Implementation Of Network Trainer

ABHISHEK MISHRA (071248)

HIMANSHU ARORA (071276)

ANIRUDH SINGH CHAUHAN (071322)

KISHEN CHAND UPADHYAY (071440)

Under the Supervision of

Brig (Retd.) S.P. Ghrera

May – 2011

Submitted in partial fulfillment of the Degree of

BACHELOR OF TECHNOLOGY

(CSE & / IT)

JAYPEE UNIVERSITY OF INFORMATION TECHNOLOGY

WAKNAGHAT

SOLAN , HIMACHAL PRADESH, INDIA

2011

ACKNOWLEDGEMENT

No venture can be completed without the blessing of Almighty.We consider it our bounded duty to bow to Almighty whose kind blessings always inspire us on the right path of life This project is our combined effort and realizes the potential of team and gives us a chance to work in co-ordination.

Science has caused many frontiers so has human efforts towards human research.Our revered guide Brig (Retd.) S.P. Ghrera, H.O.D., Department of Computer Science and IT, JUIT, has indeed acted as a light house showing us the need of sustained effort in the field of Networking to learn more and more.So we take this opportunity to thank him, for lending us stimulating suggestions, innovative quality guidance and creative thinking. He provides us the kind of strategies required for the completion of a task.We are grateful to him for the support , he provided us in doing thinks at our pace and for being patient for our mistakes.

SUMMARY

The project entitled as **IMPLEMENTATION OF NETWORK TRAINER** is a system developed as a pedantic tool for checking the efficiency and throughput of a network .This system is especially adopted in laboratories to make the students understand the intricacies behind the implementation of a network system. There are various parameters that define the successful and secured transmission of data over a network. The network trainer incorporates them all and generates a result against these parameters.

The network trainer works on diverse transmission protocols including ALOHA, CSMA, CSMA/CD among others, with each protocol having its unique characteristics. The systems are connected through varied topologies like BUS, RING, STAR and TOKEN-RING. These topologies determine how a particular set of systems would share data between them. We have been successful in implementing the transfer of data through a few protocols for a number of topologies and have enlisted the results accordingly.

The basic idea behind this project is to provide a good method for calculating the efficiency of transmission of data between systems and generating results that signify whether the transmission was successful or not.

TABLE OF CONTENTS

CERTIFICATE.....2

ACNOWLEDGEMENT.....3

SUMMARY.....4

LIST OF FIGURES. 7

LIST OF ABBREVIATIONS.....9

CHAPTER 1.....10

Introduction.....10

1.1 Scope.....10

1.2 Problem Statement.....10

1.3 Software Architecture.....11

1.4 Frame Format.....12

1.5 Address Field Format.....13

1.6 Transmit/Receive Mode.....13

CHAPTER 2.....15

Experiments.....15

2.1 Data Link Layer.....15

2.1.1 DLL basics.....15

2.1.2 Packet Transmission.....16

2.1.3 Stop and Wait Protocol.....19

2.1.4 Sliding Window Protocol.....23

2.2 Medium Access Control.....26

2.2.1 MAC basics.....26

2.2.2 Aloha.....26

2.2.3 Carrier Sense Multiple Access(CSMA).....29

2.2.4 CSMA with collision detection 31

2.2.5 Token Passing Bus.....33

2.2.6 Token Passing Ring.....34

CHAPTER 3……..……..……..……37

Explanation of the Software for LAN Trainer ……..……..……37

CHAPTER 4……..……41

Software Requirement Specification……..……..……..41

CHAPTER 5……..……..……...43

System Requirements……..……..……43

CHAPTER 6……..……..……..44

Model Used……..……..……44

Conclusion……..……..……..47

Future Scope of Work……...……...……...48

References……..……..……....49

APPENDICES……..……..……50

LIST OF FIGURES

S. NO.	Table Description	Page No.
Figure 1	LAN-T Software Architecture	11
Figure 2	LAN Trainer Frame Format	12
Figure 3	The LLC layer provides reliable data transfer to the Application	16
Figure 4	Space-time diagram showing the operation of the stop-and-wait protocol	21
Figure 5	Sender FSM for stop-and-wait (simplified)	22
Figure 6	Receiver FSM for stop-and-wait (simplified)	22
Figure 7	Complete FSMs for stop-and-wait (a) sender (b) receiver	23
Figure 8	Sliding window protocol, sender's window 1= 4, receiver's window = 1 goback-N	25
Figure 9	Theoretical throughput of ALOHA as a function of offered load	28
Figure 10	Space-time diagram showing collisions in CSMA	30
Figure 11	A ring LAN formed with unidirectional, point-to-point links	35

Figure 12 Transceiver operating in (a) bypass mode
(b) transmit mode 36

LIST OF ABBREVIATIONS

Career sense multiple access	-	CSMA
Career sense multiple access (collision detection)	-	CSMA/CD
File transfer protocol	-	FTP
Medium access control	-	MAC
Data link layer	-	DLL
Network interface unit	-	NIU
Network emulator unit	-	NEU

1. INTRODUCTION

As we approach the new millenium, the Internet is becoming all-pervasive, promising to touch the lives of everyone. As such, a good understanding of computer networking is fast becoming essential for the computer and electronics professionals of tomorrow.

NETWORK TRAINER SYSTEM enables the student to experiment with various network topologies, access methods, and higher-layer network protocols.The student can become familiar with the key concepts in most of today's LANs : multiple-access to a shared medium, reliable data transfer in the face of errors, connection management, bridging and routing, and networked applications.Performance evaluation through measurements and simple models is stressed.

The trainer features include:

- User configurable data rates
- Generation of bit errors and frame errors between nodes
- Variable network size
- User configurable delays between nodes
- ALOHA, CSMA, CSMA/CD, Token Bus, Token Ring, Star, Stop-and-Wait and various window protocols for reliable data transfer

1.1 SCOPE

The LAN-T is particularly suited for:

- Students at B.Tech/B.S. & M.S. levels in engineering/technical institutes (CS & EE)
- Technical training centres in network organisations
- R&D personnel and practicing engineers in research labs and industry

1.2 PROBLEM STATEMENT

Implement a Network Trainer also called Local area network training system. It will consists of LAN protocol simulator & analyzer software. Network Trainer system will be designed to help students understand the basic concepts, modes of operation and protocols involved in networking. The trainer system will be provided with software for analysis of different network layers protocols, and measurement of error rate and throughput. Students can easily do connections in different topologies and can learn actual data transfer either through hardware or through simulated network concept. Facility will be provided into system software to introduce errors into packets being sent and analyze the effect of error on different protocols and hence find the effect on throughput graph as well.

1.3 SOFTWARE ARCHITECTURE

The software involved in the LAN-T is distributed between the NIU card and the PC. The software in the NIU card handles most of the protocol and driver related issues, while the software in the PC side is available as a library of DLL to be used by the application programs. The software architecture implemented in the LAN-T is shown in Figure 1.

Application program in DOS	Application program in Windows
C-Library	Dynamic Link Library (DLL)

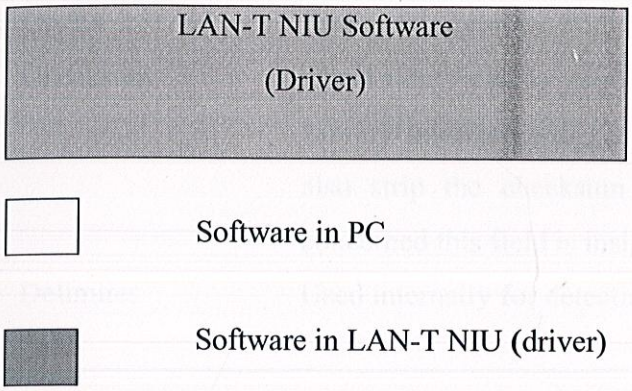


Figure 1: LAN-T Software Architecture

Each NIU card emulates two nodes and more than one card can be used in the same PC. Since more than one card can be used in a single PC, the card number is inferred from the node number. For example, if we have two cards, then node-0 and node-1 represent card-1 and node-2 and node-3 represent card 2. Thus except during the initialization phase, the interface looks alike with Node-Id being the discriminator.

