of a network simulator like NS-2, is interesting because it is using real world data from real world network stacks. Emulation offers something simulation never can: it is performed on a real network, using actual equipment and real software. However, it is very limited compared to simulation in other areas; for example scale. The WAND Emulation Network described earlier requires a lot of setup and is expensive, yet only contains 24 emulation machines. There is no theoretical limit to the number of nodes a simulator can handle, and increasing the size of a simulation does not cost anything. The factors to consider are RAM, disk space and the small amount of time taken to change a simulation script. In general, changing the simulation is a simple step, though it would be complex in the case a huge amount of nodes being required (a million, for example).

Also, network emulation must of course be run in real time, where simulation can sometimes simulate large time periods in a small amount of time. In the case of a million nodes, the simulation might run in greater than real time because the hardware it is run on would limit performance.

Introduction to Network Simulators

Network simulators implemented in software are valuable tools for researchers to develop, test, and diagnose network protocols. Simulation is economical because it can carry out experiments without the actual hardware. It is flexible because it can, for example, simulate a link with any bandwidth and propagation delay. Simulation results are easier to analyze than experimental results because important information at critical points can be easily logged to help researchers diagnose network protocols.

Network simulators, however, have their limitations. A complete network simulator needs to simulate networking devices (e.g., hosts and routers) and application programs that generate network traffic. It also needs to provide network utility programs to configure, monitor, and gather statistics about a simulated network. Therefore, developing a complete network simulator is a large effort. Due to limited development resources, traditional network simulators usually have the following drawbacks:

• Simulation results are not as convincing as those produced by real hardware and software equipment. In order to constrain their complexity and development cost, most network simulators

usually can only simulate real-life network protocol implementations with limited details, and this may lead to incorrect results.

• These simulators are not extensible in the sense that they lack the standard UNIX POSIX application programming interface (API). As such, existing or to-be-developed real-life application programs cannot run normally to generate traffic for a simulated network. Instead, they must be rewritten to use the internal API provided by the simulator (if there is any) and be compiled with the simulator to form a single, big, and complex program.

To overcome these problems, Wang invented a kernel re-entering simulation methodology and used it to implement the Harvard network simulator. Later on, Wang further improved the methodology and used it to design and implement the NCTUns network simulator and emulator.

Different types of simulators

Some of the different types of simulators are as follows:-

- **MIT's NETSIM**
- **NIST**
- **CPSIM**
- **INSANE**
- **NEST**
- **REAL**
- **NS**
- **OPNET**
- **NCTUns**

A **brief explanation of some of the above simulators is as follows:-**

REAL

REAL (REalistic And Large) is a network simulator written at Cornell University by S. Keshav and based on a modified version of NEST 2.5.

Use

NEST is intended for studying the dynamic behavior of flow and congestion control schemes in packet-switched data networks (namely TCP/IP).

The package

REAL provides 30 modules written in C that emulate flow-control protocols such as TCP, and 5 scheduling disciplines such as FIFO, Fair Queuing, DEC congestion avoidance and Hierarchical Round Robin.

The description of the network topology, protocols workload and control parameters are transmitted to the server using a simple ASCII representation called NetLanguage where the network is modeled as a graph. This latest release now includes a GUI written in Java.

Implementation of the simulator

The NEST code has been rewritten to make it less general, cleaner and faster. REAL is still implemented as a client-server program. The code is freely available to anyone willing to modify it. Node functions implement computation at each node in the network whereas queue management and routing functions manage buffers in nodes and packet switching. Routing is static and is based on Dijkstras's shortest path algorithm. A node could be a source, a router or a sink. Source nodes implement TCP-like transport layer functionality. Routers implement the scheduling disciplines, while the sinks are universal receivers that only acknowledge packets.

Since NEST didn't not allow for timers, REAL sends out a timer packet from a source back to itself to return after some specified time, but timers cannot be reset using this method.

NCTUns

Introduction

NCTUns is open source, high quality, and supports many types of networks.The NCTUns is a high-fidelity and extensible network simulator and emulator capable of simulating various protocols used in both wired and wireless IP networks. Its core technology is based on the novel kernel re-entering methodology invented by Prof. S.Y. Wang [1, 2] when he was pursuing his Ph.D. degree at Harvard University. Due to this novel methodology, NCTUns provides many unique advantages that cannot be easily achieved by traditional network simulators such as ns-2 [3] and OPNET [4].

After obtaining his Ph.D. degree from Harvard University in September 1999, Prof. Wang returned to Taiwan and became an assistant professor in the Department of Computer Science and Information Engineering, National Chiao Tung University (NCTU), Taiwan, where he founded his “Network and System Laboratory.” Since that time, Prof. S.Y. Wang has been leading and working with his students to design and implement NCTUns (the NCTU Network Simulator) for more than five years.

Salient features of NCTUns

The NCTUns network simulator and emulator has many useful features listed as follows:

It can be used as an emulator. An external host in the real world can exchange packets (e.g., set up a TCP connection) with nodes (e.g., host, router, or mobile station) in a network simulated by NCTUns. Two external hosts in the real world can also exchange their packets via a network simulated by NCTUns. This feature is very useful as the function and performance of real-world devices can be tested under various simulated network conditions.

It directly uses the real-life Linux 's TCP/IP protocol stack to generate high-fidelity simulation results. By using a novel kernel re-entering simulation methodology, a real-life UNIX (e.g., Linux) kernel's protocol stack can be directly used to generate high-fidelity simulation results.

It can use any real-life existing or to-be-developed UNIX application program as a traffic generator program without any modification. Any real-life program can be run on a simulated network to generate network traffic. This enables a researcher to test the functionality and performance of a distributed application or system under various network conditions. Another important advantage of this feature is that application programs developed during simulation studies can be directly moved to and used on real-world UNIX machines after simulation studies are finished. This eliminates the time and effort required to port a simulation prototype to a real-world implementation if traditional network simulators are used.

It can use any real-life UNIX network configuration and monitoring tools. For example, the UNIX route, ifconfig, netstat, tcpdump, traceroute commands can be run on a simulated network to configure or monitor the simulated network.

In NCTUns, the setup and usage of a simulated network and application programs are exactly the same as those used in real-world IP networks. For example, each layer-3 interface has an IP address assigned to it and application programs directly use these IP addresses to communicate with each other. For this reason, any person who is familiar with real-world IP networks can easily learn and operate NCTUns in a few minutes. For the same reason, NCTUns can be used as an educational tool to teach students how to configure and operate a real-world network.

It can simulate fixed Internet, Wireless LANs, mobile ad hoc (sensor) networks, GPRS networks, and optical networks. A wired network is composed of fixed nodes and point-to-point links. Traditional circuit-switching optical networks and more advanced Optical Burst Switching (OBS) networks are also supported. A wireless networks is composed of IEEE 802.11 (b) mobile nodes and access points (both the ad-hoc mode and infra-structure mode are supported). GPRS cellular networks are also supported.

It can simulate various networking devices. For example, Ethernet hubs, switches, routers, hosts, IEEE 802.11 (b) wireless stations and access points, WAN (for purposely delaying/dropping/reordering packets), Wall (wireless signal obstacle), GPRS base station, GPRS phone, GPRS GGSN, GPRS SGSN, optical circuit switch, optical burst switch, QoS DiffServ interior and boundary routers, etc.

It can simulate various protocols. For example, IEEE 802.3 CSMA/CD MAC, IEEE 802.11 (b) CSMA/CA MAC, learning bridge protocol, spanning tree protocol, IP, Mobile IP, Diffserv (QoS), RIP, OSPF, UDP, TCP, RTP/RTCP/SDP, HTTP, FTP, Telnet, etc.

Its simulation speed is high. By combining the kernel re-entering methodology with the discrete-event simulation methodology, a simulation job can be finished quickly.

Its simulation results are repeatable. If the chosen random number seed for a simulation case is fixed, the simulation results of a case are the same across different simulation runs even though there are some other activities (e.g., disk I/O) occurring on the simulation machine.

It provides a highly integrated and professional GUI environment. This GUI can help a user (1) draw network topologies, (2) configure the protocol modules used inside a node, (3) specify the moving paths of mobile nodes, (4) plot network performance graphs, (5) playing back the animation of a logged packet transfer trace, etc. All these operations can be easily and intuitively done with the GUI.

Its simulation engine adopts an open-system architecture and is open source. By using a set of module APIs provided by the simulation engine, a protocol module writer can easily implement his (her) protocol and integrate it into the simulation engine. NCTUns uses a simple but effective syntax to describe the settings and configurations of a simulation job. These descriptions are generated by the GUI and stored in a suite of files. Normally the GUI will automatically transfer these files to the simulation engine for execution. However, if a researcher wants to try his (her) novel device or network configurations that the current GUI does not support, he (she) can totally bypass the GUI and generate the suite of description files by himself (herself) using any text editor (or script program). The non-GUI-generated suite of files can then be manually fed to the simulation engine for execution.

It supports remote and concurrent simulations. NCTUns adopts a distributed architecture. The GUI and simulation engine are separately implemented and use the client-server model to communicate. Therefore, a remote user using the GUI program can remotely submit his (her) simulation job to a server running the simulation engine. The server will run the submitted simulation job and later return the results back to the remote GUI program for analyzes. This scheme can easily support the cluster-computing model in which multiple simulation jobs are performed in parallel on different server machines. This can increase the total simulation throughput.

