1987

A distributed channel-sense non-preemptive static priority ring for local area computer networks and its performance

Isaac Ghansah
Iowa State University

INFORMATION TO USERS

While the most advanced technology has been used to photograph and reproduce this manuscript, the quality of the reproduction is heavily dependent upon the quality of the material submitted. For example:

- Manuscript pages may have indistinct print. In such cases, the best available copy has been filmed.
- Manuscripts may not always be complete. In such cases, a note will indicate that it is not possible to obtain missing pages.
- Copyrighted material may have been removed from the manuscript. In such cases, a note will indicate the deletion.

Oversize materials (e.g., maps, drawings, and charts) are photographed by sectioning the original, beginning at the upper left-hand corner and continuing from left to right in equal sections with small overlaps. Each oversize page is also filmed as one exposure and is available, for an additional charge, as a standard 35mm slide or as a 17"x 23" black and white photographic print.

Most photographs reproduce acceptably on positive microfilm or microfiche but lack the clarity on xerographic copies made from the microfilm. For an additional charge, 35mm slides of 6"x 9" black and white photographic prints are available for any photographs or illustrations that cannot be reproduced satisfactorily by xerography.

Ghansah, Isaac

A DISTRIBUTED CHANNEL-SENSE NON-PREEMPTIVE STATIC PRIORITY RING FOR LOCAL AREA COMPUTER NETWORKS AND ITS PERFORMANCE

Iowa State University PH.D. 1987

PLEASE NOTE:

In all cases this material has been filmed in the best possible way from the available copy. Problems encountered with this document have been identified here with a check mark ✓.

1. Glossy photographs or pages _____
2. Colored illustrations, paper or print _____
3. Photographs with dark background _____
4. Illustrations are poor copy _____
5. Pages with black marks, not original copy ✓
6. Print shows through as there is text on both sides of page _____
7. Indistinct, broken or small print on several pages ✓
8. Print exceeds margin requirements _____
9. Tightly bound copy with print lost in spine _____
10. Computer printout pages with indistinct print _____
11. Page(s) _____ lacking when material received, and not available from school or author.
12. Page(s) _____ seem to be missing in numbering only as text follows.
13. Two pages numbered ***. Text follows.
14. Curling and wrinkled pages _____
15. Dissertation contains pages with print at a slant, filmed as received ✓
16. Other *** Two pages numbered 51, 120, 142. Text follows.

A distributed channel-sense non-preemptive static priority ring for local area computer networks and its performance

by

Isaac Ghansah

A Dissertation Submitted to the
Graduate Faculty in Partial Fulfillment of the
Requirements for the Degree of
DOCTOR OF PHILOSOPHY

Department: Electrical Engineering and Computer Engineering
Major: Computer Engineering

Approved:

Signature was redacted for privacy.

In Charge of Major Work

Signature was redacted for privacy.

For the Major Department

Signature was redacted for privacy.

For the Graduate College

Iowa State University
Ames, Iowa

1987

TABLE OF CONTENTS

CHAPTER 1. INTRODUCTION

Computer Communication Networks

More than three decades ago when computing systems were first installed a few peripherals such as printers, card readers, terminals, etc., operated as 'slaves' connected close to a single mainframe (or host) computer.

Since that time, we have seen a tremendous decrease in hardware costs coupled with increase in processing speeds and memory capacities. The capabilities of the present day ubiquitous microprocessor serves as a testimony to that.

As a natural consequence of these developments three main things began to happen [1,2]. First, we had computing systems in which hundreds of these peripherals (especially terminals), now more intelligent, were installed to access one (or more than one) mainframe. At the same time, for convenience, these terminals were geographically dispersed so it became apparent that a cost-effective way should be found to allow them to access the hosts. The telephone network, already reaching most homes, became the obvious choice.

Secondly, when it became apparent that the host computer could no longer cope with the processing functions special-purpose computers called front-end processors and later more sophisticated communication processors were developed to off-load to the communication-handling tasks from the host computers.

Thirdly, several very large, powerful and specialized computers geographically distributed in a wide area had been installed. The

potential users were also geographically dispersed. These large computers were so expensive that to duplicate them would not be cost effective so it was necessary to have them interconnected. This enabled users connected to any host access to any other host along with any specialized software and/or hardware resources. This was the very situation in the late 1960s which fueled the initiation of ARPA (Advanced Research Projects Agency) network by the U.S. Department of Defense.

These three somewhat parallel activities, among others, brought about the concept, development, and implementation of data and computer communication networks, thousands of which are deployed worldwide. They range from small networks which provide communications among devices within a single building or small geographic area to long-haul or wide area networks (WANs) which extend over large geographical areas such as countries, and in some cases literally spanning the globe.

The small networks, generally called local area networks (LANs) are a more recent development with characteristics different from the WANs although they fulfill similar objectives namely, resource-sharing.

Local Area Networks (LANs)

Typically a LAN has the following characteristics [3]:

1. Relatively high data rates (0.1-100 megabits per second, Mbps).
2. Users of the network are located with in a small geographical area (0.1-50 kilometers).
3. Low error rates (10^{-8} - 10^{-11}).

4. Ownership by a single organization (e.g., office, hospital, university, factory, laboratory, etc.).

LAN development began in the mid 1970s and was born in 1976 with the publication of the description of Ethernet [4] by Metcalfe and Boggs. The research into the experimental Ethernet began in 1972 [5], however. Since that time several different Ethernet-like and non Ethernet-like commercial LANs have been offered. These networks have proliferated within such a short period for the following reasons:

1. Although hardware costs for intelligent devices such as computers have decreased peripheral devices such as printers are still relatively expensive. LANs provide a way to share these expensive resources.
2. Centralized data bases that are costly to duplicate and maintain can easily be shared by the devices connected to the network.
3. The network is able to survive failures because key systems can be duplicated to provide backup thus enhancing availability.
4. Equipment from different manufacturers can be accommodated since the network software serves as a "translator." This is an advantage for customers.

In spite of these advantages, Stallings [3] cautions about disadvantages LANs might cause:

- The problem of data security and privacy.
- The problem of concurrent update of replicated databases.

- The fact that because of ease of adding new devices an organization might end up with more than necessary resources. To borrow from Stallings this is the problem of "creeping escalation."

The most common types of devices that can be connected to LANs are: computers, mass storage devices, terminals, printers, word processors and sensors (temperature, humidity, security alarm sensors).

LANs have or will most likely have applications in the following areas [6]:

1. Campus computing systems.
2. Office automation (electronic mail, word processing).
3. Factory automation (CAD/CAM, inventory control).
4. Library systems (book inventory, document retrieval, electronic copying).
5. Medical facilities (i.e., hospitals).

As alluded to earlier, several different LANs have been offered by vendors. The main differences between them can generally be found in: the type of transmission medium (twisted pair, coaxial cable, optical fiber); the network topology (bus, star, ring); and the medium access control (MAC) protocol (the set of rules that determine how users gain access to the channel). We will discuss more about LAN topology and MAC protocols in the next chapter.

Dissertation Objectives and Outline

LANs and their MAC protocols are the subjects of this dissertation. This work describes one such LAN called the distributed channel-sense

priority ring (DCPR). Both analytic and simulation modeling techniques are used to determine its performance characteristics and methods are proposed to make the MAC protocol more efficient in terms of throughput-delay performance. Finally, performance comparison of DCPR with known LANs with different MAC procedures are reported.

The contribution is in the modeling and analysis of the original DCPR, proposal for the improved DCPR protocol and, modeling and performance analysis of the two protocols.

Chapter 2 discusses LANs in more detail focusing on MAC protocols both implemented and proposed which are found in the professional literature. Chapter 3 introduces DCPR. Chapter 4 presents analytic models of DCPR and comprehensive performance analysis of it. Chapter 5 provides a simulation model of DCPR and motivation for it with a view to obtaining more accurate performance results. Chapter 6 consists of the improved DCPR protocol and its performance. In Chapter 7, comparison of DCPR and the improved DCPR with known MAC protocols for LANs are discussed. Finally, Chapter 8 concludes this dissertation.

CHAPTER 2. OVERVIEW OF LANs AND THEIR MAC PROTOCOLS

In a Local Area Computer Communication Network a set of independent distributed users (i.e., computers, terminals, etc.) are supposed to communicate with each other via a communication channel.

The communication channel is a shared resource. Therefore there is the need to ensure that only one user uses it at a time.

The rules, procedures or algorithms which are used to determine how the resource should be accessed and used reliably is generally called the Protocol.

For obvious reasons such protocols for Local Area Networks are called Multiaccess Protocols or Medium Access Control (MAC) Protocols.

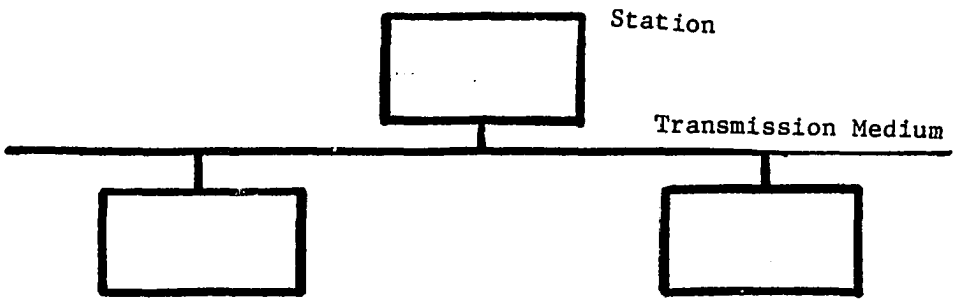
In this chapter we will review various MAC protocols in the literature--both implemented and unimplemented. Because the network topology--the way in which the communication channel is connected to stations--is of prime importance in LAN technology we will also discuss various LAN topologies in this chapter. We begin with the latter.

LAN Topologies

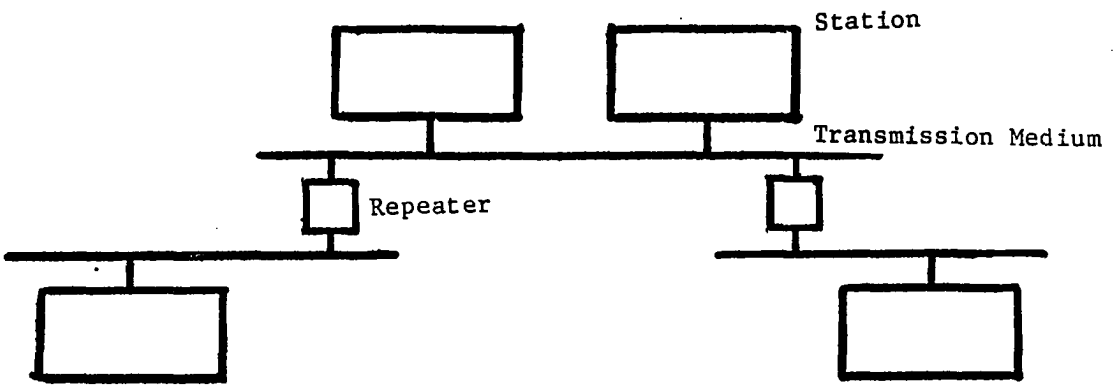
There are three main LAN topologies: Broadcast/bus, star, and ring.

Broadcast/bus topology

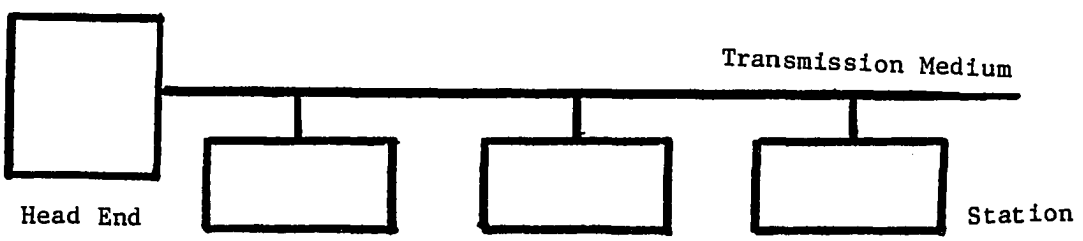
In a broadcast topology all stations share the same channel in the sense that the link between the stations is a multistation link (Figure 2-1a). The channel is a bidirectional bus and signals propagate away from the originating station in both directions to the ends of the bus.



a. Linear Bus



b. Non-rooted tree



c. Centralized transmission signal control: CATV network

Figure 2-1. Broadcast/bus networks

Multiple buses can be interconnected by repeaters so long as signal transmission path on the buses do not form a loop. This type of modified bus can be considered as a non-rooted tree (Figure 2-1b).

In this topology the signal transmission control can be centralized or distributed. Figures 2-1a and 2-1b are examples of the distributed control. In centralized transmission control, all signals are first transmitted to a central station and then broadcast over the bus to all the stations on the network to reach the ultimate destination. The CATV (Community Antenna Television) broadcast system (Figure 2-1c) is an example of a bus system which uses centralized signal transmission control. In this system transmission to the central controller, called headed (HE) is done in one frequency and the transmission from the HE to the receiver is done at a different frequency. CATV systems are configured in single or dual trunk configurations (Figures 2-2a and 2-2b). The dual trunk configuration uses two buses and requires signals to be transmitted through the forward bus to the HE amplifier, where from they are transmitted onto the return trunk to the receiver. The single trunk configuration uses a single bus with the bandwidth of the cable divided into forward and receiving channels. Information signals are transmitted through the forwarding channel to the HE frequency translator, where the information signals are modulated into another (higher or lower) frequency for transmission through the receiving channel, to the receiver.

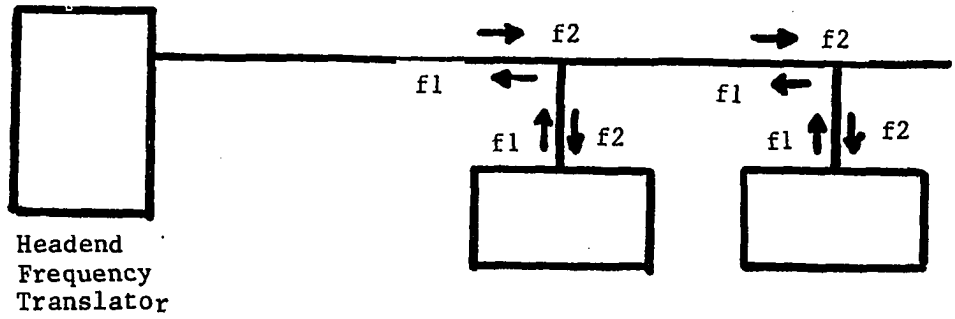
The reliability of a bus topology with distributed signal transmission control is higher than in centralized control because the

Legend:

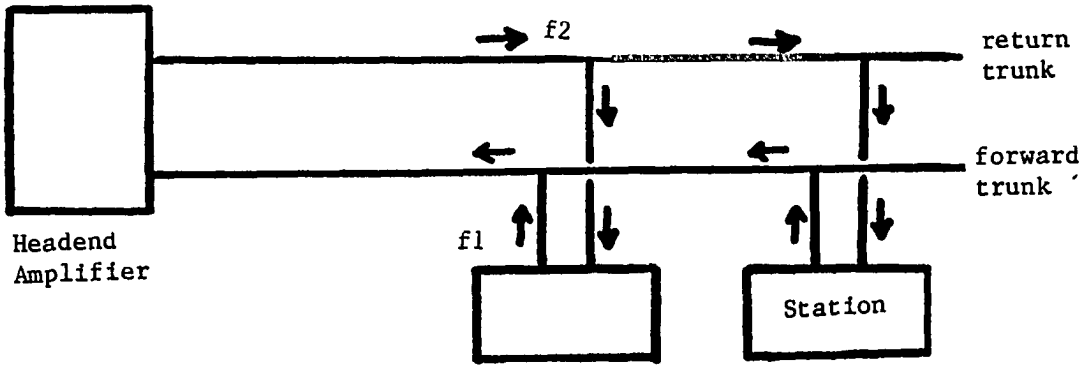
⟶ Direction of signal flow

f1 Forward frequency

f2 Return frequency



a. Single trunk CATV network



b. Dual trunk CATV network

Figure 2-2. CATV network types

network functioning is not susceptible to single station failure.

Star topology

In a star topology (Figure 2-3), each node (or station) is connected to a central node through point-to-point links and all messages pass through this central node from where they are switched to their destinations.

The reliability of this LAN topology is dependent upon the physically central node failing. Network reliability can be improved by implementing the central node with full redundancy.

Ring topology

In a ring topology (Figure 2-4), each node is connected to the adjacent node through point-to-point uni or bidirectional transmission links and arranged to form a closed loop. Unidirectional ring topologies are more common. The sending node transmits the message which is passed from one node to the next until it reaches the receiving node. Depending upon the implementation, either the receiving node can remove the message from the ring or the sending node can remove it when it returns. Each node must be able to recognize messages addressed to it and each must be able to retransmit (i.e., repeat) the message to the next node. Thus each node is an active repeater. The reliability of the ring network depends upon the failure of any one node in the network or the need to add a new node to the network. Any break in the configuration will cause the whole network to go down. Reliability can be ensured by implementing measures to bypass a failed

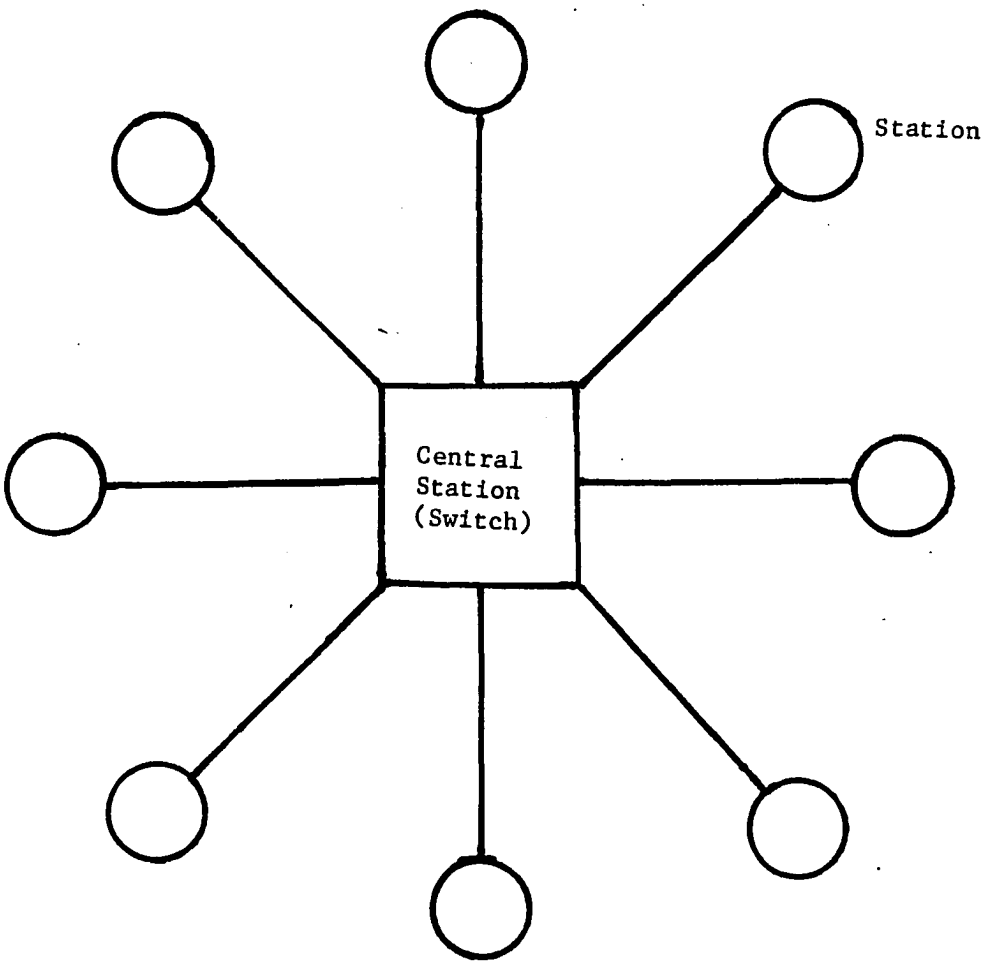


Figure 2-3. Star topology

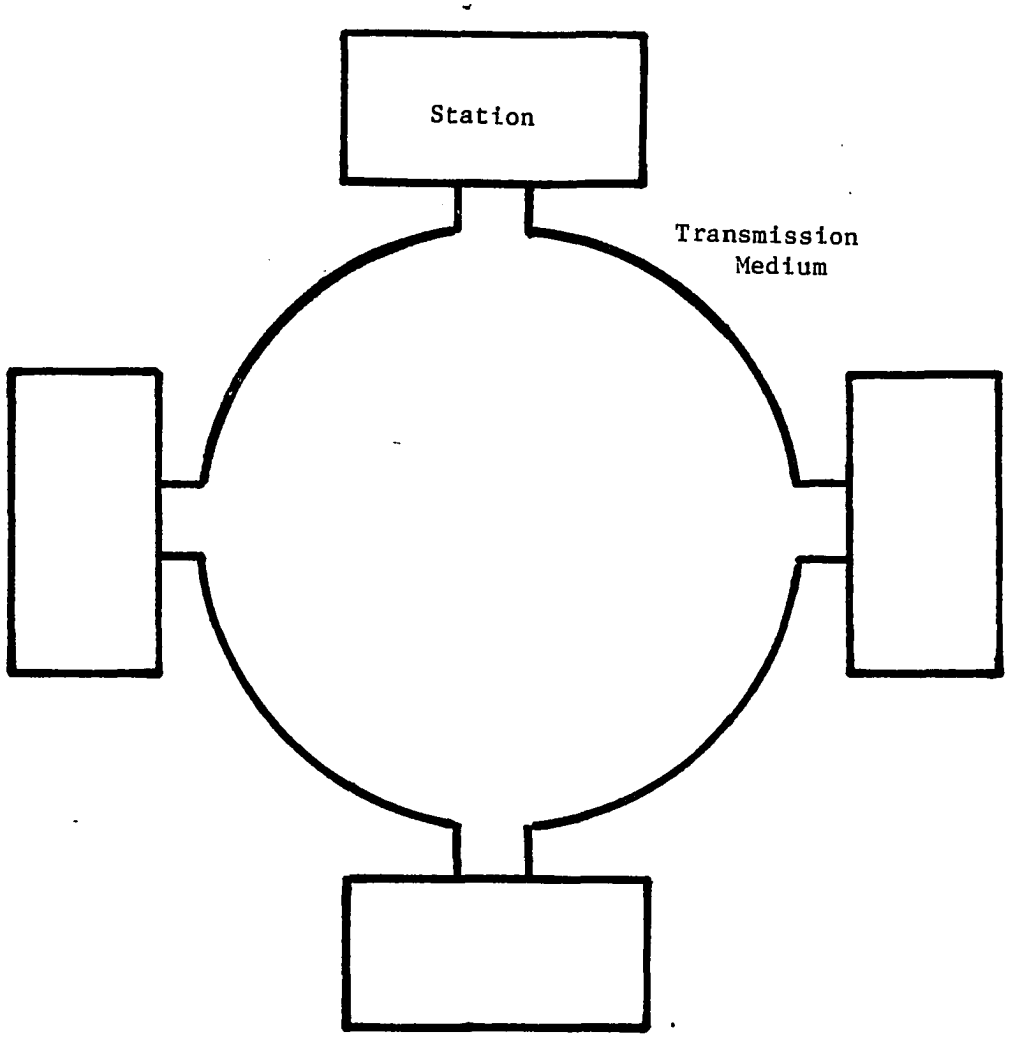


Figure 2-4. Ring topology

node. These measures result in increased complexity in the nodes. Ring networks will be discussed further later in this chapter.

Medium Access Control (MAC) Protocols

Several Medium Access Control Schemes have been proposed and analyzed. They are separated into the following classes:

Random Access Schemes [7-9]

Carrier Sense Multiple Access with Collision Detection (CSMA/CD) (Figure 2-5) is an example of such a scheme in which a station senses the channel until it is found to be free. If so, it transmits. If it detects a collision it broadcasts a jamming signal to alert all stations, waits a random amount of time and tries again. Hopefully it is eventually successful. Ethernet, a LAN supported by DEC, Intel, and Xerox is an example of a Network which uses CSMA/CD. It should be pointed out that CSMA/CD or random access schemes are, in general, implemented on broadcast or bus networks.

Polling schemes [10-11]

In such schemes a station has to receive a "permit" before it can transmit. Thus, there is no collision or contention. If the permit is always sent by a master-station then we have what is called a roll-call polling technique (Figure 2-6a). This is a centralized system. If the permit is sent by the station which last transmitted then we have a decentralized polling or hub-polling scheme (Figure 2-6b).

This is what is commonly called Token-Passing protocol. The "token" is the same as the "permit" described above. In fact the main

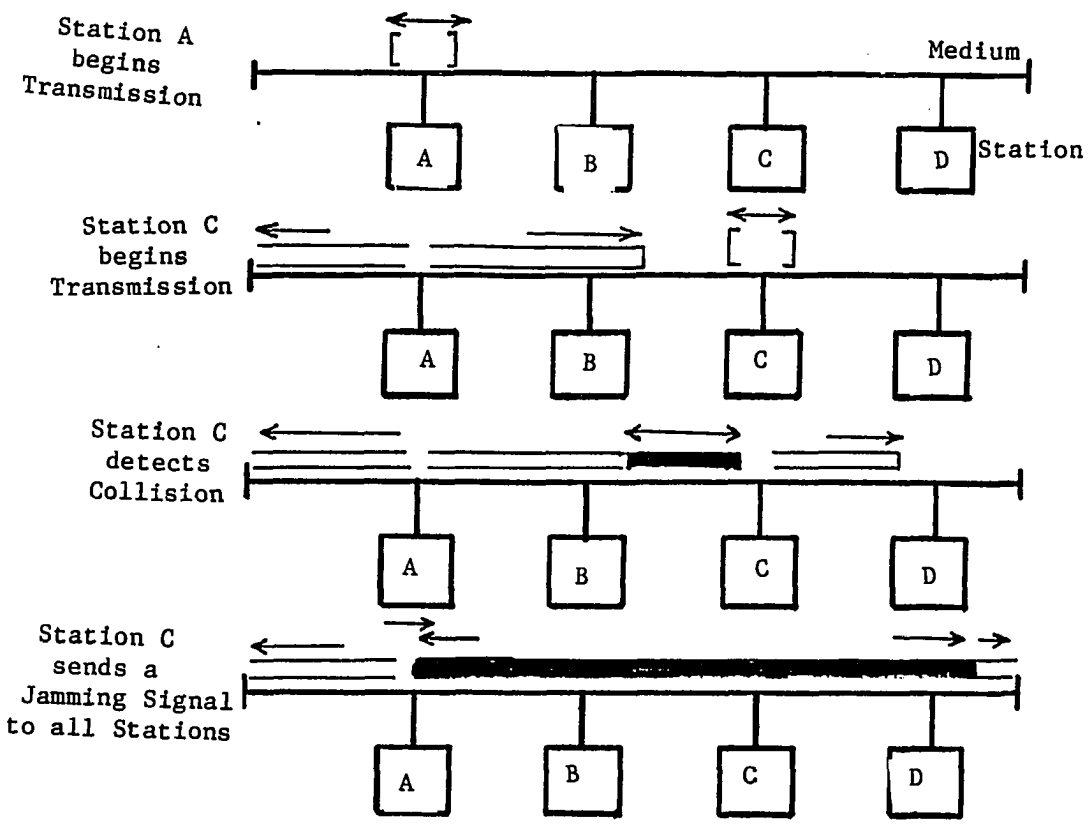
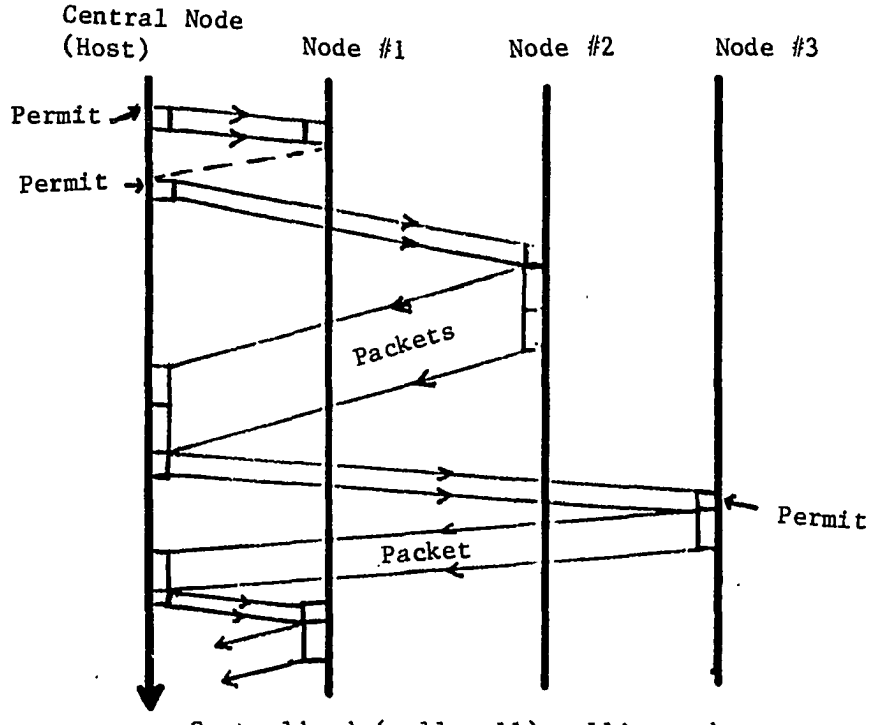
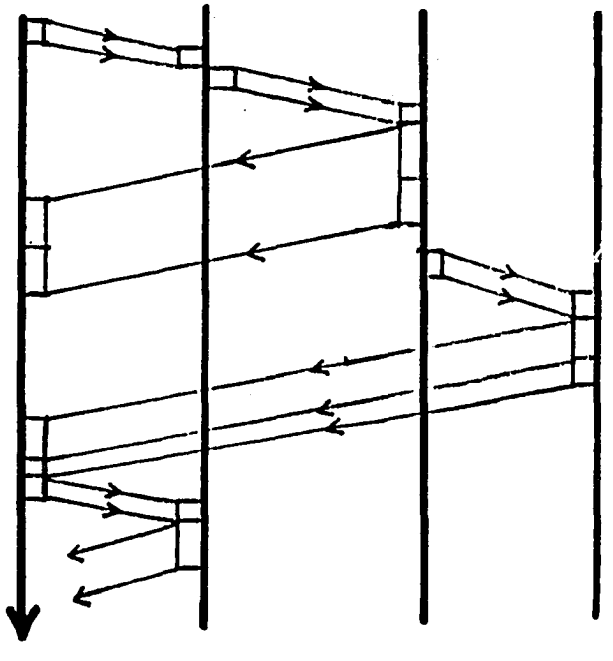


Figure 2-5. CSMA/CD operation: Random access



a. Centralized (roll-call) polling scheme



b. Decentralized (hub) polling scheme

Figure 2-6. Polling schemes

difference between hub-polling and token-passing is that in the former messages are always sent to a central station whereas in the latter messages can be sent to any station thus requiring them to be more intelligent. If the token-passing scheme is implemented on a bus topology we have a Token-Bus LAN [12]. See Figure 2-7. If it is implemented on a ring topology we have a Token-Ring LAN [13]. See Figure 2-8. There are several commercial LANs using the Token-Bus scheme. Although there is IEEE 802 standard for the Token-Ring protocol there are far less companies are known to market LANs supporting it than CSMA/CD.

The token ring is discussed further subsequently.

Ring access protocols [6, 13-18]

Recall that a ring network consists of a series of point-to-point channels between stations in a closed loop with messages travelling along a fixed route through network interfaces at each station. See Figure 2-9.

DCPR is a ring network. Therefore ring networks are discussed in some detail in this section.