1.4 Frame Format

The format of the data frames used in the LAN-T is shown in Figure 2.

1 Byte	1 Byte	0-998 Bytes	2 Bytes	2 Bytes
DA	SA	DATA	Checksum	Delimiter

Figure 2: LAN Trainer Frame Format

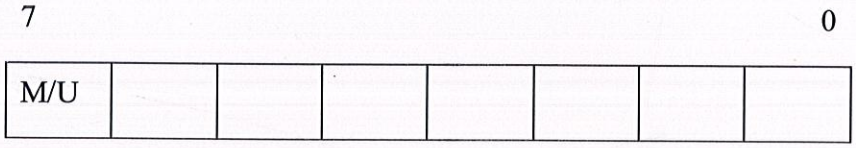
DA Destination Address

SA Source Address

DATA	Data portion can be from 0 - 998 bytes
Checksum	Checksum field is generated internally by the NIU library. The Library interface will generate the checksum while transmission, and also strip the checksum on reception. As far as the student is concerned this field is insignificant.
Delimiter	Used internally for detecting the end of frame.

1.5 Address Field Format

LAN Trainer supports unicast and multicast addresses. The most significant bit of the address specifies whether it is an unicast or multicast address.



M/U = 1 Multicast address

M/U = 0 Unicast address

The address 0xff is the broadcast address, and all the nodes will receive such frames.

1.6 Transmit / Receive Modes

Data transmission through the network by the application program on the PC can be done in number of ways. The widely used modes that are available in the NIU are:

Packet Polling	The packet is received by the low-level driver and stored internally. The user has to poll and read the data. In this mode there is a high probability of packet getting missed at high data rates.
Packet Interrupt	In this mode whenever the packet is received by the driver, it sends an interrupt to the user.

2.EXPERIMENTS

2.1 DATA LINK LAYER

2.2.1 DLL Basics

Physically connecting two or more computers together using media such as electrical cable or optical fiber is only the first step in building a useful computer network. The physical connection enables the computers to exchange electrical or optical signals representing streams of bits or bytes. The next step is to devise mechanisms for the orderly exchange of large units of information, *packets*, between specific pairs of computers in the network. This is the task of the *data link layer (DLL)*.

The DLL needs to be able to identify the start and end of each packet, to ensure reliable delivery of packets even in the face of bit errors on the physical link, and to allow several applications to share one link. Considering only two computers, these issues are covered in this chapter.

An application such as file transfer or remote database access cannot tolerate such packet loss. Every byte of a file must be written into the remote file in exactly the same order as in the local file. Hence, it is necessary to have a layer of network software between the MAC layer and the application that ensures reliable data transfer. This function is usually provided by the *Logical Link Control (LLC)* and/or the Transport layers in the OSI model (see Figure 3). In this chapter we look at various reliable data transfer mechanisms that are used in these layers.

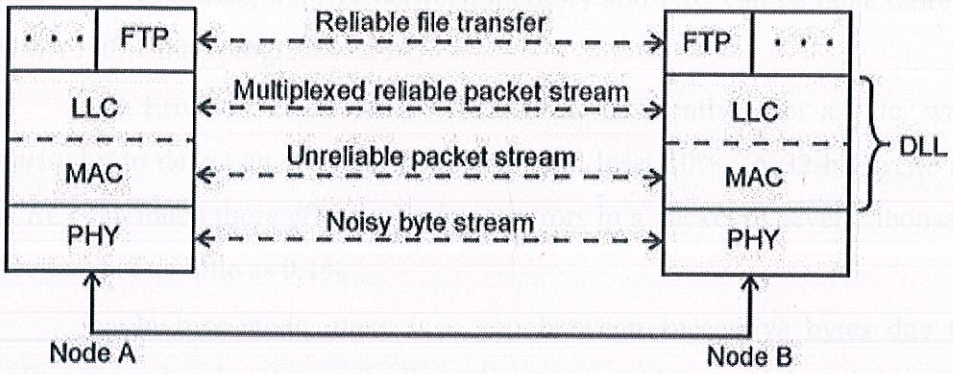


Figure 3: The LLC layer provides reliable data transfer to the application

In a LAN with a number of computers, it is also necessary to have rules such that only one computer transmits at a time. This *medium access control (MAC)* function is a sub-layer of the data link layer. We cover various MAC protocols in the next chapter.

2.1.2 Packet Transmission

Problem

Transmit packets from one node to another. One node, the *sender* reads a message from the keyboard and transmits it in a single packet. The other node, the *receiver* receives the packet and writes the message to the display.

Concepts

- Packet communication

Program Development

Combining a number of bytes into a single packet has several advantages:

- The NIU hardware can be programmed to interrupt the host CPU only once per packet rather than once per byte, thus reducing the overhead per byte.

- Likewise, transfer between memory and NIU can be done more efficiently using DMA rather than programmed I/O.
- Error detection can be done more efficiently. For a byte, we need at least 1 *parity* bit to detect an error, an overhead of at least 10%. A 32-bit *cyclic redundancy code (CRC)* can much more effectively detect errors in a packet of several thousand bytes with an overhead of as little as 0.1%
- In byte-mode, there is a gap between successive bytes due to hardware and software overheads. Hence, the line utilization could be much less than 100%. With packet-mode transfer, bytes in a packet are transmitted without delay, resulting in higher line utilization.
- In a network with more than 2 nodes, each unit of data needs a destination address to indicate for which node it is intended and the address of the source of the data. With packet-mode, this is required only once per packet.

Of course, these benefits are not free: packet transmission requires more complex hardware and software. A typical packet is shown below:

SOP	A special bit pattern or code indicating start of packet
Length	Number of bytes in the packet, in case this can vary
Data	The actual information
CRC	Cyclic redundancy code for error detection
EOP	A special bit pattern or code indicating end of packet

```
Sender()
 {
    Initiallize network card in packet mode

    do
       {
            Read a message from the keyboard
            Form the message into a packet
            Write the packet to the NIU

       } while (message is not empty)
 }

Receiver()
 {
    Initiallize network card in packet mode

    do
       {
            Read a packet from the NIU
            Depacketize the message
            Write message to the display

       } while (message is not empty)

    Get statistics from NIU and print
 }
```

Note that we use an empty message to cleanly terminate the program. To read a one-line message, we can use the C library function gets() . For a multi-line message, we can use gets() repeatedly until an end-of-message indication is read. The NIU automatically generates and checks the CRC for each packet.

2.1.3 Stop-and-Wait Protocol

Problem

Provide reliable data transfer between two nodes over an unreliable network using the stop-and-wait protocol.

Concepts

* error correction
* timeouts
* state machines

Program Development

Let us consider an analogy. You want to send a message to a friend in a distant town. The message is not urgent but it is important that it reaches your friend. You do not want to spend very much so you decide to write a letter. The postal service is like the MAC layer - it usually delivers a letter within a few days, but there are no guarantees. The letter may not reach, it may reach after a delay of months, it may be delivered soiled or torn and hence be unreadable.

If your letter is delivered in good condition and your friend's reply is similarly delivered to you, the message has been reliably transferred. If something goes wrong, after waiting for a reasonable period, you send a copy of your original letter. The waiting period depends on the normal delivery time (transmission delay) and low long you expect your friend to take in writing the reply (processing time) plus some margin for unexpected delays.

Say 3 days to go, 2 days to reply, 3 days to return and 6 days margin: you would resend after 2 weeks.

This is precisely the sequence of actions in the stop-and-wait protocol. The sender transmits a data packet D1 and waits for an ACK packet from the receiver. If it receives the ACK, it discards the data packet and repeats the procedure with the next data packet, D2. However, if it does not receive the ACK within a time-out period, it retransmits the data packet (Figure 4).