It supports more realistic wireless signal propagation models. In addition to providing the simple (transmission range = 250 m, interference range = 550 m) model that is commonly used in the ns-2, NCTUns provides a more realistic model in which a received bit's BER is calculated based on the used modulation scheme, the bit's received power level, and the noise power level around the receiver. Large-scale path loss and small-scale fading effects are also simulated.

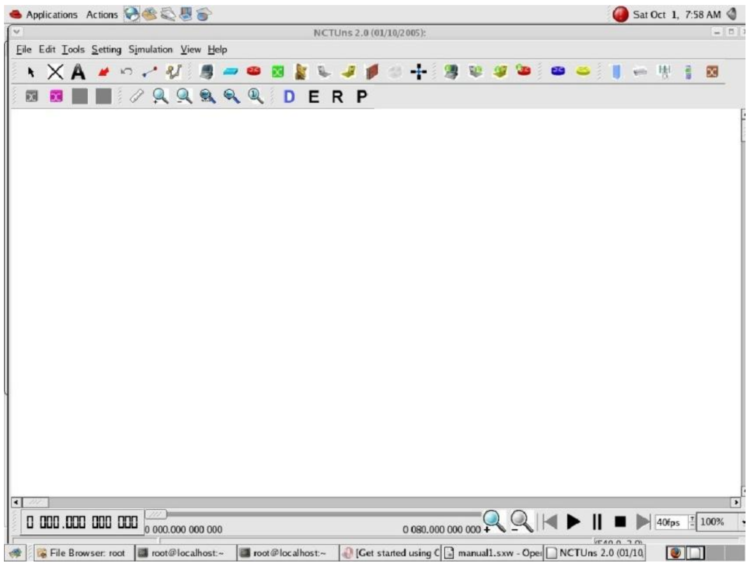
GETTING STARTED

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. **Set up environment variables:**
   Before the user can run up the dispatcher, coordinator, or NCTUns GUI program he/she must set up the NCTUNSHOME environment variable.
2. **Start up the dispatcher on terminal 1.**
3. **Start up the coordinator on terminal 2.**
4. **Start up the nctunsclient on terminal 3.**

After the above steps are followed, the starting screen of NCTUns disappears and the user is presented with the working window as shown below:



Drawing a Network Topology

To draw a new network topology, a user can perform the following steps:

Choose Menu->File->Operating Mode-> and make sure that the "Draw Topology" mode is checked. This is the default mode of NCTUns when it is launched. It is only in this mode that a user can draw a new network topology or change an existing simulation topology. When a user

switches the mode to the next mode " Edit Property", the simulation network topology can no longer be changed.

1. **Move the cursor to the toolbar.**

2. **Left-Click the router icon on the toolbar.**

3. **Left-Click anywhere in the blank working area to add a router to the current network topology. In the same way we can add switch, hub, WLAN access point, WLAN mobile node, wall (wireless signal obstacle) etc.**

4. **Left-Click the host icon on the toolbar. Like in step 4, add the required number of hosts to the current topology.**

5. **To add links between the hosts and the router, left-click the link icon on the toolbar to select it.**

6. **Left-Click a host and hold the mouse button. Drag this link to the router and then release the mouse left button on top of the router. Now a link between the selected host and the router has been created.**

7. **Add the other, required number of links in the same way. This completes the creation of a simple network topology.**

8. **Save this network topology by choosing Menu->File->Save. It is saved with a .tpl extension.**
9. **Take the snapshot of the above topology.**

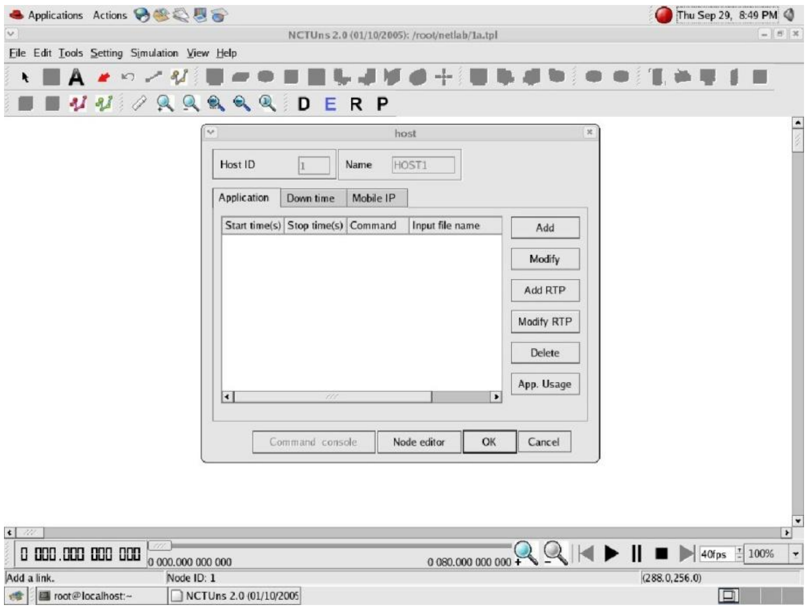
Editing Node's Properties

1. **A network node (device) may have many parameters to set. For example, we may have to set the maximum bandwidth, maximum queue size etc to be used in a network interface. For another example, we may want to specify that some application programs (traffic generators) should be run on some hosts or routers to generate network traffic.**

2. **Before a user can start editing the properties of a node, he/she should switch the mode from the "Draw Topology" to "Edit Property" mode. In this mode, topology changes can no longer be made. That is, a user cannot add or delete nodes or links at this time.**

3. **The GUI automatically finds subnets in a network and generates and assigns IP and MAC addresses to layer 3 network interfaces.**

4. **A user should be aware that if he/she switches the mode back to the "Draw Topology" mode when he/she again switches the mode back to the "Edit Topology" mode, node's IP and MAC addresses will be regenerated and assigned to layer 3 interfaces. Therefore the application programs now may use wrong IP addresses to communicate with their partners.**



Running the Simulation

When a user finishes editing the properties of network nodes and specifying application programs to be executed during a simulation, he/she can start running the simulation.

2. In order to do so, the user must switch mode explicitly from "Edit Property" to "Run Simulation". Entering this mode indicates that no more changes can (should) be made to the simulation case, which is reasonable. This simulation is about to be started at this moment; of course, any of its settings should be fixed.

3. Whenever the mode is switched to the "Run Simulation" mode, the many simulation files that collectively describe the simulation case will be exported. 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 " main File Name.sim" directory, where main Filename is the name of the simulation case chosen in the "Draw Topology" mode.

Playing Back the Packet Animation Trace

After the simulation is finished, the simulation server will send back the simulation result files to the GUI program after receiving these files, the GUI program will store these files in the "results directory" .It will then automatically switch to "play back mode".

1. These files include a packet animation trace file and all performance log files that the user specifies to generate. Outputting these performance log files can be specified by checking some output options in some protocol modules in the node editor. In addition to this, application programs can generate their own data files.

3. The packet animation trace file can be replayed later by the packet animation player. The performance curve of these log files can be plotted by the performance monitor.

Post Analysis

1. When the user wants to review the simulation results of a simulation case that has been finished before, he /she can run up the GUI program again and then open the case's topology file

2. The user can switch the mode directly to the "Play Back" mode. The GUI program will then automatically reload the results (including the packet animation trace file and performance log file.

3. After the loading process is finished, the user can use the control buttons located at the bottom of the screen to view the animation.

Simulation Commands

The following explains the meaning of each job control command:

- **Run: Start to run the simulation.**
- **Pause: Pause the currently -running simulation.**
- **Continue: Continue the simulation that was just paused.**
- **Stop: Stop the currently -running simulation**
- **Abort: Abort the currently running simulation. The difference between "stop" and "abort" is that a stopped simulation job's partial results will be transferred back to GUI files.**
- **Reconnect: The Reconnect command can be executed to reconnect to a simulation job that was previously disconnected. All disconnected jobs that have not finished their simulations or have finished their simulations but the results have not been retrieved back to be a GUI program by the user will appear in a session table next to the "Reconnect" command. When executing the reconnect command, a user can choose a disconnected job to reconnect from this session table.**
- **Disconnect: Disconnect the GUI from the currently running simulation job. The GUI now can be used to service another simulation job. A disconnected simulation will be given a session name and stored in a session table.**

Part A

EXPERIMENT 1

Simulate a three-node point-to-point network with a duplex link between them. Set the queue size and vary the bandwidth and find the number of packets dropped.

STEPS:

Step1: Select the hub icon on the toolbar and drag it onto the working window.

Step2: Select the host icon on the toolbar and drag it onto the working window. Repeat this for another host icon.

Step3: Select the link icon on the toolbar and drag it on the screen from host (node 1) to the hub and again from host(node 2) to the hub. Here the hub acts as node 3 in the point-to-point network. This leads to the creation of the 3-node point-to-point network topology. Save this topology as a .tpl file.

Step4:Double-click on host(node 1), a host dialog box will open up. 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 for switches and routers and half duplex for hubs, and in log Statistics, select Number of Drop Packets, Number of Collisions, Throughput of incoming packets and Throughput of outgoing packets. Select FIFO and set the queue size to 50 and press OK. Then click on Add. Another dialog box pops up. Click on the Command box and type the Command according to the following syntax:

```
stg [-t duration(sec)] [-p port number]HostIPaddr
```

and click OK.