The ring interface can be in one of three possible states: listen, transmit and bypass (Figure 2-10). In the listen state, each received bit is retransmitted with a small delay, ideally one bit time, required to allow the repeater to perform the following functions:

- scan passing bit stream for pertinent patterns (e.g., address, token)
- copy each incoming bit and send it to the attached station

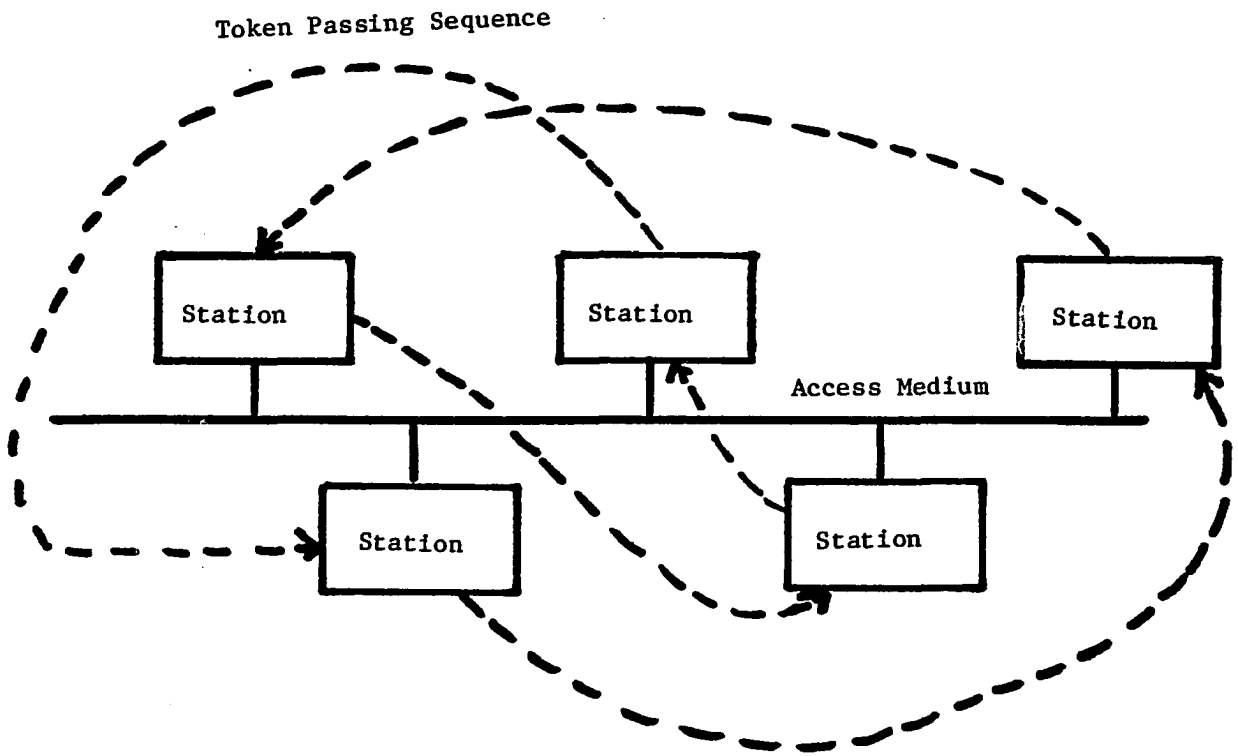
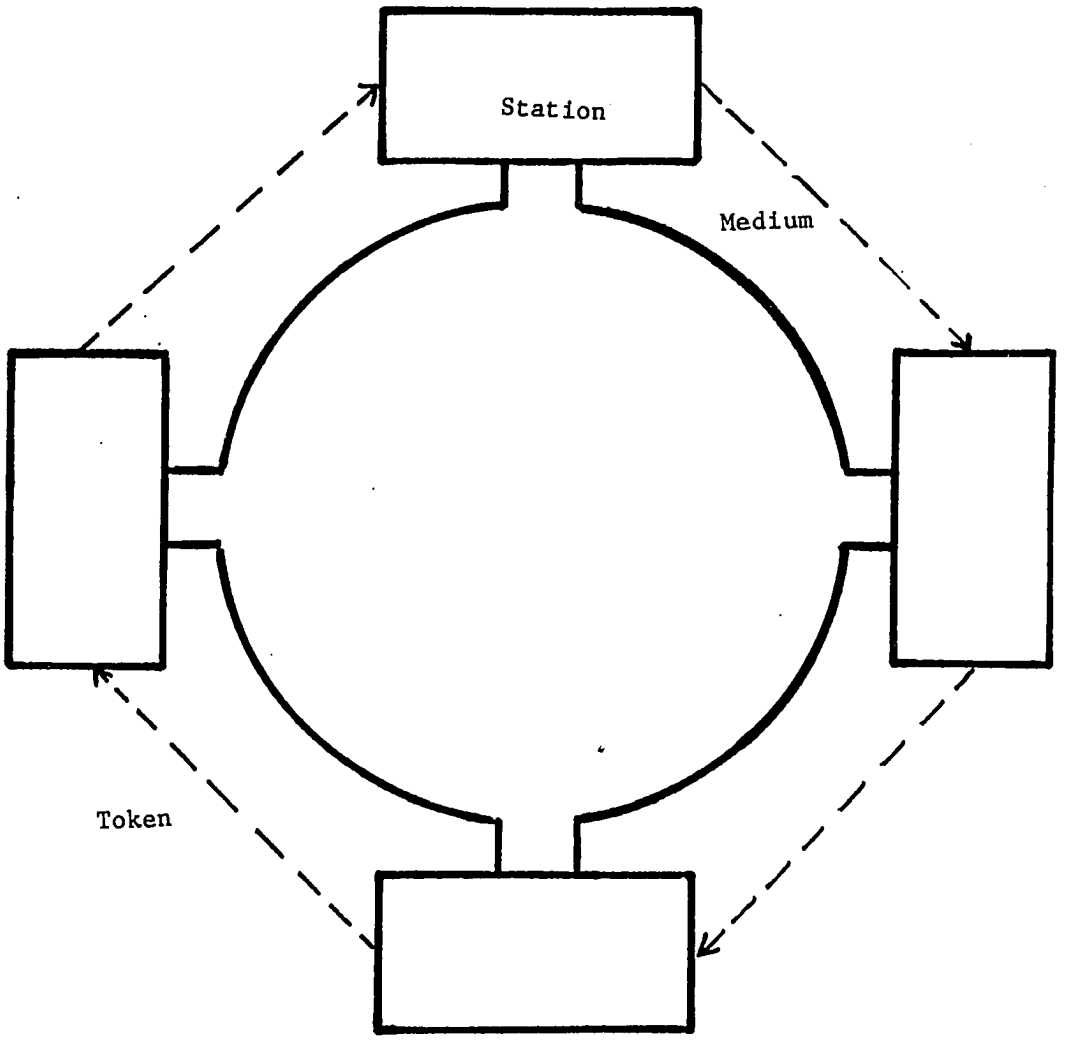


Figure 2-7. Token-bus: Logical ring, physical bus



Legend - - →: Logical Token-passing sequence

Figure 2-8. Token-ring: Physical ring, logical ring

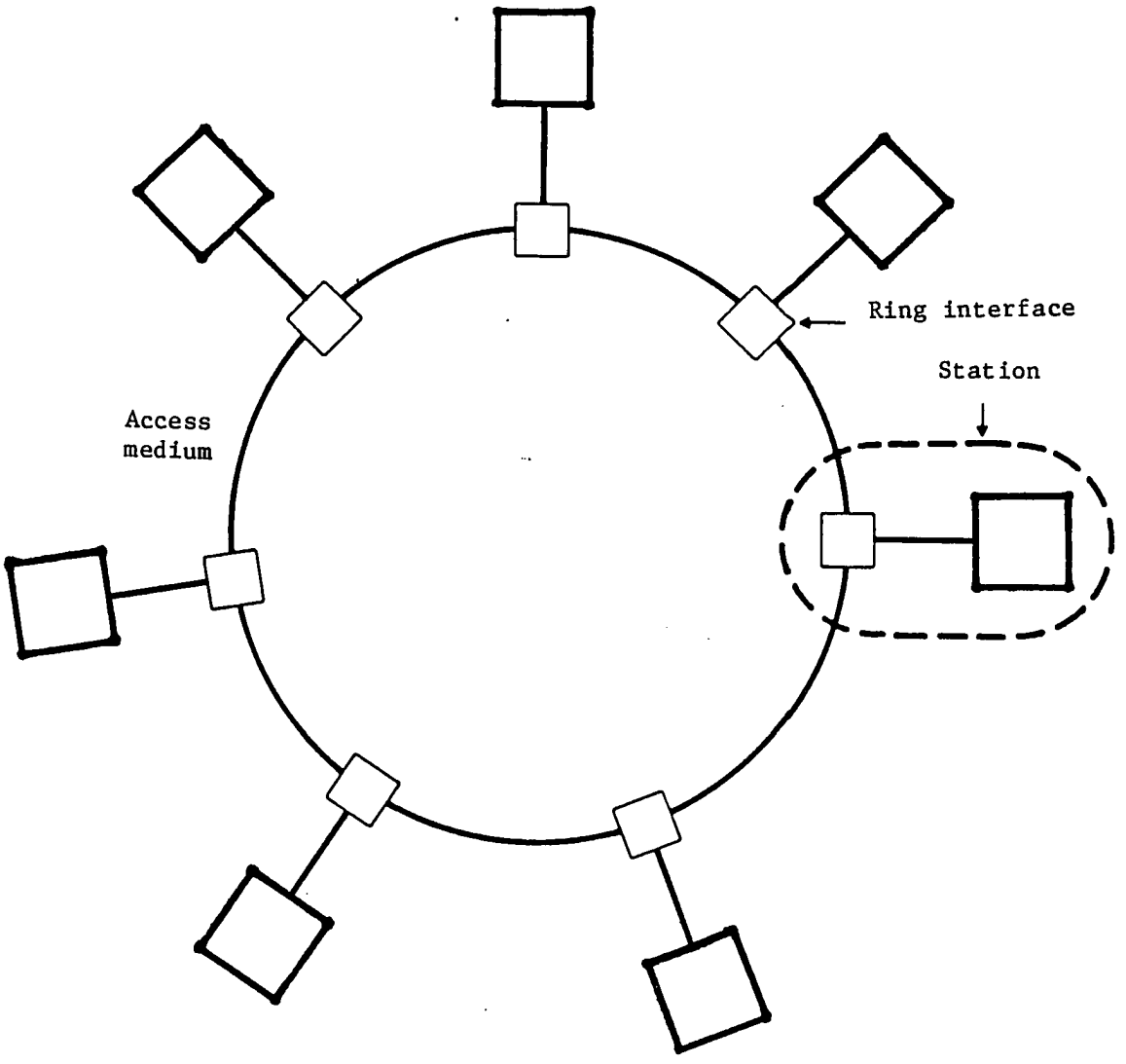


Figure 2-9. Ring network with interface units

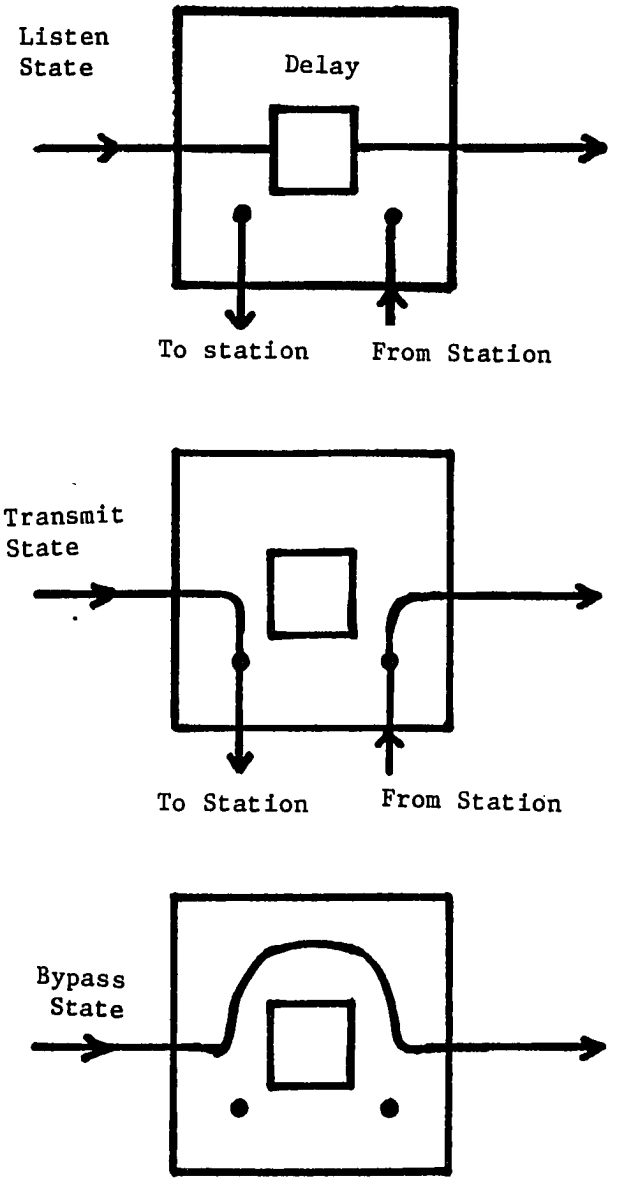


Figure 2-10. Ring interface states

- modify a bit as it passes by (e.g., to indicate an acknowledgment).

In the transmit state, the repeater receives bits from the station and retransmits them on its outgoing link. The bypass state can be used to improve performance by eliminating repeater delays for those stations that are not active on the network.

One issue related to ring networks is message removal. Owing to the closed-loop structure of ring networks a packet, once transmitted onto the network, must be removed, or the ring will become congested. This message removal can be done by either the source station or the destination station.

If the message is to be removed by the destination station, it requires that every message be delayed at every intermediate ring interface long enough to recognize whether or not it is destined for that interface. This delay must be long enough to recognize the destination address field (typically 8 or 16 bits). A disadvantage of this method is that imposing this 8 or 16 bit station delay may be undesirable in some applications. An advantage is that minimal bandwidth is used for acknowledgments because a short ACK message is used.

On the other hand, if the message is to be removed by the source station, there is no need for any additional delay at each station than is necessary to regenerate the pulses (usually, 1 bit delay). There are two advantages of this method: (1) automatic acknowledgment; (2) multicast addressing--one packet sent to multiple stations

simultaneously. A disadvantage of this method is that bandwidth may be wasted in carrying the message back from the destination station to the source station, especially if the message is very long.

The advantages of ring networks are:

- Ease of implementation;
- Long distance coverage because messages can be regenerated at each station;
- Efficiency does not degrade rapidly with load;
- Potential for fiber optics application;

The main disadvantages are:

- Its reliability;
- Propagation delay is proportional to the number of stations.

However, the reliability problem has recently been solved by researchers at both M.I.T. [16] and IBM [13].

Several Ring networks have been designed in the industry, universities, and research laboratories. Detailed discussions are given by Hammond and O'Reilly [6] and Dallas and Spratt [15].

Throughout the review of the literature of ring networks, it was found that the three major access protocols are: token ring, slotted ring, and the register insertion ring.

1) Token Ring

This access scheme has already been discussed. Here, we look at it in some detail. The token is a dedicated bit pattern which may be in one of two states: free or busy. A station that has a message to transmit can seize the free token (Figure 2-11a), change its state to

busy, and append to it its message and control bits. Each packet is transmitted onto the ring passing from station to station and is regenerated as it passes through each node (Figure 2-11b). Upon receipt of a packet a station verifies the packet's integrity and address but only copies messages addressed to it. Normally the source station removes the packet and issues new a free token to the next node (Figure 2-11c).

Generally, there are three variations of operation: multiple-token—multiple token, multiple packet; single-token--single token, multiple packet; and single-packet--single token, single packet. For multiple-token operation, the transmitting station generates a new token and places it on the ring immediately following the last bit of transmitted data. This operation permits several busy tokens and one free token on the ring at a time, hence the name, multiple-token.

The single-token requires that a transmitting station wait until it has erased its own busy token before generating a new free token.

For a single-packet operation, a station does not issue a new free token until after it has circulated completely around the ring. The IEEE 802 standard specifies the single-token scheme.

2) Slotted Ring [17]

In this scheme a sequence of fixed length slots travel around the ring with each slot marked empty (free) or busy (full) (Figure 2-12). A ready station (i.e., a station with a message to transmit) can only transmit in an empty slot. The slot's status can be changed from busy to empty by the source station or the destination station. The Cambridge

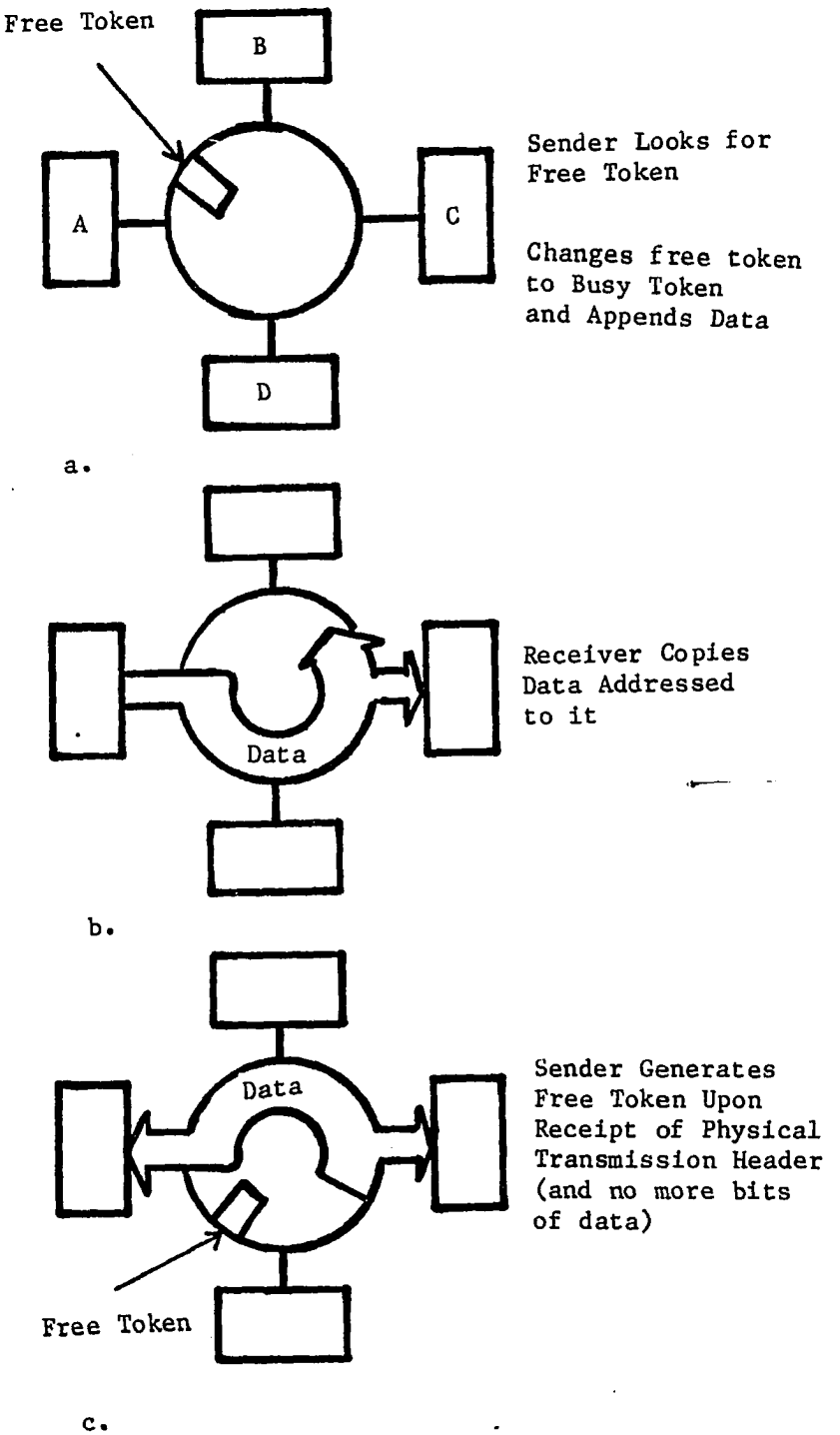


Figure 2-11. Token ring operations (single-token scheme)

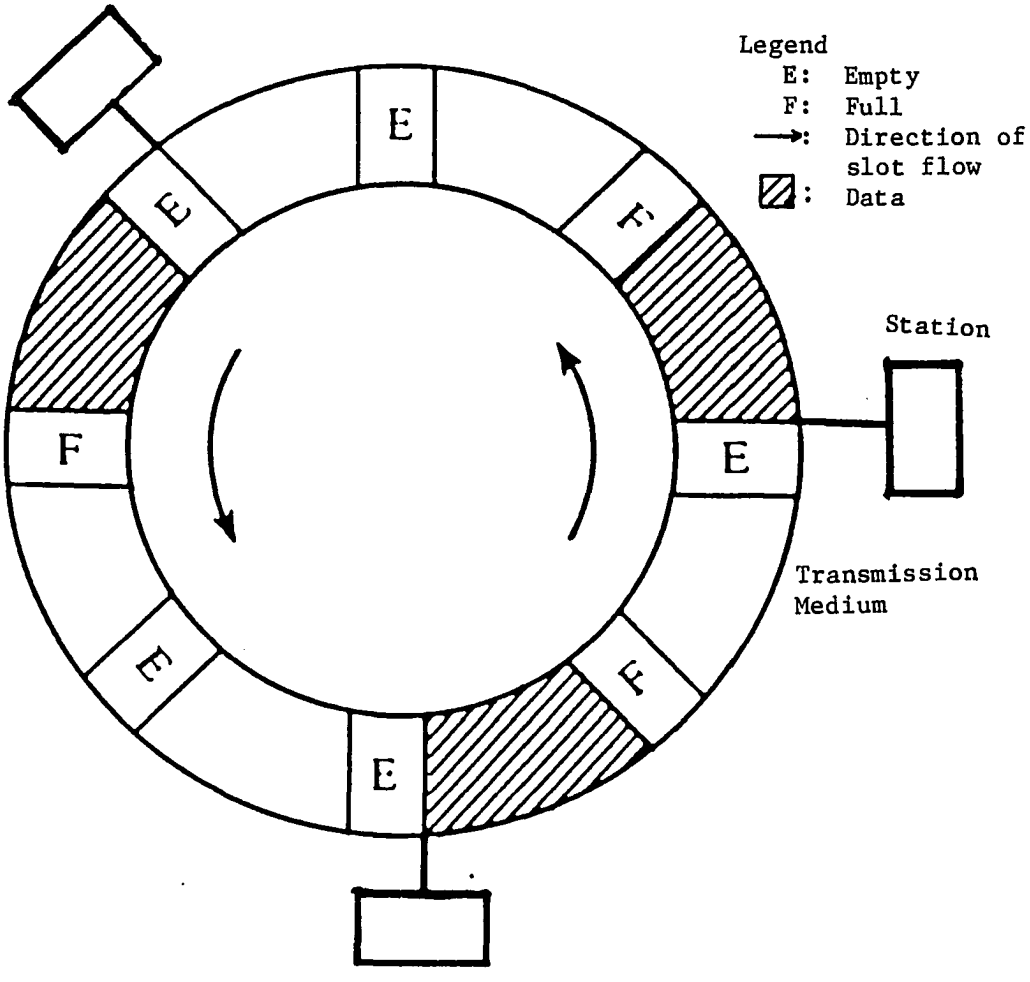


Figure 2-12. Slotted ring

Ring [17] is an example of a slotted ring. Although slotted rings are not popular in the United States and hence are not being considered as a standard by the IEEE 802 Committee on LANs, they are very popular and heavily supported by European computer manufacturers.

The main disadvantage of slotted rings is:

- waste of bandwidth due to large overhead-to-data ratio (e.g., in Cambridge ring this ratio is 22/16) and the fact that messages are broken up into fixed length minipackets.

The principal advantages are:

- Prevention of ring hogging;
- Simplicity of implementation.

3) Register Insertion Ring

The scheme grew out of research at Ohio State University [18]. It makes use of two shift registers at each station (Figure 2-13): an insertion buffer and a transmit buffer. The buffer is equal in size to the maximum packet length. The operation is as follows.

Initially when the ring is idle and there are no messages to send the input pointer points to the rightmost bit of the insertion buffer. When a bit arrives from the ring it is shifted into the insertion register with the input pointer moving 1 bit to the left. As soon as the address field has arrived, the interface will determine if it is the addressee or not. If so, the rest of the packet is diverted to the station removing the packet from the ring. The input pointer is then moved to point to the right most position. Note that the packet can also be retransmitted onto the ring if the source station is the one to

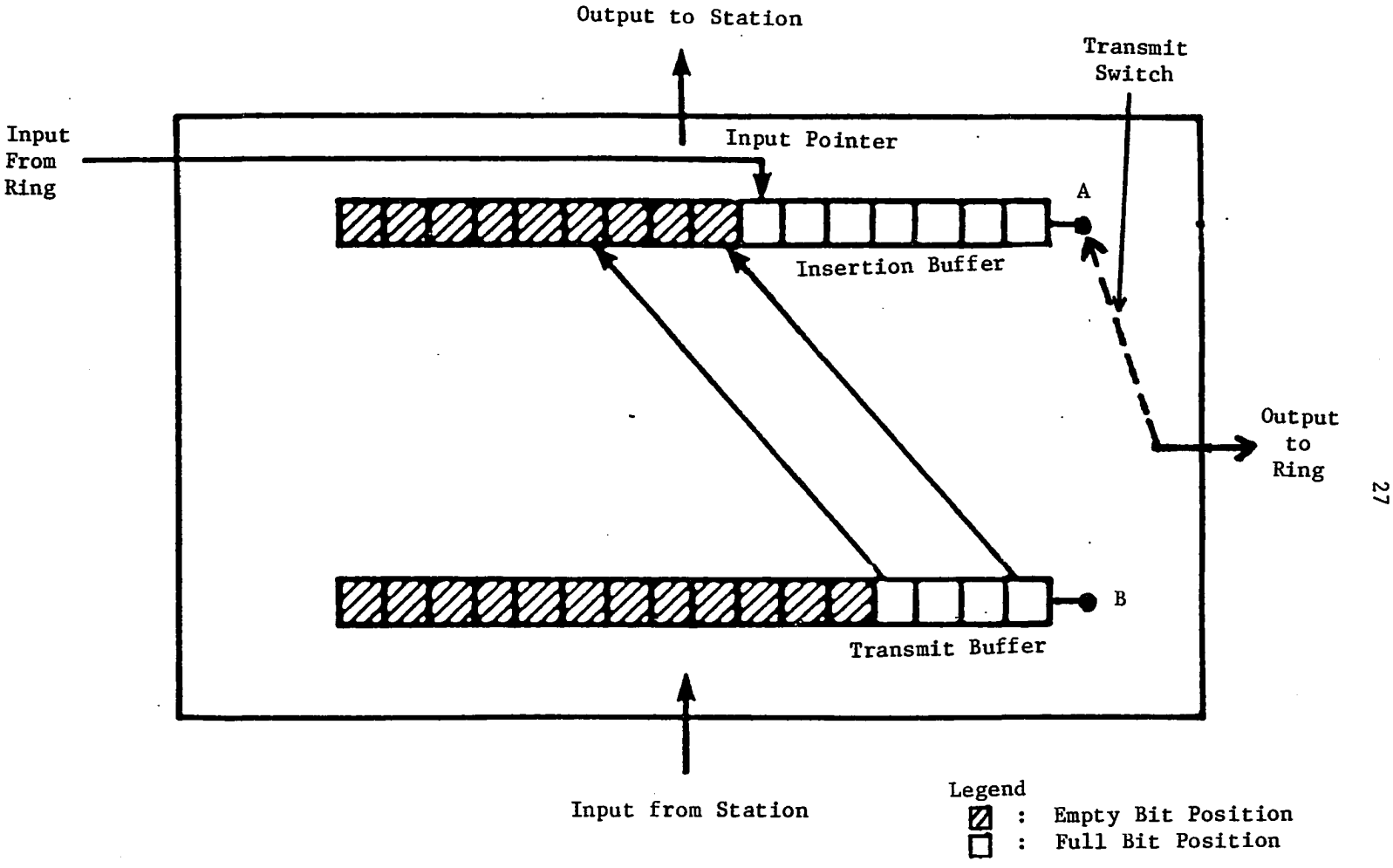


Figure 2-13. Register insertion ring

remove the packet although this is not normally done.

On the other hand, if the station is not the addressee, the packet is shifted bitwise as it arrives onto the ring.

Now consider output from the station. A packet to be transmitted is placed in the transmit buffer. The transmit switch is flipped and data is output from this buffer onto the ring if, and only if two conditions occur: (1) There is a packet waiting in this transmit buffer and, (2) the number of empty bits in the insertion buffer is at least as large as the size of the packet waiting. Alternatively, the waiting packet can be parallel transferred to the empty portion of the insertion. This does not require switching [19].

Disadvantages of this scheme are:

- Recognition of the address bits whether packets are removed by the sending station or the receiving station;
- Complex interface.

The principal advantages are:

- Maximum ring utilization;
- Prevents ring hogging.

4) Other rings

These are normally hybrid schemes examples of which are the Contention Ring [20] which is a Token-CSMA/CD hybrid and partial insertion slotted ring [21] whose scheme is obvious.

A detailed discussion of ring networks can be found in Refs. 6, 15, and 20.

Other Schemes

Although these other protocols are not as popular as the ones already mentioned, they have application in Local Area Networks. Some of them (the hybrid schemes) are too complex to be practical at the moment. They can be grouped in the following classes.

Fixed assignment schemes (FDMA and TDMA) [22,23]

In Frequency Division Multiple Access (FDMA) each user is confined to a fixed portion of the bandwidth assigned a priori. In Time Division Multiple Access (TDMA) each user is assigned a predetermined time slot yet owns the whole bandwidth.

Reservation schemes [24]

In this scheme the channel is reserved by a user before it can be used. Thus as is the case in some other schemes the purpose is to avoid packet collision. Conceptually there exists reservation and data subchannels which can be separated either physically or logically. Normally, if they exist separately, the reservation subchannel is of lower speed. TDMA, FIFO or even random access schemes can be used to access the reservation subchannel.

Hybrid schemes [25-29]

These hybrid schemes are so called for obvious reasons. It is also difficult to include them in the above ones although one could arguably do so. They come under such names and descriptions as Group Random Access Schemes [8], Reservation Decision Tree Algorithm [29], the Urn Protocol [30], Adaptive Polling [27], Reservation Upon Collision [31],

Distributed Scheduling Conflict-Free Multiple Access (DSMA) [26], and Multiple-Level-Multiple-Access (MLMA) [28], to name just a few. A summary of most of the MAC schemes can be found in [30-33].

It should be mentioned that apart from the protocols for ring networks, all the other protocols were designed for broadcast networks or bus systems. As such, they are adaptable for satellite or packet radio networks which are broadcast networks.

We need to mention certain disadvantages of one of the more popular schemes, the random access scheme, of which CSMA/CD is an example. Two of these disadvantages are redundant traffic due to packet collision when the LAN is operating under high load, and unpredictable access delay for packets.

It is especially because of the second disadvantage that a lot of conflict-free multi-access methods have been proposed, for, in many applications such as voice/data networks where a real-time deadline should be met there is the need to have deterministic delays. In fact, recently Ethernet, which uses CSMA/CD, has been modified to include message-based priority functions so as to accommodate the practical real-time situations [34]. The main advantage of CSMA/CD is that it offers the least delay under light load. This is expected because under light load only one station is likely to transmit at a time. Thus there is no redundant traffic whatsoever. The token passing schemes operate better than the random access schemes under heavy load.

Ring Networks are becoming more and more popular because they can make use of high speed, reliable and error-free fiber optic channels.

It is currently impractical to implement a multipoint broadcast network using fiber optics [14].

CHAPTER 3. THE DISTRIBUTED CHANNEL-SENSE PRIORITY RING (DCPR)

Introduction

In this chapter, the Distributed Channel-Sense Priority Ring (DCPR) and its access protocol will be described and a formal state-oriented specification using the State Architecture Notation (SAN) will be presented. The simulation model for DCPR, which will be presented in Chapter 5 for obtaining performance characteristics of the protocol will be derived directly from the state specification.

DCPR is a Distributed Channel-Sense Asynchronous Non-Preemptive Static Priority Ring for Local Area Computer Communications (see Figure 3-1). It is so called for the following reasons:

Distributed: because we have a system in which the users are connected to stations which are separated by point-to-point communication channels.

Channel-Sense: because before any station can send a message it has to sense the channel first to determine its state. If the channel is free it will send a request to the other stations informing them that it wants to use the channel.

Asynchronous: because there is no central controller to tell what the stations must do (i.e., synchronize the activities of the stations). Each station implements the same algorithm. Moreover, when a station gains access to the channel, it sends a message whose length may be different from others. This is different from other protocols such as a slotted ring or token-ring with non-exhaustive discipline

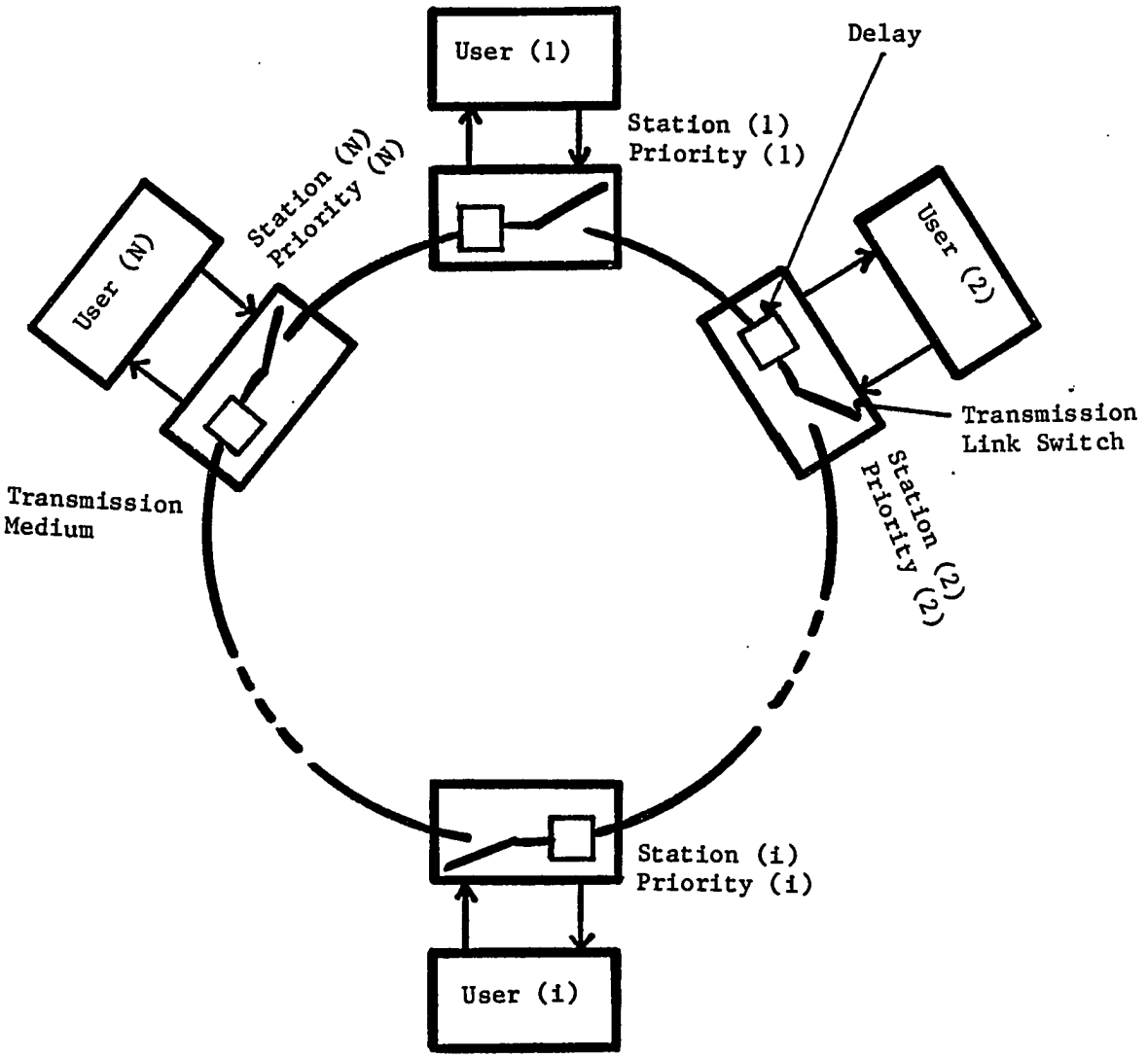


Figure 3-1. The DCPR system structure

where fixed length packets are sent all the time.