A few details need to be taken care of. Suppose the receiver receives D1 and sends the ACK which gets lost. The sender timesout and retransmits D1 which the receiver receives. How does the receiver know that this is not the next data packet D2? For this purpose, the sender puts a *sequence number* in every packet. The receiver remember the last sequence number that it received. After receiving D1 it only accepts and passes on to the application packet D2. The receiver still acknowledges the duplicate D1 so that the sender does not timeout again. Likewise, the sender may receive several ACKs for the same data packet. So the ACK must contain the sequence number of the data packet that it is acknowledging.

The Finite State Machine Approach

The above description of the stop-and-wait is reasonably easy to comprehend and could be converted into a program in a straight forward manner. Most network protocols are far more complicated. A prose description could lead to ambiguities and errors in implementation.

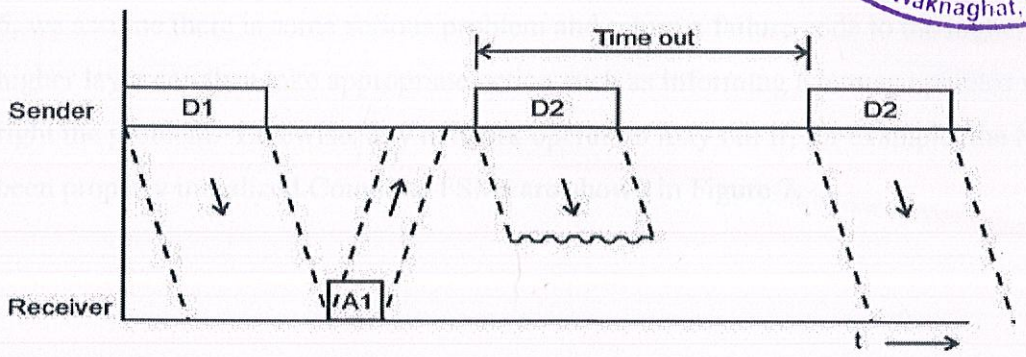


Figure 4: Space-time diagram showing the operation of the stop-and-wait protocol.

Hence they require the use of system atic techniques for specification and correct implementation. Most commonly used is the finite state machine (FSM) approach. This is useful for a system whose behaviour is determined by events such as receipt of a packet or user input. When an event occurs, the system performs some action that depends on the current state and the event. The action may include moving to a new state. Thus, the specification of the system consists merely of listing the possible states and events and the actions to be taken for each <state, event> pair.

The FSM for the sender in the stop-and-wait protocol is shown in Figure 5. Initially, it is in the idle state. The only event of interest here is a SendData command from the higher layer. The sender transmits the data packet and goes to the AwaitACK state. If there are no errors, it will receive the ACK packet from the receiver and return to the Idle state. If the ACK does not come, there will be a Timeout event upon which the sender initiates retransmission.

Similarly, the receiver FSM is shown in Figure 6. This is simpler since the receiver does not wait, it merely responds to data packets from the sender.

For clarity, we have not shown all <state, event> pairs in the figure. For instance, a Timeout in the Idle state is an error. A RcvACK in the Idle state must be a duplicate that is ignored. Suppose the receiver is not running, or the network has failed, the sender will never get an ACK. Rather than have it timeout and retransmit repeatedly, we count the number of successive retries for a given data packet. If this exceeds some limit, say MAX_RETRIES =

5, we assume there is some serious problem and return a failure code to the higher layer. The higher layer can then take appropriate action such as informing a human operator who can set right the problem. Likewise, any network operation may fail if, for example, the NIU has not been properly initialized.Complete FSMs are shown in Figure 7.

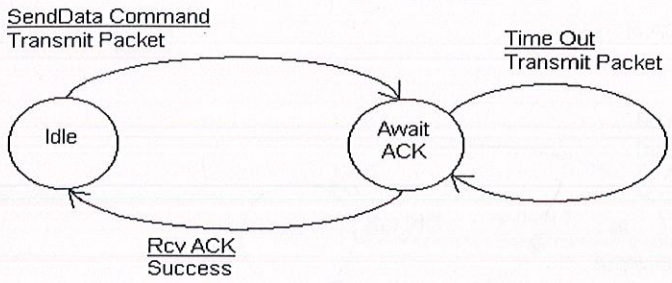


Figure 5: Sender FSM for stop-and-wait (simplified).

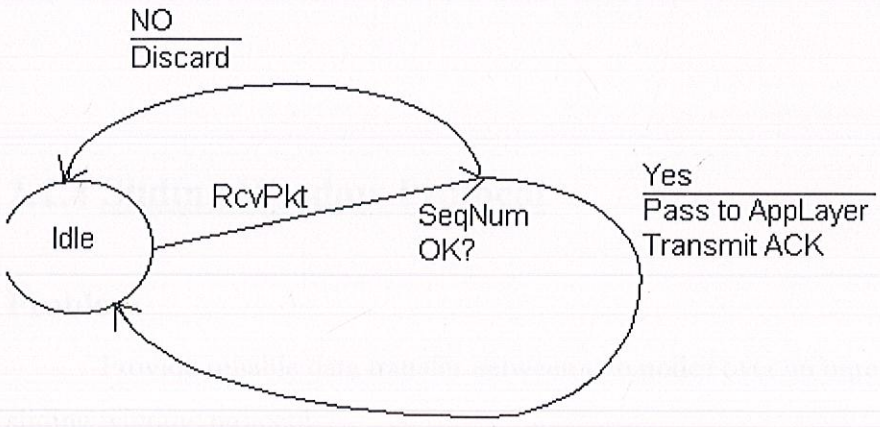


Figure 6: Receiver FSM for stop-and-wait (simplified).

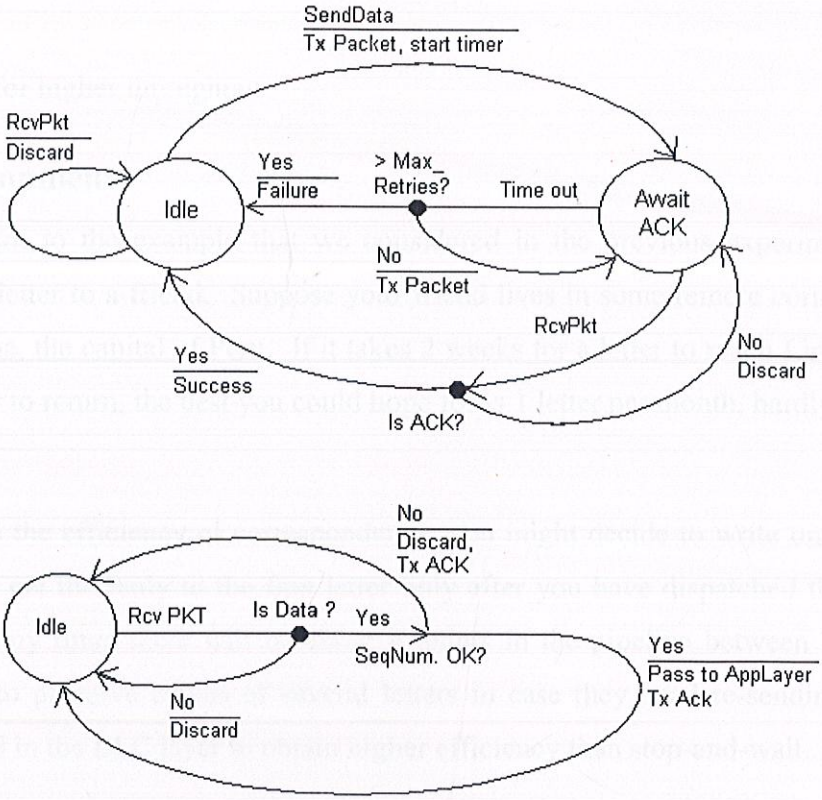


Figure 7: Complete FSMs for stop-and-wait (a) sender (b) receiver.

2.1.4 Sliding Window Protocol

Problem

Provide reliable data transfer between two nodes over an unreliable network using the sliding window protocol.