Step 5: Double-click on host (node 2), and follow the same step as above with only change in command according to the following syntax:

```
rtg [-t] [-w log]   [-p port number]
```

and click OK.

Step 6: Double click on the link between node 1 and the hub to set the bandwidth to some initial value say, 10 Mbps. Repeat the same for the other node.

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

Step 8: Now go to Menu->Simulation->Run. Executing this command will submit he 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.

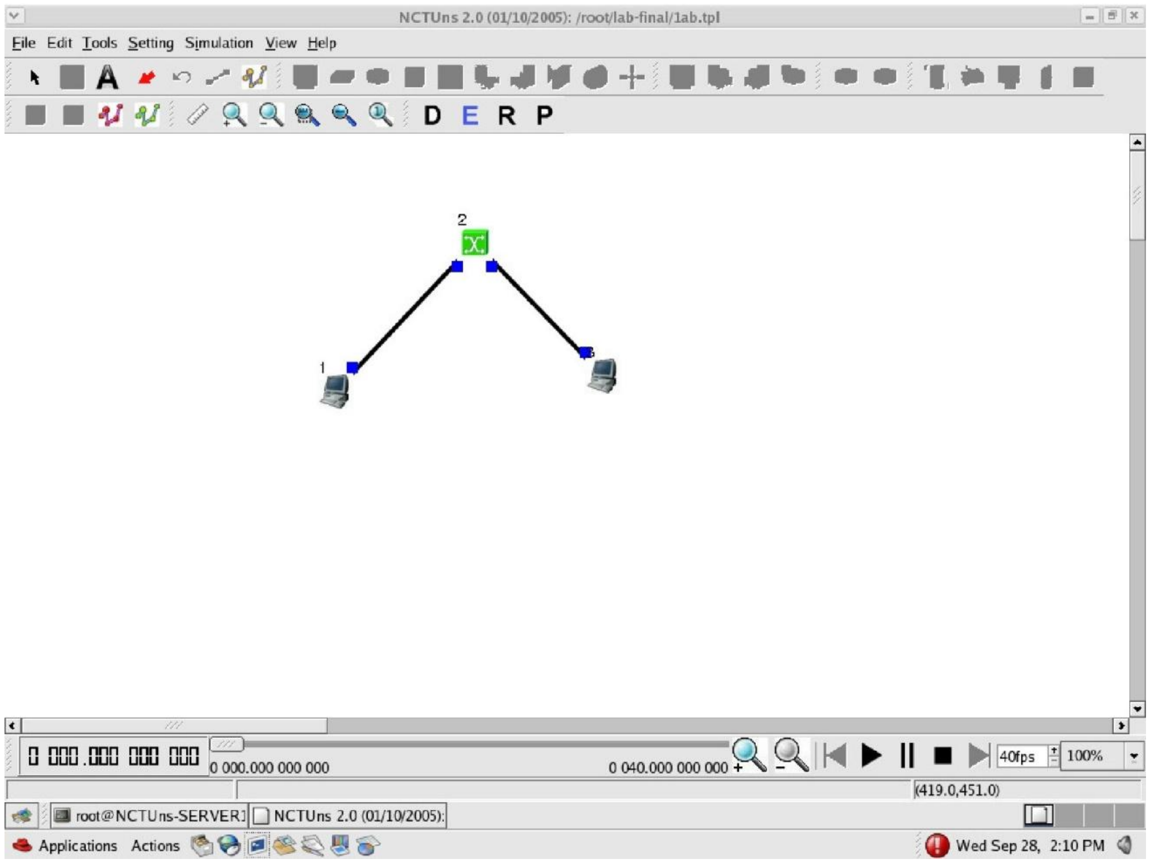
Step 9:To start the playback, the user can left-click the start icon(|>) of the time bar located at the bottom. The animation player will then start playing the recorded packet animation.

Step 10: Change the bandwidth say, 9 Mbps, and run the simulation and compare the two results.

Step 12: To view the results, go to the filename. results folder.

Note: To get the syntax of any command, double click on the host icon. Host dialog boxes appear and then choose App. Usage.

The screenshot below explain the topology.



EXPERIMENT 2

Simulate a four-node point-to-point network and connect the link as follows: Apply a TCP agent between n0 to n3 and apply a UDP agent between n1 and n3. Apply relevant applications over TCP and UDP agents changing the parameters and determine the number of packets sent by two agents.

STEPS:

Step 1: Create the topology as specified in the question, in the draw mode of the simulator.

Step 2: Go to edit mode and save the topology.

Step 3: Setup a TCP connection between node 1 and node 3 using the following commands:

```
stcp [-p port] [-l writesize] hostIPaddr
rtcp [-p port] [-l readsize]
```

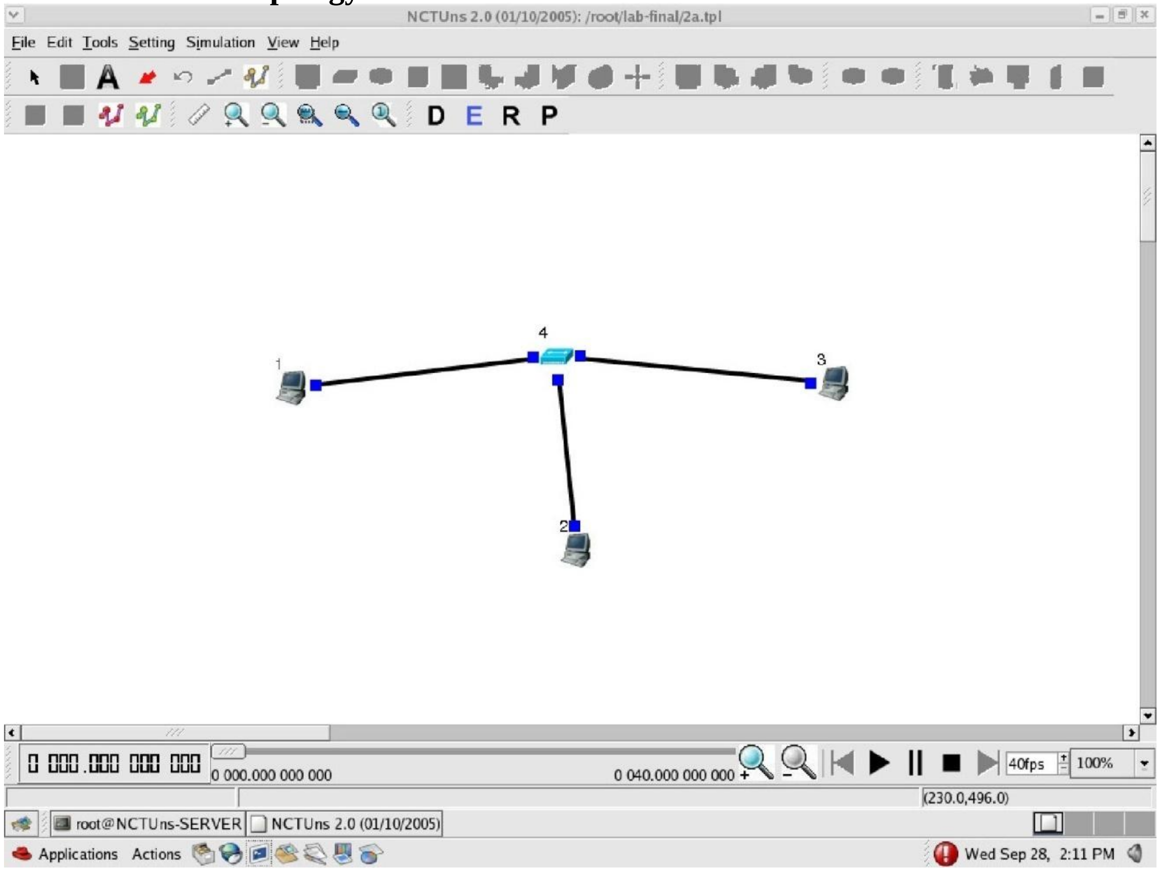
Step 4: Setup a UDP connection between node 2 and node 3 using the following commands:

```
stg [-u payload size duration] [Host address]
rtg [-u] [-w log]
```

Step 5:Set the output throughput log to determine the number of packets sent by TCP/UDP as described in experiment 1.

Step 6:To view the results, go to the filename.results folder.

The screenshot of the topology is shown below:



EXPERIMENT 3

Simulate the transmission of ping messages over a network topology consisting of 6 nodes and find the number of packets dropped due to congestion.

STEPS:

Step 1: Click on the subnet icon on the toolbar and then click on the screen of the working window.

Step 2: Select the required number of hosts and a suitable radius between the host and the switch.

Step 3: In the edit mode, get the IP address of one of the hosts say, host 1 and then for the other host say, host2 set the drop packet and no: of collisions statistics as described in the earlier experiments.