Non-preemptive: because when a station is involved in a message transmission no station (including higher priority ones) is allowed to interrupt the transmission of the message. This is unlike preemptive priority schemes where a higher priority station can interrupt a station of low priority such that it will delay transmission till a later time. Normally, the message is broken into fixed length packets in such systems so only transmission of subsequent packets are preempted (see [35]).

Static priority: because each station has a fixed priority. They are used to resolve conflicts in case more than one station is ready to send a message. The fixed priorities do not provide fairness in certain applications so dynamic priorities will be suggested as alternatives in Chapter 8.

Ring: because DCPR protocol is implemented on a ring topology.

The protocol - word description

a) Each station has an address which specifies its priority. The lower the address the higher the priority.

b) A station which has a message to send senses to see if the channel is idle. If it is not it waits until it is idle.

c) When the channel becomes idle, it opens its link and sends a

"Request For Reservation" (RFR) which is essentially the station's address, to the next station.

d) If any station receives an RFR, it checks to see if it is a higher priority request. If so it closes its link regardless of whether it has sent an RFR by itself or not.

e) If a station receives a lower priority RFR, it keeps its link open if it has a message to send. Otherwise, it closes the its link.

f) If any station, expecting to receive its RFR back, does not receive it after a specified time interval (worst case round trip transmission time of an RFR) it goes to step c. Note that assuming no errors occur the only stations in this category are the higher priority stations.

g) It is clear that after steps c, d, e, and f are repeated for a while only the highest priority competing RFR will be received back by the appropriate station because all the links will have been closed by then.

The winning station transmits one message and waits for an acknowledgment. After this is received the transaction is complete and the channel is up for grabs so we go back to step b.

The protocol-algorithmic description

The following algorithm written in PASCAL-like notation is a more concise definition of the DCPR protocol.

```
1. REPEAT
       Sense Channel
   UNTIL channel_is_idle;
```

```
   Open Link;
2. IF RFR_rcvd THEN Wait-for-Message & Close Link
   ELSE IF Rdy THEN BEGIN
      Send RFR; Start Timer; WAIT
      IF higher_Priority_RFR_rcvd THEN Close Link
      ELSE IF lower_Priority_RFR_rcvd OR timeout THEN Go To 2
      ELSE IF my_RFR_rcvd THEN BEGIN
         Send_Message
         Wait for ACK
         IF ACK_rcvd THEN Go To 1
   END IF
```

DCPR example

We next show by example a typical DCPR protocol scenario under the following assumptions:

A1. A total of 4 stations are connected on the ring.

A2. Stations becoming ready after the reservation or scheduling process has begun do not take part in the bidding for transmission. [Note that although the real DCPR will not operate in this manner this assumption has been made in order not to complicate the example which is supposed to illustrate only the essence of the DCPR algorithm.]

A3. The Ring operates in a unidirectional manner (i.e., information moves in one direction). In reality that is how Ring Networks normally operate.

A4. Stations are equally spaced apart timewise. In other words,

the propagation delay between any station, i, and the adjacent station, 1 + imodN, (where N is the total number of stations) is the same for all $i \in \{1, \ldots, N\}$.

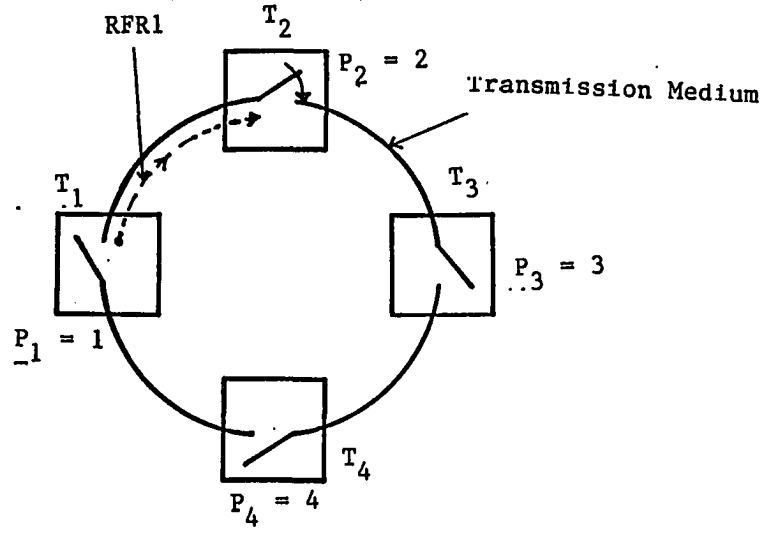
Let the stations be arranged in decreasing order of priority such that station T_i has priority $P_i = i$. In other words, station T_1 will have priority $P_1 = 1$ (highest priority) station T_2, priority $P_2 = 2$, ..., station T_4, priority $P_4 = 4$ (lowest priority). This arrangement is seen in Figure 3-2.

Case 1: Only station T_1 has a message to transmit (ready).

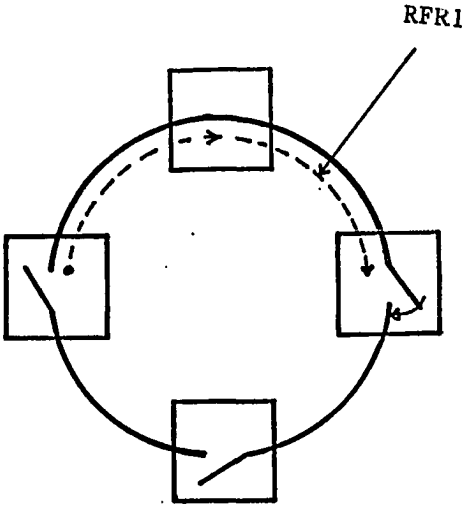
Initially after detecting the channel idle all stations open their links. Station T_1 first sends its RFR to Station T_2 and starts a timer. Station T_2 receives the RFR and closes its link (see Figure 3-2a). Station T_1 then times out and then sends the RFR again. This time the RFR passes through Station T_2 uninhibited until it reaches station T_3. The latter station then closes its link (Figure 3-26). T_1 times out again and sends the RFR again (Figure 3-2c). Eventually (as in Figure 3-2d) all links except that of T_1 are closed so T_1 will receive its RFR back. At this point T_1 knows that all the links are closed meaning all the other stations are waiting for a message. This is the end of the reservation (or scheduling) period. During the transmission period T_1 transmits a variable length message.

Case 2: Similar to Case 1 except that all stations are ready.

Again after detecting idle channel all stations open their links and then send an RFR to the adjacent station (see Figure 303). Since all the receiving stations, with the exception of Station T_1, are lower

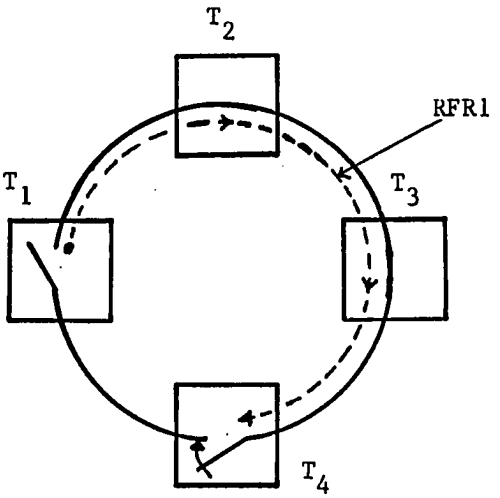


a. Station T_1 sends its RFR to T_2; T_2 receives it and closes its link; T_1 times out

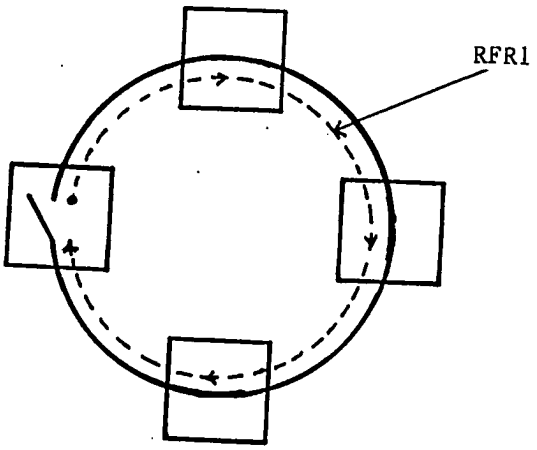


b. After timeout T_1 sends RFR again; T_3 receives it and closes its link; T_1 times out

Figure 3-2. DCPR example: Case 1 (station priority in decreasing order; only station T_1 ready)



c. After timeout T_1 sends RFR again; T_4 receives RFR1 and closes its link; T_1 times out



d. After timeout T_1 sends RFR1 again; T_1 receives RFR1 back

Figure 3-2. (continued)

priority stations they close their links (Figure 3-3a). According to the protocol T_1 sends its RFR again as soon as it receives a lower priority RFR. Because all stations are waiting for a message (i.e., links closed), it receives its RFR back (see Figure 3-3b).

Case 3: Only stations T_1 and T_3 are ready.

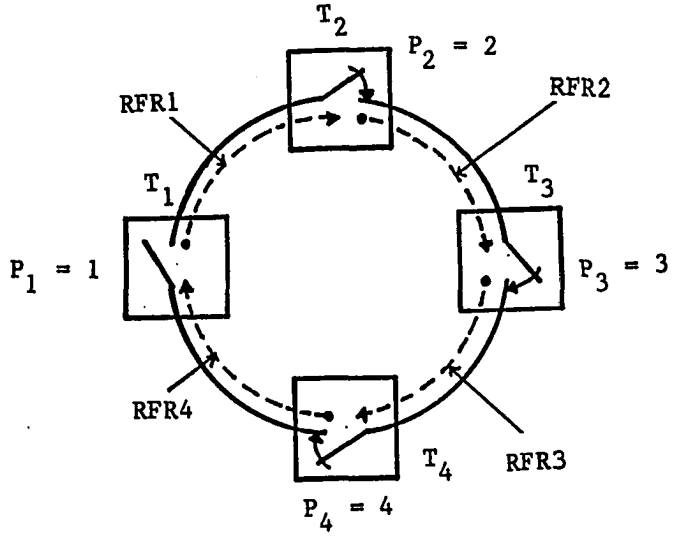
From Figure 3-4, we see that T_1 gets its RFR back after three tries. Notice that for all the three cases the time for reservation is different. This shows that the reservation time is a function of the number of ready stations in a particular configuration (i.e., priority ordering). We will discuss more about the reservation time later in Chapters 4, 5, and 6.

As will soon be seen, although the DCPR protocol seems to be quite simple it becomes complex when the possibility of communication errors are incorporated and also when we look at it in detail. This necessitates the need to have a systematic and formal way to present the protocol at the detailed level. This requires the use of formal models. This is the subject of the next section.

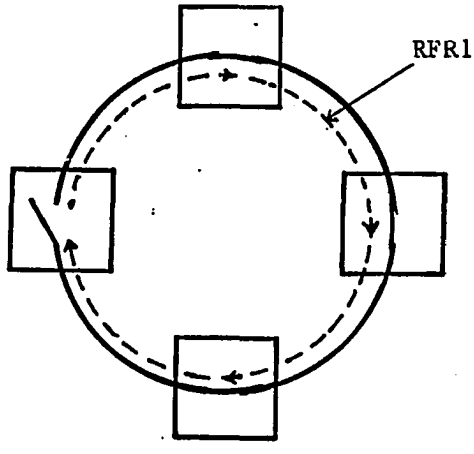
The protocol - a formal model

In this section we develop a formal model of DCPR for two reasons: 1) to provide a detailed, clear description of DCPR for benefit of the reader, 2) to provide a model which, with modification, will be used in Chapter 4 for analytical studies and Chapter 5 for simulation studies.

The need to clearly specify a protocol and its intended behavior is widely acknowledged [36]. The paramount reason for this need is because a protocol architecture involves independent processes (or users)

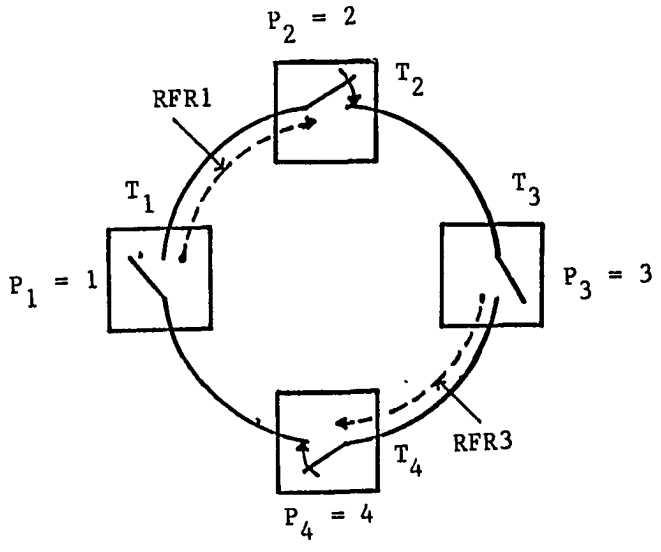


a. T_1 sends RFR1 to T_2; T_2 sends RFR2 to T_3; T_3 sends RFR3 to T_4; T_4 sends RFR4 to T_1; T_2, T_3, and T_4 receive higher prioty RFR's so they close their links

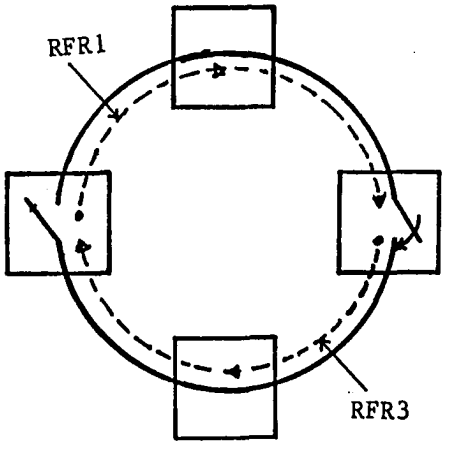


b. T_1 sends RFR1 again after receiving one from T_4 and gets it back later

Figure 3-3. DCPR example: Case 2 (station priority in decreasing order; all stations ready)

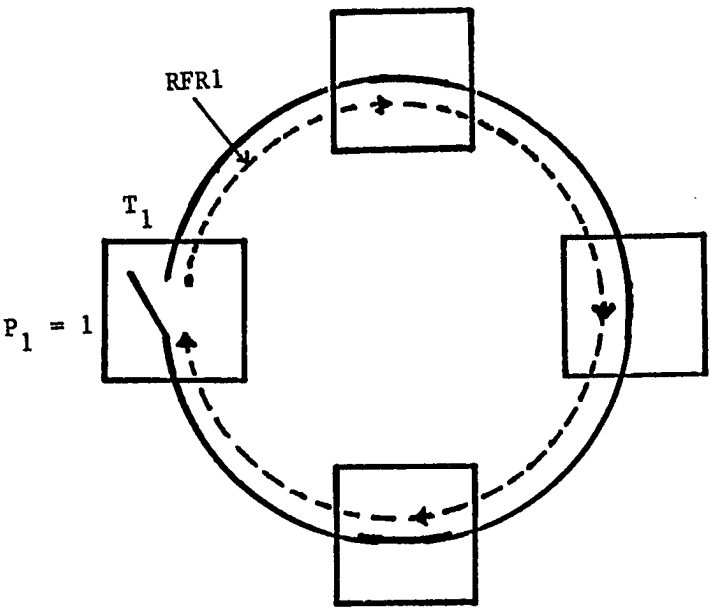


a. T_1 sends RFR1 to T_2; T_3 sends RFR3 to T_4; T_2 and T_4 close their links; T_1 and T_3 timeout



b. After timeout T_1 and T_3 send RFR1 and RFR3, respectively, again; T_1 receives RFR3; T_3 receives RFR1 (a higher priority RFR) so it closes its link; T_1 times out

Figure 3-4. DCPR Example: Case 3 (station priority in decreasing order; only station T_1 and T_3 ready)



c. After timeout T_1 sends RFR1 again and gets it back later

Figure 3-4. (continued)

running concurrently on physically separated systems that can communicate only via imperfect message channels. The reader will notice that this is precisely the case with DCPR.

With the help of a few figures, a formal specification of DCPR will be presented.

A formal specification is nothing but a formal description which specifies unambiguously the behavior of the protocol.

A programming language like PASCAL is an example of a formal description technique that can be used to specify some type of algorithm formally. Such descriptions, as opposed to English-language descriptions (also called prose), are normally unambiguous. The algorithmic description of the DCPR protocol presented earlier in this chapter is an example of such unambiguous descriptions.

A state diagram is another example of a formal description technique that can be used to specify the behavior of a sequential circuit. In the DCPR specification, a formal state-oriented approach is used.

The specification method that is used here is that of the State Architecture Notation (SAN) of Piatkowski [37]. The difference between SAN and other state transition descriptions is that it is based on the concept that any protocol can be described as a well-defined set of interconnected canonical components. Most of these components are likely to be Finite State Machines (FSMs). Others are Clocks, Combinational Functions, Queues, Delays, and so on. What makes the author prefer this over other methods is: 1) It is heavily graphic-

oriented, 2) the interconnections of components are well-specified, 3) input/output variables of components have well-defined attributes, 4) it requires the use of both block diagrams of the system and graphical description of the internal representation of components.

Figure 3-5 shows a block diagram of the DCPR system. Figure 3-6 is the block diagram of a single DCPR station showing the DCPR station manager or Medium Access Control (MAC) manager and associated timers. This manager is the decision-maker which actually emulates the DCPR protocol. In an implementation this manager may be a programmable processor, a hardwired processor or a combination of both. Figure 3-7 is the SAN model showing the internal behavior of the MAC manager. This model is actually a type of state transition diagram. This figure, together with the block diagrams in Figures 3-5 and 3-6 specifies completely and unambiguously the behavior of the MAC manager in each DCPR station.

In Figure 3-7 the vertical lines represent states and the horizontal lines, transition to states. Just above each horizontal line is the event which causes the transition and below it is a resulting output event, if any. These events are either pulsed inputs or outputs as the case might be. They may also be a combination of pulsed and static inputs or outputs. To clarify matters, we state that a pulsed variable is an instantaneous variable whereas a static variable is not.

Figure 3-8 together with Table 3-1 is hopefully a more familiar description of everything in Figure 3-7.

Legend:

UPM = Unspecified Protocol Machine

DELP = Pulsed Delay (SAN canonical component)

Figure 3-5. DCPR system block diagram (graphical SAN)

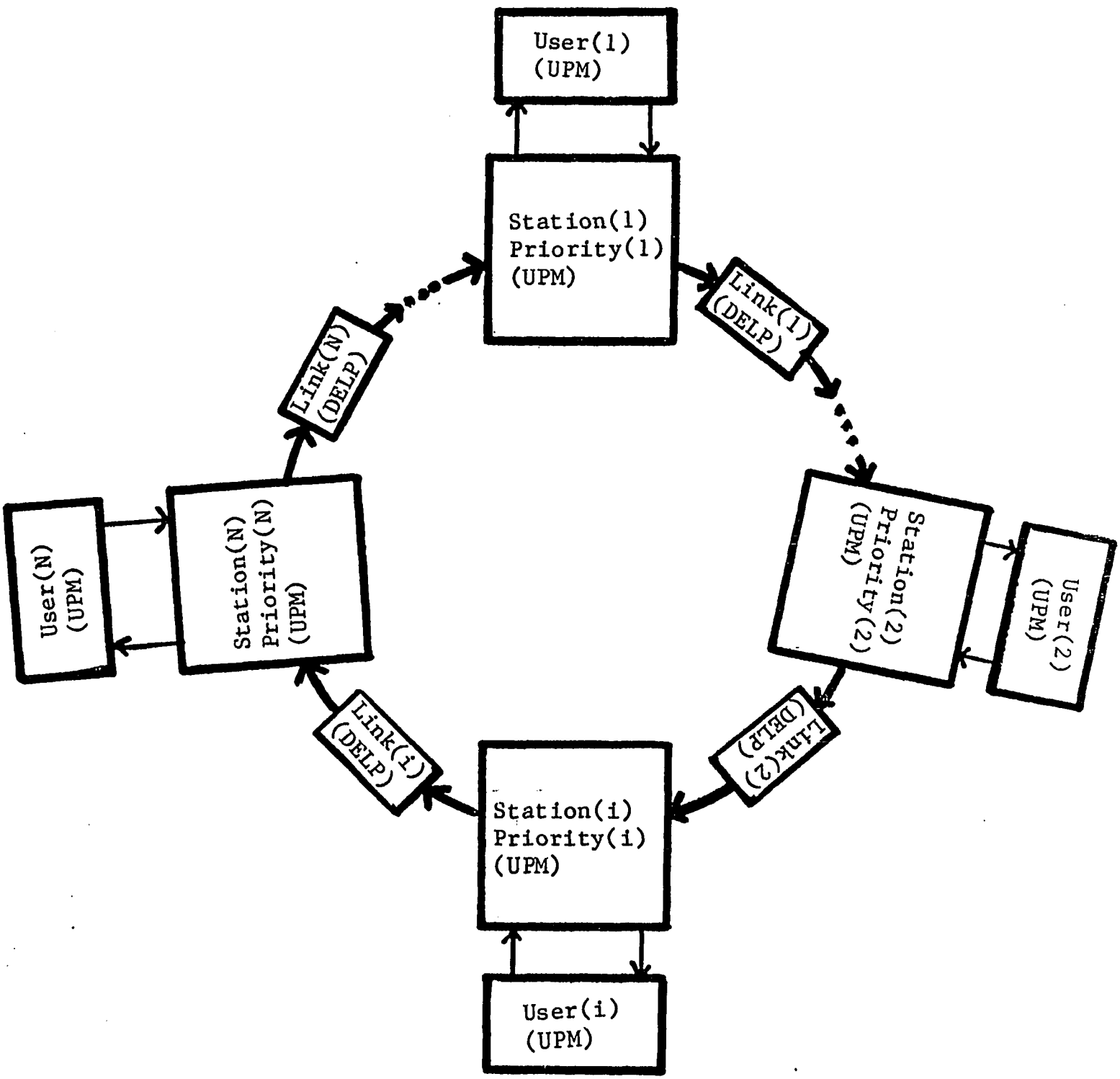
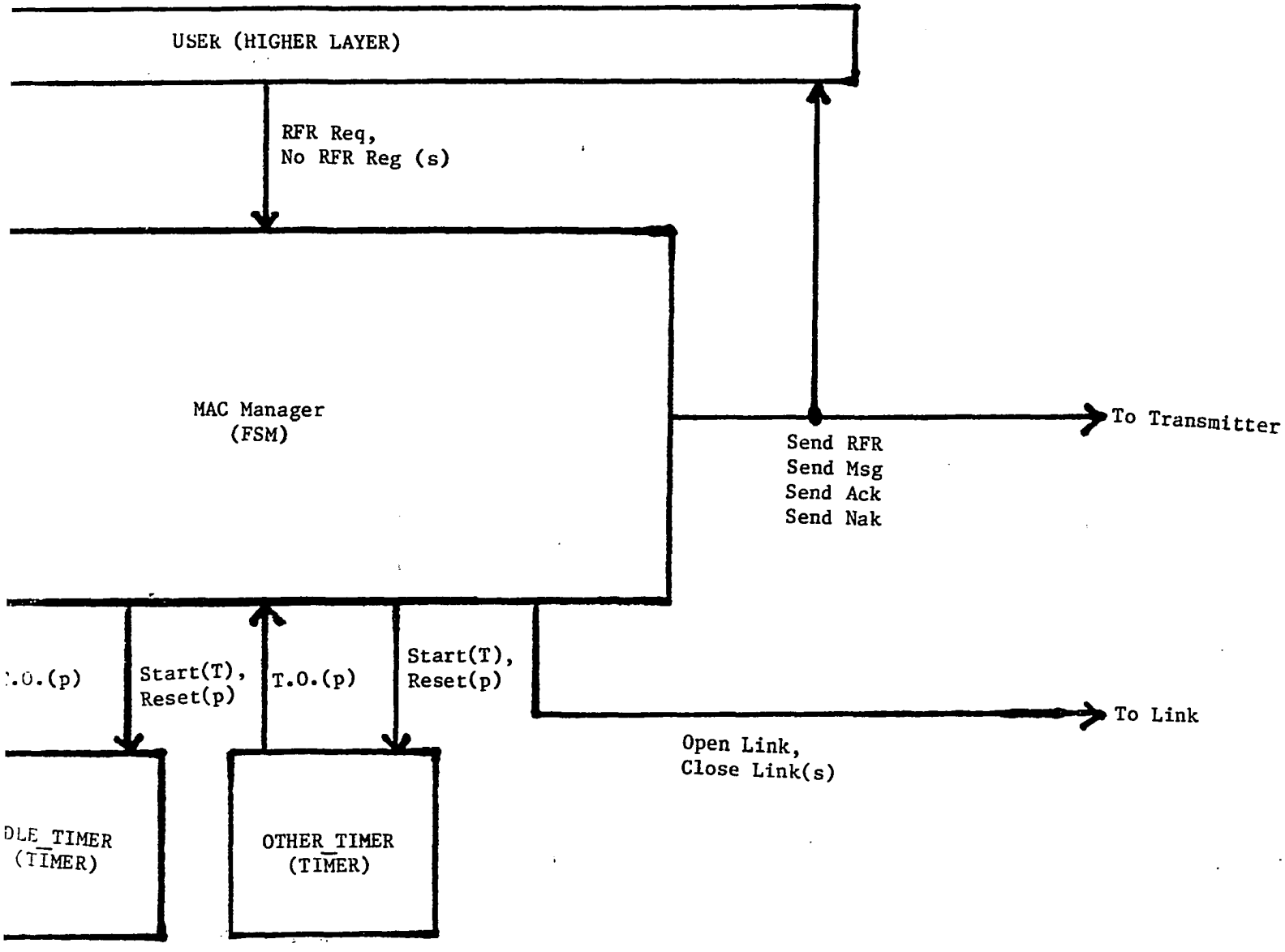
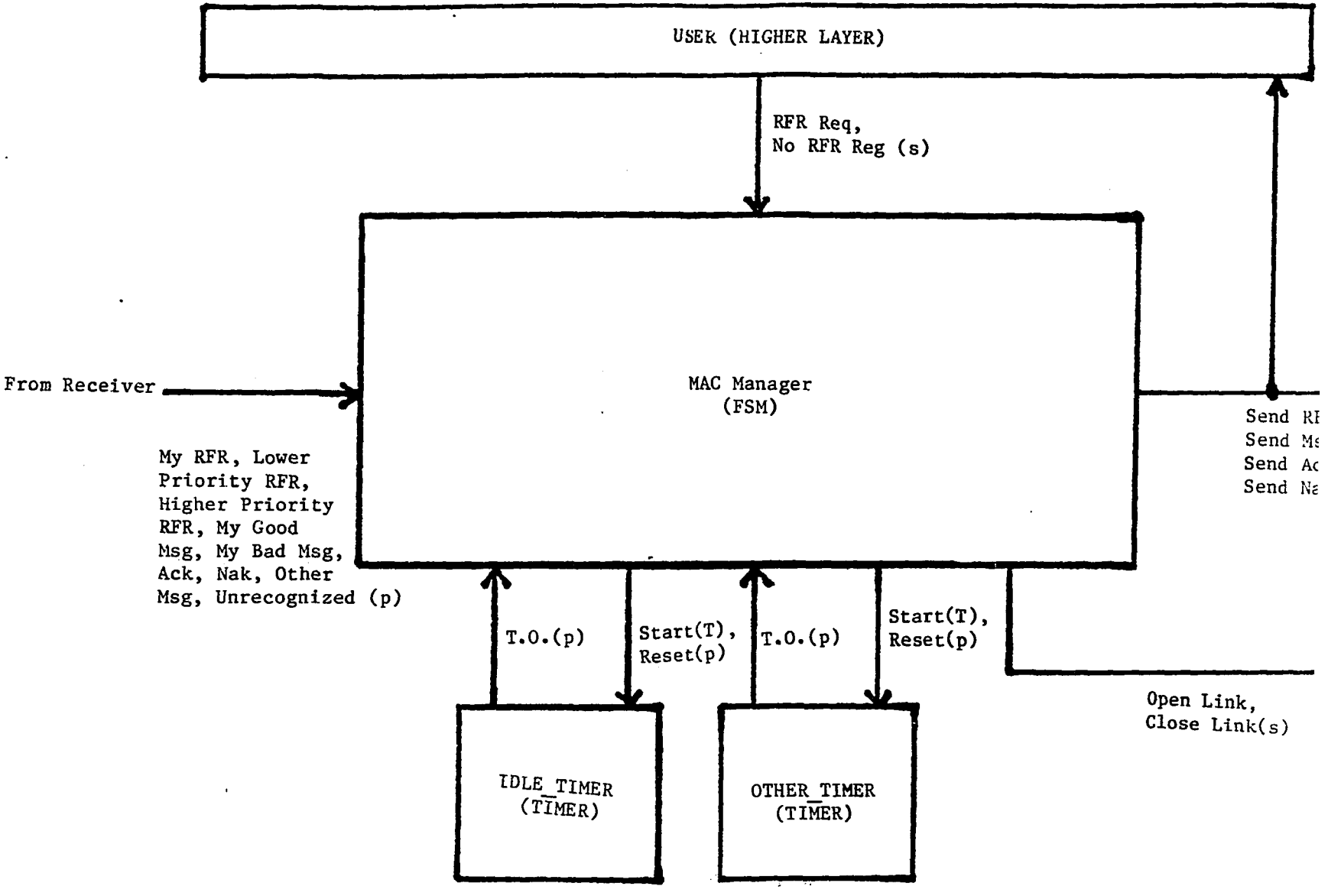

User(1)
(UPM)
Station(1)
Priority(1)
(UPM)
Link(1)
(DELP)
Station(2)
Priority(2)
(UPM)
User(2)
(UPM)
Link(2)
(DELP)
Station(i)
Priority(i)
(UPM)
User(i)
(UPM)
Link(i)
(DELP)
Station(N)
Priority(N)
(UPM)
User(N)
(UPM)
Link(N)
(DELP)

Figure 3-6. DCPR station: MAC manager context

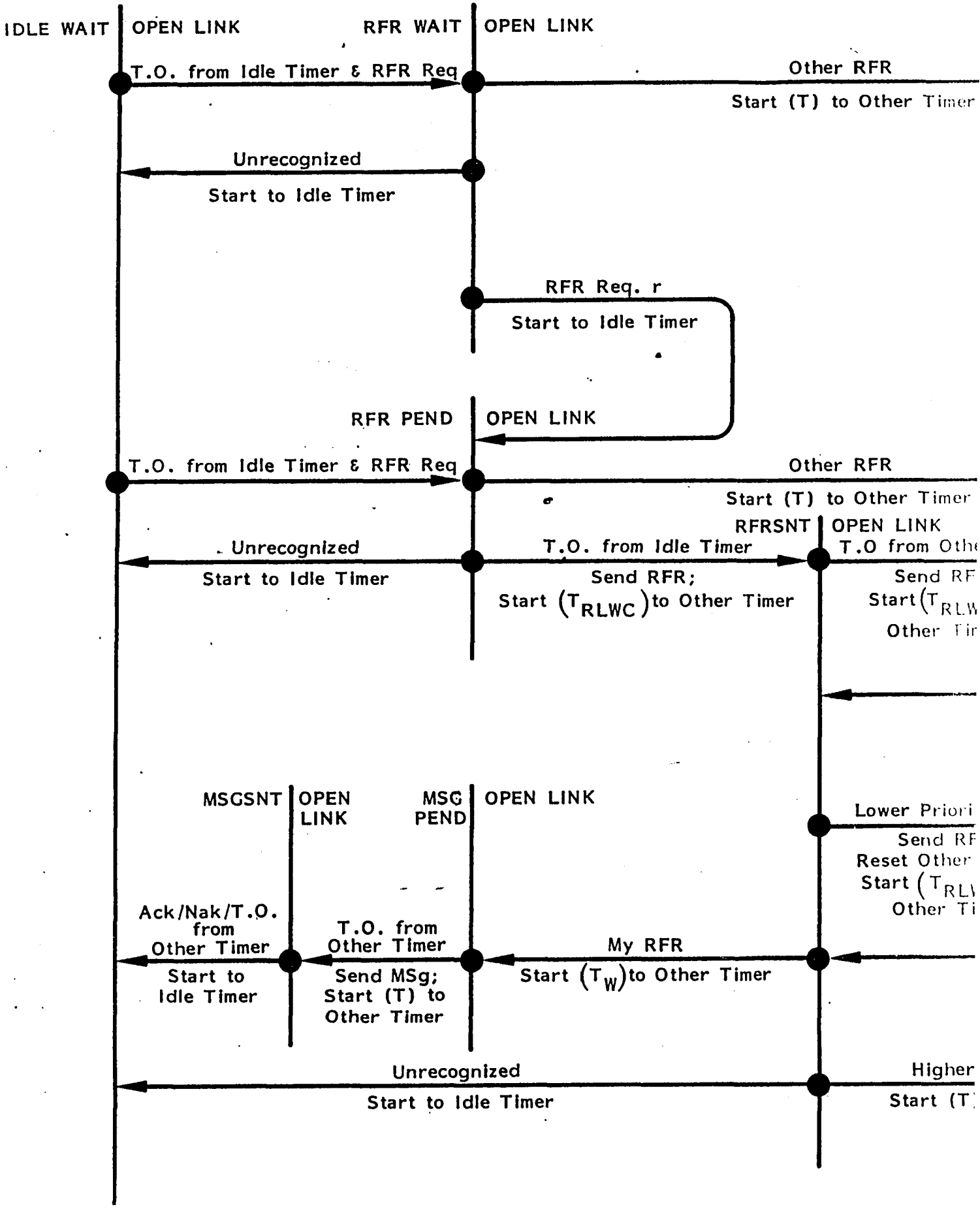

USER (HIGHER LAYER)
RFR Req,
No RFR Reg (s)
MAC Manager
(FSM)
To Transmitter
Send RFR
Send Msg
Send Ack
Send Nak
.O.(p)
Start(T),
Reset(p)
T.O.(p)
Start(T),
Reset(p)
To Link
Open Link,
Close Link(s)
DLE_TIMER
(TIMER)
OTHER_TIMER
(TIMER)