Concepts

Pipelining for higher throughput

Program Development

Let us return to the example that we considered in the previous experiment, viz. reliably sending a letter to a friend. Suppose your friend lives in some remote corner of the world, such as Lima, the capital of Peru. If it takes 2 weeks for a letter to reach Lima, and 2 weeks for the reply to return, the best you could hope for is 1 letter per month, hardly a lively correspondence.

To improve the efficiency of correspondence, you might decide to write one letter a week. You would get the reply to the first letter only after you have dispatched the fourth letter. In fact, at any time, there will be about 4 letters in the pipeline between here and Lima. You have to preserve copies of several letters in case they need re-sending. This scheme can be used in the LLC layer to obtain higher efficiency than stop-and-wait.

The sender has to have several packet buffers for packets that have been transmitted but not yet acknowledged. In order to limit the memory usage at the sender, it is customary to set a maximum on the number of such outstanding packets, referred to as the sender's *window*.

Suppose the sender's window is 4 packets. It can send packets 1..4 without waiting for an ack. When it receives ack 1, it can discard packet 1 and slide the window forward to <2..5> and so on (Figure 8). As in stop-and-wait, when a packet is transmitted, a retransmission timer is started. If the timer expires without receiving the corresponding ack, the packet is re-transmitted.

The receiver may receive packets out of order, e.g., it receives 1, 2, 3, 5 (4 is lost). Should the receiver accept 5 or not? If it accepts 5, this must be buffered in the LLC layer until 4 is received and only then passed to the application in the proper order. In a simple receiver, any packet received out of order is simply discarded. The receiver has only a single packet buffer. If there is a timeout, the sender must re-transmit the lost packet and every succeeding packet. This is referred to as sliding-window, *goback-N*.

Goback-N uses up extra network bandwidth in retransmitting packets that have already reached the receiver. A more efficient alternative is to have several packet buffers in the receiver's LLC layer for saving out-of-order packets. Then, the sender need retransmit only the lost packets (4 in the example). Thus, we have sliding window, *selective-repeat*. It is often convenient to have equal windows at the sender and receiver, but in general, they could be different depending on the available memory.

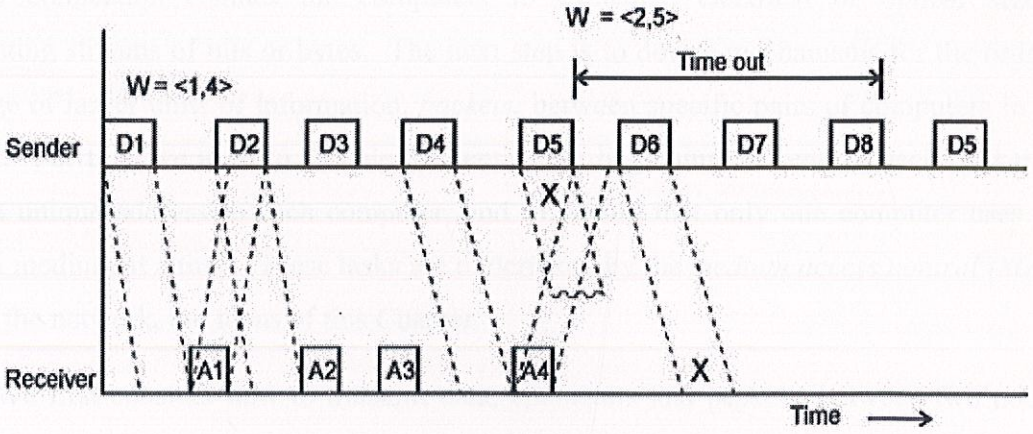


Figure 8: Sliding window protocol, sender's window 1= 4,receiver's window = 1 goback-N

2.2 MEDIUM ACCESS CONTROL

2.2.1 MAC Basics

Physically connecting two or more computers together using media such as electrical cable or optical fibre is only the first step in building a useful computer network. The physical connection enables the computers to exchange electrical or optical signals representing streams of bits or bytes. The next step is to devise mechanisms for the orderly exchange of larger units of information, *packets*, between specific pairs of computers in the network. For this, we need to be able to identify the beginning and end of each packet, to assign a unique address to each computer, and to ensure that only one computer uses the physical medium at a time. These tasks are undertaken by the *medium access control (MAC)* layer of the network, the focus of this Chapter.

We first consider how to transmit data, characters and packets, between two nodes. Then, we look at the more complex problem of transmitting between a number of nodes. In the latter case, we need a mechanism to ensure that only one of the many nodes transmits at a time.

2.2.2 ALOHA

Problem

Implement the ALOHA protocol for packet communication between a number of nodes connected to a common bus.

Concepts

* Multiple access to a shared medium
* Addresses

Program Development

We have seen how two nodes can exchange packets over a point-to-point link.When we have a number of nodes to be interconnected in a local area network, the simplest topology is a bus.When any node transmits a packet on a bus, it can be received by every other node, i.e., a bus is a *broadcast topology*. This has two implications. First, every node must have a unique address and each packet must have the address of the destination (the source address is usually also included). Thus, only the node for which the packet is intended will actually process it; others will discard it. Second, since only one node can transmit at any instant in time, a mechanism is needed to ensure that each node gets a turn to transmit. If two or more nodes transmit at the same time, their packets will *collide* and be lost.

In this experiment, we accommodate the first of the above by adding two fields to the packet used in Experiment E??:

Dest	address of the destination node, typically ranges from 8 bits to 48 bits
Src	address of the source node

Note that the destination address is placed before the source address. Since the packet is received serially, this enables the hardware to check the address as soon as possible and decide whether to discard or receive the packet.

The simplest mechanism by which a node can gain access to the bus is for it to make the optimistic assumption that no other node wants to transmit. It just goes ahead and transmits as and when it wants. This may not seem a very useful strategy. However, it was successfully used in about 1970 in one of the earliest local area networks at the University of Hawaii, the *ALOHA* net.

If the number of nodes is small and each transmits infrequently, the probability that two or more will choose to transmit at the same time is low. Thus, this strategy usually works well under light loads. As the load increases, we expect collisions to occur more

frequently. Under heavy loads almost every tramsmission will be lost due to collisions and the throughput will drop to close to zero. Indeed, it can be shown that the maximum throughput of the ALOHA strategy is about 18.3% and occurs when the rate at which packets arrive is equal to the capacity of the bus. Figure 9 shows how throughput varies as offered load increases in an ideal ALOHA network in which each of a large number of nodes transmits packets at the same rate.

The program is similar to the sender of Experiment E?? with two differences. First, the program runs in a loop in which it waits for some time and then transmits a packet. At the end of transmission, it checks the NIU status to determine whether the packet was successfully transmitted or suffered a collision. Since we are interested in the effect of collisions, not of bit errors, we set the bit error rate to 0. As a result, if a packet is successfully transmitted, it will be successfully received. Hence, we can use all available nodes for transmission and calculate throughput based on statistics from the senders.

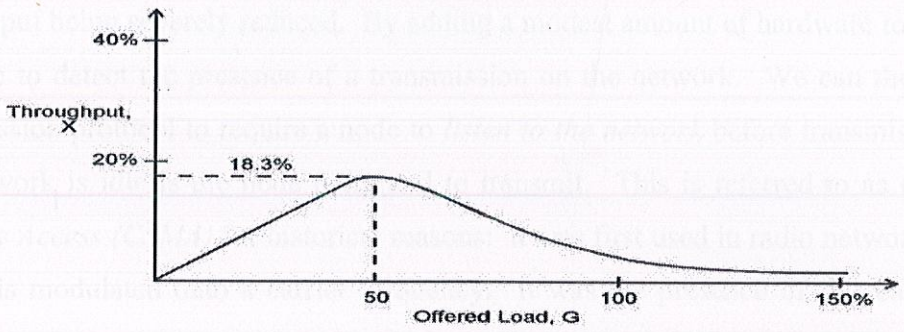


Figure 9: Theoretical throughput of ALOHA as a function of offered load (both expressed as percentages of the network capacity).