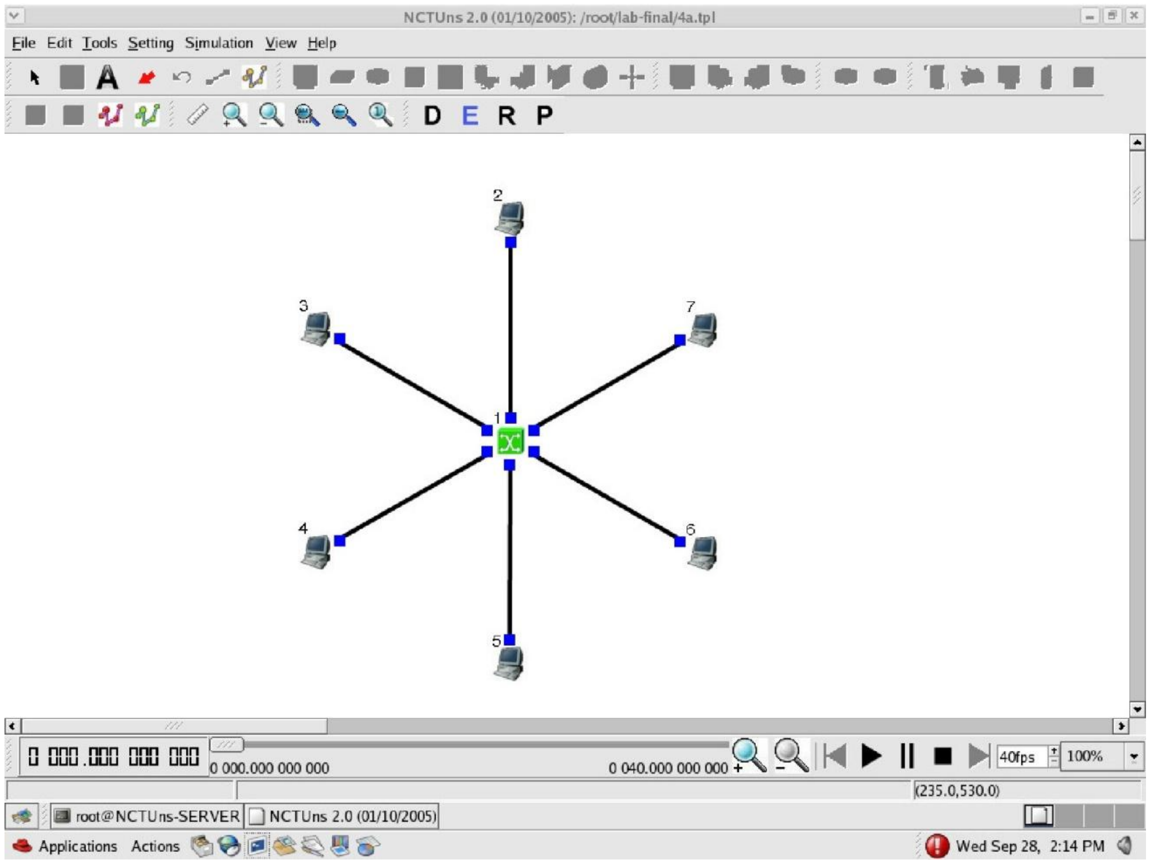
Step 4: Now run the simulation.

Step 5: Now click on any one of the hosts and click on command console and ping the destination node.

ping IP Address of the host

Note: The no: of drop packets are obtained only when the traffic is more in the network. For checking the no of packets dropped press ctrl+C

The screenshot of the topology is shown below:



EXPERIMENT 4

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

STEPS:

Step 1: Connect one set of hosts with a hub and another set of hosts also through a hub and connect these two hubs through a switch. This forms an Ethernet LAN.

Step 2: Setup a TCP connection between a host on one hub and host on another hub using the following command:

```
stcp [-p port] [-l writesize] hostIPaddr
rtcp [-p port] [-l readsize]
```

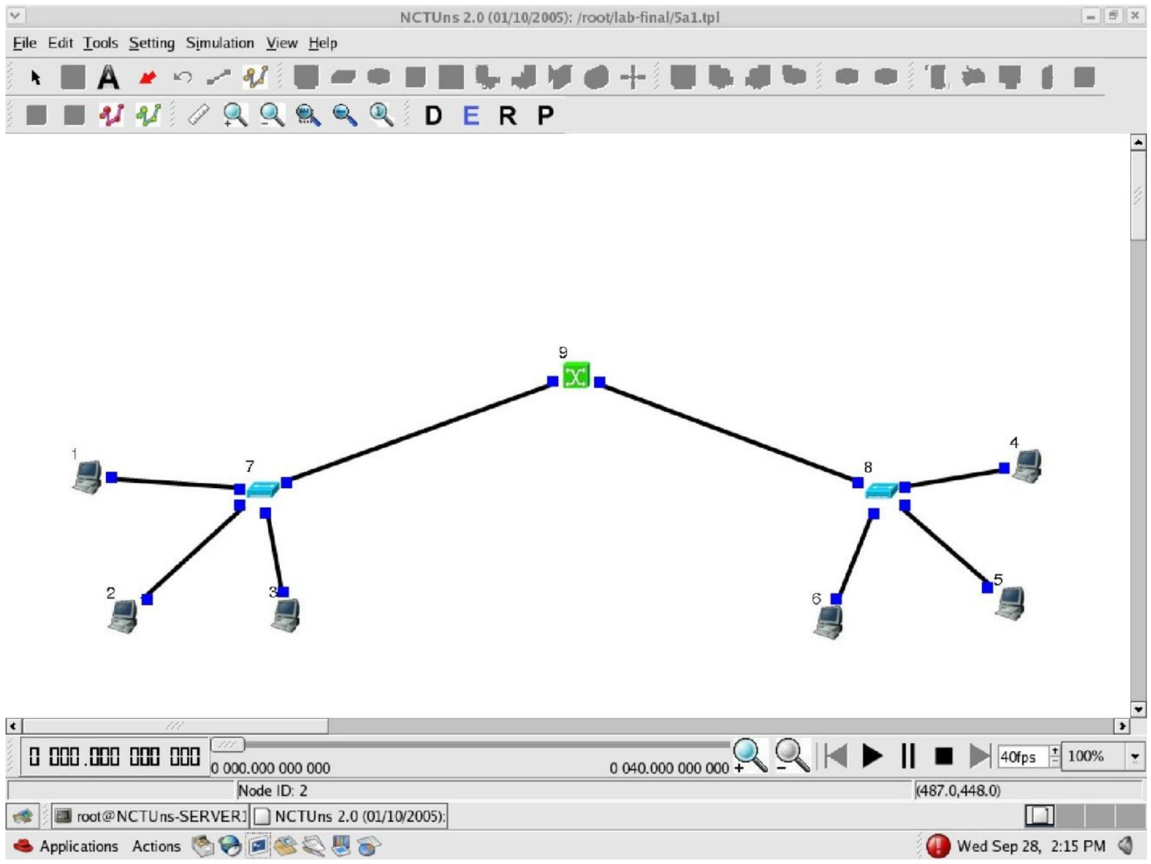
Step 3: Setup the error rate, data rate in the physical layer, input and output throughput in the mac layer as described above.

Step 4: Change error rate and data rate and compare the

throughputs. Step 5: View the results in the filename.results.

Computer Networks Lab

The screenshot of the topology is shown below:



EXPERIMENT 5

Simulate an Ethernet LAN using N nodes and set multiple traffic nodes and plot congestion window for different source/destination.

STEPS:

Step 1: Connect one set of hosts with a hub and another set of hosts also through a hub and connect these two hubs through a switch. This forms an Ethernet LAN.

Step 2: Setup multiple traffic connections between the hosts on one hub and hosts on another hub using the following command:

```
stcp [-p port] [-l writesize] hostIPaddr
rtcp [-p port] [-l readsize]
```

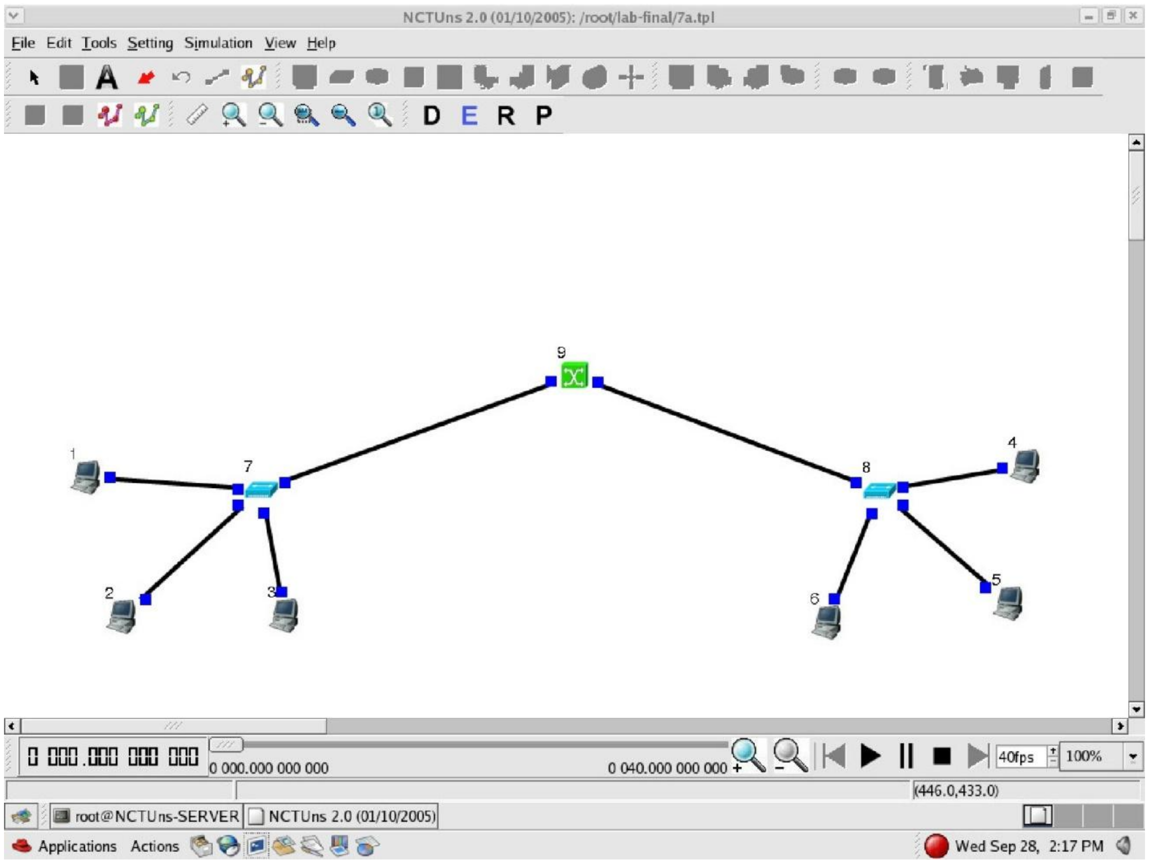
Step 3: Setup the collision log at the destination hosts in the MAC layer as described in the earlier experiments.