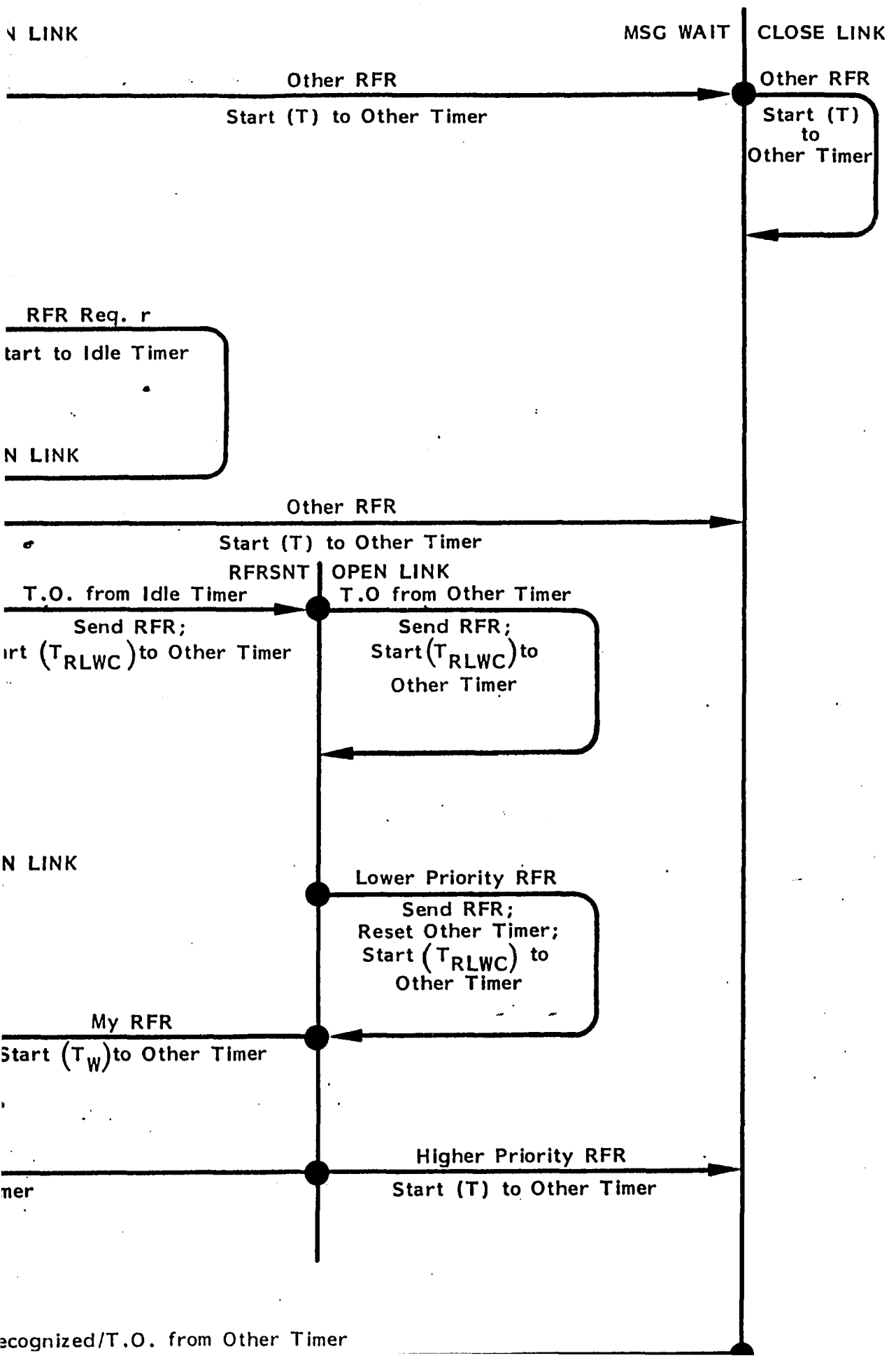

USER (HIGHER LAYER)
RFR Req,
No RFR Reg (s)
From Receiver
MAC Manager
(FSM)
My RFR, Lower
Priority RFR,
Higher Priority
RFR, My Good
Msg, My Bad Msg,
Ack, Nak, Other
Msg, Unrecognized (p)
T.O.(p)
Start(T),
Reset(p)
T.O.(p)
Start(T),
Reset(p)
Open Link,
Close Link(s)
IDLE_TIMER
(TIMER)
OTHER_TIMER
(TIMER)

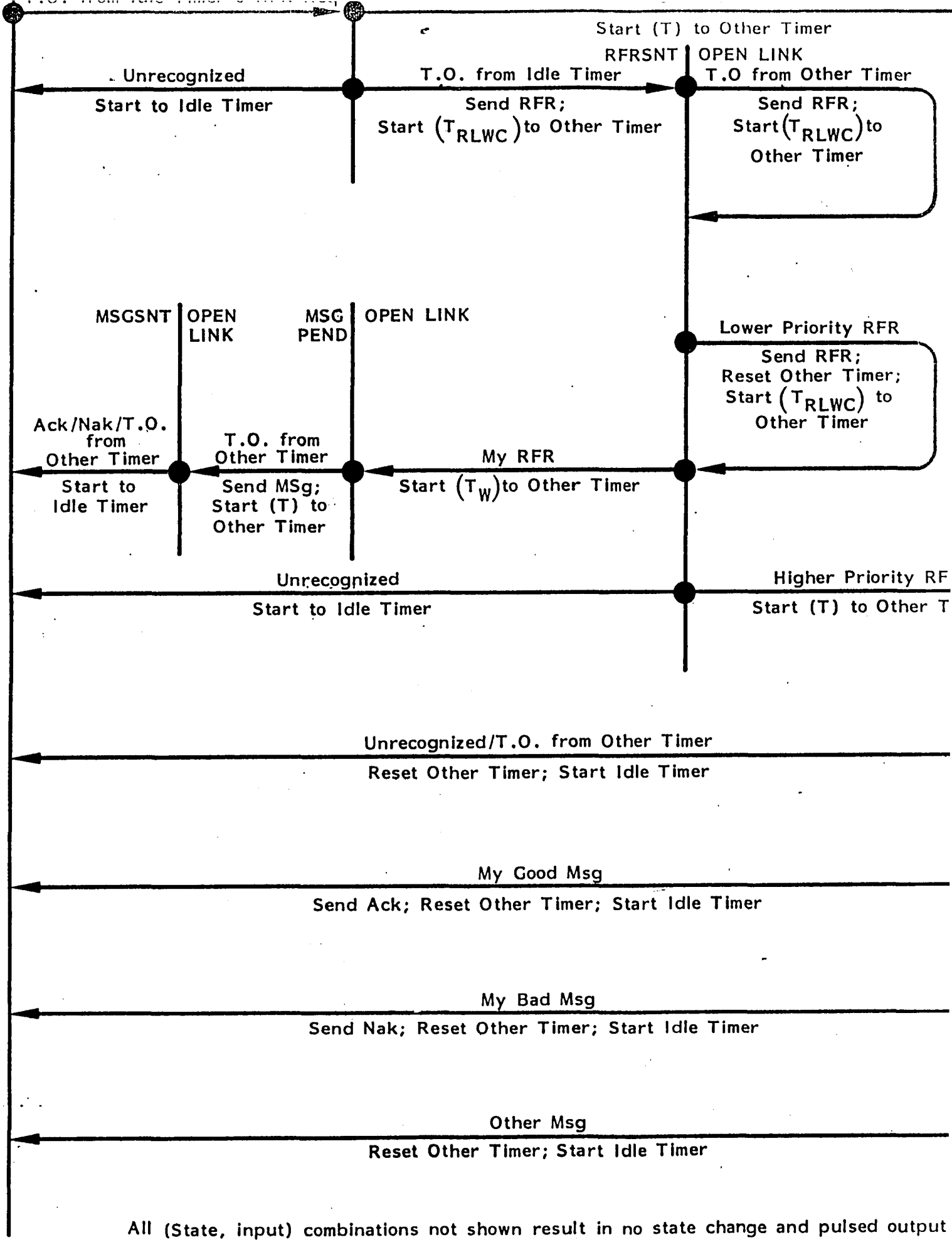
Legend: I_T = Idle_Timer
O_T = Other_Timer
T.O. = Time Out
T_{rlw} = Worst case ring propagation delay
T^{W} = Wait time before sending message, Ideally $T_{w} = 0$
T^{W} = Implementation dependent time

All (state, input) combinations not shown result in not state change and no pulsed output

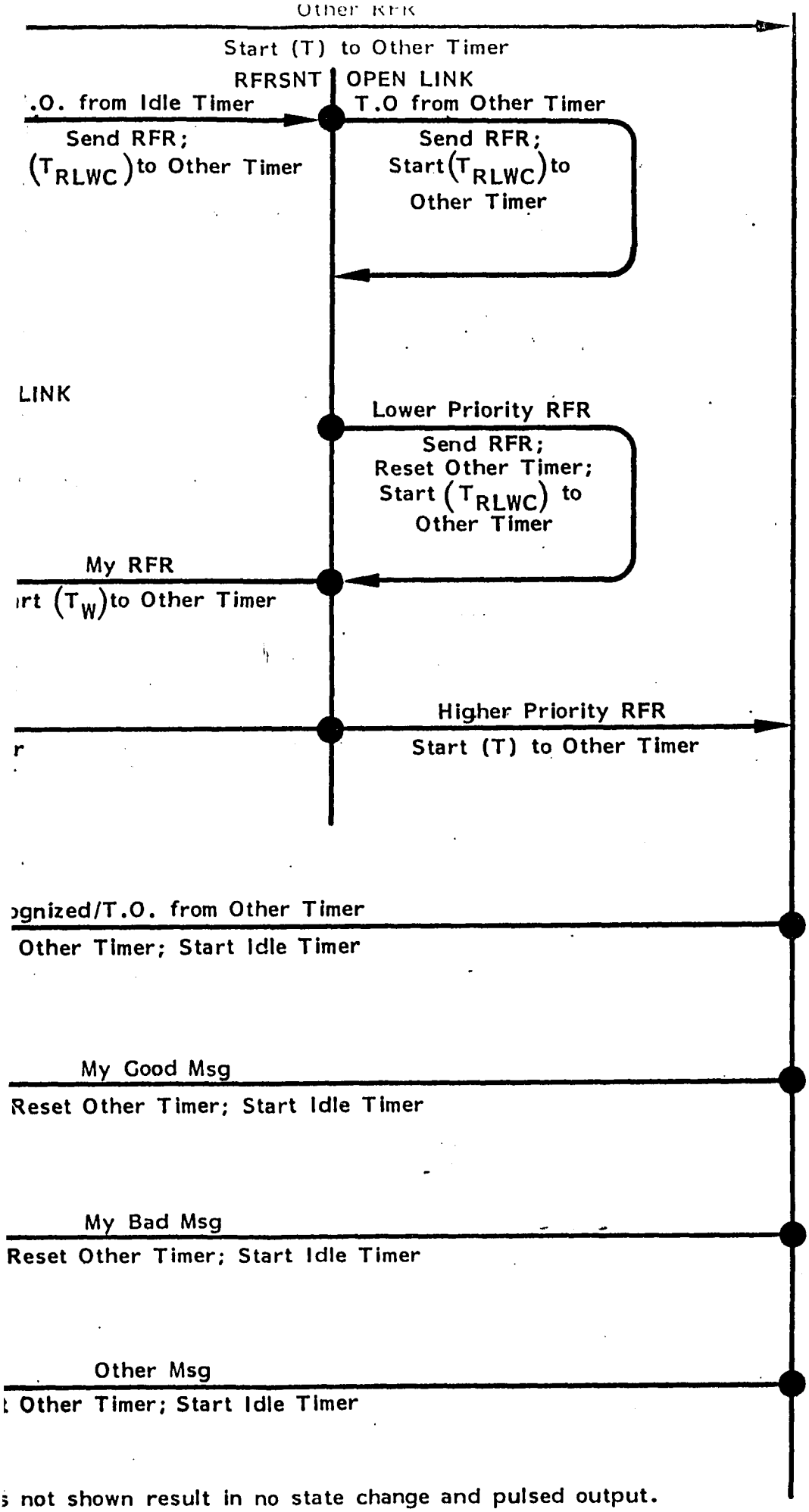
Figure 3-7. SAN model of the internal behavior of the MAC manager


IDLE WAIT
OPEN LINK
RFR WAIT
OPEN LINK
T.O. from Idle Timer & RFR Req
Other RFR
Start (T) to Other Timer
Unrecognized
Start to Idle Timer
RFR Req. r
Start to Idle Timer
RFR PEND
OPEN LINK
T.O. from Idle Timer & RFR Req
Other RFR
Start (T) to Other Timer
RFRSNT
OPEN LINK
Unrecognized
Start to Idle Timer
T.O. from Idle Timer
Send RFR;
Start (T_RLWC) to Other Timer
T.O from Othe
Send RF
Start (T_RLW
Other Tir
MSGSNT
OPEN LINK
MSG PEND
OPEN LINK
Lower Priori
Send RF
Reset Other
Start (T_RLW
Other Ti
Ack/Nak/T.O. from Other Timer
Start to Idle Timer
T.O. from Other Timer
Send MSg; Start (T) to Other Timer
My RFR
Start (T_W) to Other Timer
Unrecognized
Start to Idle Timer
Higher
Start (T


N LINK
MSG WAIT
CLOSE LINK
Other RFR
Start (T) to Other Timer
Other RFR
Start (T) to Other Timer
RFR Req. r
tart to Idle Timer
N LINK
Other RFR
Start (T) to Other Timer
RFRSNT
OPEN LINK
T.O. from Idle Timer
T.O from Other Timer
Send RFR;
irt (T_RLWC) to Other Timer
Send RFR;
Start (T_RLWC) to Other Timer
N LINK
Lower Priority RFR
Send RFR;
Reset Other Timer;
Start (T_RLWC) to Other Timer
My RFR
Start (T_W) to Other Timer
Higher Priority RFR
ner
Start (T) to Other Timer
ecognized/T.O. from Other Timer



All (State, input) combinations not shown result in no state change and pulsed output



s not shown result in no state change and pulsed output.

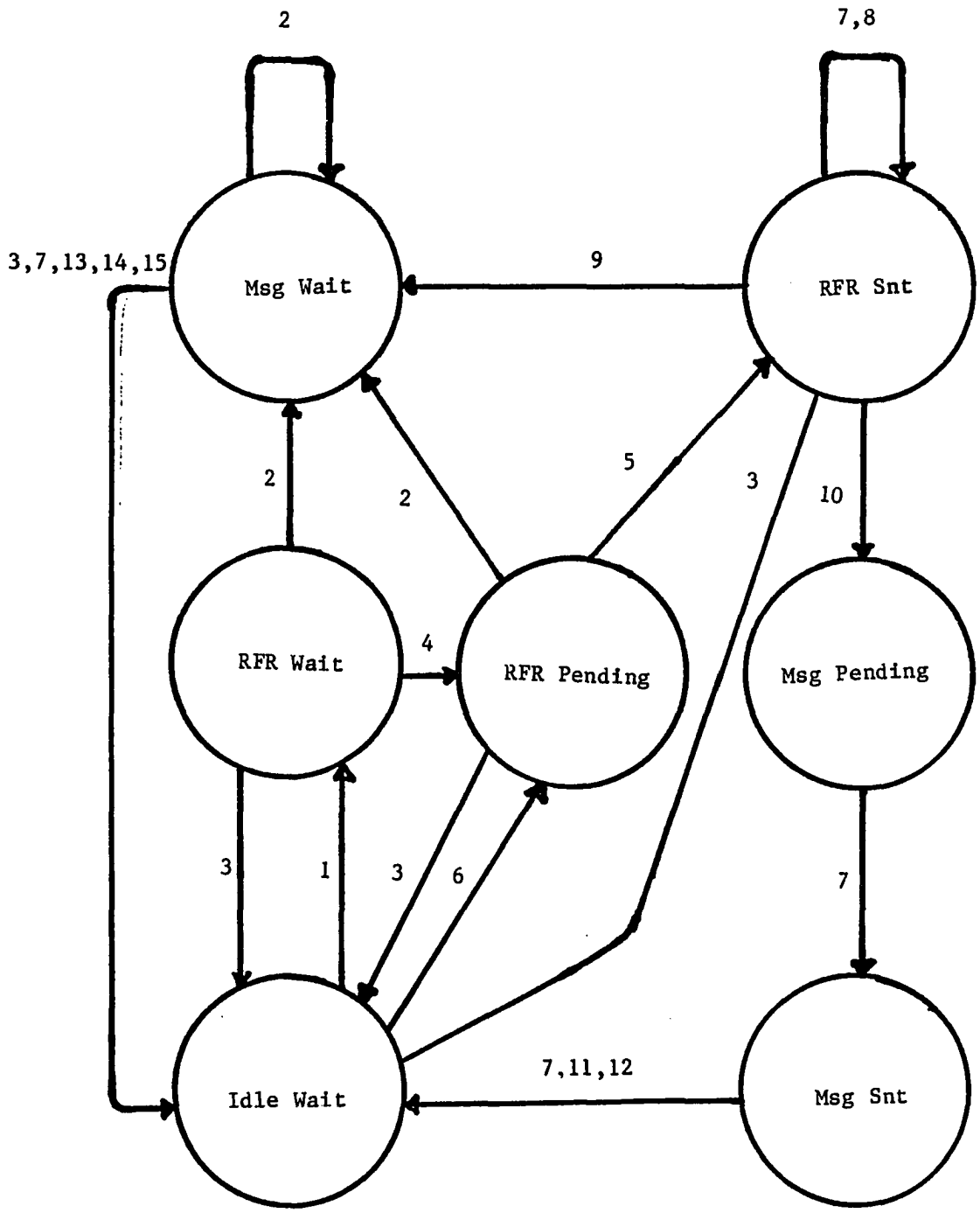


Figure 3-8. State diagram of the DCPR MAC manager. For the meaning of the numbers see Table 3-1

Table 3-1. Meaning of numbers used in Figure 3-1

Number	Meaning
1	T.O.[a] from Idle_Timer and No RFR Reg
2	Other RFR
3	Unrecognized
4	RFR Reg
5	T.O. from Idle_Timer
6	T.O. from Idle_Timer & RFR Reg
7	T.O. from Other_Timer
8	Lower Priority RFR.
9	Higher Priority RFR
10	My RFR
11	Ack
12	Nack
13	My Good Msg
14	My Bad Msg
15	Other Msg

[a]T.O. = Time Out.

CHAPTER 4. ANALYTIC MODELING OF DCPR

Introduction

In this chapter analytic models of DCPR are derived. In particular mathematical techniques are used to obtain performance measures (such as throughput, response time, etc.) of DCPR. We begin with a simple model making it more complex as we move along.

Recall that in DCPR each station has to reserve the channel before it can use it. Moreover, only the highest priority station which is ready before the reservation period begins eventually wins the bidding. Subsequently, it transmits its message and waits for an acknowledgment. After the latter has been received, a new reservation process begins. This situation is shown in Figure 4-1.

Let L_m = message length (bits)

L_a = ack, nack, etc length (bits)

L_h = header/trailer length (bits)

L_{RFR} = RFR length (bits)

C_b = channel capacity (bits/sec)

b = probability of 1 bit in error

λ_{mi} = source intensity (messages/sec) at station i

N = number of stations on the ring

t_m = message transmission time (sec)

t_a = ack, nack, etc. transmission time (sec)

t_{RFR} = RFR transmission time (sec)

t_{iw} = idle wait time (sec)

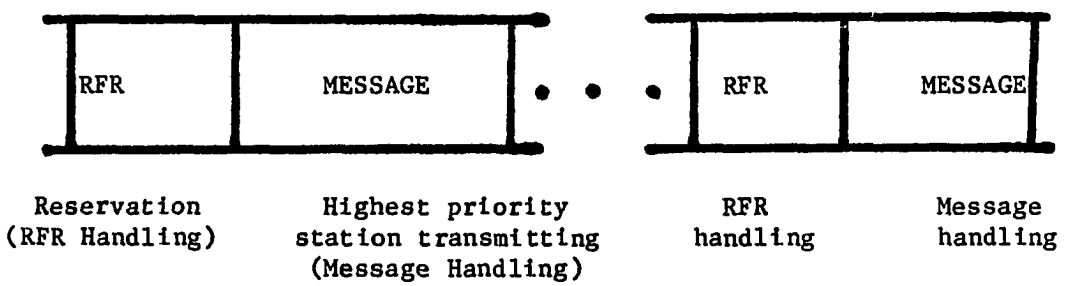


Figure 4-1. Time model of DCPR

t_w = waiting time before a message is transmitted after winning bidding (sec)

t_r = round trip transmission medium propagation delay (sec)

t_{rl} = ring latency (sec)

t_{dRFR} = time delay experienced by a RFR at each station (sec)

B_{dRFR} = bit delay experienced by a message at each station (bits)

t_{dm} = time delay experienced by a message at each station (sec)

B_{dm} = bit delay experienced by a message at each station (bits)

We also make the following general assumptions for the sake of analytical tractability:

A4.1) Messages randomly arrive at each station according to a Poisson process.

A4.2) The arrival rate at station k is λ_k messages/second; k=1,2, ..., N where N is the total number of stations.

A4.3) The message length at station k is a random variable L_k(bits), whose distribution is general.

Case 1: Lightly-loaded DCPR with no errors

Assumptions:

1) Only one station is ready just before each reservation process begins.

2) There is no competition for the channel (This is implied from A1).

3) No station failures occur.

4) No link (transmission medium) errors occur.

5) $\lambda_1 = \lambda_2 = \ldots = \lambda_k = \lambda_N = \lambda$

6) $E[_{M1}] = E[_{M2}] = \ldots = E[_{Mk}] = \ldots = E[_{MN}] = E[L]$

Applying these assumptions to Figure 3-7, a simplified Timed State Diagram (TSD) can be obtained as seen in Figure 4-2. From the TSD we want to determine the total time for the reservation period. This is calculated by adding together the times it takes the ready station to stay in each of the states involved in the reservation. The time it takes the ready station to stay in Idle Wait, RFR Pending and MSG Pending states are straightforward as can be seen in Figure 4-2. However, the time it takes to stay in the RFR Sent state requires a more careful look.

The system that is being analyzed is similar to the system that was discussed in Case 1 of the DCPR example of Chapter 3. In Figure 3-2, it was found that the ready station (in this case station T_1) times out 3 times before finally receiving its RFR. This means that in general, for N stations, the time the ready station stays in the RFR Sent state is given by

$$t_{RFR\ Sent} = (N-1)t_{RLWC} + T_{RFR} + t_r + NB_{dRFR}/C \qquad (4.1)$$

where T_{RLWC} is the worst case RFR round trip time and

$$t_{RFR} = L_{RFR}/C \qquad (4.2)$$

Thus the total time for the reservation period is given by

$$t_{RH} = 2t_{iw} + T_{RFRSent} + t_W \qquad (4.3)$$

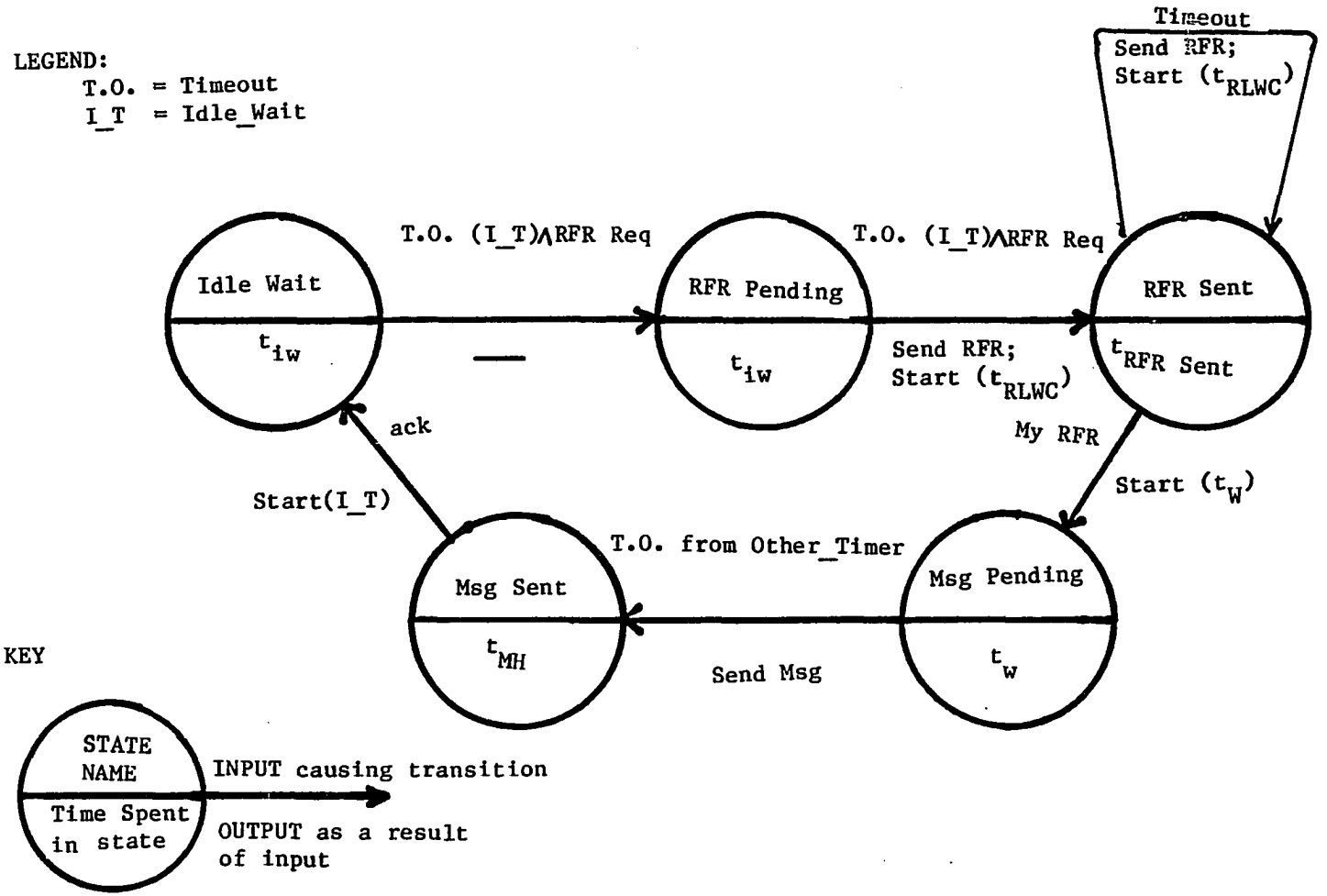


Figure 4-2. Times state diagram of DCPR MAC mgr when lightly loaded with no errors

and the total average time for handling one message transmission and subsequent acknowledgment is given by

$$t_{MH} = E[L]/C + L_h/C + L_a/C + t_r + NB_{dm}/C \qquad (4.4)$$

One parameter of importance in all ring networks is ring latency. It is defined as the round trip propagation delay experienced along the ring. In bus networks the roundtrip propagation delay is simply twice the end-to-end propagation delay where the latter is just the transmission medium delay.

In ring networks it is not that simple. Since each station is an active repeater in this case, delays the minimum of which is 1 bit, occur at each station. The maximum depends upon several factors which will be explained in detail later. For the moment, suffice it to say that it depends upon the ring access control protocol and the number of bits of control information in each packet. Thus the round trip propagation delay depends upon both the station delays and transmission medium delays.

In DCPR, the station delay experienced by an RFR may be different from that experienced by a message, in general. Thus, we define an RFR ring latency as

$$t_{RLR} = t_r + NB_{dRFR}/C \qquad (4.5)$$

and message ring latency as

$$t_{RLM} = t_r + NB_{dm}/C \qquad (4.6)$$

To simplify matters we let

$$B_{dRFR} = B_{dm} = B \tag{4.7}$$

and define a single ring latency as

$$t_{rL} = t_r + NB/C \tag{4.8}$$

Substituting the latter equation in Equations (4.1), (4.3), and (4.4), we obtain

$$t_{RH} = 2t_{iw} + (N-1)t_{RLWC} + t_{RFR} + t_R + NB/C + t_W \tag{4.9}$$

$$t_{RH} = 2t_{iw} + (N-1)t_{RLWC} + t_{RFR} + t_{RL} + t_W \tag{4.10}$$

and

$$t_{MH} = E[L]/C + t_h + t_a + t_{rL} \tag{4.11}$$

where

$$t_h = L_h/C$$

and $t_a = L_a/C$.

Since the total time it takes to send one message is

$$t_{cyc} = t_{RH} + t_{MH} \tag{4.12}$$

the Protocol efficiency is given by

$$PE = \frac{t_{MH}}{t_{cyc}} \tag{4.13}$$

The Effective Bit Rate is given by

$$EBR = \frac{E[L]}{t_{cyc}} = E[L]\lambda_m \tag{4.14}$$

where λ_m = message throughput

$$\therefore \lambda_m = \frac{1}{t_{cyc}} \tag{4.15}$$

The Normalized Effective Bit Rate is given by

$$NEBR = \frac{EBR}{C} = E[L]\lambda_m/C \tag{4.16}$$

Equation (4.12) gives a measure of average response time for each station. Equation (4.13) gives the fraction of cycle time that is used in actual transmission of data and receipt of acknowledgment. This Protocol Efficiency can be increased by reducing t_{RH}. Equation (4.14) is the average throughput in bits/sec whereas Equation (4.15) gives the same measure in messages/sec. Equation (4.16) gives the fraction of channel capacity that is utilized in sending actual data. It is a measure of the efficiency of channel usage.

Case 2: Lightly-loaded with errors

This case is similar to Case 1 with the following modified assumptions:

1) RFR and message errors can occur.

2) RFR and message errors are independent.

3) ACK/NAK frames are always delivered and correctly.

4) No other errors occur.

The Timed State Diagram (TSD) with state transition probabilities which results from the above assumptions is shown in Figure 4-3 where the following quantities are defined:

P_{re} = probability of a RFR being in error

P_{me} = probability of a message being in error.