2.2.3 Carrier Sense Multiple Access (CSMA)

Problem

Implement the *CSMA* protocol for packet communication between a number of nodes connected to a common bus.

Concepts

* Listen-before-transmit to improve efficency
* Effect of propagation delay

Program Development

In ALOHA, a node transmits whenever it wishes without regard to the activities of other nodes. This may cause a collision with the transmission of another node, resulting in both packets being lost. Under heavy load, we have seen that frequent collisions can result in throughput being severely reduced. By adding a modest amount of hardware to the NIU, it is possible to detect the presence of a transmission on the network. We can then modify the transmission protocol to require a node to *listen to the network* before transmission. Only if the network is idle is the node permitted to transmit. This is referred to as *Carrier Sense Multiple Access (CSMA)* for historical reasons: it was first used in radio networks in which a packet is modulated onto a carrier frequency. It was the presence of this carrier that was sensed.

It is still possible to get collisions due to the non-zero delay between sensing idle and actually starting transmission and the propagation delay of the signal along the bus to other stations. Refer to Figure 10 which shows activity on the network over a period of time. The X-axis represents time, while the Y-axis represents space. Assume that a node A at one end of the network decides to transmit a packet. At time t_0 , it senses the network and finds it idle. After some hardware/software latency of d, it actually starts transmitting at time t_1 . This signal propagates down the cable at a high but finite speed, and reaches node B at the other end of the cable at time t_2 . Note that $t_2 - t_1 = t*d$, where t is the propagation time per metre of the electrical signal (approximately 5ms/m) and d is the distance between nodes A and B.

Thus, even after A has started transmitting, B could sense the cable as being idle and start transmission at time $t_3 < t_2$. As shown in the figure, B's packet propagates towards A and at some point both packets collide and are lost. Notice that t_3 could even be earlier than t_1. In fact, t_3 could be anywhere in the range $(t_1\text{-}t\text{-}d,\ t_1\text{+}t\text{+}d)$ and cause a collision (see Figure 4.3). This interval is referred to as the *vulnerable period*. If B started transmitting before $t_1\text{-}t\text{-}d$, A would find the network busy at t_1 and would not transmit, thus avoiding a collision.

As the vulnerable period increases, the probability of a collision for each packet increases and hence throughput decreases. The vulnerable period is proportional to the end-to-end propagation delay = td. For a given physical medium, t is a constant. Hence, the vulnerable period depends on the length of the cable.

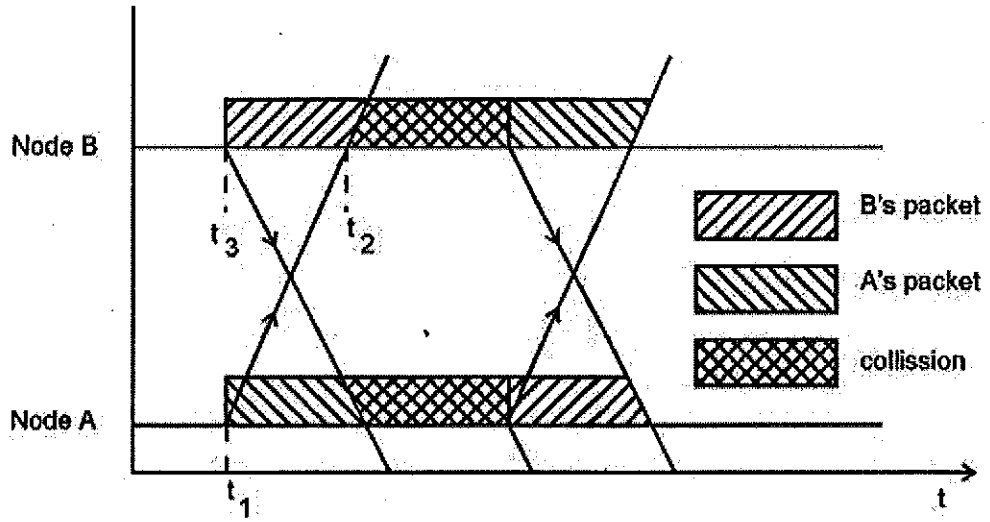


Figure 10: Space-time diagram showing collisions in CSMA

2.2.4 CSMA with Collision Detection

Problem

Implement the CSMA/CD protocol for packet communication between a number of nodes connected to a common bus.

Concepts

* Listen-while-transmit to improve efficiency

Program Development

In CSMA, once a node starts transmitting, it continues until the end of its packet even if there is a collision. Since the collision occurs during the vulnerable period, which is usually much smaller than the packet transmission time, there is considerable unnecessary transmission. By *detecting* the collision and aborting transmission immediately, the channel is freed for other transmission attempts.

Collision detection is usually implemented in hardware. Essentially, this hardware compares what the NIU receives with what it transmits. If the two are different, a collision has occurred:transmission is aborted and the NIU's collision status flag set. (Optionally, the NIU generates an interrupt to notify the host.)

After starting transmission of a packet, the program polls the status flag for one of two events: normal end-of-transmission or collision. In the first case, the packet has been successfully transmitted so the program can go on to the next packet. In case of a collision, transmission is immediately aborted. Next, the node must retry the same packet after some time.

To avoid another collision between the same set of nodes, each node waits a random time before retrying using a *back-off algorithm*. Assume that n nodes had collided. If one node were to retransmit immediately, another after 1 packet transmission time, a third after 2 packet times, and so on, after n packet times all nodes would have successfully transmitted their packets without further collision.

There are two difficulties in implementing this: first, a node does not know how many other nodes were involved in the collision; second, without communicating amongst themselves, the nodes cannot agree on an order. To overcome the second problem, the best that we can do is to randomly select one of the n slots and transmit in that one. There is a fairly good chance that other nodes will choose some other slot. However, it is possible to get another collision.

For the first problem, we can start out with the optimistic assumption that only 2 nodes are involved in the collision. I.e., we set n', our estimate of n, to 2. If there are actually more than 2 nodes involved, there is a very good chance that there will be another collision. So, if we experience a second collision for the same packet, we suspect that n' is too low an estimate of n, and we increase it. On every successive collision, we further increase the estimate. Eventually, n' will become large enough and the packet will be successfully transmitted.

At each stage, we could increase n' by incrementing it. This is *linear incremental back-off*. This has the disadvantage that if in a large network under heavy traffic, 10s of nodes are involved in a collision, it would take many successive collisions before n' reaches n. To adapt faster, we could double n' on each collision, resulting in *binary exponential back-off*. If, say, 100 nodes collided, it would take only 7 steps for n' to reach 128, greater than n.

Note that during the back-off period, the node cannot transmit any other packet. In effect, the node's transmitter is temporarily disabled, thus reducing the load on the network. The back-off thus acts as a regulatory mechanism in case of overload. As a result, even with high offered loads, throughput in the case of CSMA/CD does not drop to zero as in the case of CSMA and ALOHA.

2.2.5 Token-Passing Bus

Problem

Implement a token-passing access method for a bus LAN.

Concepts

* Demand assignment versus random access
* Priorities
* Token management

Program Development

The *random* access methods that we have seen so far have the advantages of simplicity of implementation and robustness in the face of errors. In many situations, they give good performance also. However, they suffer from some disadvantages. Under heavy load, especially with high bit rates and large networks, the collision rate becomes high and throughput drops. Due the the random nature of the access method, the delay in transmission of a particular packet cannot be guaranteed. Even under light load, it is possible for a packet to suffer many collisions and hence large delay. This is unacceptable in real-time applications such as industrial control where delay must be bounded.