Step 4: To plot the congestion window go to Menu->Tools->Plot Graph->File->open->filename.results->filename.coll.log

Step 5: View the results in the filename.results.

Computer Networks Lab

The screenshot of the topology is shown below:



EXPERIMENT 6

Simulate simple ESS and with transmitting nodes in wireless LAN by simulation and determine the performance with respect to transmission of packets.

STEPS:

Step 1: Connect a host and two WLAN access points to a router.

Step 2: Setup multiple mobile nodes around the two WLAN access points and set the path for each mobile node.

Step 3: Setup a ttcp connection between the mobile nodes and host using the following command:

Mobile Host 1

```
ttcp –t –u –s –p 3000 IPAddrOf Receiver
```

Mobile Host 1

```
ttcp –t –u –s –p 4000 IPAddrOf Receiver
```

Host(Receiver)

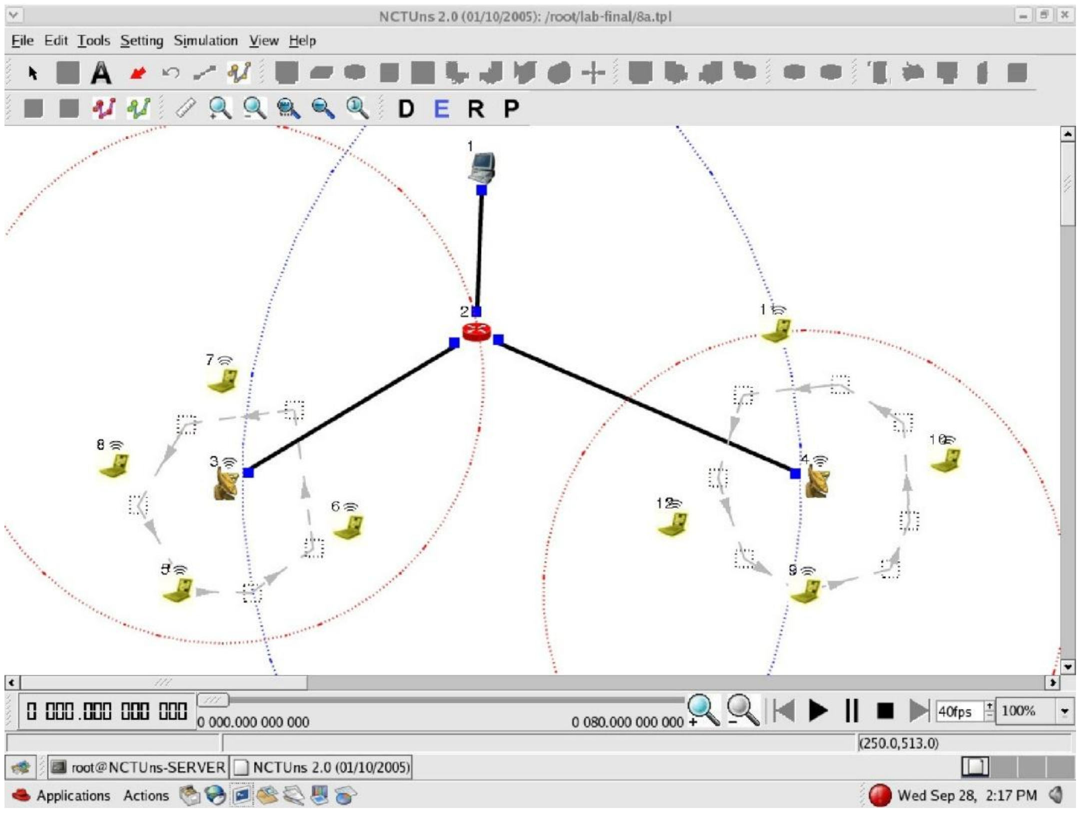
```
ttcp –r –u –s –p 3000
ttcp –r –u –s –p 4000
```

Step 4: Setup the input throughput log at the destination host.

Step 5: To set the transmission range go to Menu->Settings->WLAN mobile node->Show transmission range.

Step 5: View the results in the filename. results.

Screenshot


NCTUns 2.0 (01/10/2005): /root/lab-final/8a.tpl
File Edit Tools Setting Simulation View Help
D E R P
1
2
3
4
5
6
7
8
9
10
11
12
0 000.000 000 000
0 080.000 000 000
40fps
100%
(250.0,513.0)
root@NCTUns-SERVER
NCTUns 2.0 (01/10/2005)
Applications Actions
Wed Sep 28, 2:17 PM

Part B Programs

Experiment No 1
CRC

Problem Statement
Write a program for error detecting code using CRC-CCITT (16-bits).

Theory
It does error checking via polynomial division. In general, a bit string

$$b_{n-1}\ b_{n-2}\ b_{n-3}\ \ldots b_2\ b_1\ b_0$$

As

$$b_{n-1}X^{n-1} + b_{n-2}X^{n-2} + b_{n-3}X^{n-3} + \ldots b_2X^2 + b_1X^1 + b_0$$

Ex: -

10010101110

As

$$X^{10} + X^7 + X^5 + X^3 + X^2 + X^1$$

All computations are done in modulo 2

Algorithm:-
1. **Given a bit string, append 0^S to the end of it (the number of 0^s is the same as the degree of the generator polynomial) let B(x) be the polynomial corresponding to B.**
2. **Divide B(x) by some agreed on polynomial G(x) (generator polynomial) and determine the remainder R(x). This division is to be done using Modulo 2 Division.**
3. **Define T(x) = B(x) –R(x)**

   (T(x)/G(x) => remainder 0)

4. **Transmit T, the bit string corresponding to T(x).**
5. **Let T' represent the bit stream the receiver gets and T'(x) the associated polynomial. The receiver divides T^1(x) by G(x). If there is a 0 remainder, the receiver concludes T = T' and no error occurred otherwise, the receiver concludes an error occurred and requires a retransmission.**

Program

```
#include<stdio.h>
#include<string.h>
#include<conio.h>
#define N strlen(g)

char t[128], cs[128], g[]="10001000000100001";
int a, e, c;

void xor() {
  for(c=1;c<N;c++) cs[c]=((cs[c]==g[c])?'0':'1');
}

void crc() {
  for(e=0;e<N;e++) cs[e]=t[e];
  do {
    if(cs[0]=='1') xor();
    for(c=0;c<N-1;c++) cs[c]=cs[c+1];
    cs[c]=t[e++];
  }while(e<=a+N-1);
}

void main() {
  clrscr();
  printf("\nEnter poly : "); scanf("%s",t);
  printf("\nGenerating Polynomial is : %s",g);
  a=strlen(t);
  for(e=a;e<a+N-1;e++) t[e]='0';
  printf("\nModified t[u] is :  %s",t);
  crc();
  printf("\nChecksum is : %s",cs);
  for(e=a;e<a+N-1;e++) t[e]=cs[e-a];
  printf("\nFinal Codeword is : %s",t);
  printf("\nTest Error detection 0(yes) 1(no) ? : ");
  scanf("%d",&e);
  if(e==0) {
    printf("Enter position where error is to inserted : ");
    scanf("%d",&e);
    t[e]=(t[e]=='0')?'1':'0';
    printf("Errorneous data   : %s\n",t);
  }
  crc();
  for (e=0;(e<N-1)&&(cs[e]!='1');e++);
  if(e<N-1) printf("Error detected.");
  else printf("No Error Detected.");
  getch();
}
```

<u>Output</u>

```
Enter poly : 1011101
Generating Polynomial is : 10001000000100001
Modified t[u] is :  10111010000000000000000
Checksum is : 1000101101011000
Final Codeword is : 10111011000101101011000
Test Error detection 0(yes) 1(no) ? : 0
Enter position where you want to insert error : 3
Errorneous data   : 10101011000101101011000
Error detected.

Enter poly : 1011101
Generating Polynomial is : 10001000000100001
Modified t[u] is :  10111010000000000000000
Checksum is : 1000101101011000
Final Codeword is : 10111011000101101011000
Test Error detection 0(yes) 1(no) ? : 1
No Error Detected.
```

Experiment No 2

Distance Vector Routing

Problem Statement

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 nonadaptive. Nonadaptive 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 ■*T* 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.