We want to determine the following performance parameters:

$\bar{\tau}_{cyc}$ = average time per cycle (cycle is defined as the movement from the Idle Wait State through RFR Pending and RFR Sent (and possibly MSG Pending and MSG Sent) states back to the Idle Wait State)

$\bar{N}_{cyc}$ = average number of cycles required to send 1 good message

$\bar{N}_m$ = average number of good messages per cycle

λ_m = message throughput

λ_b = bit throughput

$\bar{\tau}$ = average delay (the average response time from initiation of an RFR request till the receipt of a positive acknowledgment)

From Figure 4-3:

$$\bar{\tau}_{cyc} = t_{iw} + 1(t_{iw} + 1(t_{RFRSent} + (1-P_{re})(t_w + 1(t_{MH}))))$$
$$= t_{iw} + t_{iw} + t_{RFRSent} + (1-P_{re})(t_w + t_{MH}) \quad (4.17)$$

$$\bar{\tau}_{cyc} = t_{RH} - t_w + (1-P_{re})(t_{MH} + t_w) \quad (4.18)$$

$$\bar{N}_m = (1 - P_{me})(1\bullet(1\bullet((1 - P_{re})(1\bullet(1))))) \quad (4.19)$$

$$\bar{N}_m = (1 - P_{me})(1 - P_{re}) \quad (4.20)$$

$$\bar{N}_{cyc} = \frac{1}{\bar{N}_m} = \frac{1}{(1 - P_{me})(1 - P_{re})} \quad (4.21)$$

$$\lambda_m = \frac{\bar{N}_m}{\bar{\tau}_{cyc}} = \frac{(1 - P_{me})(1 - P_{re})}{t_{RH} + (1 - P_{re})t_{MH}} \quad (4.22)$$

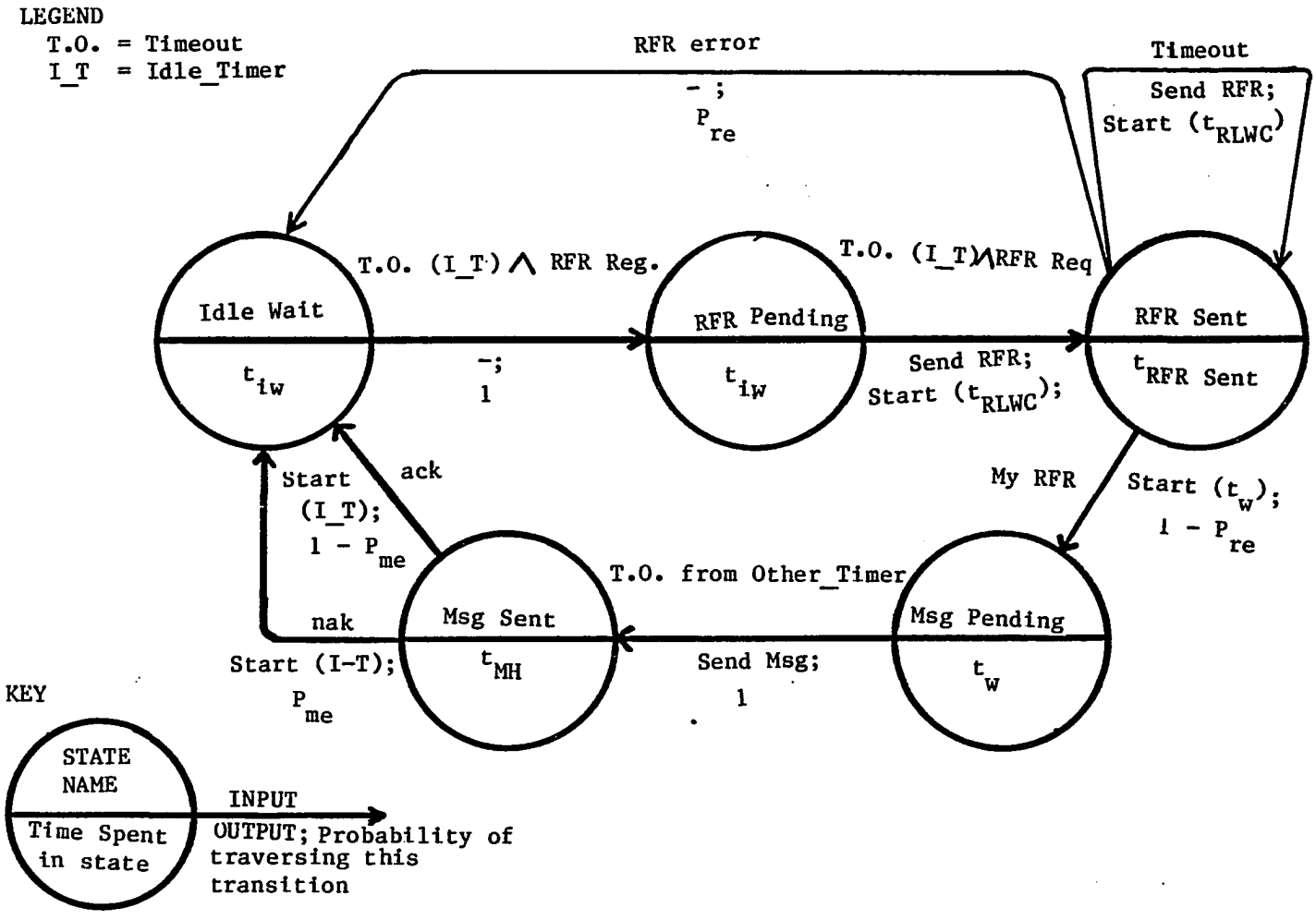


Figure 4-3. Time state diagram of MAC mgr with lightly loaded with errors

$$\lambda_b = \lambda_m E[L] = \frac{E[L](1 - P_{me})(1 - P_{re})}{t_{BH} + (1 - P_{re})t_{MH}} \tag{4.23}$$

$$\bar{\tau} = \bar{\tau}_{cyc}\, \bar{N}_{cyc} = \frac{(t_{BH} - t_w) + (1 - P_{re})(t_{MH} + t_w)}{(1 - P_{me})(1 - P_{re})} \tag{4.24}$$

The last expression could have been obtained using the following alternative method:

Let $\bar{N}_R$ = average number of attempts to sent 1 good RFR and $\bar{N}$ = average number of attempts to send 1 good message. Then

$$\bar{\tau} = \bar{N}\,\{\bar{N}_R(t_{RH} - t_w) + t_{MH} - t_w\} \tag{4.25}$$

$$\text{Now}\quad \bar{N} = \sum_{k=1}^{\infty} k\ Pr\{N=k\} = \sum_{k=1}^{\infty} k(1 - P_{me})P_{me}^{k-1} \tag{4.26}$$

$$= \frac{1}{1 - P_{me}} \tag{4.27}$$

$$\text{Similarly,}\quad \bar{N}_R = \frac{1}{1 - P_{re}} \tag{4.28}$$

Substituting 4.27 and 4.28 in 4.25 we obtain

$$\bar{\tau} = \frac{1}{1 - P_{me}}\ \frac{1}{1 - P_{re}}(t_{RH} - t_w) + t_{MH} + t_w$$

$$= \frac{1}{(1 - P_{me})(1 - P_{re})}\ \{t_{RH} - t_w + (1 - P_{re})t_{MH} + t_w\} \tag{4.29}$$

Now let $E[L] = L_m$

Since b = Probability{1 bit in error} (4.30)

P_{me} = Probability{at least 1 bit in error} (4.31)

$$= 1 - (1 - b)^{L_h + L_m} \tag{4.32}$$

Similarly, $P_{be} = 1 - (1 - b)^{b}$ (4.33)

Now assuming $b \ll 1$, (4.34)

which is valid for most systems, Eq. (4.32) becomes

$$\tilde{P}_{me} = 1 - \{1 - (L_h + L_m)b\}^{*}$$

$$= (L_h + L_m)b \tag{4.35}$$

Similarly, Eq. (4.33) becomes,

$$\tilde{P}_{be} = L_b b^{**} \tag{4.36}$$

Case 3: More than one station transmitting with no errors

From the operating rules of each DCPR station it can be seen that the DCPR LAN is a station-based static priority system with the communication channel reserved for the highest priority ready station. Thus, when there are messages available at the buffers of the station the channel is being used alternately for reservation and transmission (Figure 4-4).

Let S_i = ith reservation period

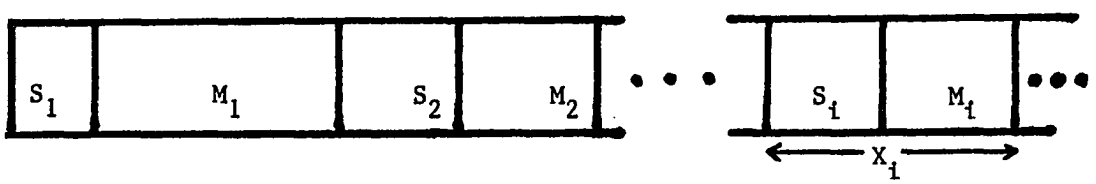
and M_i = ith message transmission period.

Then we make the following assumptions in addition to the assumptions made at the beginning of this chapter:

---

*In fact Eq. (4.35) is true if and only if $(L_m + L_b)b \ll 1$ for all L_m, L_h, b. Otherwise the approximation cannot be made.

**This approximation is true if and only if $bL_b \ll 1$.



S_i = ith reservation period

M_i = ith message handling period

X_i = ith service time = $S_i + M_i$

Figure 4-4. DCPR channel transactions

A4.4) The set $\{S_i : i=1,2, \ldots\}$ consists of independent identically distributed (i.i.d.) random variables.

A4.5) The set $\{M_i : i=1,2,\ldots\}$ consists of i.i.d. random variables. This follows partly from assumption A4.3 where it is assumed that message lengths are i.i.d random variables.

A4.6) S_i and M_j with $i=1,2,\ldots$ and $j=1,2,\ldots$ are independent.

A4.7) Let $X_i = S_i + M_i$. Then from A4.4-A4.6 the set $\{X_i : i=1,2,\ldots\}$ consists of i.i.d. random variables.

Based on the fact that the DCPR LAN consists of a number of stations on a ring sharing a single channel and, as a result of assumption A4.1, A4.2, and A4.7, the DCPR LAN can be modelled as an M/G/1 nonpreemptive static priority queueing system (Figure 4-5). The ring channel is the server and the queues are the DCPR stations with fixed priorities. This is the same as the so-called head-of-the-line priority system [9].

Let p = priority of a station where p=1,2, ..., P.

Then it, can be shown (see Kleinrock [9]) that the average waiting time for mesesages of the pth priority group is given by

$$W_p = \frac{W_o}{(1 - \sigma_p)(1 - \sigma_{p+1})} \qquad p=1, \ldots, P \tag{4.37}$$

where

$$\sigma_p = \sum_{k=p}^{P} \rho_k \tag{4.38}$$

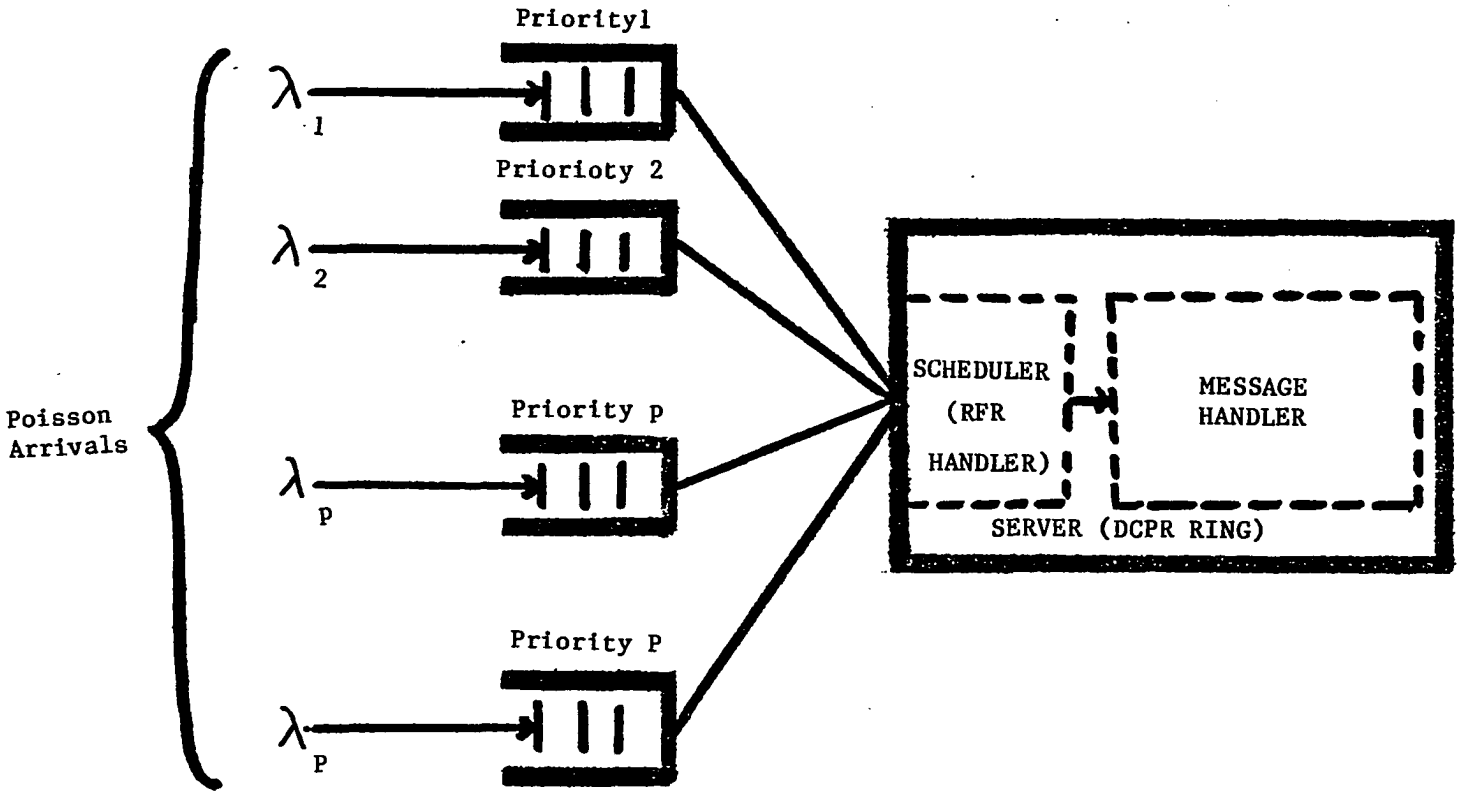


Figure 4-5. DCPR configured as an M/G/1 priority queueing system

$$\rho_k = \lambda_k E[X_k] \tag{4.39}$$

$E[X_k]$ = average service time for group k messages

ρ_k = fraction of time the server is busy with messages of group k

$$W_o = \sum_{k=1}^{P} \frac{\rho_k E[X_k^2]}{2E[X_k]} \tag{4.40}$$

under the constraint that $\sigma_p = \sum_{k=1}^{P} \rho_k < 1$

Now assuming that

A4.8) $\lambda_k = \lambda$ for k=1, ..., N

A4.9) $E[L_k] = E[L]$ and $E[L_k^2] = E[L^2]$, k=1, ..., N

From A4.9 and A4.10, $E[X_k] = E[X]$ for k=1, ..., N (4.41)

$$\sigma_p = (P - p + 1)\rho \tag{4.42}$$

$$\sigma_{p+1} = (P - p)\rho \tag{4.43}$$

where

$$\rho = \lambda E[X] \tag{4.44}$$

A4.10) $E[S_i] = S$. (Note: This assumption is strictly speaking not very reasonable for DCPR in general. The distribution of S_i is a dynamic quantity which depends upon traffic characteristics. This assumption will, however, be used to determine upper and lower bounds of average response time).

Therefore,

$$W_o = \frac{N\lambda E[X]\ E[X^2]}{2E[X]} = N\lambda E[X^2]/2 \tag{4.45}$$

Using (4.37), (4.41) the average response time of the pth priority station is given as

$$T_p = E[X] + \frac{N\lambda E[X^2]}{2[1 - (N - P + 1)\rho][1 - (N - p)\rho]} \tag{4.46}$$

Let us define the total message throughput rate (i.e., NEBR) as:

$$S = N\lambda E[L]/C \tag{4.47}$$

From Eq. (4.44)

$$\rho = \lambda[H + E[L]/C] \tag{4.48}$$

where H is a constant time overhead which includes reservation time, time to transmit message header/trailer and ACK/NAK and ring latency.

Substituting Eq. (4.47), (4.48) becomes

$$\rho = \lambda H + S/N \tag{4.49}$$

Then substituting Eq. (4.49), (4.46) becomes

$$T_p = H + \frac{E[L]}{C} + \frac{N\lambda E[X^2]}{2[1 - (N - p + 1)(\lambda H + S/N)][1 - (N-p)(\lambda H + S/N)]} \tag{4.50}$$

Normalizing the response time by the time it takes to transmit one message and, after some algebra we obtain

$$T_p' = \frac{T_p C}{E[L]} = (h' + 1) + \frac{SC^2E[X^2]/\{E[L]\}^2}{[1-(N-p+1)(h'+1)S/N][1-(N-P)(h'+1)S/N]} \tag{4.51}$$

where h' is the normalized overhead latency defined as

$$h' = \frac{HC}{E[L]} \tag{4.52}$$

Since the service time is always H + L/C, the second moment of the service time is given as

$$E[X^2] = H^2 + 2HE[L]/C + E[L^2]/C^2 \tag{4.53}$$

For fixed message lengths,

$$E[L^2_{fixed}] = \{E[L]\}^2 \tag{4.54}$$

and for exponentially distributed message lengths

$$E[L^2_{exp}] = 2\{E[L]\}^2 \tag{4.55}$$

Upper and lower bounds of the reservation time

As seen from the DCPR examples in the previous chapter, the reservation times, S, is a dynamic quantity which depends on which stations are ready before the reservation process begins. The obvious thing to do is to determine the distribution of S or its first two moments. However, this determination is a non-trivial problem, which in the opinion of the author is not analytically tractable without making further assumptions. This is the reason why simulation is used in the next chapter to determine more accurate performance results. The complexity of this problem and the simulation approach will be discussed in detail in the next chapter.

For the moment upper and lower bounds of S will be used to determine the upper and lower bounds of the response times of the stations.

It is clear from the examples in Chapter 3 that the upper bound

of S is achieved when only one station is ready before reservation begins and the lower bound is achieved when all stations are ready before reservation begins. The former case has already been analyzed in this chapter (see Case 1; Eq. 4.10). Thus, the upper bound of S is:

$$S_{max} = 2t_{iw} + (N-1)t_{RLWC} + t_{RFR} + t_{rL} + t_{w} \tag{4.56}$$

where T_{iw} is the time it takes for the stations to detect idle links before starting to request for channel. This time, which occurs at the end of each message transmission and acknowledgment, can be considered as a time to allow the stations to synchronize. t_{RLWC} is a time fixed during the design of the network. It is the worst case time it takes an RFR to go one-round trip around the ring. Any time an RFR is sent the OTHER_TIMER is set to timeout after this time. The other quantities have already been defined.

When all stations are ready each time reservation begins Figure 3-3 clearly shows that for N stations, the lower bound of S is:

$$S_{min} = 2t_{iw} + t_{RFR} + t_{rL}/N + t_{RFR} + t_{rL} + t_{w} \tag{4.57}$$

Therefore,

$$S_{min} = 2t_{iw} + 2t_{RFR} + t_{rL}(N+1)/N + t_{w} \tag{4.58}$$

From Eqs. (4.56) and (4.58) the minimum average overhead is:

$$H_{min} = S_{min} + t_{a} + t_{h} + t_{rL} \tag{4.59}$$

and the maximum is:

$$H_{max} = S_{max} + t_a + t_h + t_{rL} \quad (4.60)$$

Equations (4.56) - (4.60) are used in (4.52) and (4.51) to obtain bounds on the normalized delays for both fixed and exponentially distributed message lengths with the help of Eqs. (4.53)-(4.55).

In particular, the delays of interest are those of the highest priority (the station with p = N) and lowest priority (the one with p=1) stations.

The numerical results of all three cases are discussed next.

Numerical Results

The purpose of this section is to show throughput-delay graphs for various number of stations and message lengths where the latter is either fixed or exponentially distributed.

The following were the DCPR LAN parameters that were used:

Header/trailer length,	L_h = 56 bits
Ack/Nak length,	L_a = 24 bits
RFR length,	L_{RFR} = 8 bits
Wait time before message transmission,	Tw = 0
Length of ring cable,	D = 1 km
Ring cable propagation delay,	$t_{pd} = 5 \times 10^{-6}$ s/km
Ring round trip propagation delay,	$t_r = 5 \times 10^{-6}$s
Latency per station,	B = 1
Channel capacity,	C = 1Mbps
Worst case ring latency,	t_{RLWC} = 1 ms
Idle wait time,	t_{iw} = 1 ms

The ring round trip propagation delay was calculated from the formula

$$t_r = Dt_{pd}$$

and the worst case ring latency was calculated using Eq. (4.8) based on the assumption that the maximum ring cable length, D_{max} = 10 km, the maximum station latency is B_{max} = 8 bits, the maximum number of stations, N_{max} = 100 and the minimum channel capacity, C_{min} = 1 Mbps. Thus,

$$\begin{aligned} t_{RLWC} &= D_{max}tpd + B_{max}N_{max}/C_{min} \qquad (4.61) \\ &= 10 \times 5 \times 10^{-5} + 8 \times 100/1 \times 10^{6} \\ &= 850 \times 10^{-6} = .85ms \end{aligned}$$

Cases 1 and 2 where only one station is ready to transmit at a time are discussed first. Clearly, Case 1 is a special form of Case 2 with error probability equal to zero. The purpose of Case 2 is to look at the performance of the error control aspect of the protocol. It can be seen that the basic ARQ (Automatic Repeat Request) scheme is being used to control message errors. What makes the present analysis different from others is that RFRs are also susceptible to errors and so provision is made to ensure that errors thereof are also controlled.

Figures 4.6-4.8 show a plot of the Normalize Effective Bit Rate (NEBR) against the number of stations (N) for various values of bit error probability. The range of values of bit error probability used was 0 to 10^{-4}.

Figure 4.6 shows this variation with an average message length of 1000 bits. The figure shows that the maximum NEBR that can be achieved for this configuration is a little less than 0.5. Furthermore, it does

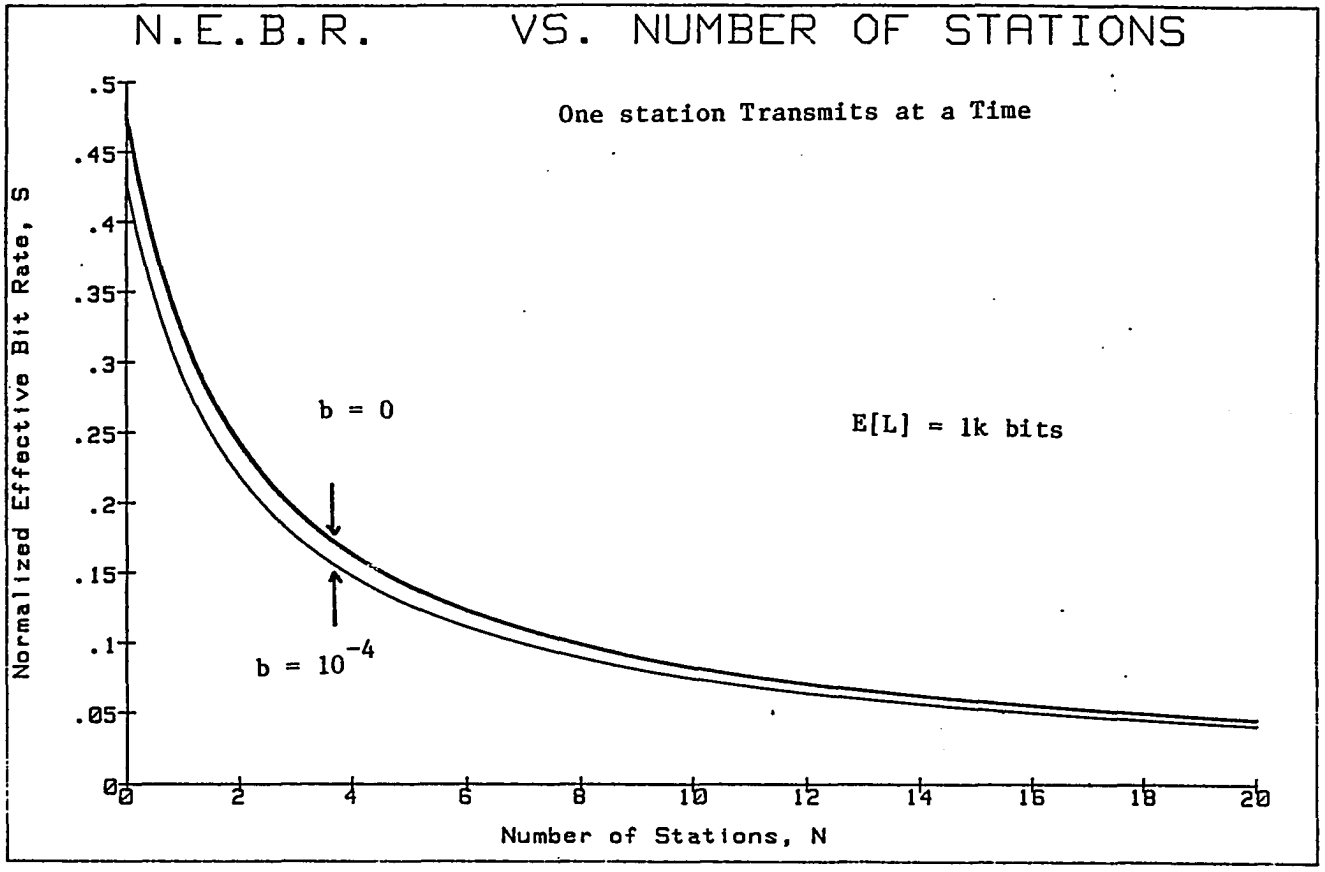


Figure 4-6. Normalized throughput as a function of number of stations with bit error probability fixed. E[L] = 1k bits

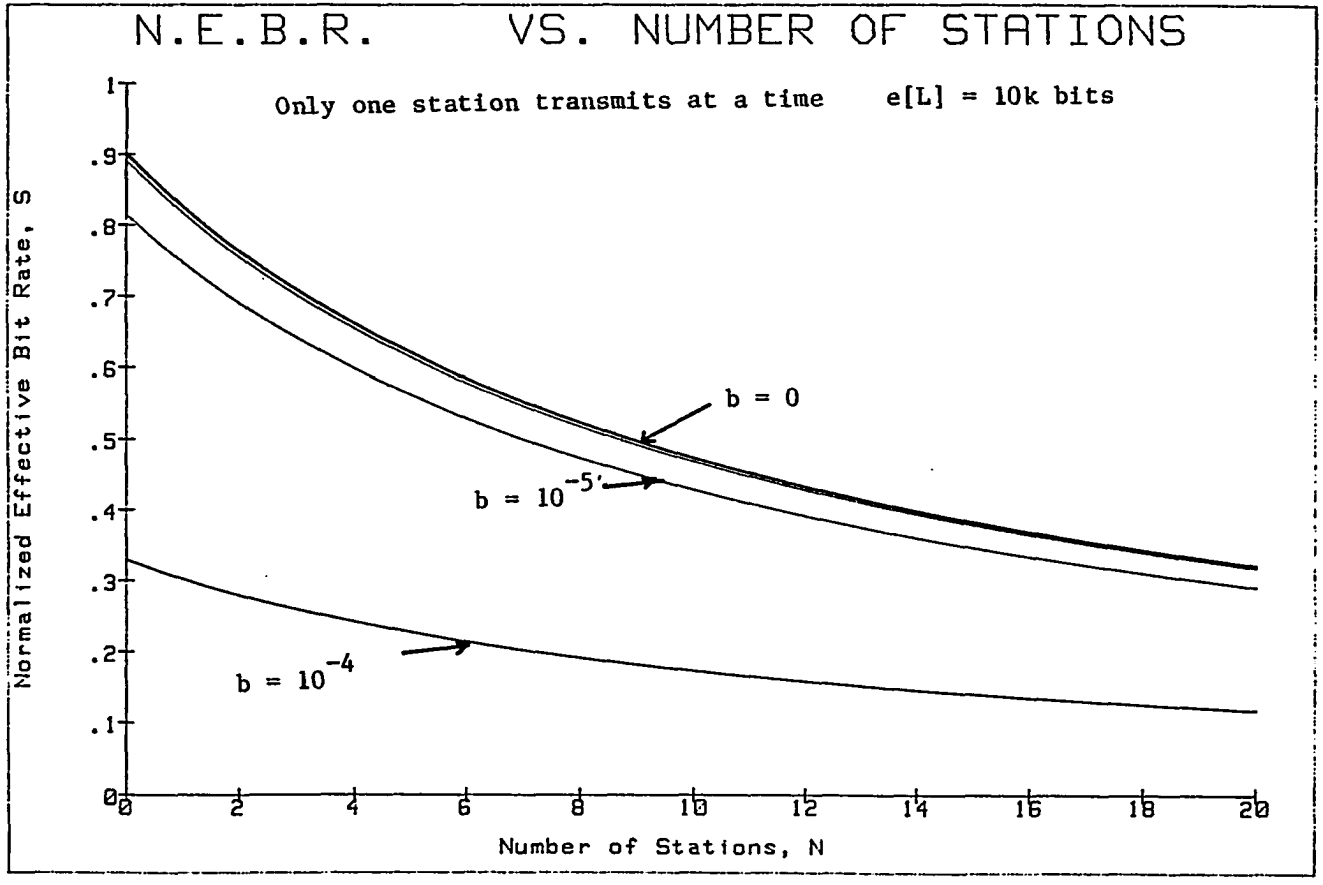


Figure 4-7. Normalized throughput as a function of number of stations with bit error probability varying. E[L] = 10k bits

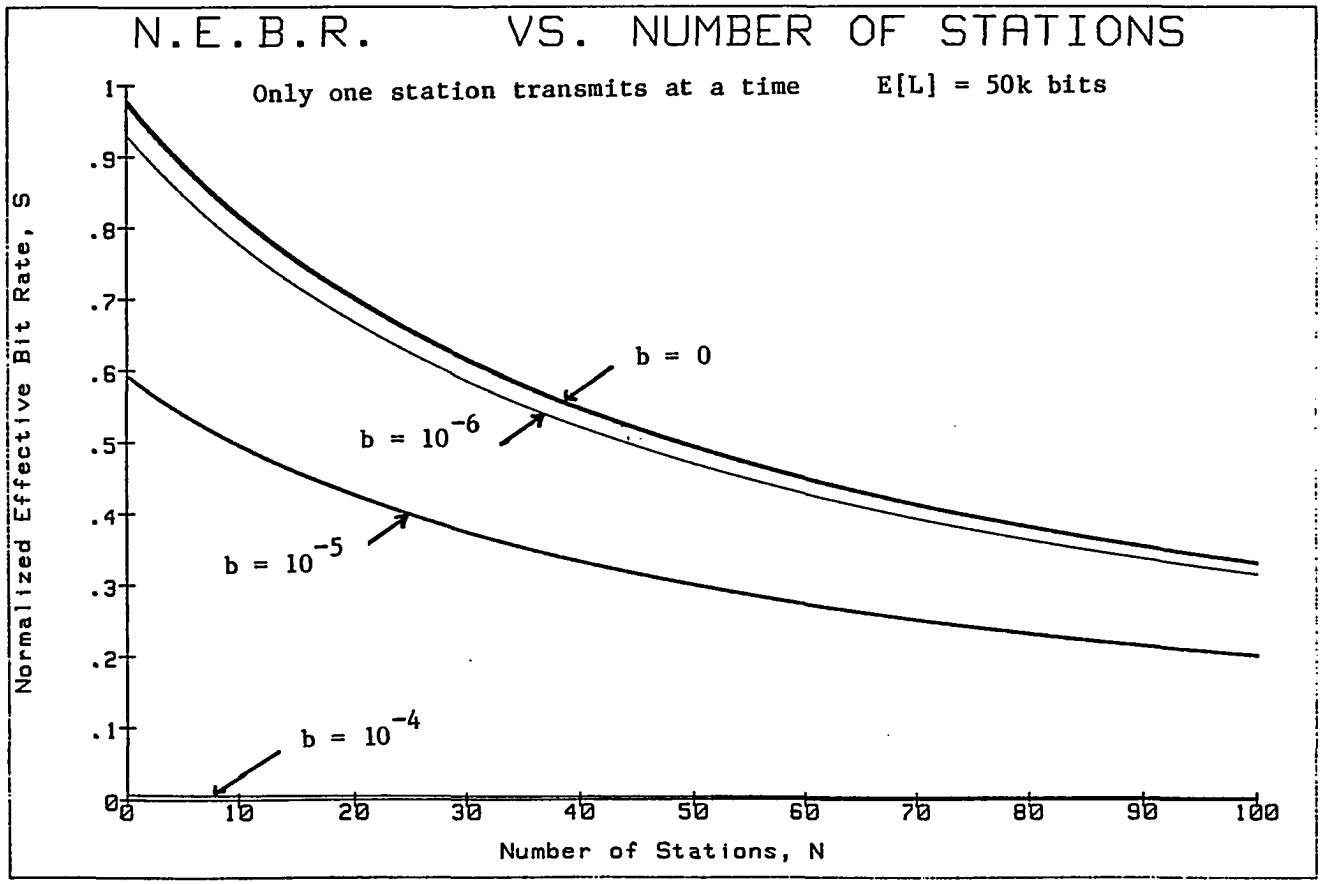


Figure 4-8. N.E.B.R. vs. number of stations. E[L] = 50k bits

not change much with error.

Figure 4.7 is similar to Figure 4.6 except that the average message length is 10 kbits. In contrast with Figure 4.6, Figure 4.7 gives a very high NEBR of .9. Also, a bit error probability of 10^{-6} is as good as error-free operation.

Figure 4.8, with average message length of 50 kbits provides an even better performance with NEBR very close to the maximum value of 1.

All three figures (4.6-4.8) show that NEBR, which can be considered as fraction of channel capacity used to process actual messages, decreases as the number of stations increases. They also show that the decrease is sharper with low values of average message length such as 1 kbits (see Figure 6). The reason for this is because with small message lengths most of the channel capacity is used for reservation. Since the ready station has to query all stations to close their links before it can transmit the reservation time is heavily dependent on N (see Eqs. (4.10) and (4.61)). Moreover, as message length increases, it becomes more advantageous to have lower error probability (i.e., permit less errors with subsequent retransmissions). Clearly, with large message lengths, message processing times dominate. Since each error requires retransmission of the whole message, limiting errors will cause fewer retransmissions and hence, higher channel utilization (NEBR).