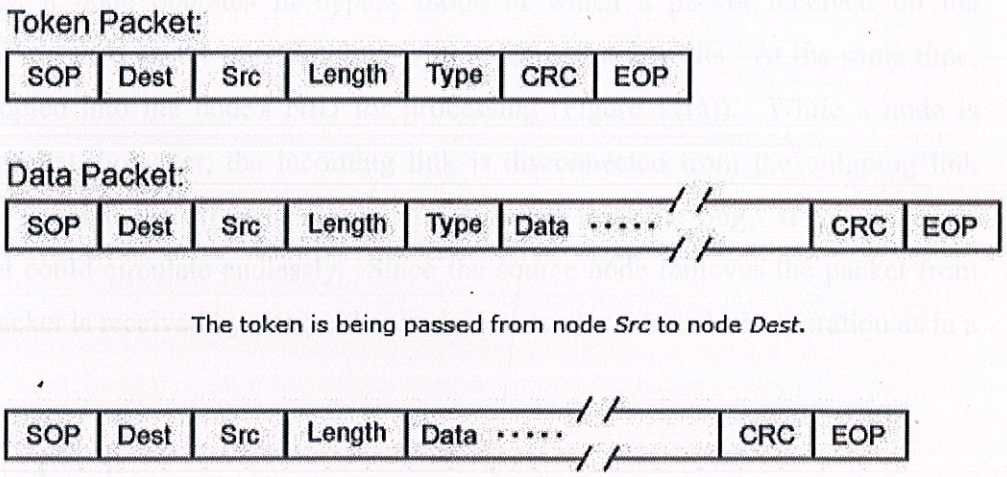
To overcome these disadvantages, we need an access method that avoids collision and that gives each node a chance to transmit at deterministic intervals. Collisions will be eliminated if only one node is allowed to transmit at a given point in time. A simple way of achieving this is to have exactly one *token*, similar to the baton in a relay race. A node can transmit only if it holds the token. By passing the token from node to node in a specific order, we can ensure that all nodes get a turn at regular intervals. Given the number of nodes, the time taken for passing the token, and the packet transmission time, we can calculate the maximum delay.

In a random access protocol, a node can transmit a packet as soon as it becomes active. In a token-passing protocol, on the other hand, there are two phases:

1. the network must be initialized -- the order in which the token is to be passed is determined, and then the token is created by one node

2. only after the network has been correctly initialized can the data transfer phase begin in which nodes transmit and receive data packets.

The token itself is usually a packet with *type* field containing the value TOKEN while a data packet has *type* = DATA as shown below



2.2.6 Token Ring

Problem

Implement a token-passing access method for a ring LAN.

Concepts

* Demand assignment versus random access
* Priorities
* Token management

Program Development

The *ring topology* is useful with point-to-point media such as optical fibre. Each node is connected to the ring via a transceiver. The transmitter of one transceiver is connected to the receiver of the neighbouring transceiver to form a uni-directional ring (Figure 11).

Normally, a node operates in bypass mode in which a packet received on the incoming link is repeated on the outgoing link with a delay of a few bits. At the same time, the packet is copied into the node's NIU for processing (Figure 12(a)). While a node is transmitting a packet, however, the incoming link is disconnected from the outgoing link (Figure 12(b)). This has the effect of removing the packet from the ring. If this were not done, the packet could circulate endlessly. Since the source node removes the packet from the ring, each packet is received by every other node, i.e., we have broadcast operation as in a bus.

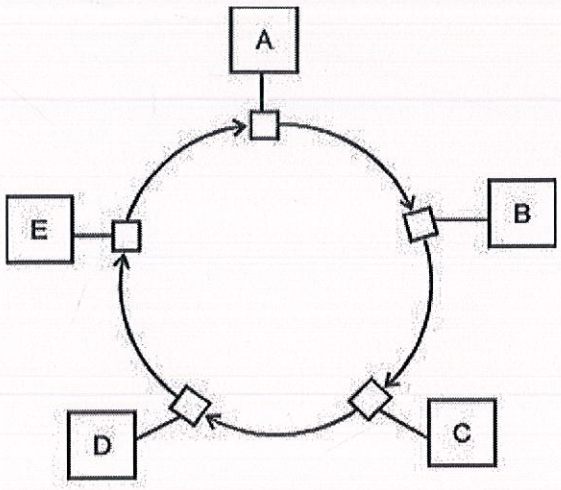


Figure 11: A ring LAN formed with unidirectional, point-to-point links.

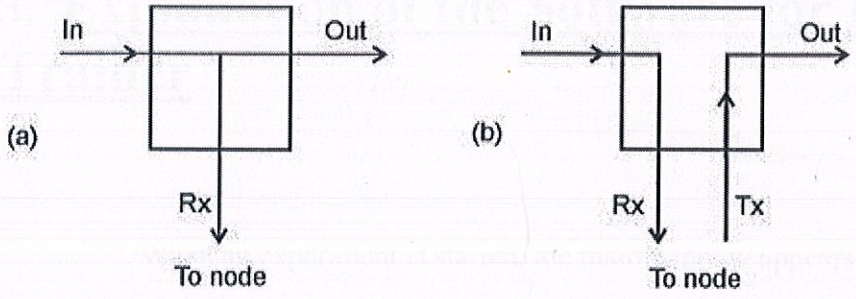
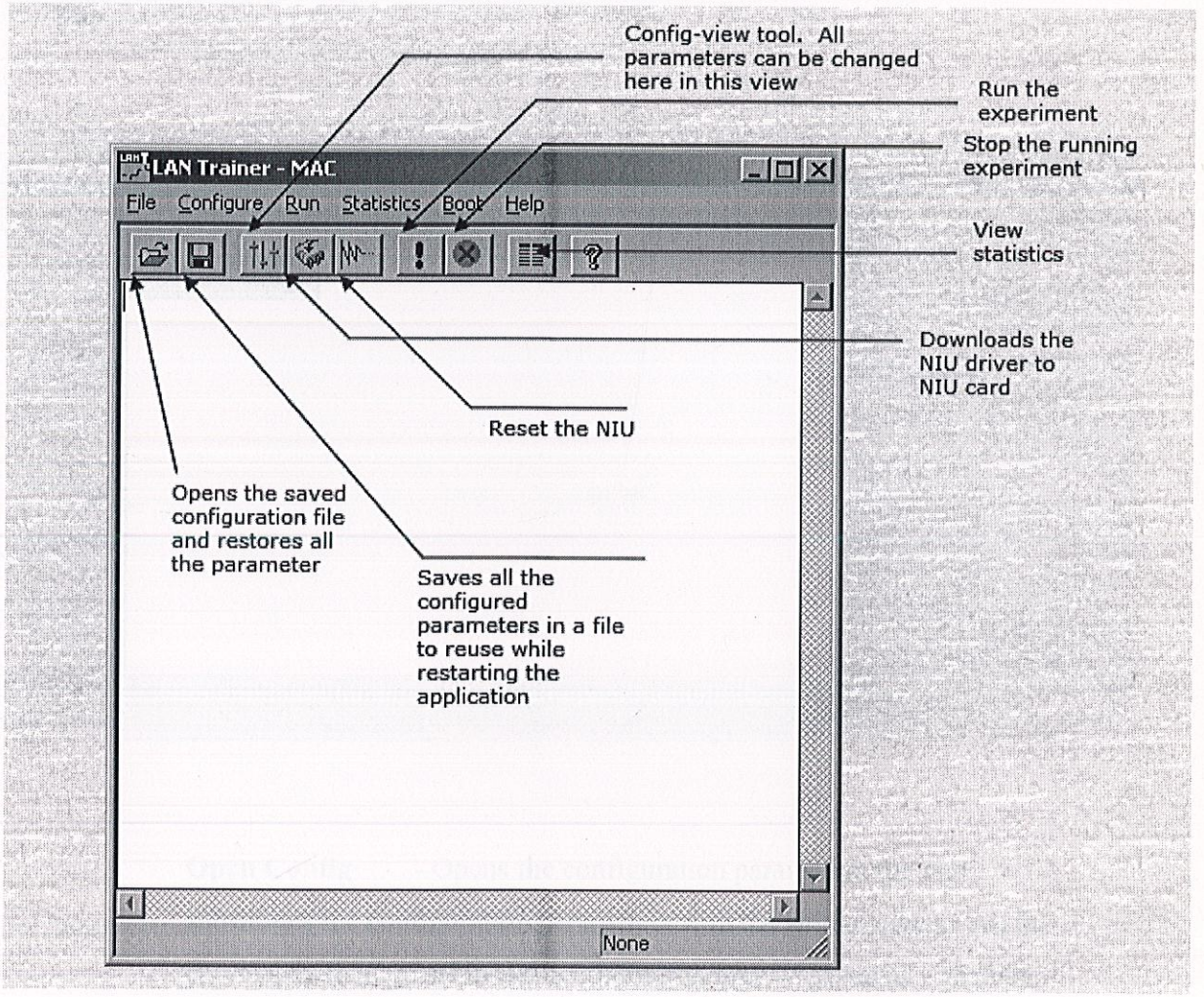


Figure 12: Transceiver operating in (a) bypass mode (b) transmit mode.