The distance vector routing algorithm is sometimes called by other names, including the distributed Bellman-Ford routing algorithm and the Ford-Fulkerson algorithm, after the researchers who developed it (Bellman, 1957; and Ford and Fulkerson, 1962). It was the original ARPANET routing algorithm and was also used in the Internet under the RIP and in early versions of DECnet and Novell's IPX. AppleTalk and Cisco routers use improved distance vector protocols.

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 out going 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.

The Count to Infinity Problem.

Distance vector routing algorithm reacts rapidly to good news, but leisurely to bad news. Consider a router whose best route to destination *X* is large. If on the next exchange neighbor *A* suddenly reports a

Computer Networks Lab

short delay to *X*, the router just switches over to using the line to *A* to send traffic to *X*. In one vector exchange, the good news is processed.

To see how fast good news propagates, consider the five node (linear) subnet of following figure, where the delay metric is the number of hops. Suppose A is down initially and all the other routers know this. In other words, they have all recorded the delay to A as infinity.

A	B	C	D	E	
	∞	∞	∞	∞	Initially
	1	∞	∞	∞	After 1 exchange
	1	2	∞	∞	After 2 exchange
	1	2	3	∞	After 3 exchange
	1	2	3	4	After 4 exchange

A	B	C	D	E	
	1	2	3	4	Initially
	3	2	3	4	After 1 exchange
	3	3	3	4	After 2 exchange
	5	3	5	4	After 3 exchange
	5	6	5	6	After 4 exchange
	7	6	7	6	After 5 exchange
	7	8	7	8	After 6 exchange
		:			
	∞	∞	∞	∞	

Many ad hoc solutions to the count to infinity problem have been proposed in the literature, each one more complicated and less useful than the one before it. The split horizon algorithm works the same way as distance vector routing, except that the distance to *X* is not reported on line that packets for *X* are sent on (actually, it is reported as infinity). In the initial state of right figure, for example, *C* tells *D* the truth about distance to *A* but *C* tells *B* that its distance to *A* is infinite. Similarly, *D* tells the truth to *E* but lies to *C*.

Program

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

#define FSize 3

typedef struct packet{int SeqNum; char Data[FSize+1];}packet;

struct packet *readdata, *transdata;

int divide(char *msg) {
  int msglen, NoOfPacket, i, j;
  msglen = strlen(msg);
  NoOfPacket = msglen/FSize;
  if ((msglen%FSize)!=0) NoOfPacket++;
  readdata = (struct packet *)malloc(sizeof(packet) * NoOfPacket);
  for(i = 0; i < NoOfPacket; i++) {
     readdata[i].SeqNum = i + 1;
     for (j = 0; (j < FSize) && (*msg != '\0'); j++, msg++)
         readdata[i].Data[j] = *msg;
     readdata[i].Data[j] = '\0';
  }
  printf("\nThe Message has been divided as follows\n");
  printf("\nPacket No.         Data\n\n");
  for (i = 0; i < NoOfPacket; i++)
    printf("   %2d               %s\n", readdata[i].SeqNum,
           readdata[i].Data);
  return NoOfPacket;
}

void shuffle(int NoOfPacket) {
  int *Status;
  int i, j, trans;
  randomize();
  Status=(int * )calloc(NoOfPacket, sizeof(int));
  transdata = (struct packet *)malloc(sizeof(packet) * NoOfPacket);
  for (i = 0; i < NoOfPacket;) {
     trans = rand()%NoOfPacket;
     if (Status[trans]!=1) {
         transdata[i].SeqNum = readdata[trans].SeqNum;
         strcpy(transdata[i].Data, readdata[trans].Data);
         i++;     Status[trans] = 1;
     }
  }
  free(Status);
}

void sortframes(int NoOfPacket) {
  packet temp;
  int i, j;
  for (i = 0; i < NoOfPacket; i++)
    for (j = 0; j < NoOfPacket - i-1; j++)
       if (transdata[j].SeqNum > transdata[j + 1].SeqNum) {
           temp.SeqNum = transdata[j].SeqNum;
           strcpy(temp.Data, transdata[j].Data);
           transdata[j].SeqNum = transdata[j + 1].SeqNum;
           strcpy(transdata[j].Data, transdata[j + 1].Data);
           transdata[j + 1].SeqNum = temp.SeqNum;
           strcpy(transdata[j + 1].Data, temp.Data);
       }

}

void receive(int NoOfPacket) {
  int i;
  printf("\nPackets received in the following order\n");
  for (i = 0; i < NoOfPacket; i++) printf("%4d", transdata[i].SeqNum);
  sortframes(NoOfPacket);
  printf("\n\nPackets in order after sorting..\n");
  for (i = 0; i < NoOfPacket; i++) printf("%4d", transdata[i].SeqNum);
  printf("\n\nMessage received is :\n");
  for (i = 0; i < NoOfPacket; i++) printf("%s", transdata[i].Data);
}

void main() {
  char *msg;
  int NoOfPacket;
  clrscr();
  printf("\nEnter The message to be Transmitted :\n");
  scanf("%[^\n]", msg);
  NoOfPacket = divide(msg);
  shuffle(NoOfPacket);
  receive(NoOfPacket);
  free(readdata);
  free(transdata);
  getch();
}
```

Output

```
Enter The messgae to be Transmitted :
hi, it was nice meeting u on sunday

The Message has been divided as follows
Packet No.         Data

    1              hi,
    2               it
    3               wa
    4              s n
    5              ice
    6               me
    7              eti
    8              ng
    9              u o
   10              n s
   11              und
   12              ay

Packets received in the following order
   4   2   6   3   5   1   8   9  11   7  12  10

Packets in order after sorting..
   1   2   3   4   5   6   7   8   9  10  11  12

Message received is :
hi, it was nice meeting u on sunday
```

Experiment No 3

TCP Socket

Problem Statement

Using TCP/IP sockets, write a client-server program to make client sending the file name and the server to send back the contents of the requested file if present.

Overview

Unix sockets is just like two way FIFO's. All data communication will take place through the socket's interface, instead of through the file interface. Although unix socket's are a special file in the file system(just like FIFO's), there's usage of socket(), bind(), recv(),etc and not open(), read().

When programming with socket's, usually there's creation of server and client programs. The server will sit listening for incoming connections from clients and handling. This is similar to the situation that exists with internet sockets but with fine differences.

For instance, when describing which unix socket that has to be used (i.e the path to the special file that is the socket). The structure "struct sockaddr_un" has the following fields:

```
struct sockaddr_un{
                    unsigned short  sa_family;      // Address family,AF_XXXX
                    char            sa_data;        // 14 bytes of protocol address
                    };
```

This is the structure you will be passing to the bind() function, which associates a socket descriptor(a file descriptor) with a certain file(the name for which is in the sun_path field).

The structure "struct sockaddr_in" is used when we need IP address and Port number to be binded to the Sockets. It has following fields:

```
struct sockaddr_in {
                    short int            sin_family;   // Address family
                    unsigned short int   sin_port;     // Port number
                    struct in_addr       sin_addr;     // Internet address
                    unsigned char        sin_zero[8]   // Same size as struct sockaddr
                   };
        // Internet adress
         struct in_addr
                  {
                     unsigned long s_addr;          // 32 bits or 4 bytes long
                   };
```

ii. BACKGROUND REQUIRED:

1. **UNIX File I/O system calls**
2. **UNIX IPC system calls**

3. UNIX socket programming

iii. THEORETICAL CONCEPTS:

Most interprocess communication uses the client server model. These terms refer to the two processes which will be communication with each other. One of the two processes , the client , connects to the other process, the server, typiceally to make a request for information. A good analogy is a person who makes a phone call to another person.

Notice that the client needs to know of the existence of and the address of the server, but the server does not need to know the adresss of(or even the existence of) the client prior to the connection being established. Notice also that once a connection is established, both sides can send and receive information.

The system calls for establishing a connection are somewhat different for the client and the server, but both involve the basic construct of a socket. A socket is one end of an interprocess communication channel. The two processes each establish their own socket.

The steps involved in establishing a socket on the client side are as follows-

1. **Create a socket with the socket() system call .**
2. **Connect the socket to the address of the server using the connect() system call.**
3. **Send and receive data.There are a number of ways to do this, but the simplest is to use the read() and write() systen calls.**

The stepped involved in establishing a socket on the server side are as follows-

1. **Create a socket with the socket() system call.**
2. **Bind the socket to an address using the bind() system call. For a server socket on the internet, an address consists of a port number on the host machine.**
3. **Listen for connections with the listen() system call.**
4. **Accept a connection with the accept() system call. This call typically blocks until a client connects with the server.**
5. **Send and receive the data.**

Socket Types:

When a socket is created, the program has to specify the address domain and the socket type. Two processes can communicate with each other only if their sockets are of the same type and in the same domain, in which two processes running on any two hosts on the Internet communicate. Each of these has it's own adress format.

The address of a socket in the Unix domain is a character string which is basically an entry in the file system.

The address of a socket in the Internet domain consists of the Internet address of the host machine (every computer on the Internet has a unique 32 bit address, often reffered to as it's IP address). In addition , each socket needs a port number on that host. Port numbers are 16 bit unsigned integers. The lower numbers are reserved in Unix for standard services. For eg, the port number for the FTP server is 21.

There are two widely used socket types, stream sockets , and datgram sockets. Stream sockets treat communications as a continuous stream of characters, while datagram sockets have to read entire messages at once. Each uses it's own communications protocol. Stream sockets use

TCP , which is a reliable , stream oriented protocol, and datagram sockets use UDP, which is unreliable and message oriented.

The primary socket calls are as follows:-

1. **socket() - Create a new socket and return it's descriptor.**
2. **bind() - Associate a socket with a port and address .**
3. **Listen() -Establish a queue for connection requests.**
4. **Accept()- Accepts a connection request.**
5. **Connect()- Initiate a connection to a remote host.**
6. **Recv() - Receive data from a socket descriptor.**
7. **Send() - Send data to a socket descriptor.**
8. **Close() - “one-way” close of a socket descriptor,**

The other system calls used are as follows:-

1. **gethostbyname- given a hostname , returns a structure which specifies it's DNS name(s) and IP address(es).**
2. **getservbyname – given service name and protocol , returns a structure which specifies its name(s) and its port address.**

The socket utility functions are as follows:-

1. **htons/ntohl- convert short/long from host byte order to network byte order.**
2. **inet_ntoa/inet_addr- converts 32 bit IP address (network byte order to/from a dotted decimal string)**

<u>Algorithm (Client Side)</u>

1. **Start.**
2. **Create a socket using socket() system call.**
3. **Connect the socket to the address of the server using connect() system call.**
4. **Send the filename of required file using send() system call.**
5. **Read the contents of the file sent by server by recv() system call.**
6. **Stop.**

<u>Algorithm (Server Side)</u>

1. **Start.**
2. **Create a socket using socket() system call.**
3. **Bind the socket to an address using bind() system call.**
4. **Listen to the connection using listen() system call.**
5. **accept connection using accept()**
6. **Receive filename and transfer contents of file with client.**
7. **Stop.**

Experiment No 4

FIFO IPC

Problem Statement

Implement the above program using as message queues or FIFO as IPC channels.

Algorithm (Client Side)

1. **Start.**
2. **Open well known server FIFO in write mode.**
3. **Write the pathname of the file in this FIFO and send the request.**
4. **Open the client specified FIFO in read mode and wait for reply.**
5. **When the contents of the file are available in FIFO, display it on the terminal**
6. **Stop.**

Algorithm (Server Side)

1. **Start.**
2. **Create a well known FIFO using mkfifo()**
3. **Open FIFO in read only mode to accept request from clients.**
4. **When client opens the other end of FIFO in write only mode, then read the contents and store it in buffer.**
5. **Create another FIFO in write mode to send replies.**
6. **Open the requested file by the client and write the contents into the client specified FIFO and terminate the connection.**
7. **Stop.**

Program

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

#define MAX 10
int n;

class router {
    char adj_new[MAX],   adj_old[MAX];
    int  table_new[MAX], table_old[MAX];

  public:
    router(){
      for(int i=0;i<MAX;i++) table_old[i]=table_new[i]=99;
    }

    void copy(){
      for(int i=0;i<n;i++) {
        adj_old[i]  =adj_new[i];
        table_old[i]=table_new[i];
      }
    }

    int equal() {
      for(int i=0;i<n;i++)
        if(table_old[i]!=table_new[i]||adj_new[i]!=adj_old[i])return 0;
      return 1;
    }

    void input(int j) {
      cout<<"Enter 1 if the corresponding router is adjacent to router"
```

```
          <<(char)('A'+j)<<" else enter 99: "<<endl<<"             ";
        for(int i=0;i<n;i++)
          if(i!=j) cout<<(char)('A'+i)<<" ";
        cout<<"\nEnter matrix:";
        for(i=0;i<n;i++) {
          if(i==j)
            table_new[i]=0;
          else
            cin>>table_new[i];
          adj_new[i]= (char)('A'+i);
        }
        cout<<endl;
      }

      void display(){
        cout<<"\nDestination Router: ";
        for(int i=0;i<n;i++) cout<<(char)('A'+i)<<" ";
        cout<<"\nOutgoing Line:      ";
        for(i=0;i<n;i++) cout<<adj_new[i]<<" ";
        cout<<"\nHop Count:          ";
        for(i=0;i<n;i++) cout<<table_new[i]<<" ";
      }

      void build(int j) {
        for(int i=0;i<n;i++)
          for(int k=0;(i!=j)&&(k<n);k++)
            if(table_old[i]!=99)
              if((table_new[i]+r[i].table_new[k])<table_new[k]) {
                table_new[k]=table_new[i]+r[i].table_new[k];
                adj_new[k]=(char)('A'+i);
              }
      }
  } r[10];

  void build_table() {
    int i=0, j=0;
    while(i!=n) {
      for(i=j;i<n;i++) {
        r[i].copy();
        r[i].build(i);
      }
      for(i=0;i<n;i++)
        if(!r[i].equal()) {
          j=i;
          break;
        }
    }
  }

  void main() {
    clrscr();
    cout<<"Enter the number the routers(<"<<MAX<<"): "; cin>>n;
    for(int i=0;i<n;i++) r[i].input(i);
    build_table();
    for(i=0;i<n;i++) {
      cout<<"Router Table entries for router "<<(char)('A'+i)<<":-";
      r[i].display();
      cout<<endl<<endl;
    }
    getch();
  }
```

<u>Output</u>

```
Enter the number the routers: 5
Enter 1 if the corresponding is adjacent to router A else enter 99:
             B C D E
Enter matrix:1 1 99 99
Enter 1 if the corresponding is adjacent to router B else enter 99:
             A C D E
Enter matrix:1 99 99 99
Enter 1 if the corresponding is adjacent to router C else enter 99:
             A B D E
Enter matrix:1 99 1 1
Enter 1 if the corresponding is adjacent to router D else enter 99:
             A B C E
Enter matrix:99 99 1 99
Enter 1 if the corresponding is adjacent to router E else enter 99:
             A B C D
Enter matrix:99 99 1 99

Router Table entries for router A
Destination Router: A B C D E
Outgoing Line:      A B C C C
Hop Count:          0 1 1 2 2
Router Table entries for router B
Destination Router: A B C D E
Outgoing Line:      A B A A A
Hop Count:          1 0 2 3 3
Router Table entries for router C
Destination Router: A B C D E
Outgoing Line:      A A C D E
Hop Count:          1 2 0 1 1
Router Table entries for router D
Destination Router: A B C D E
Outgoing Line:      C C C D C
Hop Count:          2 3 1 0 2
Router Table entries for router E
Destination Router: A B C D E
Outgoing Line:      C C C C E
Hop Count:          2 3 1 2 0
```

Experiment No 5

RSA Algorithm

Problem Statement

Write a program for simple RSA algorithm to encrypt and decrypt the data.

Theory

Cryptography has a long and colorful history. The message to be encrypted, known as the plaintext, are transformed by a function that is parameterized by a key. The output of the encryption process, known as the ciphertext, is then transmitted, often by messenger or radio. The enemy, or intruder, hears and accurately copies down the complete ciphertext. However, unlike the intended recipient, he does not know the decryption key and so cannot decrypt the ciphertext easily. The art of breaking ciphers is called cryptanalysis the art of devising ciphers (cryptography) and breaking them (cryptanalysis) is collectively known as cryptology.

There are several ways of classifying cryptographic algorithms. They are generally categorized based on the number of keys that are employed for encryption and decryption, and further defined by their application and use. The three types of algorithms are as follows:

1. **Secret Key Cryptography (SKC): Uses a single key for both encryption and decryption. It is also known as symmetric cryptography.**
2. **Public Key Cryptography (PKC): Uses one key for encryption and another for decryption. It is also known as asymmetric cryptography.**
3. **Hash Functions: Uses a mathematical transformation to irreversibly "encrypt" information**

Public-key cryptography has been said to be the most significant new development in cryptography. Modern PKC was first described publicly by Stanford University professor Martin Hellman and graduate student Whitfield Diffie in 1976. Their paper described a two-key crypto system in which two parties could engage in a secure communication over a non-secure communications channel without having to share a secret key.

Generic PKC employs two keys that are mathematically related although knowledge of one key does not allow someone to easily determine the other key. One key is used to encrypt the plaintext and the other key is used to decrypt the ciphertext. The important point here is that it does not matter which key is applied first, but that both keys are required for the process to work. Because pair of keys is required, this approach is also called asymmetric cryptography.

In PKC, one of the keys is designated the public key and may be advertised as widely as the owner wants. The other key is designated the private key and is never revealed to another party. It is straight forward to send messages under this scheme.

The RSA algorithm is named after Ron Rivest, Adi Shamir and Len Adleman, who invented it in 1977. The RSA algorithm can be used for both public key encryption and digital signatures. Its security is based on the difficulty of factoring large integers.