Figures 4.9-4.10 show a plot of the number of stations against the normalized average response time for various values of bit error probability. Figure 4.9 was an average message length (L_m) of 1k whereas Figure 4.10 was an average message length of 10k. As expected

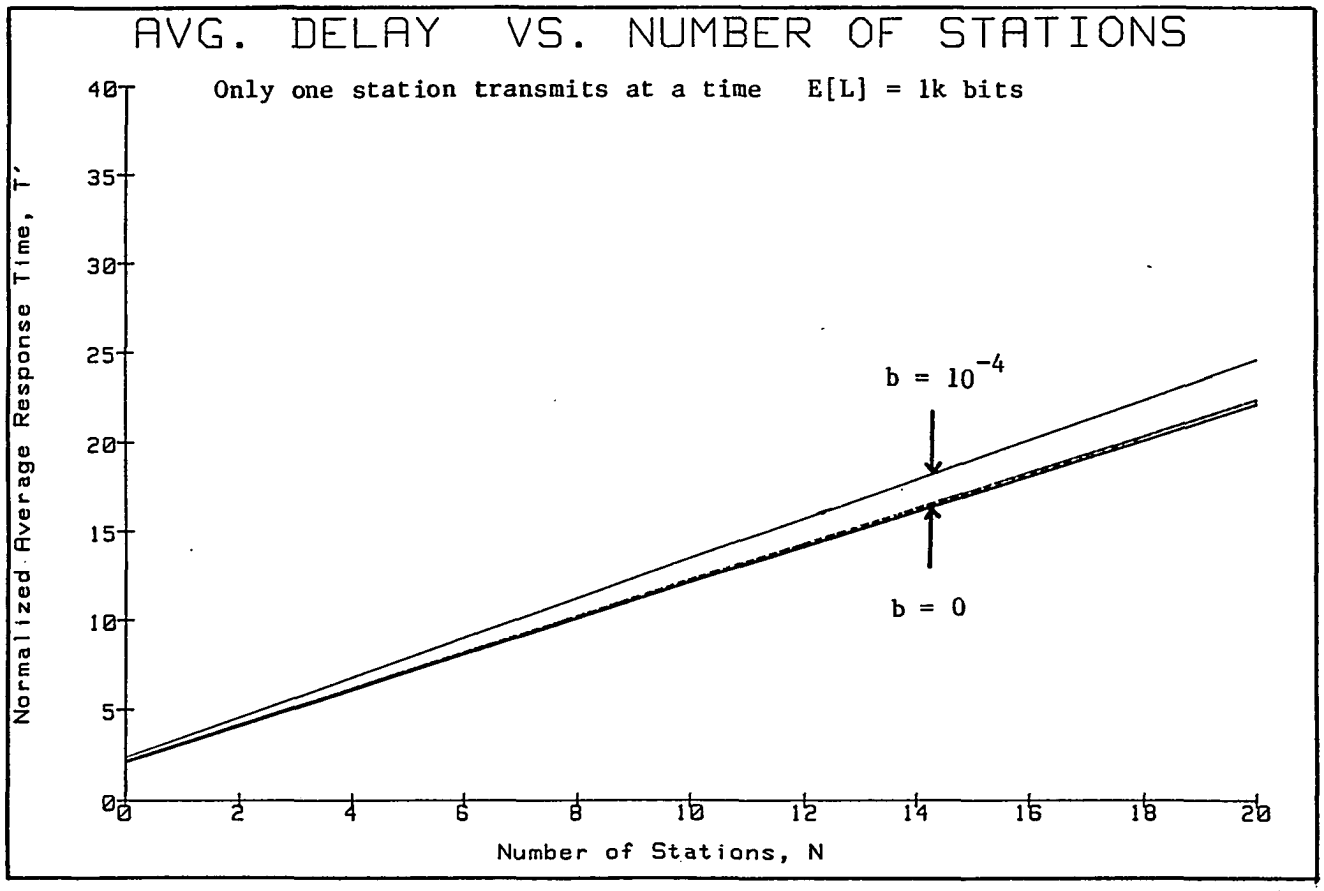


Figure 4-9. Normalized average response time as a function of number of stations. E[L] = 1k bits

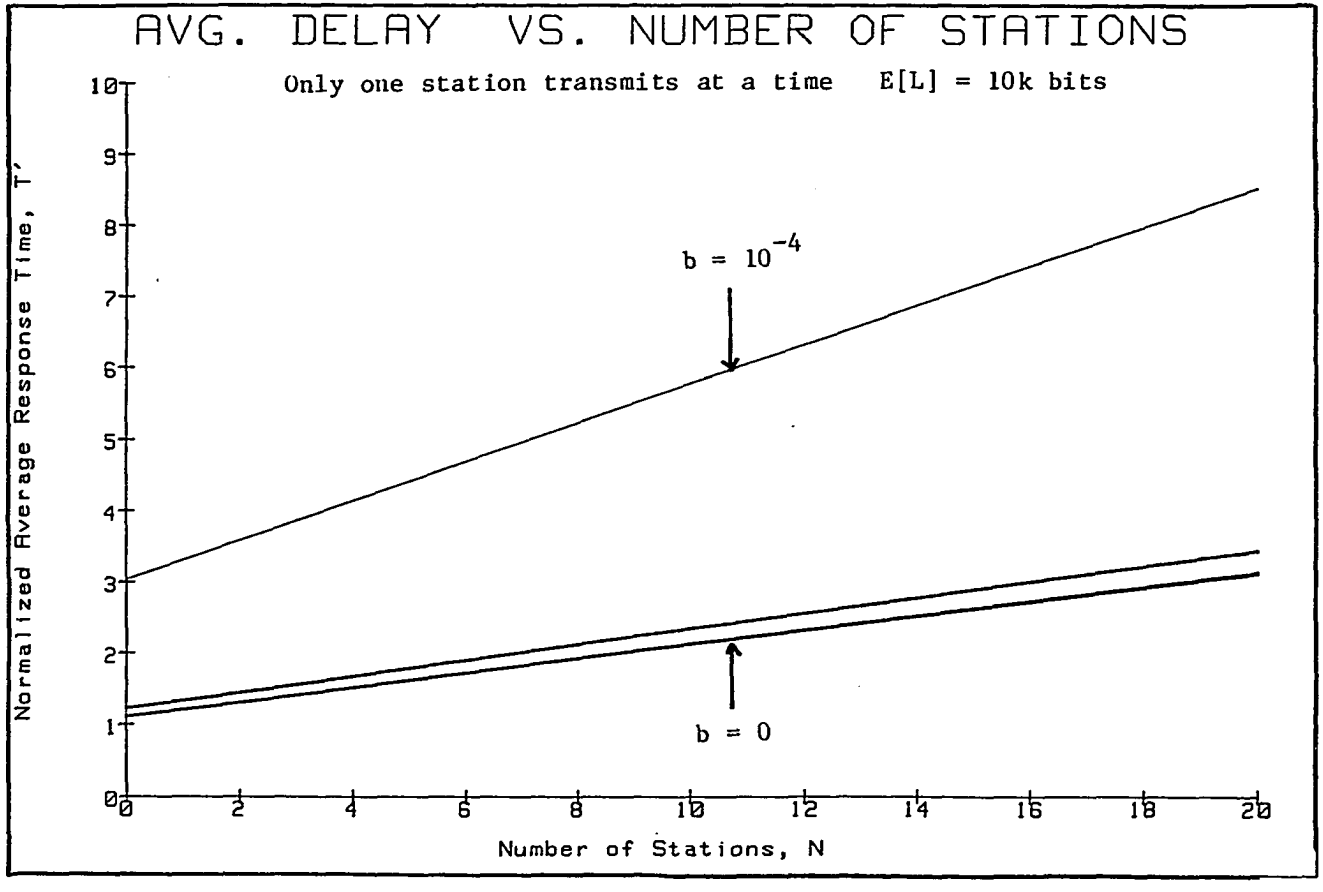


Figure 4-9. Normalized average response time as a function of number of stations. E[L] = 1k bits

to normalized average response time goes low with increase in message length. Also, it increases linearly with the number of stations. This is due to the linear dependence of the reservation time on N as seen in Eq. (4.10). Since the maximum achievable normalized average response time is 1, the graphs show that the system performs very well when there are fewer stations and larger message lengths.

Figures 4.11-4.14 provide us with similar information as seen in Figures 4.6-4.10 except that the latter has a wider range of values of $\bar{L}_m$. As seen earlier, better performance of high NEBR and low normalized response time is achieved with small number of stations, large message lengths, and few errors. Those figures also show that there is an optimal message length for each bit error probabilty.

The next discussion relates to the M/G/1 static priority queueing model based on the assumption of multiple ready stations taking part in the reservation process (Case 3). This model will be used to determine the bounds on the performance of the DCPR LAN, the motivation for which has already been discussed. This necessitates using maximum and minimum reservation times and highest and lowest priority stations. Intuitively, the lowest priority station achieves the lowest average delay because it has to wait until no other station has a message to transmit. Moreover, it takes the maximum time to reserve the channel because it has query all stations sequentially to close their links before it can transmit. During this 'long' time a higher priority station may become ready and with the reservation for the channel and so it has to keep on waiting. Clearly, the lowest priority exhibits the

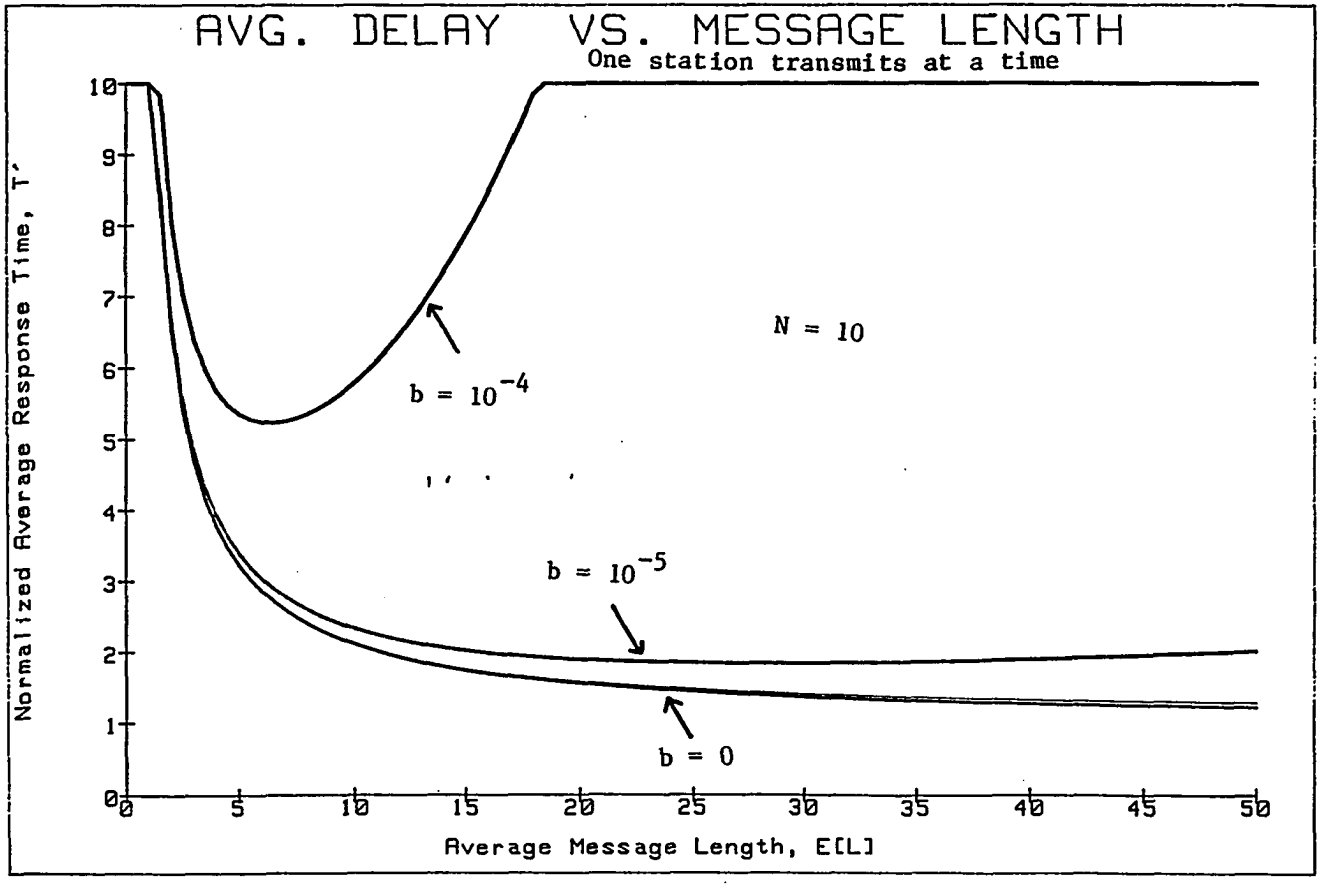


Figure 4-11. Normalized averaged response time as a function of the average message length. N = 10

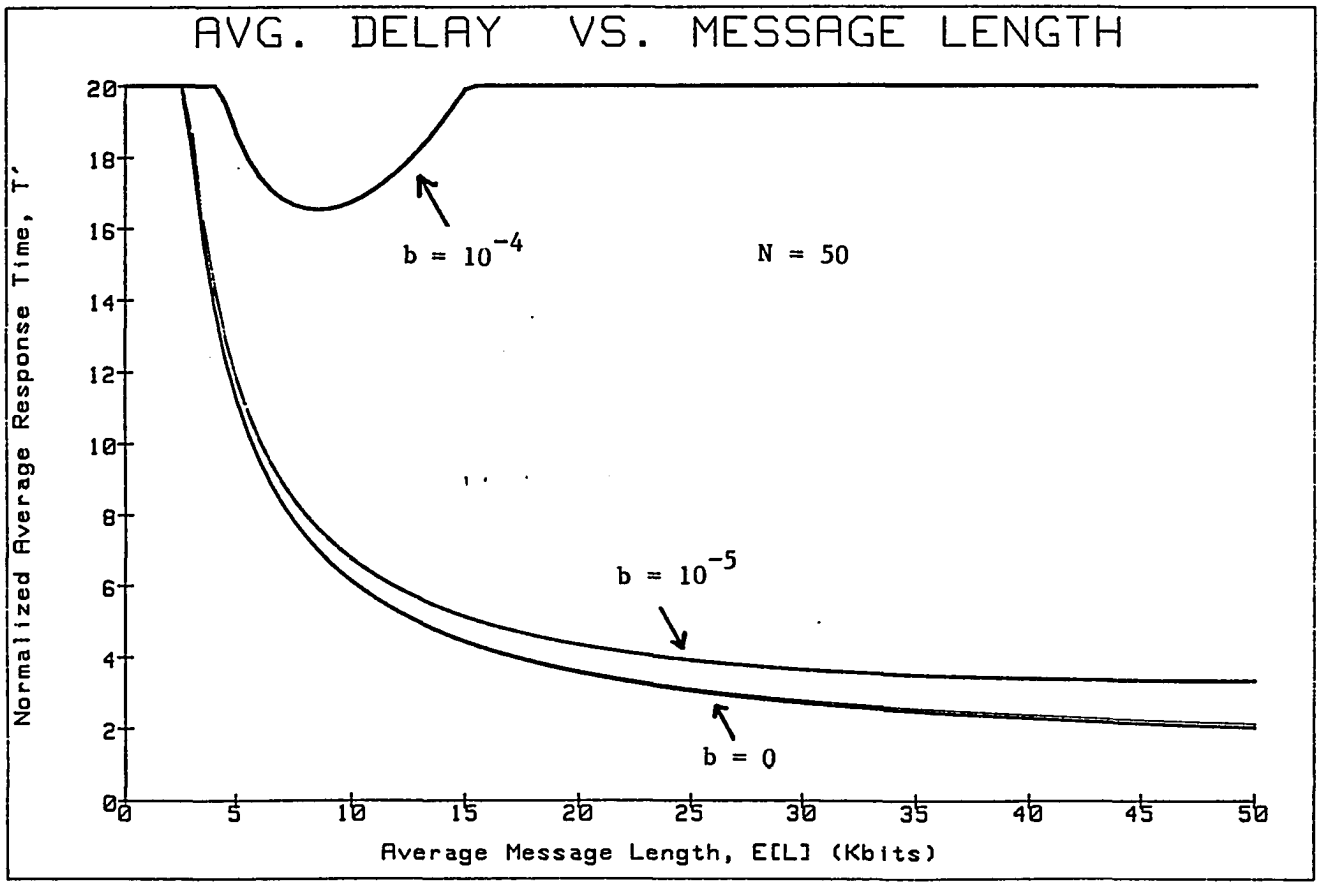


Figure 4-12. Normalized average response time as a function of average message length. N = 50

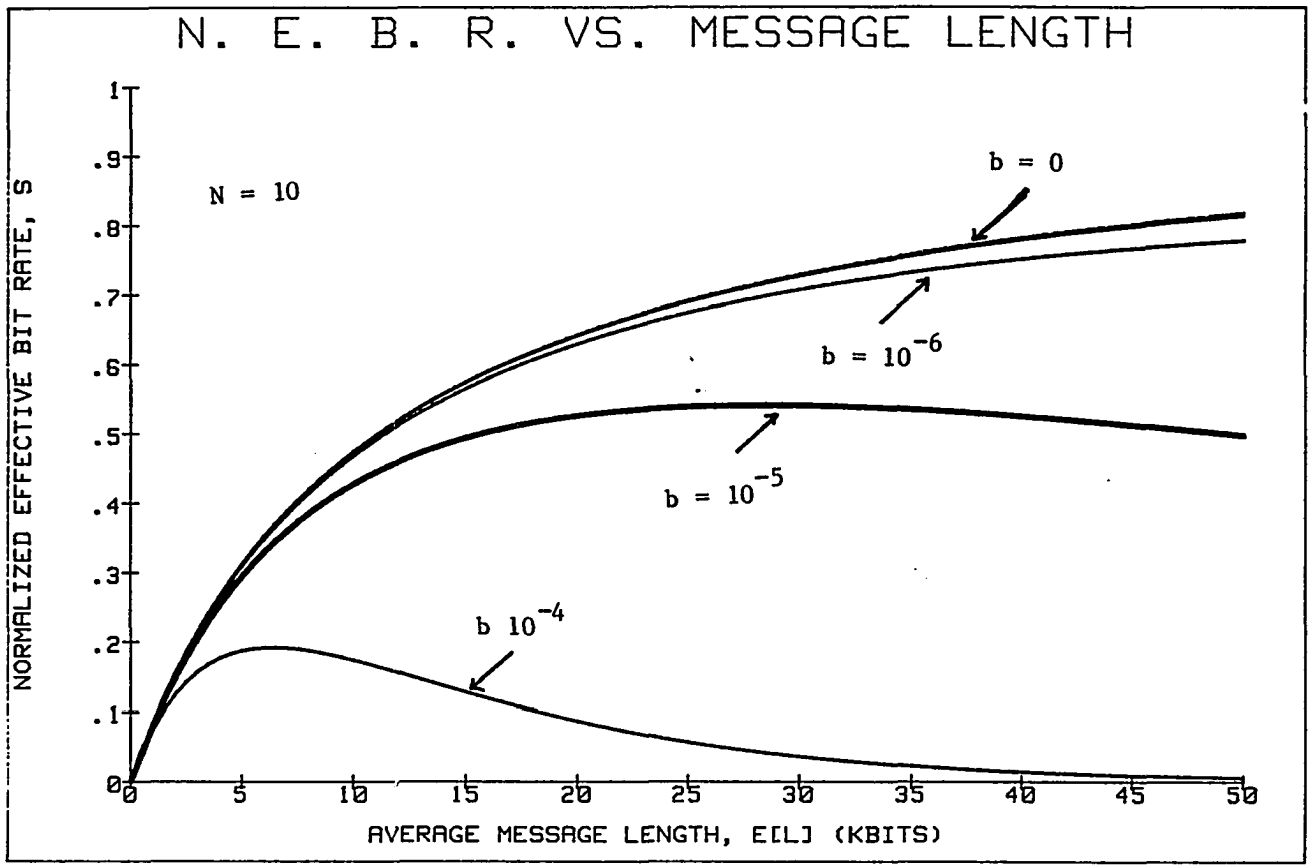


Figure 4-13. Normalized throughput as a function of average message length. N = 10

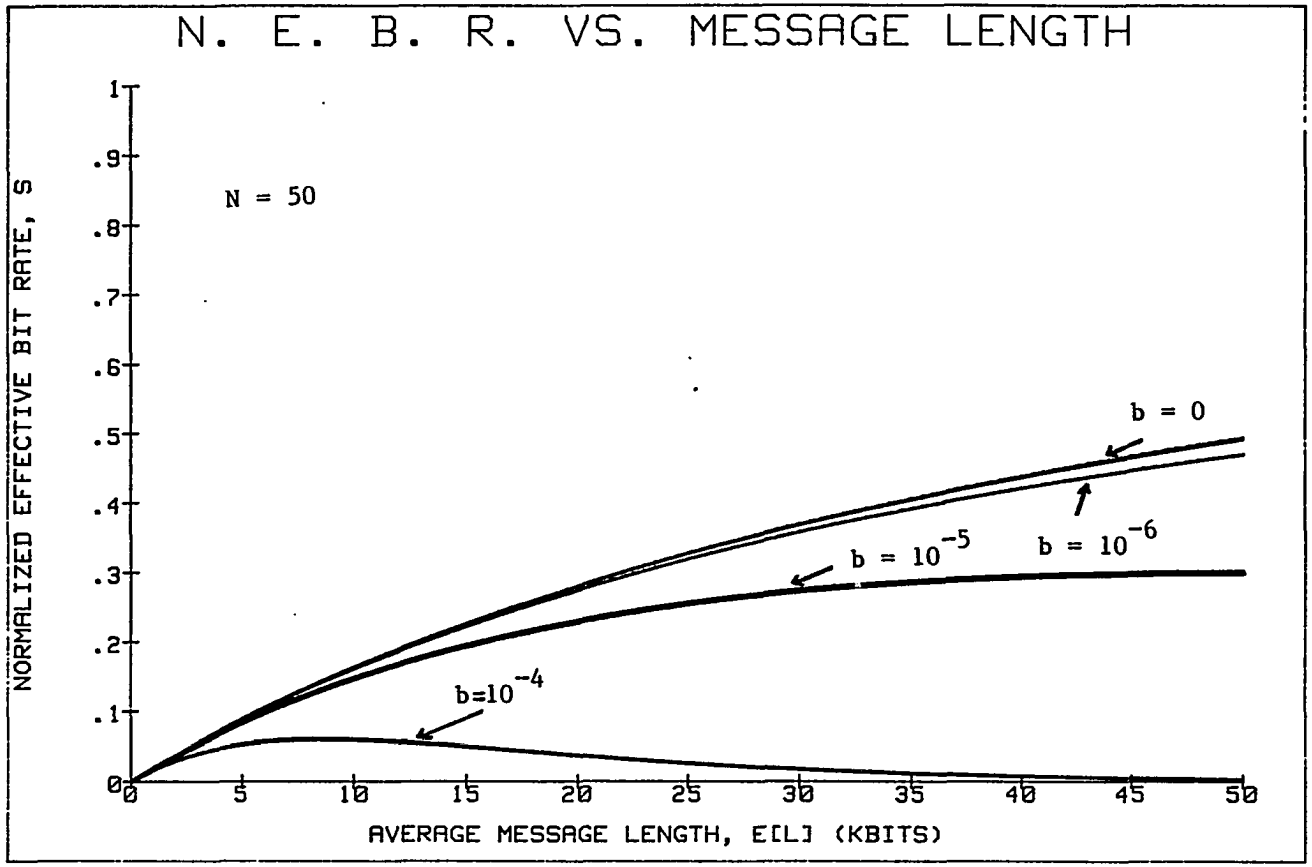


Figure 4-13. Normalized throughput as a function of average message length. N = 50

worst case average response time in every sense of the word. The foregoing means that a fixed reservation time of S_{max} is used together with a priority index of p=1 in Eq. (4.51) to obtain the worst case normalized average response time, Twc'.

Figures 4.15-4.16 show a plot of Twc' against NEBR for various values of average message length with fixed number of stations and constant message length. Figures 4.17-4.18 show the same thing with message lengths assumed to be exponentially distributed. Realizing the fact that Twc' is the worst case response time achievable by any station, it can be conjectured from the figures that an average message length of 1000 bits should never be used for the DCPR LAN. Message lengths of 10 kbits and above are desirable however, if NEBR greater or equal to 0.4 is the yardstick.

The best case delays are achieved by the highest priority station (the one with p=N where N is the number of stations). It does so at the expense of the lower priority stations. When all stations are ready before the reservation period the reservation time is minimum. These facts are used to obtain the plot of best case normalized average response time, T'_{bc}, against NEBR in Figure 4.19 for both fixed message and exponentially distributed message lengths.

Finally, Figure 4.20 shows a plot of Tp' versus NEBR for both best and worst cases.

In conclusion, it can be said that the DCPR LAN performs the best with fewer number of stations and larger message lengths.

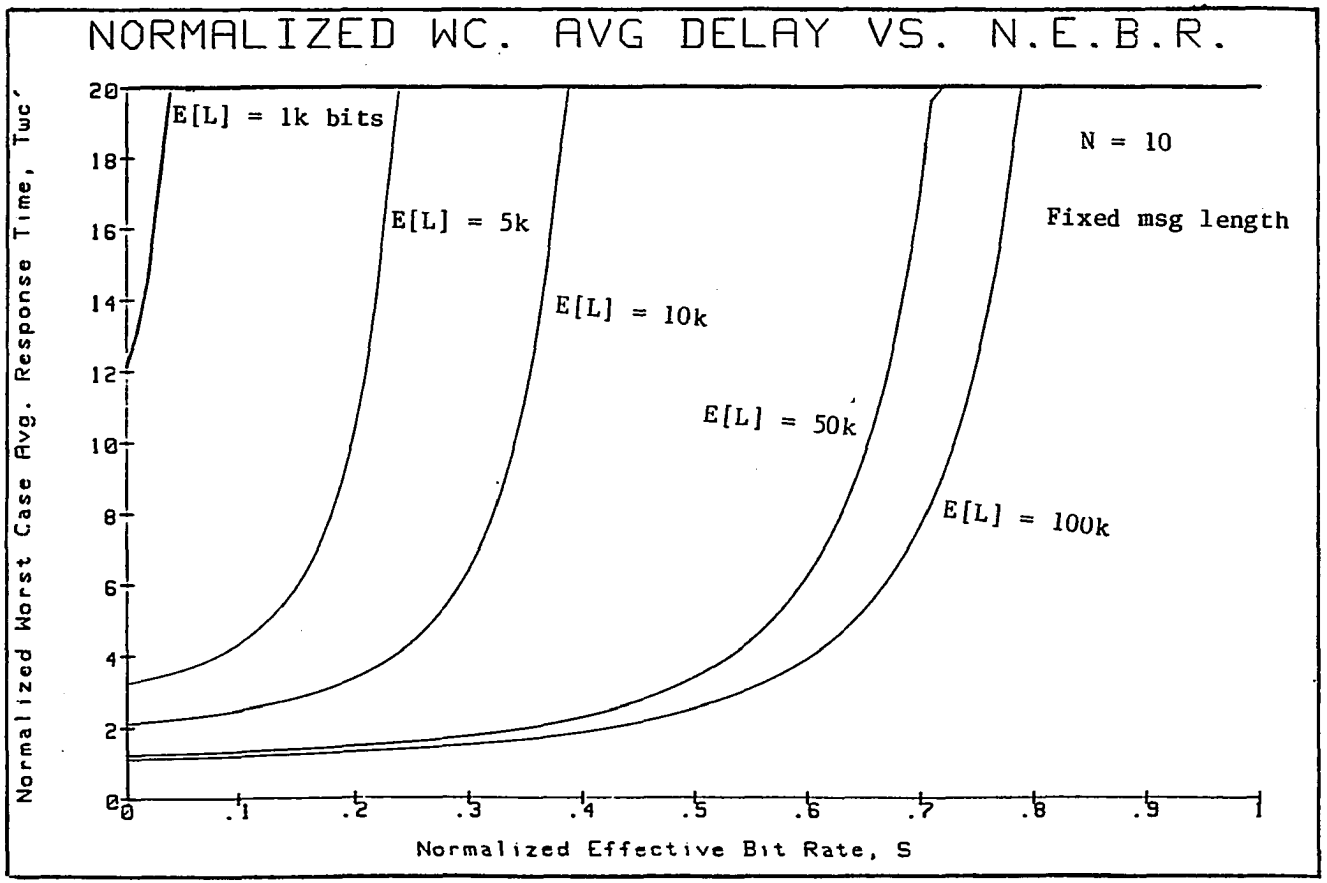


Figure 4-15. Normalized worst case average response time as a function of normalized throughput. N = 10. Fixed msg length

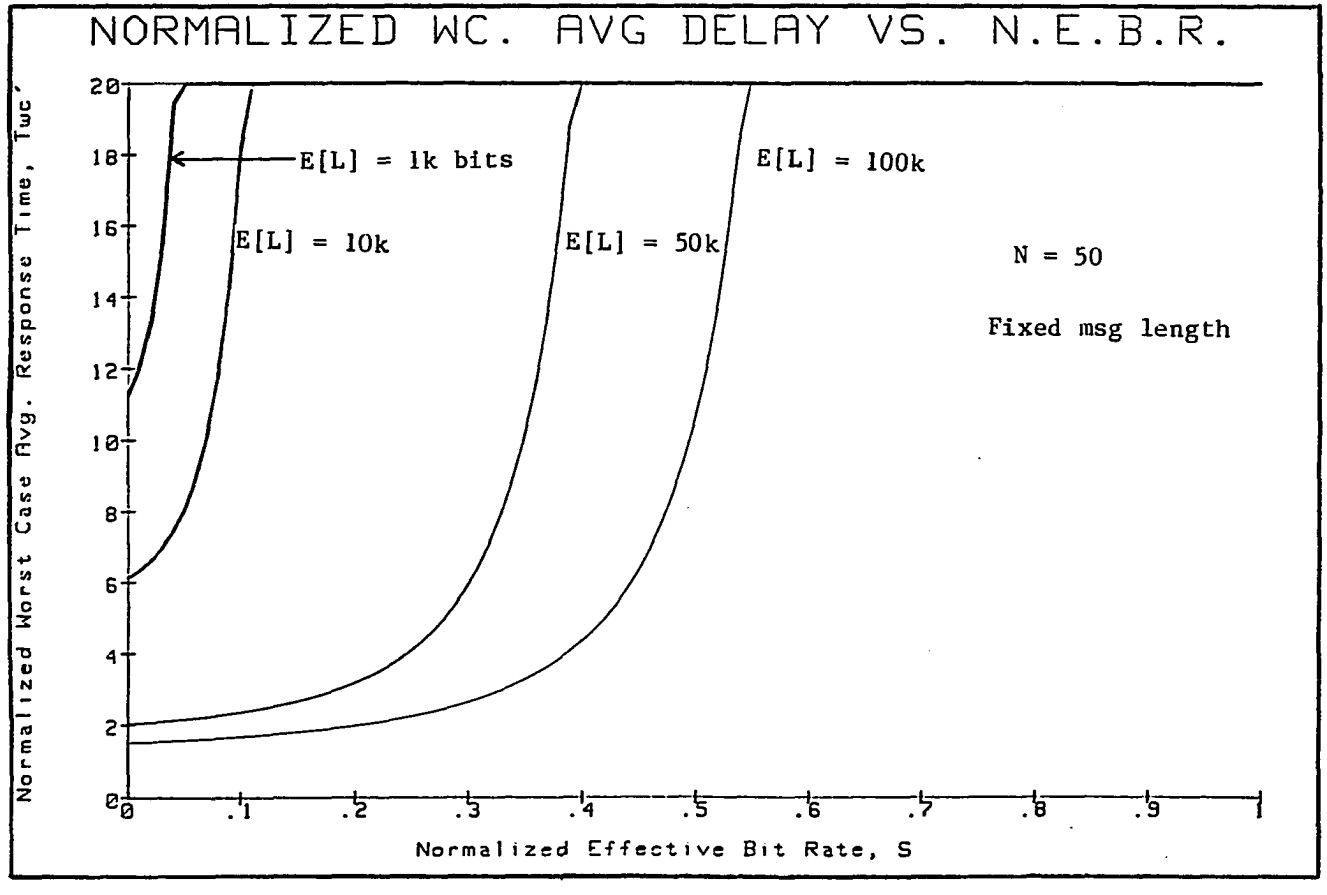


Figure 4-16. Normalized worst case average response time as a function of normalized throughput. N = 50. Fixed msg length