3. Explanation of the Software for LAN Trainer

When an experiment is started, the main window appears.



The LAN Trainer Shell

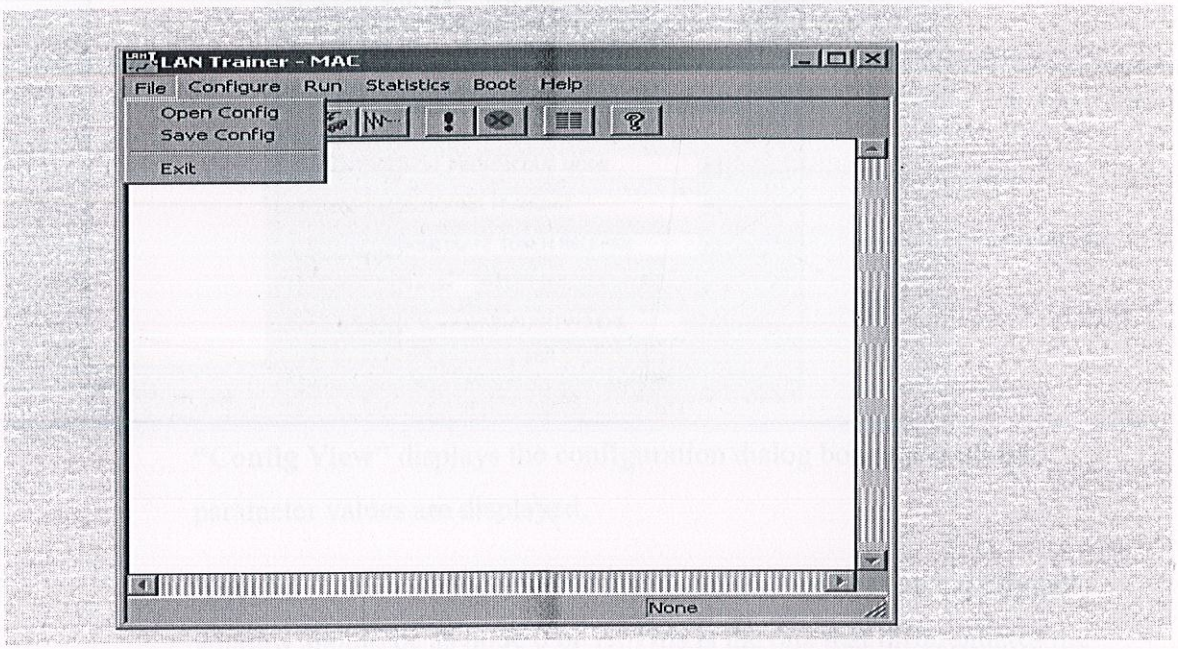
All the LAN Trainer experiments have a common user-interface. This allows configuring of the NIU plugged into the PC, which must match the settings in the Emulator

Unit. It also allows setting of software parameters such as experiment duration and packet length.

The LAN Trainer Shell, called as **LT_Shell** (or ShellVxy for latest versions with xy representing the version) appears with main menu in the application/experiment window as shown in Fig A-1.

The POP-UP menus are explained below:

File

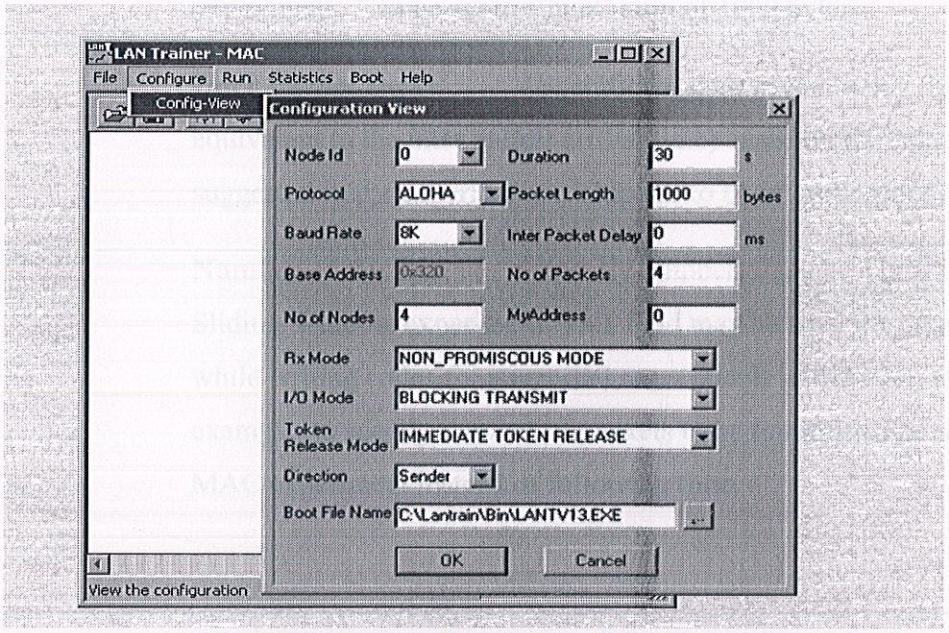


Open Config - Opens the configuration parameters file and substitutes the configuration parameters with the stored values. All the parameters that are set in the configuration may be saved as a file using "Save Config" and these are the ones stored while opening.

Save Config - Saves the configuration parameters in a file. The saved details may be applied as and when required using "Open Config".

Several configuration files for experiments may be saved such that they can be applied whenever an experiment is started and only the required parameter for that experiment can be changed.

Configure



"Config View" displays the configuration dialog box where all the parameter values are displayed.

Node Id - Specify that that particular application/experiment window should be as node 0 or 1. This is the one that differentiates the two applications in the same PC.

Protocol - Specify the MAC Protocol that NIU has to emulate. By default, it is set to Aloha

Baud Rate - Specify the data rate between 8Kbps and 1Mbps. (Only for CSMA/CD and Ring mode, this needs to be set both in the Application window and the NEU. For Aloha and CSMA, settings in NEU are sufficient.)

No of Nodes - Specify the number of nodes in the network that are used to do Token Ring experiment. This field may be used for other purposes while writing codes for suggested experiments in exercise also.

Duration - Specify the duration of the experiment in seconds.

Packet Length - Specify the length of packets that are used in the experiment. Can be set to a maximum of 1000 bytes.

Inter Packet Delay - Specify the inter packet delay (that is equivalent to the inter packet arrival time) based on the calculation suggested in the experiments. This has to be in milliseconds.

Number of Packets - Specify number of packets in a window for Sliding Window experiment. This field may be used for other purposes while writing codes for suggested experiments in exercise, say for example, to specify number of packets to be transmitted or received in a MAC experiment instead of following time.

4. SOFTWARE REQUIREMENTS SPECIFICATION

Ultimately the requirement phase translates the ideas whatever is in the mind of client (the input) into a formal document (the output of the requirement phase.). In a more general way the SRS is a document that completely describes "What" the proposed system should do without describing "How" the software will do it.