Algorithm

1. **Generate two large random primes, P and Q, of approximately equal size.**
2. **Compute $N = P \times Q$**
3. **Compute $Z = (P-1) \times (Q-1)$.**
4. **Choose an integer E, $1 < E < Z$, such that GCD $(E, Z) = 1$**
5. **Compute the secret exponent D, $1 < D < Z$, such that $E \times D \equiv 1 \pmod{Z}$**

6. The public key is (*N*, *E*) and the private key is (*N*, *D*).

Note: The values of *P*, *Q*, and *Z* should also be kept secret.
The message is encrypted using public key and decrypted using private key.

An example of RSA encryption

1. **Select primes *P*=11, *Q*=3.**
2. **$N = P \times Q = 11 \times 3 = 33$**
   $Z = (P-1) \times (Q-1) = 10 \times 2 = 20$
3. **Lets choose *E*=3**
   Check GCD(*E*, *P*-1) = GCD(3, 10) = 1 (i.e. 3 and 10 have no common factors except 1), and check GCD(*E*, *Q*-1) = GCD(3, 2) = 1
   therefore GCD(*E*, *Z*) = GCD(3, 20) = 1
4. **Compute *D* such that $E \times D \equiv 1 \pmod{Z}$**
   compute $D = E^{\wedge -1} \bmod Z = 3^{\wedge -1} \bmod 20$
   find a value for *D* such that *Z* divides (($E \times D$)-1) find *D* such that 20 divides 3D-1.
   Simple testing (D = 1, 2, ...) gives D = 7
   Check: ($E \times D$)-1 = 3.7 - 1 = 20, which is divisible by *Z*.
5. **Public key = (*N*, *E*) = (33, 3)**
   Private key = (*N*, *D*) = (33, 7).

Now say we want to encrypt the message m = 7,
Cipher code = $M^{\wedge E} \bmod N$
$= 7^{\wedge 3} \bmod 33$
= 343 mod
33 = 13.
Hence the ciphertext c = 13.

To check decryption we compute Message' = $C^{\wedge D} \bmod N = 13^{\wedge 7} \bmod 33$
= 7.

Note that we don't have to calculate the full value of 13 to the power 7 here. We can make use of the fact that a = bc mod n = (b mod n).(c mod n) mod n so we can break down a potentially large number into its components and combine the results of easier, smaller calculations to calculate the final value.

Program:

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

int mult(unsigned int x, unsigned int y, unsigned int n) {
  unsigned long int k=1;
  int j;
  for (j=1; j<=y; j++) k = (k * x) % n;
  return (unsigned int) k;
}
```

```
void main () {
  char msg[100];
  unsigned int pt[100], ct[100], n, d, e, p, q, i;
  printf("Enter message : "); gets(msg);


  //strcpy(pt, msg);
  for(i=0;i<strlen(msg);i++)
   pt[i]=msg[i];
  n = 253; d = 17; e = 13;
  printf("\nCT = ");
  for(i=0; i<strlen(msg); i++) ct[i] = mult(pt[i], e,n);
  for(i=0; i<strlen(msg); i++) printf("%d ", ct[i]);
  printf("\nPT = ");
  for(i=0; i<strlen(msg); i++) printf("%c", pt[i]);
  for(i=0; i<strlen(msg); i++) pt[i] = mult(ct[i], d,n) ;
}
```

Output

```
Enter message : alpha
CT = 113 3 129 213 113
PT = alpha
```

Experiment No 6

Leaky Bucket

Problem Statement

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.

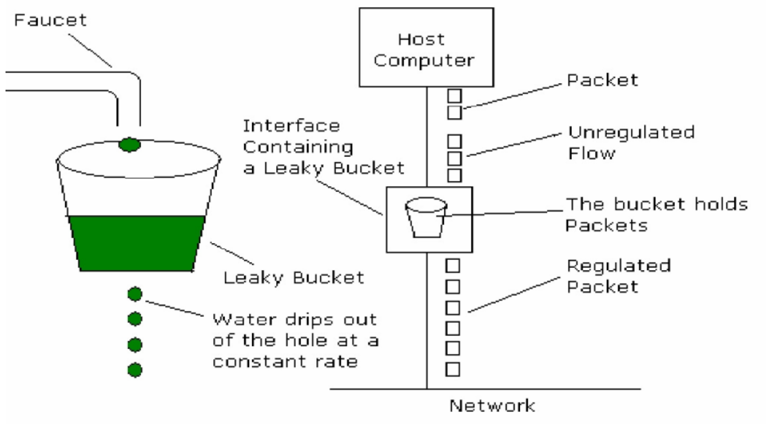
In contrast, closed loop solutions are based on the concept of a feedback loop if there is any congestion. Closed loop algorithms are also divided into two sub categories: explicit feedback and implicit feedback. In explicit feedback algorithms, packets are sent back from the point of congestion to warn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



Program:

```
#include<iostream.h>
#include<dos.h>
#include<stdlib.h>
#define bucketSize 512

void bktInput(int a,int b) {
  if(a>bucketSize)
    cout<<"\n\t\tBucket overflow";
  else {
    delay(500);
    while(a>b){
      cout<<"\n\t\t"<<b<<" bytes outputted.";
      a-=b;
      delay(500);
    }
    if (a>0) cout<<"\n\t\tLast "<<a<<" bytes sent\t";
    cout<<"\n\t\tBucket output successful";
  }
}

void main() {
  int op, pktSize;
  randomize();
  cout<<"Enter output rate : "; cin>>op;
  for(int i=1;i<=5;i++){
    delay(random(1000));
    pktSize=random(1000);
    cout<<"\nPacket no "<<i<<"\tPacket size = "<<pktSize;
    bktInput(pktSize,op);
  }
}
```

Output

```
Enter output rate : 100

Packet no 0  Packet size = 3
                Bucket output successful
                Last 3 bytes sent
Packet no 1  Packet size = 33
                Bucket output successful
                Last 33 bytes sent
Packet no 2  Packet size = 117
                Bucket output successful
                100 bytes outputted.
                Last 17 bytes sent
Packet no 3  Packet size = 95
                Bucket output successful
                Last 95 bytes sent
Packet no 4  Packet size = 949
                Bucket overflow
```

Viva Questions

1. **What are functions of different layers?**
2. **Differentiate between TCP/IP Layers and OSI Layers**
3. **Why header is required?**
4. **What is the use of adding header and trailer to frames?**
5. **What is encapsulation?**
6. **Why fragmentation requires?**
7. **What is MTU?**
8. **Which layer imposes MTU?**
9. **Differentiate between flow control and congestion control.**
10. **Differentiate between Point-to-Point Connection and End-to-End connections.**
11. **What are protocols running in different layers?**
12. **What is Protocol Stack?**
13. **Differentiate between TCP and UDP.**
14. **Differentiate between Connectionless and connection oriented connection.**
15. **Why frame sorting is required?**
16. **What is meant by subnet?**
17. **What is meant by Gateway?**
18. **What is an IP address?**
19. **What is MAC address?**
20. **Why IP address is required when we have MAC address?**
21. **What is meant by port?**
22. **What are ephemerical port number and well known port numbers?**
23. **What is a socket?**
24. **What are the parameters of socket()?**
25. **Describe bind(), listen(), accept(), connect(), send() and recv().**
26. **What are system calls? Mention few of them.**
27. **What is IPC? Name three techniques.**
28. **Explain mkfifo(), open(), close() with parameters.**
29. **What is meant by file descriptor?**
30. **What is meant by traffic shaping?**
31. **How do you classify congestion control algorithms?**
32. **Differentiate between Leaky bucket and Token bucket.**
33. **How do you implement Leaky bucket?**
34. **How do you generate busty traffic?**
35. **What is the polynomial used in CRC-CCITT?**
36. **What are the other error detection algorithms?**
37. **What is difference between CRC and Hamming code?**
38. **Why Hamming code is called 7,4 code?**
39. **What is odd parity and even parity?**
40. **What is meant by syndrome?**
41. **What is generator matrix?**
42. **What are Routing algorithms?**
43. **How do you classify routing algorithms? Give examples for each.**
44. **What are drawbacks in distance vector algorithm?**
45. **How routers update distances to each of its neighbor?**
46. **How do you overcome count to infinity problem?**
47. **What is cryptography?**
48. **How do you classify cryptographic algorithms?**
49. **What is public key?**
50. **What is private key?**
51. **What are key cipher text and plaintext?**

52. **What is simulation?**
53. **What are advantages of simulation?**
54. **Differentiate between Simulation and Emulation.**
55. **What is meant by router?**
56. **What is meant by bridge?**
57. **What is meant by switch?**
58. **What is meant by hub?**
59. **Differentiate between route, bridge, switch and hub.**
60. **What is ping and telnet?**
61. **What is FTP?**
62. **What is BER?**
63. **What is meant by congestion window?**
64. **What is BSS?**
65. **What is incoming throughput and outgoing throughput?**
66. **What is collision?**
67. **How do you generate multiple traffics across different sender-receiver pairs?**
68. **How do you setup Ethernet LAN?**
69. **What is meant by mobile host?**
70. **What is meant by NCTUns?**
71. **What are dispatcher, coordinator and nctunsclient?**
72. **Name few other Network simulators**
73. **Differentiate between logical and physical address.**
74. **Which address gets affected if a system moves from one place to another place?**
75. **What is ICMP? What are uses of ICMP? Name few.**
76. **Which layer implements security for data?**
77. **What is connectionless and connection oriented protocol?**
78. **What is Contention?**
79. **What is Congestion window?**
80. **What is flow control?**