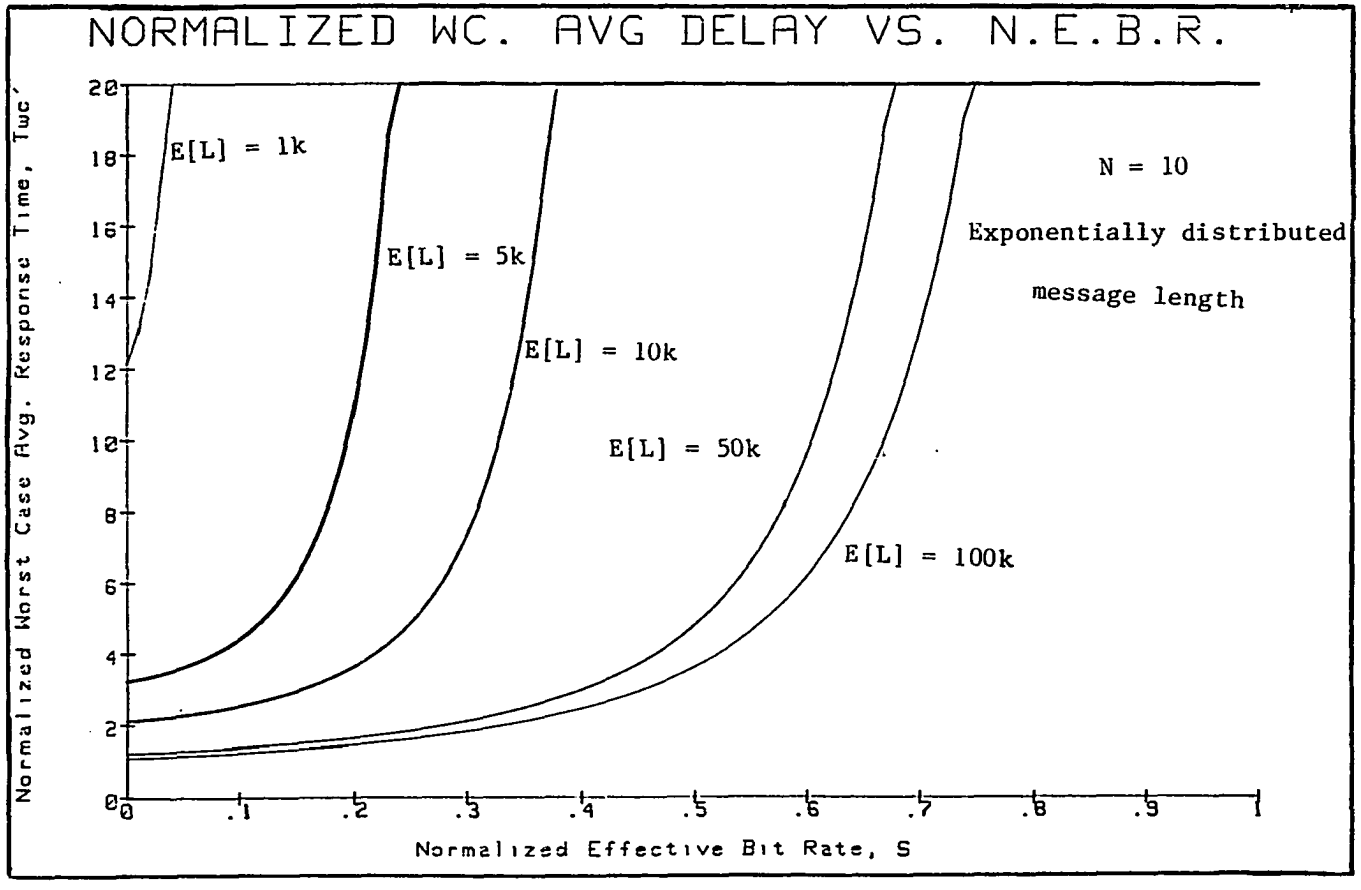


Figure 4-17. Normalized worst case average response time as a function of normalized throughput. N = 10. Exponentially distributed message length

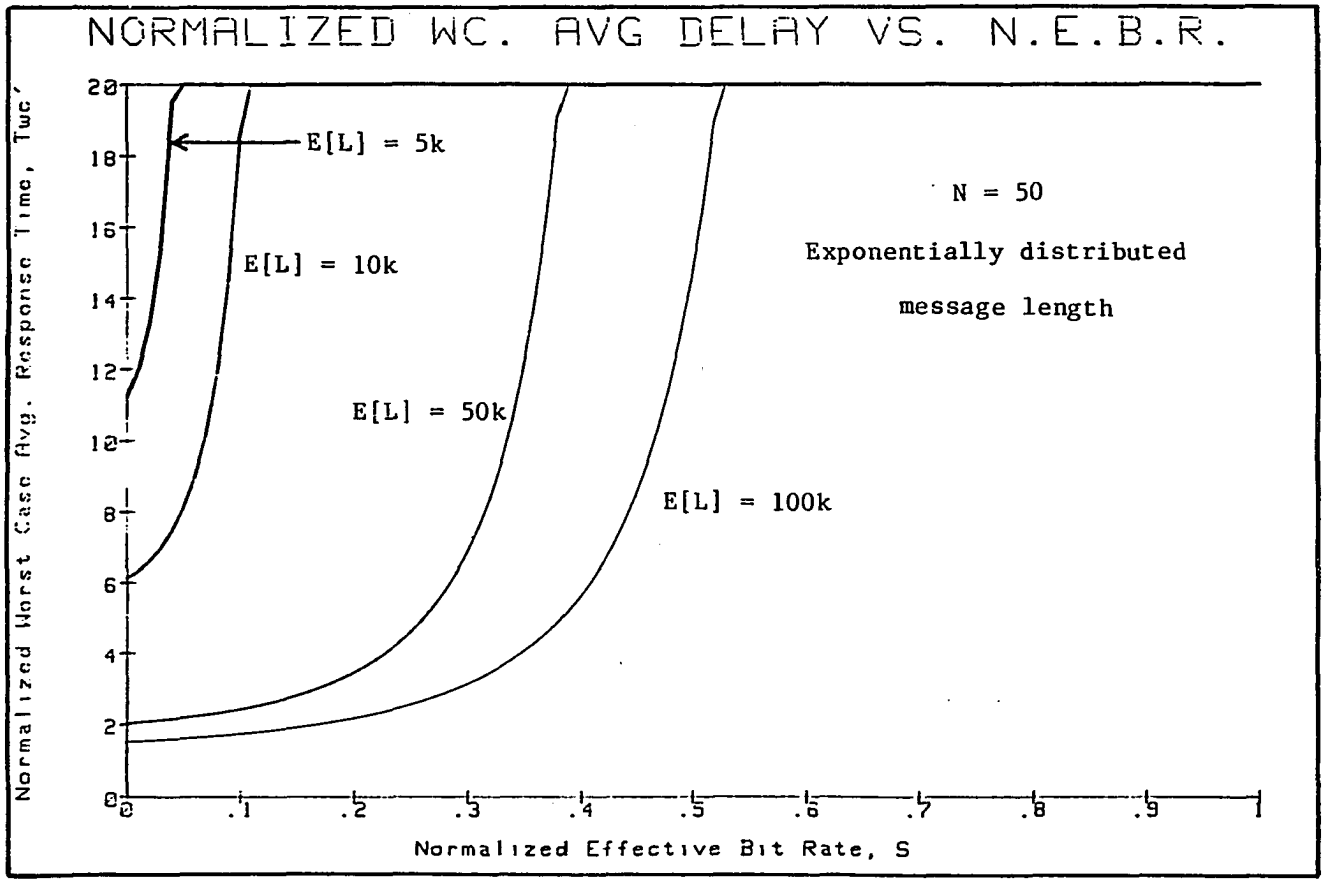


Figure 4-18. Normalized worst case average response time as a function of normalized throughput. N = 50.

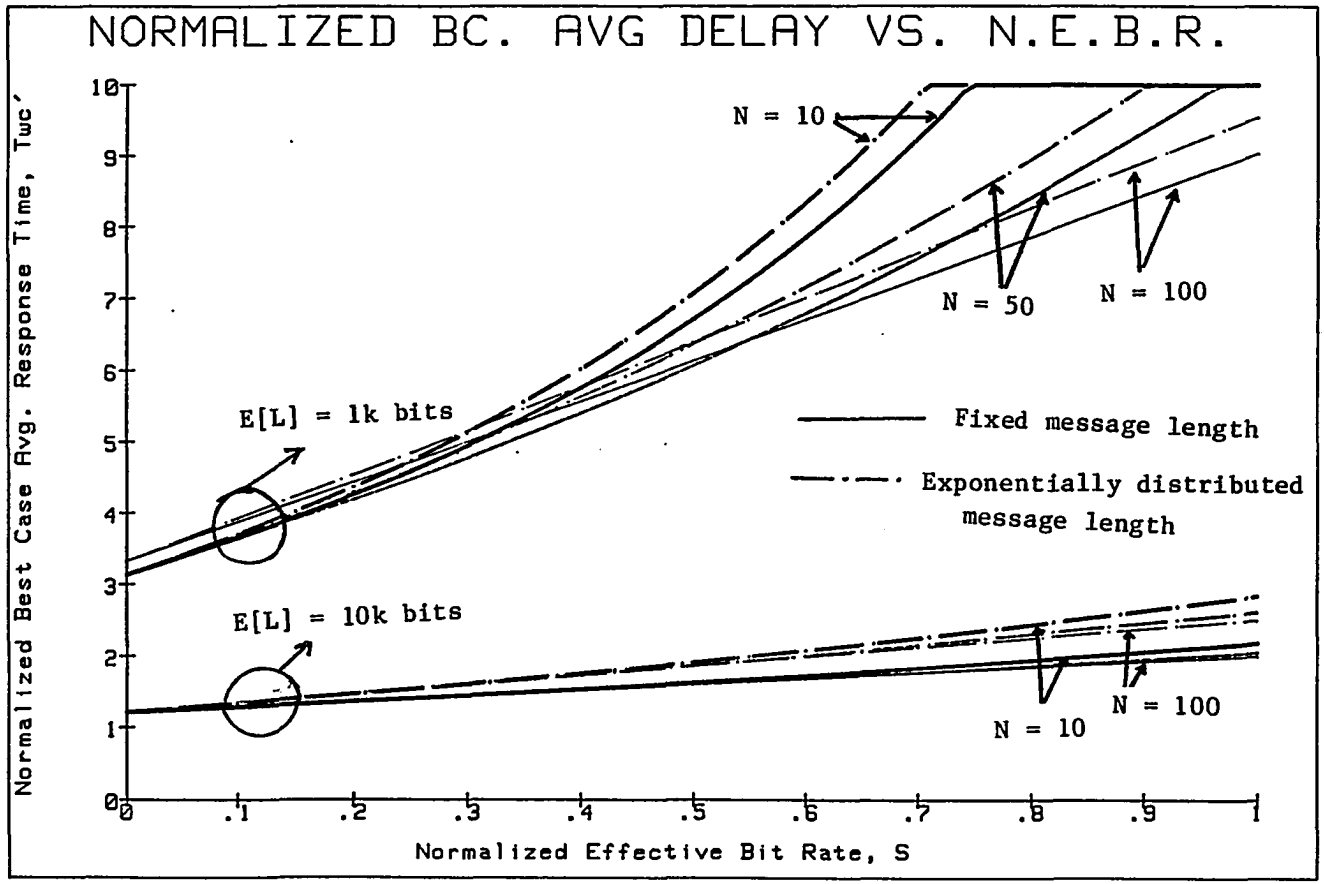


Figure 4-19. Normalized best case average response time as a function of normalized throughput for both fixed and exponentially distributed message lengths

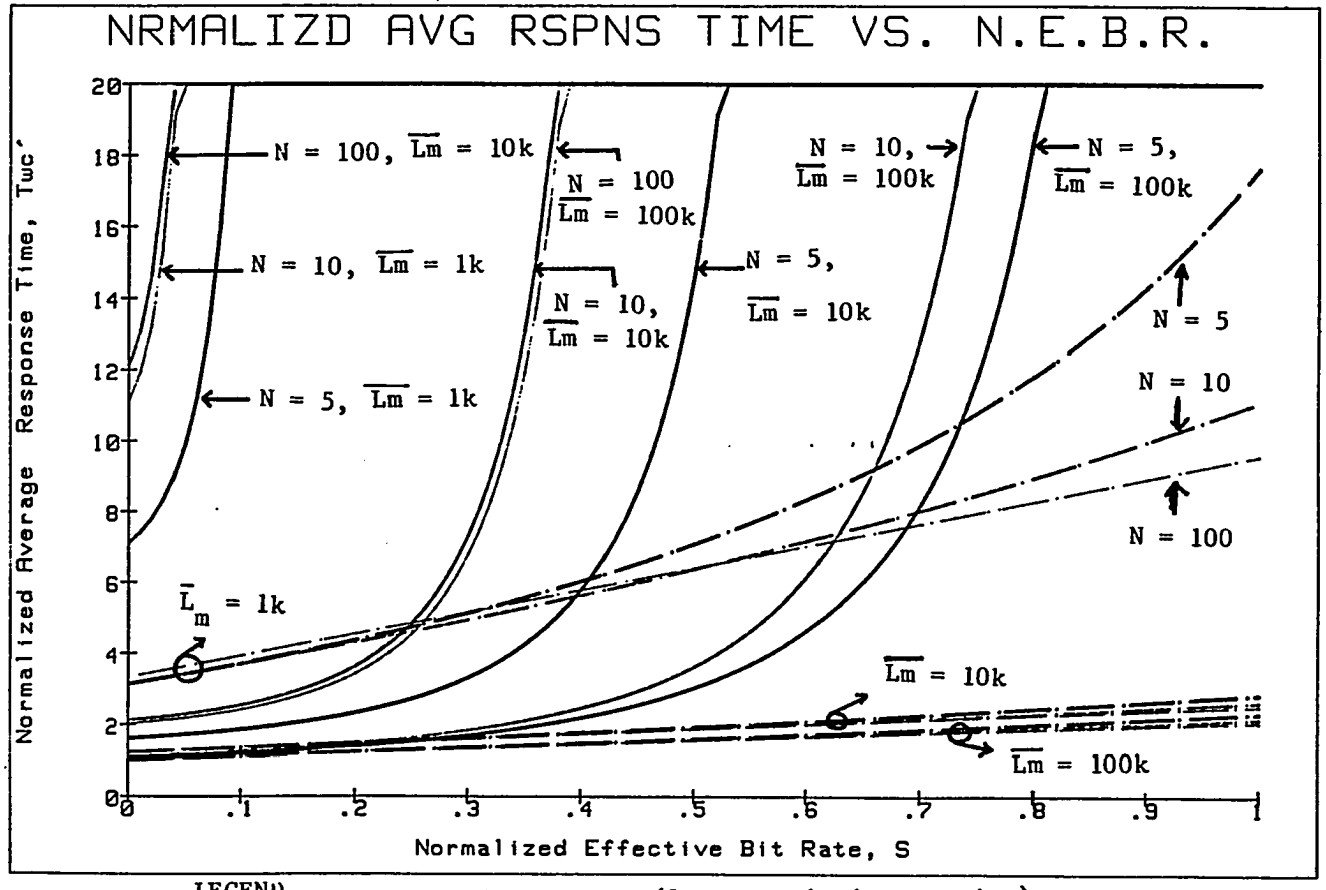


Figure 4-20. Normalized best and worst case average response time as a function of normalized throughput for exponentially distributed message lengths

CHAPTER 5. SIMULATION MODELING OF DCPR

Introduction

In this chapter a discrete-event computer simulation technique is used to model and analyze the DCPR LAN to obtain an accurate performance measurement. The limitations of the analytical methods of the previous chapter which motivated the need for simulation are discussed. To enable effective discussion in this chapter the following definitions are needed.

Definition 5.1

A station is said to be ready if it has a message in its buffer.

Definition 5.2

The readiness of a station is given by the binary random variable R_i defined as

$$R_i = 1 \text{ if station } T_i \text{ is ready, } i \in \{1, \ldots, N\}$$
$$= 0 \text{ otherwise} \qquad (5.1)$$

Definition 5.3

T_{rsvmax} = maximum reservation time for any configuration

T_{rsvmin} = minimum reservation time for any configuration.

Motivation for Simulation

The analytical method of the previous chapter made use of the M/G/1 static priority queueing model in which arrivals are based on a Poisson process, the service time is based on a general probability

distribution and the ring channel is considered as a single server. Furthermore as in all queueing systems, it is assumed that service times are independent and identically distributed (i.i.d) random variables [38,39].

Thus, in the DCPR system messages were assumed to arrive at the various stations according to a Poisson process. This is a general assumption for most real systems and is reasonable for all practical purposes.

For DCPR, the service time includes the time it takes to reserve the channel plus the time it takes to actually transmit a message and receive an acknowledgment.

There are problems associated with using the M/G/1 queueing model. First, the service time is a dynamic quantity whose value depends upon the state of the system just before reservation begins, where 'state' is defined as the vector $(R_1, R_2, \ldots, R_N)$. The probability distribution of the reservation time is unknown and cannot be assumed as in the case of message transmission time.

Second, the equations derived so far are for only 1 out of a possible (N-1)! configurations--decreasing order of priority.

Third, as a consequence of the dynamic property of the reservation time, the service times are not independent.

To prove the point we look at the reservation times in more detail.

The following cases are considered:

Let $t_w = 0$.

Case 1: Decreasing order of priority with ranking (1,2,3,4); only one

station is ready (see Figure 3-2).

This case has already been considered in Chapter 4 in which it is found that the ready station queries the stations sequentially (3 times) before it receives its RFR back. Thus, the reservation time is (see Eq. 4.10) given by:

$$T_{rsv} = 2t_{iw} + 3t_{RLWC} + t_{RFR} + t_{RL} \tag{5.2}$$

For N stations, the reservation time is given by

$$T_{rsv} = 2t_{iw} + (N-1)\, t_{RLWC} + t_{RFR} + t_{RL} \tag{5.3}$$

which is clearly an increasing function of N and is also the maximum reservation time, T_{rsvmax} for fixed N.

Case 2: Decreasing Order of Priority with ranking (1,2,3,4); all stations are ready (see Figure 3-3).

It is seen in Figure 3-3 that a ready station queries the stations only once before getting it RFR back. It is clear from the figure that the reservation time is given by:

$$T_{rsv} = 2t_{iw} + t_{RFR} + t_{RL}/4 + t_{RFR} + t_{RL} \tag{5.4}$$

For N stations,

$$T_{rsv} = 2t_{iw} + t_{RFR} + t_{RL}/N + t_{RFR} + t_{RL} \tag{5.5}$$

Note that this time is basically dependent on the ring latency, t_{RL}, which is a constant for a given configuration. It is also the minimum reservation time, T_{rsvmin} for fixed N.

Case 3: Decreasing Order of Priority with ranking (1,2,3,4). Only stations T_1 and T_3 are ready (see Figure 3-4).

For this case, station T_1 (the highest priority station) queries the stations 2 times before they close their links. Thus by arguments similar to above the reservation time is found to be

$$T_{rsv} = 2t_{iw} + t_{RLWC} + t_{RL}/2 + t_{RFR} + t_{RFR} + t_{RL} \qquad (5.6)$$

Case 4: Decreasing Order of Priority with ranking (1,2,3,4); stations T_1 and T_2 ready (Figure 5-1).

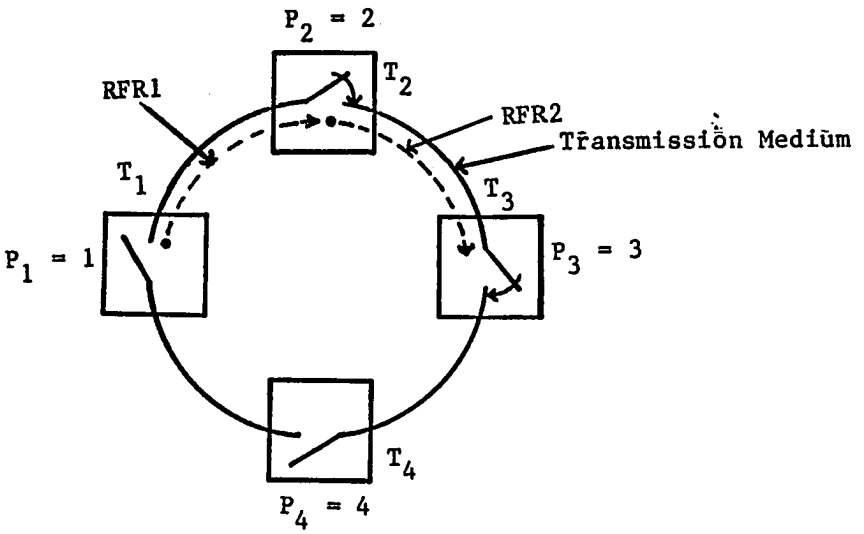
It is found from the figure that station T_1 gains access to the channel after sending 2 queries. Thus,

$$T_{rsv} = 2t_{iw} + 2t_{RLWC} + T_{RFR} + t_{RL} \qquad (5.7)$$

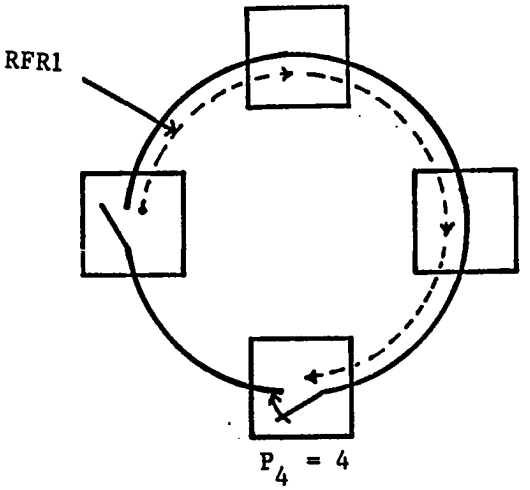
In a similar manner, the reservation times for all other possible combinations of ready stations can be determined for this configuration. A summary of these reservation times is shown in Table 5.1.

The table shows an interesting way in which the DCPR protocol operates to reduce the reservation time (overhead) as the load increases for this configuration. To speak in more general terms, as the load increases, the queries to the low priority stations to close their links adjusts from being sequential to parallel.

Furthermore, from Table 5.1, it can be seen that for a fixed configuration (i.e., priority ranking) the reservation time, T_{rsv}, depends on not only the number of ready stations, but also the identity of the ready stations. In other words, T_{rsv} is a function of the vector (R_1, R_2, R_3, R_4) as given by

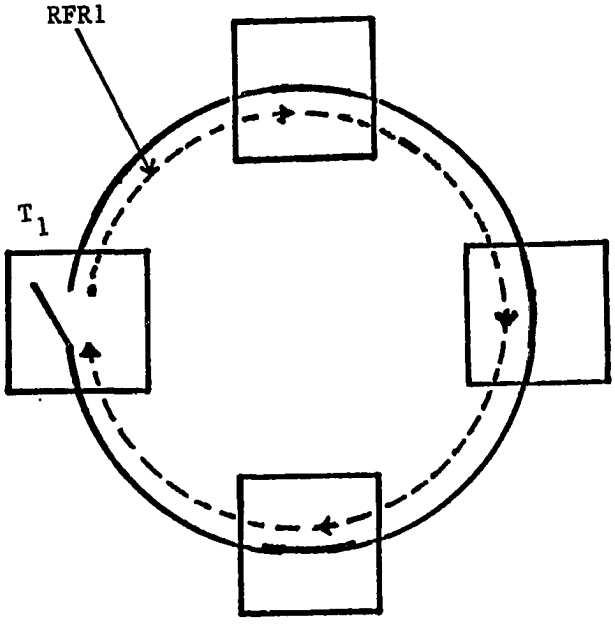


a. T_1 sends RFR1 to T_2; T_2 sends RFR2 to T_3: T_2 and T_3 close their links because T_2 receives a higher priority RFR nd T_3 is not ready



b. After timeout T_1 sends RFR1 again; T_4 receives RFR1 and closes its link

Figure 5-1. DCPR example: station priority in decreasing order; only station T_1 and T_2 ready



c. After timeout T_1 sends RFR1 and receives it back later

Figure 5-1. (continued)

Table 5.1. Reservation times for all possible combinations of ready stations for the decreasing order of priority configuration (1,2,3,4)

Station Readiness states $R_i = 1$ if T_i is ready, R = 0 otherwise				Number of queries sent by the highest priority ready station to have all low priority links closed	T_{rsv} (Reservation Time)
R_1	R_2	R_3	R_4		
0	0	0	0	0	0
0	0	0	1	3	T_{ovr}[a] + $3t_{RLWC}$
0	0	1	0	3	T_{ovr} + $3t_{RLWC}$
0	0	1	1	2	T_{ovr} + $2t_{RLWC}$
0	1	0	0	3	T_{ovr} + $2t_{RLWC}$
0	1	0	1	2	T_{ovr} + t_{RLWC} + $t_{RL}/2$ + t_{RFR}
0	1	1	0	2	T_{ovr} + $2t_{RLWC}$
0	1	1	1	1	T_{ovr} + t_{RLWC}
1	0	0	0	3	T_{ovr} + $3t_{RLWC}$
1	0	0	1	3	T_{ovr} + $2t_{RLWC}$ + $t_{RL}/4$ + t_{RFR}
1	0	1	0	2	T_{ovr} + t_{RLWC} + $t_{RL}/2$ + t_{RFR}

[a] $T_{ovr} = 2t_{iw} + t_{RFR} + t_{RL}$.

Table 5.1. (continued)

Station Readiness states R_i = 1 if T_i is ready, = 0 otherwise				Number of queries sent by the highest priority ready station to have all low priority links closed	T_{rsv} (Reservation Time)
R_1	R_2	R_3	R_4		
1	0	1	1	2	$T_{ovr} + t_{RLWC} + t_{RL}/4 + t_{RFR}$
1	1	0	0	2	$T_{ovr} + 2r_{RLWC}$
1	1	0	1	2	$T_{ovr} + t_{RLWC} + t_{RL}/4 + t_{RFR}$
1	1	1	0	1	$T_{ovr} + t_{RLWC}$
1	1	1	1	1	$T_{ovr} + t_{RL}/4 + t_{RFR}$

$$T_{rsv} = f(R_1, R_2, R_3, R_4) \tag{5.8}$$

As expected, the reservation time is maximum when only one station is ready before the reservation process begins and minimum when all stations are ready.

Now consider a different priority ranking, namely (4,3,2,1) which is an increasing order of priority.

When only one station is ready just before reservation Figure 5-2 shows that 3 queries are required before all a lower priority stations close their links. Thus in general, for this case, $T_{rsv} = T_{rsvmax}$. Also as seen in Figure 5-3, 3 queries are required to have all lower priority links closed for the case where all stations are ready. This also means $T_{rsv} = T_{rsvmax}$ for this case too.

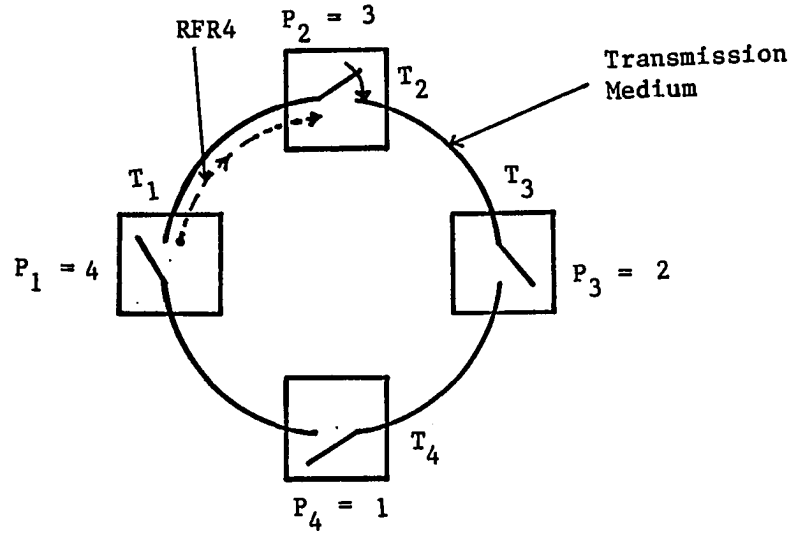
On the other hand, when stations T_1 and T_3 are ready 2 queries are required before the highest priority station (in this case T_3) is assured that it will receive its RFR back (see Figure 5-4). This means that

$$T_{rsv} = 2t_{iw} + 2t_{RLWC} + t_{RFR} + t_{RL} \tag{5.9}$$

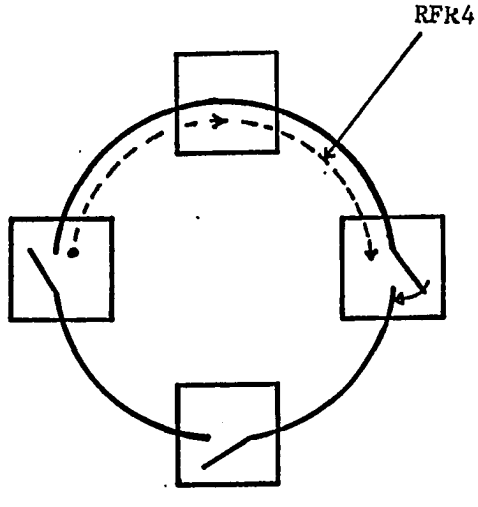
which is the same as Eq. (5.7).

In a similar manner, the reservation time for all other ready-station combinations can be obtained. These results are summarized in Table 5.2.

From the table, it is seen that for the configuration (4,3,2,1), the maximum number of queries (i.e., 3) is required to cause all low priority stations to close their links for all possible combinations of ready stations except the case where each ready station is adjacent

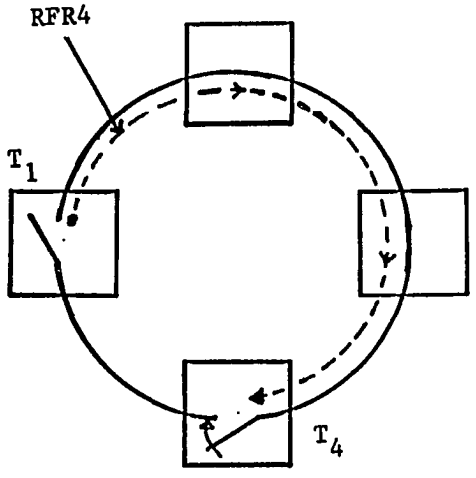


a. T_1 sends RFR4 to T_2; T_2 closes its link

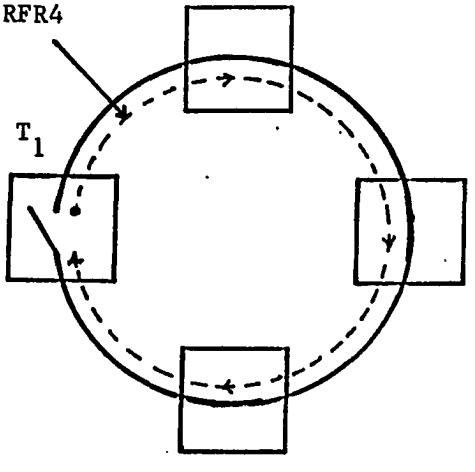


b. After timeout T_1 sends RFR4 again; T_3 receives it and closes its link

Figure 5-2. DCPR example: Station Priority in Increasing Order; only station T_1 ready

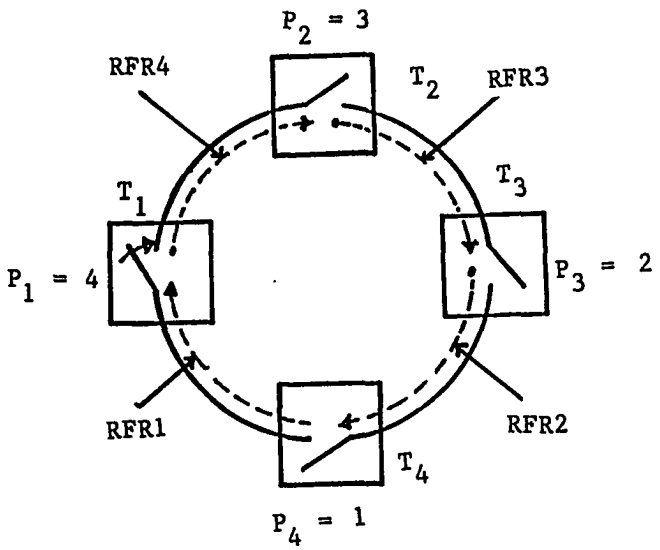


c. After timeout T_1 sends RFR4 again; T_4 receives it and closes its link

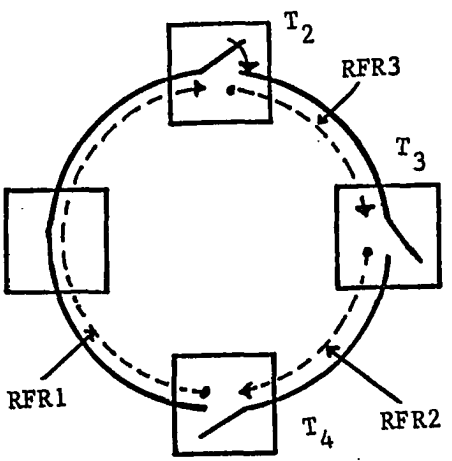


d. After timeout T_1 sends RFR4 again and receives it back later

Figure 5-2. (continued)

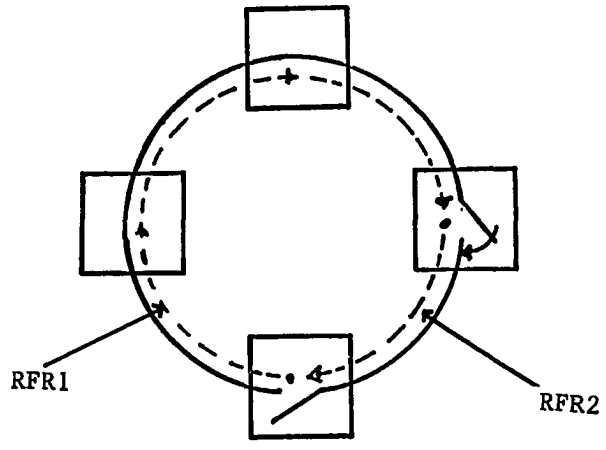


a. T_1 sends RFR4 to T_2; T_2 sends RFR3 to T_3; T_3 sends RFR2 to T_4; T_4 sends RFR1 to T_1; only T_1 receives a higher priority RFR so it closes its link