FEASIBILITY STUDY

The feasibility study concerns with the consideration made to verify whether the system fit to be developed in all terms. Once an idea to develop software is put forward the question that arises first will pertain to the feasibility aspects.

There are different aspects in the feasibility study:

- Operational Feasibility.
- Technical Feasibility.
- Economical Feasibility.

OPERATIONAL FEASIBILITY:

There in no difficulty in implementing the system, if the user has the knowledge in internal working of the system. Therefore, it is assumed that he will not face any problem in running the system. The main problem faced during development of a new system is getting acceptance from the users. As users are responsible for initiating the development of a new system this is rooted out.

TECHNICAL FEASIBILITY:

Technical feasibility deals with the study of function, performance, and constraints like resources availability, technology, development risk that may affect the ability to achieve an acceptable system.

ECONOMICAL FEASIBILITY:

One of the factors, which affect the development of a new system, is the cost it would incur. The existing resources available in the company are sufficient for implementing the proposed and hence no extra cost has to be incurred to run the system developed. Thus, the system is financially feasible.

5. SYSTEM REQUIREMENTS

PC :	Pentium II or higher One PCI slot (32-bit) required. (For Star related experiments that may be developed by user, 2 slots are required in one of the PCs) Minimum RAM as recommended by OS or by VC++ if installed
Operating System:	Windows 98 2nd Edition, Windows 2000 Professional
Number of PCs:	2 – 3
VC++ compiler:	Version 6.0 or above

6. MODEL USED

To develop software a particular development strategy is used which encompasses the process, methods and tools. The strategy is often referred to as process model of Software Engineering Paradigm. A process model for software is chosen based in the nature of the project and application, the methods and tools to be used and the controls and deliverables that are required. The model which is being used in this project development is WATERFALL MODEL.

WATERFALL MODEL

The waterfall model derives its name due to the cascading effect from one phase to the other. In this model each phase well defined starting and ending point, with identifiable deliveries to the next phase.

Note that this model is sometimes referred to as the linear sequential model or the software life cycle. The model consists of six distinct stages, namely:

- **REQUIREMENTS ANALYSIS**

In the requirements analysis phase:

(a) The problem is specified along with the desired service objectives (goals).

(b) The constraints are identified.

• SPECIFICATION PHASE

In the specification phase the system specification is produced from the detailed definitions of (a) and (b) above. This document should clearly define the product function.
Note that in some text, the requirements analysis and specifications phases are combined and represented as a single phase.

• SYSTEM AND SOFTWARE DESIGN PHASE

In the system and software design phase, the system specifications are translated into a software representation. The software engineer at this stage is concerned with:

a. Data structure
b. Software architecture
c. Algorithmic detail and
d. Interface representations

The hardware requirements are also determined at this stage along with a picture of the overall system architecture. By the end of this stage the software engineer should be able to identify the relationship between the hardware, software and the associated interfaces. Any faults in the specification should ideally not be passed 'down stream'.

• IMPLEMENTATION AND TESTING PHASE

In the implementation and testing phase stage the designs are translated into the software domain:

- Detailed documentation from the design phase can significantly reduce the coding effort.
- Testing at this stage focuses on making sure that any errors are identified and that the software meets its required specification.

- **INTEGRATION AND SYSTEM TESTING PHASE**

In the integration and system testing phase all the program units are integrated and tested to ensure that the complete system meets the software requirements. After this stage the software is delivered to the customer [Deliverable – The software product is delivered to the client for acceptance testing.]

- **MAINTENANCE PHASE**

The maintenance phase the usually the longest stage of the software. In this phase the software is updated to:

- Meet the changing customer needs
- Adapted to accommodate changes in the external environment
- Correct errors and oversights previously undetected in the testing phases
- Enhancing the efficiency of the software

Observe that feed back loops allow for corrections to be incorporated into the model. For example a problem/update in the design phase requires a 'revisit' to the specifications phase. When changes are made at any phase, the relevant documentation should be updated to reflect that change.

CONCLUSION

The network trainer works on diverse transmission protocols including ALOHA, CSMA, CSMA/CD among others, with each protocol having its unique characteristics. The systems are connected through varied topologies like BUS, RING, STAR and TOKEN-RING. These topologies determine how a particular set of systems would share data between them. We have been successful in implementing the transfer of data through a few protocols for a number of topologies and have enlisted the results accordingly.

The basic idea behind this project is to provide a good method for calculating the efficiency of transmission of data between systems and generating results that signify whether the transmission was successful or not.

FUTURE SCOPE

Future enhancements recommended are :

- The limit of message length to be increased.
- User Interface to be improved.
- More protocols can be included.
- Code can be improved to work on bigger networks.

REFERENCES

1. Network Security Management for a National ISP R. Deepak,Timothy,Timothy A Gonsalves,Hema A. Murthy, and N.Usha Rani TeneT Group, IIT Madras.
2. Cfengine – GNU project http://www.gnu.org/software/cfengine/cfengine.html
3. Mark Burgess, Cfengine: A site configuration engine,USENIX Computing systems.
4. T.A. gonsalves , Ashok Jhunjhunwala , and Hema A Murthy et al., "CygNet : Integrated Network Management for voice + Internet ," NCC 2000.
5. A.G.K Vanchynathan , N.Usha Rani ,C.Charitha , and T.A. Gonsalves , "distributed NMS for affordable Communications", NCC 2004, January 2004.
6. TECH TRENDS BY GOPAL GARG, CYPRESS SEMICONDUCTOR, AND R THIRUMURTHY, MIDAS COMMUNICATION TECHNOLOGIES.
7. DISTRIBUT ED NMS FOR AFFORDABLE COMMUNICATIONS by A.G.K. Vanchynathan, n.Usha Raniand C.Charitha,Timothy A. Gonsalves.
8. Network Security management for a national ISP R.Deepak, Timothy a. Gonsalves , Hema A. Murthy, and N.Usha Rani TeneT Group, IIT Madras.
9. http://www.uic.edu/depts/accc/network/ftp/vftp.html
10. www.elab**trainers**kits.com/category/**LAN-Trainer**.html
11. benchmark-electronics.com/Products/**lan**main.html
12. www.tetcos.com/brochure/**lantrainer**_brochure_181106.pdf

APPENDIX

CODING

```
namespace project_ppt
{
   public partial class Form1 : Form
   {
      public Form1()
      {
         InitializeComponent();
      }

      private void Form1_Load(object sender, EventArgs e)
      {

      }

      private void configViewToolStripMenuItem_Click(object sender, EventArgs e)
      {
         Form2 obj=new Form2();

         obj.Show();

      }

      private void exitToolStripMenuItem_Click(object sender, EventArgs e)
      {
         this.Close();
      }
```

```
        private void viewToolStripMenuItem_Click(object sender, EventArgs e)
        {
            Form3 f1 = new Form3();
            f1.Show();
        }

        private void toolStripButton1_Click(object sender, EventArgs e)
        {

        }

        private void runToolStripMenuItem_Click(object sender, EventArgs e)
        {

        }

        private void startToolStripMenuItem_Click(object sender, EventArgs e)
        {


        }
    }
}
```

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Text;
using System.Windows.Forms;

namespace project_ppt

{

   public partial class Form2 : Form
   {public int pl,ipd,nop,nod;
      public Form2()
      {
         InitializeComponent();
      }

      private void Form2_Load(object sender, EventArgs e)
      {

      }

      private void button1_Click(object sender, EventArgs e)
      {

         pl=Convert.ToInt32(textBox5.Text);


          nod=Convert.ToInt32(textBox1.Text);
```

```
            int g=1;
        int c=8;
        if(comboBox2.SelectedIndex == 0)
           ipd=(nod*pl)/(g*c);
        textBox6.Text = ipd.ToString();



    }

    private void comboBox2_SelectedIndexChanged(object sender,EventArgs e)
    {

    }

    private void button2_Click(object sender, EventArgs e)
    {
       this.Hide();
    }
  }
}
```