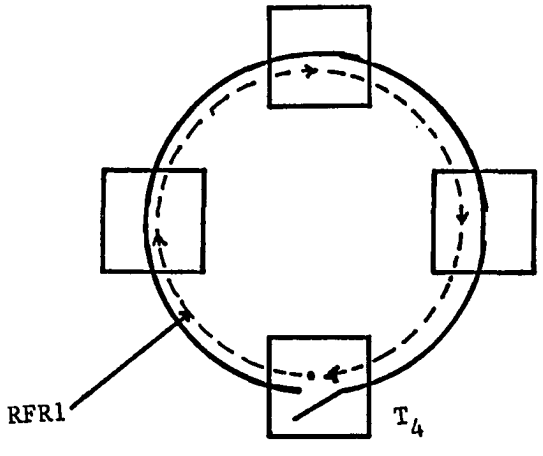


b. T_4 sends RFR1 again; T_2 sends RFR3 again; T_3 sends RFR2 again; only T_2 receives a higher priority RFR this time so it closes its link

Figure 5-3. DCPR example: station priority in increasing order; all stations ready

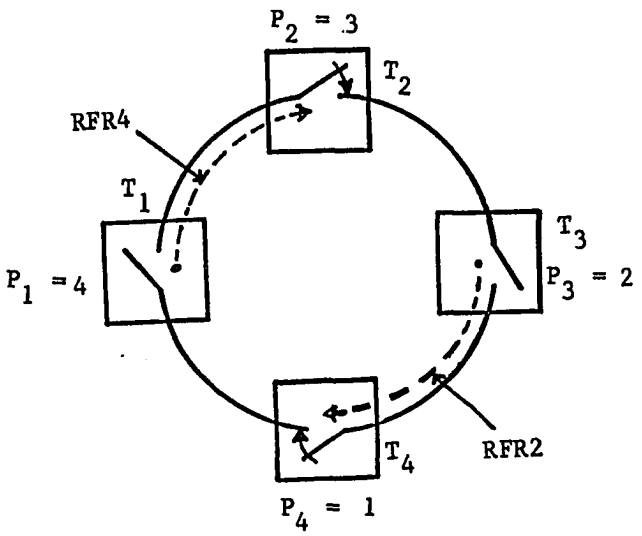


c. T_4 sends RFR1 again; T_3 sends RFR2 again; only T_3 receives a higher priority RFR so it closes its link

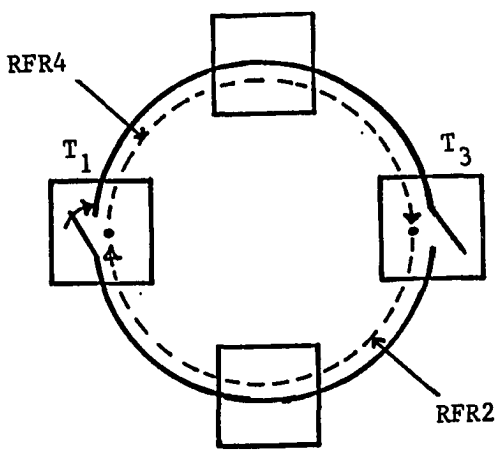


d. T_4 sends RFR1 again and finally receives it back later

Figure 5-3. (continued)

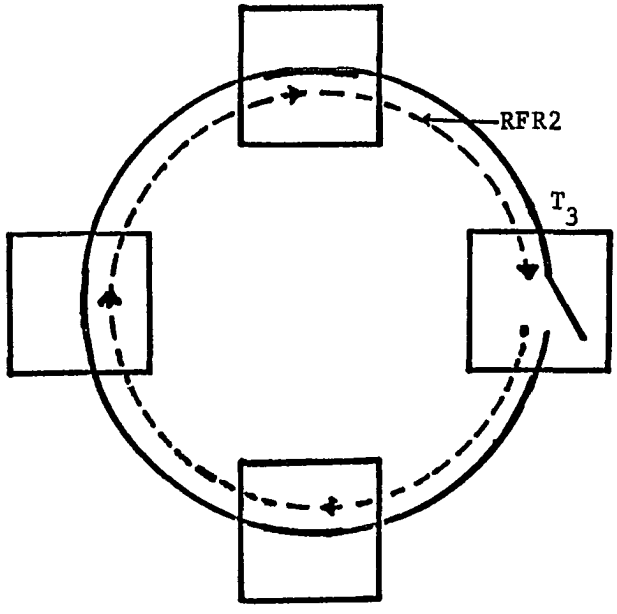


a. T_1 sends RFR4 to T_2; T_2 closes its link; T_3 sends RFR2 to T_4; T_4 closes its link



b. T_1 sends RFR4 again; T_3 sends RFR2 again; T_1 receives a higher priority RFR so it closes its link

Figure 5-4. DCPR example: station priority in increasing order; only stations T_1 and T_3 ready



c. T_3 sends RFR2 agains and receives it back

Figure 5-4. (continued)

Table 5.2. Reservation times for all possible combinations of ready stations for the increasing order of priority configuration (1,2,3,4)

Station Readiness states R_i = 1 if T_i is rdy = 0 otherwise				Number of queries required to have all low priority links closed	T_{rsv} (Reservation Time)
R_1	R_2	R_3	R_4		
0	0	0	0	0	0
0	0	0	1	3	T_{ovr}[a] + $3t_{RLWC}$
0	0	1	0	3	"
0	0	1	1	3	"
0	1	0	0	3	"
0	1	0	1	2	T_{ovr} + $2t_{RLWC}$
0	1	1	0	3	T_{ovr} + $3t_{RLWX}$
0	1	1	1	3	"
1	0	0	0	3	"
1	0	1	0	2	T_{ovr} + $3t_{RLWC}$
1	0	1	1	3	T_{ovr} + $3t_{RLWC}$
1	1	0	0	3	"
1	1	0	1	3	"
1	1	1	0	3	"
1	1	1	1	3	"

[a] $T_{ovr} = 2t_{iw} + t_{RFR} + t_{RL}$.

to a non-ready station (which requires 2 queries). In fact, it is safe to say that the reservation time for this configuration is constant at T_{rsvmax}. Therefore this configuration can be said to give the worst performance of any others.

Intuitively the decreasing order of priority seems to offer the best performance due to its dynamic way of reducing the reservation time as the load increases.

The foregoing shows that in general, T_{rsv} depends on the priority ranking of the stations as well as the number of stations. This is clear from Tables 5.1 and 5.2.

Since a station is ready (i.e., will vie for the channel) if, and only if it has a message (i.e., its queue of messages is not empty) to transmit it means that T_{rsv} also depends on the arrival rate of messages to the stations. As a result of the above arguments, it is clear that, in general,

$$A = f(\lambda_1, \ldots, \lambda_i, \ldots, \lambda_N, P_1, \ldots, P_i, \ldots, P_N, N) \quad (5.10)$$

where A is a random variable representing the reservation time,

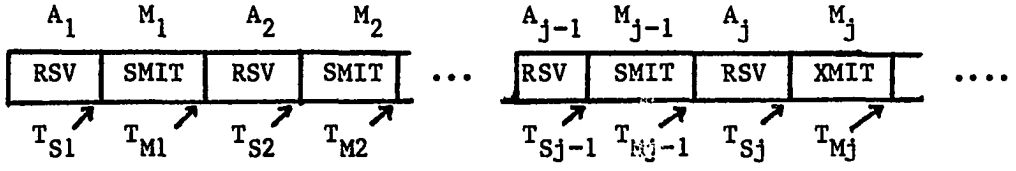
λ_i = Poisson arrival rate of messages to station T_i,

P_i = priority ranking of station T_i (note: the lower the number, the higher the priority)

and

N = the number of stations on the ring.

Now consider the time model (channel transactions) for DCPR (Figure 5-4).



LEGEND: T_{Sj} = epoch at which the j^{th} reservation of period A_j begins

T_{Mj} = epoch at which the j^{th} message transmission of period M_j begins

Figure 5-5: DCPR Time Model (channel transactions)

From the figure it is seen that the service time is given by

$$X_j = A_i + M_j \tag{5.11}$$

It is also noticed that S_j and M_j are independent random variables. The set $\{M_j : j=1,2,\ldots\}$ also consists of independent and identically distributed random variables. However,

$$A_j = f(A_{j-1}, M_{j-1}, N) \tag{5.12}$$

for a fixed configuration because the larger the values of A_{j-1} and M_{j-1} the more will there be messages queued at the stations based on the arrival rates of messages.

Equation (5.12) shows that the set of random variables $\{A_j : j=1,2,\ldots\}$ are not independent. In fact, A_j and M_{j-1} are not independent either.

Therefore, it can be claimed that the service times X_1, X_2, ..., X_j... are not independent. In fact this non-independence is especially

true for the decreasing priority ranking we used in Chapter 4 and some others. Since this independence was used in the previous models, in fact all known queueing models are baed on independence) a new approach is needed to obtain performance data on DCPR. Simulation holds the best promise of solution to provide a good analysis in the general case.

Introduction to simulation modeling

The purpose of a simulation model is to allow a designer to use a computer program to imitate the behavior of a system.

It is essential to distinguish between a simulation and an emulation here. In the latter, the programmer seeks to represent every detail of the system being studied. In simulation it is important to represent only those details which are relevant to the study being conducted.

Compared with a mathematical model a simulation model may more easily be used to create a comprehensive representation of a complex system of queues, data link protocols, etc.

A computer simulation system provides facilities to build a simulation model, takes as input a workload description, and uses a simulator mechanism to perform experiments. The simulation model and workload description determine the behavioral information that is generated and recorded for performance analysis by the simulator.

An efficient performance analysis simulator generally simulates only those events that change the system state. Such simulators, called DISCRETE EVENT or EVENT-SCHEDULING SIMULATORS, jump from event-to-event

in a simulated time. A key factor that affects the processing required to do a simulation is the total number of events. Clearly, the efficiency of the simulator is directly related to the level of detail of the simulation. A general flow diagram of an event-scheduling simulation system is shown in Figure 5-6.

The simulator begins by first obtaining the input parameters which describe the work load description. Initialization then follows. One of the things that is done during the initialization is to produce the initial event and place it on an event list. The event is then selected and processed. During the processing one or more events are produced and scheduled. Events are scheduled (i.e., placed in the event list) in a chronological order. Thus, the most imminent event is selected each time for processing. This means some events may occur at the same time. This is the key feature of an event-driven simulator--the capability of pseudo-coincident simulation activity.

Since each event spawns one or more additional events so that new events are constantly being created the simulation will terminate not by running out of events but by running past a pre-set time limit (simulation time). It should be pointed out here that this simulation time may not be necessarily continuous but, could be discrete.

There are various ways of ending the simulation. One way of doing it as shown in Figure 5-6 is to schedule a simulation-complete event in some future time. When this event is selected for processing statistics that were gathered during the simulation are reported and

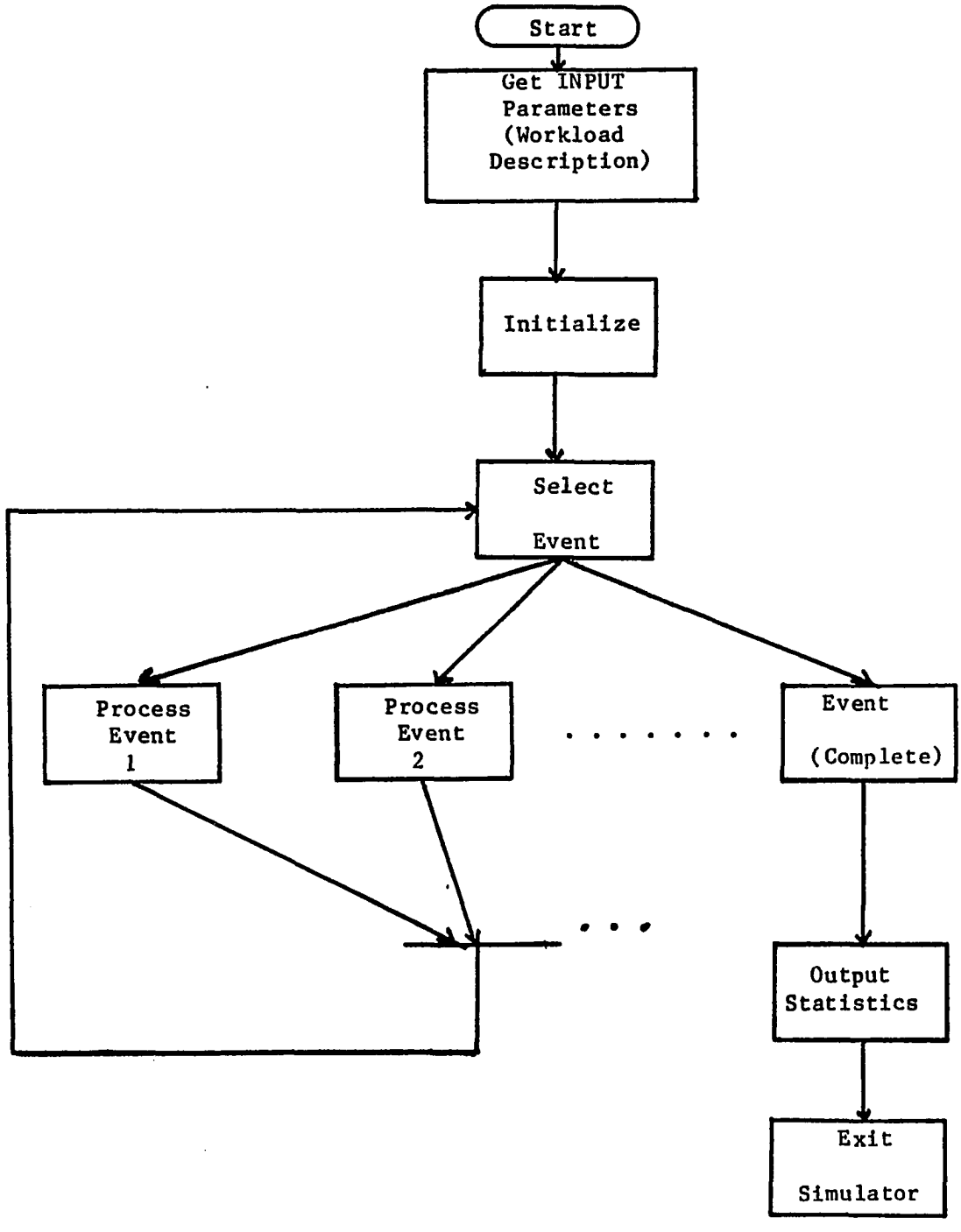


Figure 5-6. Activities in simulator

then the simulator is exited.

DCPR Simulation Model

Introduction

For the purposes of simulating the DCPR system the following assumptions can be made. These assumptions are the same as those used in the analytic model of an M/G/1 queue.

1) Messages arrive at each station, i, according to a Poisson process with a given rate λ_i.
2) The arrival processes at the stations are independent of each other.
3) The length of messages arriving at station, i, is distributed according to a general distribution with given average value $E[M_i]$. Specifically, exponentially distributed lengths and fixed lengths will be used.
4) No errors (i.e., communication line errors or station failures) occur in the DCPR system.

It should be pointed out that these assumptions are reasonable in practice. The only thing that can cause these assumptions to be changed is if statistics gathered from the real running DCPR system indicate such a change should be made.

Generally the results to be expected from such a simulation model are:

$$E[\vec{Q}],\ E[\vec{D}] = f(\vec{\lambda},\vec{p},\vec{M},N)$$

and

$$E[D] = \sum_{k=1}^{N} E[D_K] \qquad \text{: Expected total delay}$$

where

$\vec{D} = (D_1, \ldots, D_N)$: Delays

$\vec{\lambda} = (\lambda_1, \ldots, \lambda_N)$: Poisson input rates

$\vec{M} = (M_1, \ldots, M_N)$: Message lengths

N : Number of stations

$\vec{p} = (p_1, \ldots, p_N)$: Priority assignment

$\vec{Q} = (Q_1, \ldots, Q_N)$: Queue length

for some ranges of the variables, N, M, p, and . Specifically, equal message arrival rates and average message lengths will be assumed for both fixed and exponentially distributed cases. The results of interest and importance are those of the best and worst cases.

This means, for example, the best and worst cases of $E[D_1]$ and $E[D_N]$, respectively, are determined for fixed values of λ, M, and configurations $(p_1, \ldots, p_N)$ as N is varied.

System Configuration and Specification

Based on the specification of the simulation model to be built and run a block diagram of the components of a typical DCPR station (with connections) and a formal state specification of the DCPR station manager is developed.

Figure 3-5 shows a block diagram of the DCPR SAN model to be used for simulation. Each station is shown to have both an identity and a priority. For example, the identity of station i is STA(i) and its

priority is PRIORITY(i). The USER layer is an unspecified layer from where messages arrive. The USER layer will typically also receive acknowledgment.

Figure 5-7 shows a block diagram of the main components of a typical DCPR station to be used for simulation. Notice that the difference between this figure and Figure 3-7 is the queue. Strictly speaking the queues are present in the higher layer—the Logical Link Control Layer, to use IEEE 802 terminology. The reason why the queue is included in this model is to permit quantification of average queue lengths, the latter being useful information on storage requirements. The main component in Figure 5-7 is the DATA LINK MGR or the MEDIUM ACCESS CONTROL (MAC) MANAGER. This MAC MGR will be specified as a Finite State Machine (FSM). The other components are the IDLE_TIMER of type TIMER which is used to detect whether the channel is idle, the OTHER_TIMER of type TIMER which is a general purpose programmable timer used to set timeouts, the QUEUE which is used to hold messages that arrive from the user and, finally the USER which of type UPM (for Unspecified Protocol Machine). It is called UPM because for the purposes of our simulation model it is a fictitious layer above the DCPR station from where messages originate and to where acknowledgments if any, are sent.

The figure also shows the kinds of events that can occur in each component and from where they originate. All of these events are instantaneous (pulsed inputs and outputs) and they may cause changes of state in the various components.

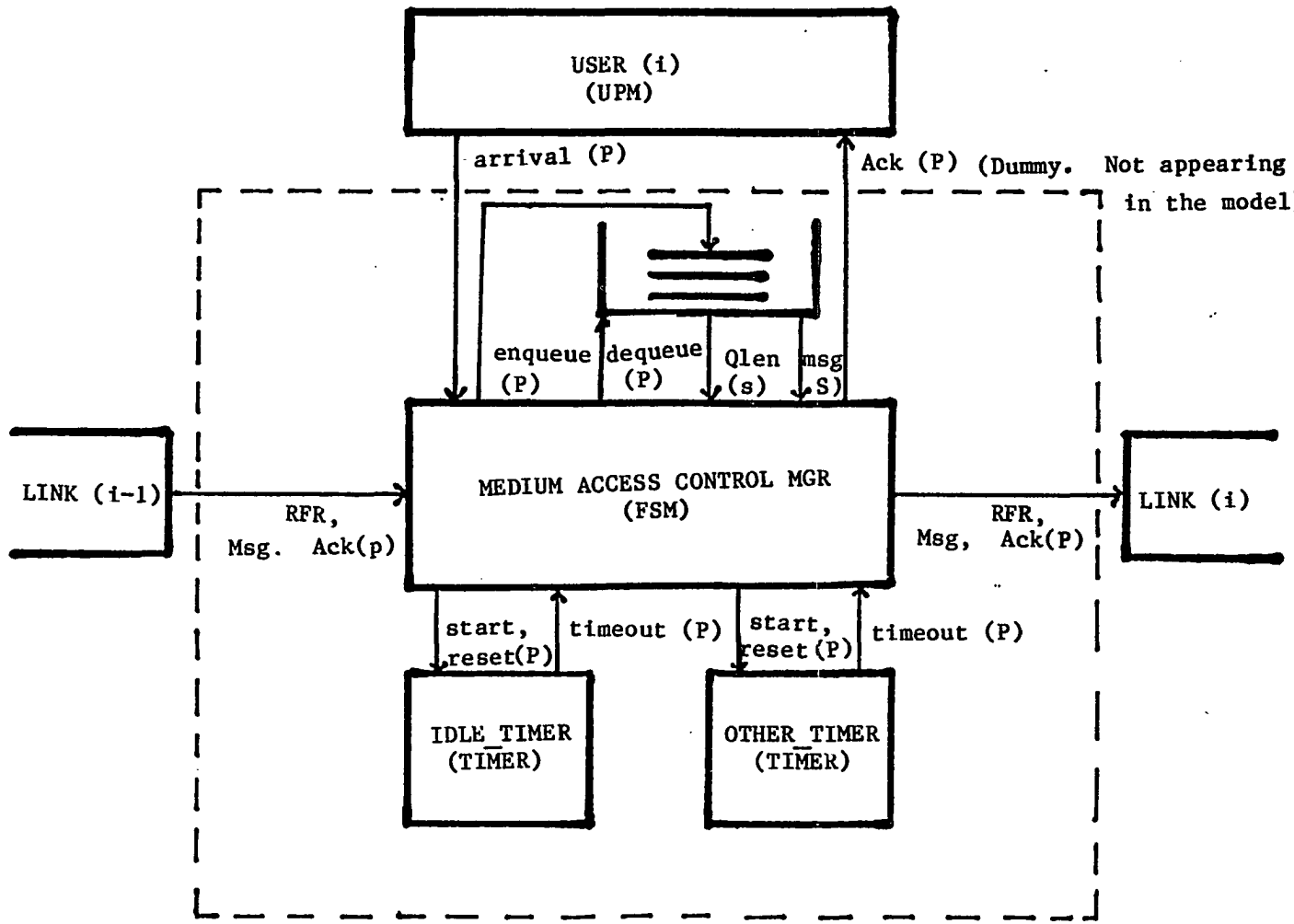


Figure 5-7. SAN block diagram showing internals of a DCPR station

Figure 5-8 shows the specification of the internal behavior of the MAC manager. Note that Figures 5-7 and 5-8 are all that are needed to describe the DCPR protocol that will be simulated. As explained earlier the vertical lines represent states. The state names are indicated on the top left edge of the corresponding vertical line representing each state. The horizontal lines represent transitions from a state to another state pointed to by the corresponding arrow. The events that cause transitions to occur are on top of the corresponding vertical line and the resultant events if any, are shown below the corresponding horizontal line. If no events occur as a result of a transition it is indicated with a '_'. The simulation code will be derived directly from Figure 5-8.

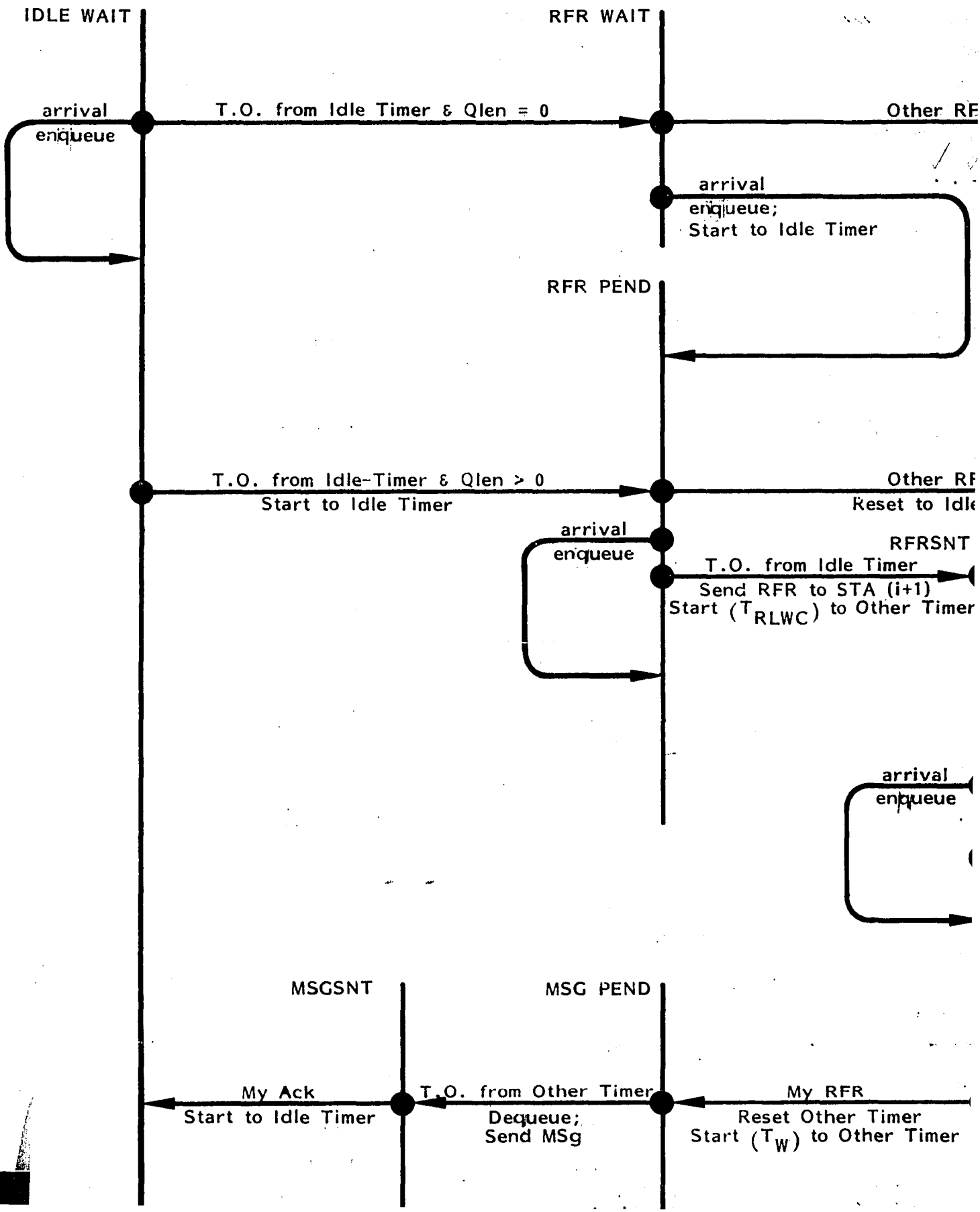
The simulation software is written in PASCAL and is designed in such a way that to check whether it agrees with Figure 5-8 is straight forward.

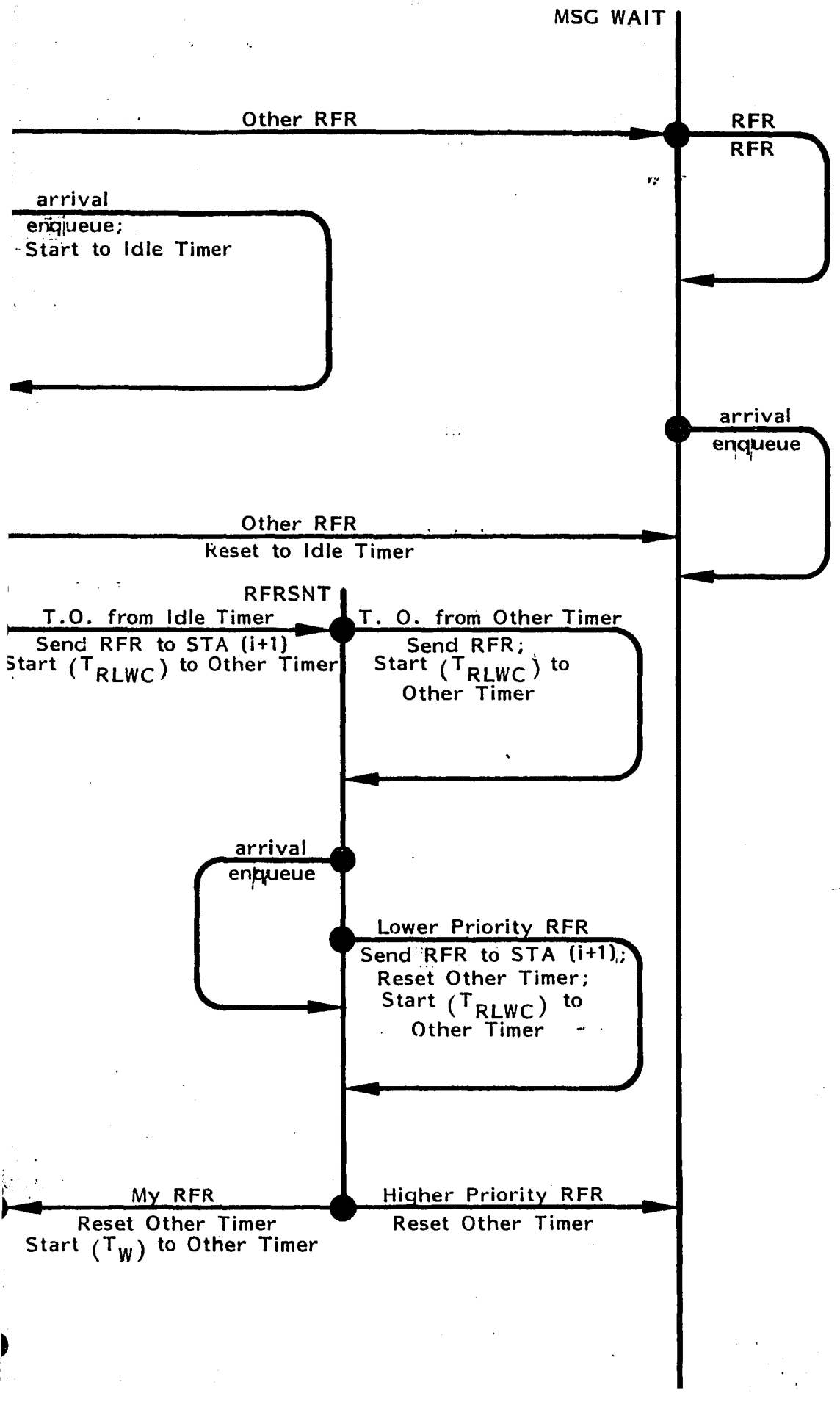
Before saying anything further about the simulation software it is necessary to point out the fact that the other components (Timers and Queues) in the DCPR station have not been described. The reason is that they are not as complex as the MAC mgr. Morever they will be treated as global components in the simulation software.

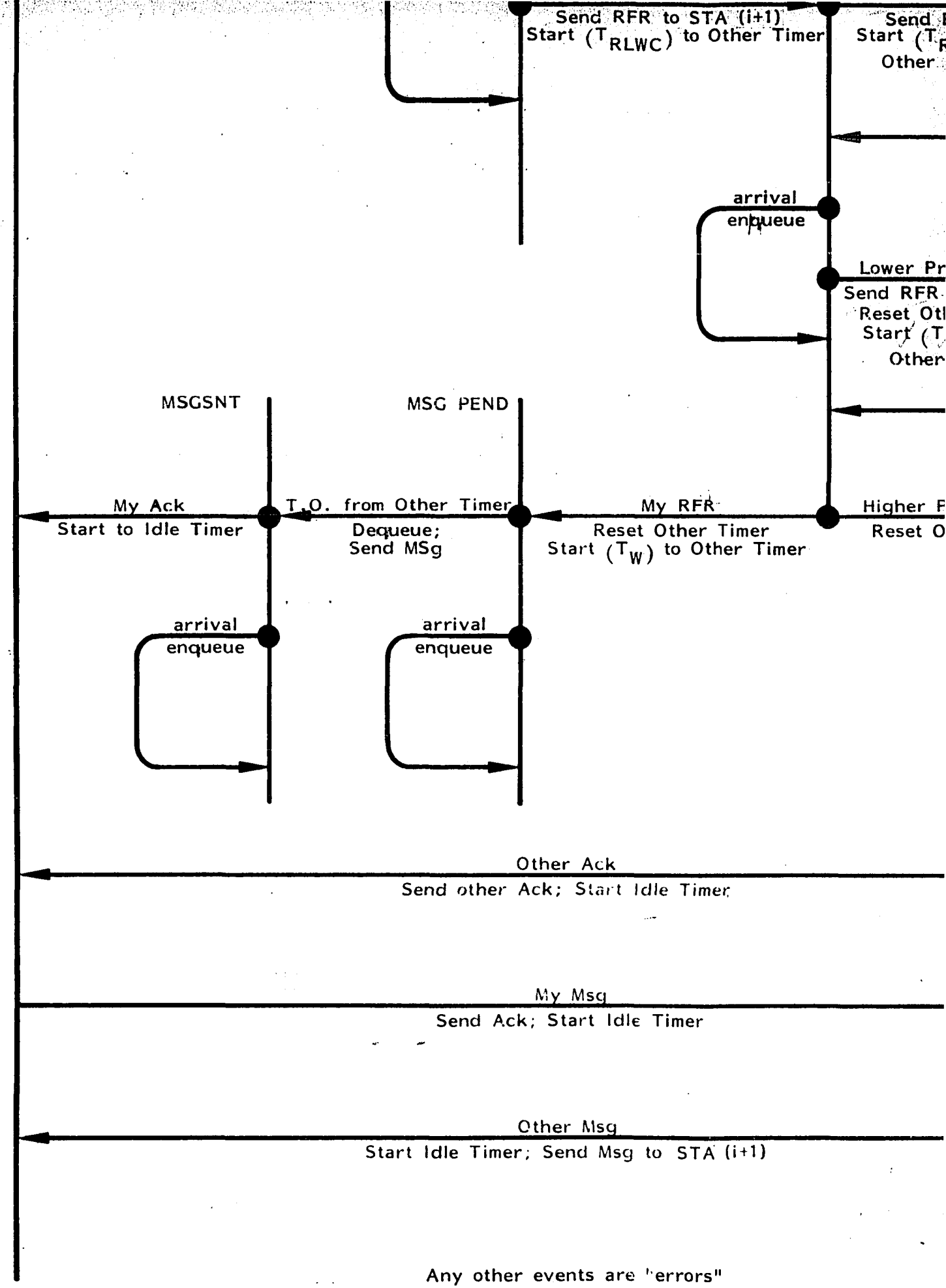
Now a word should be said about states, events and meaning of events before attention is turned to the simulation software design.

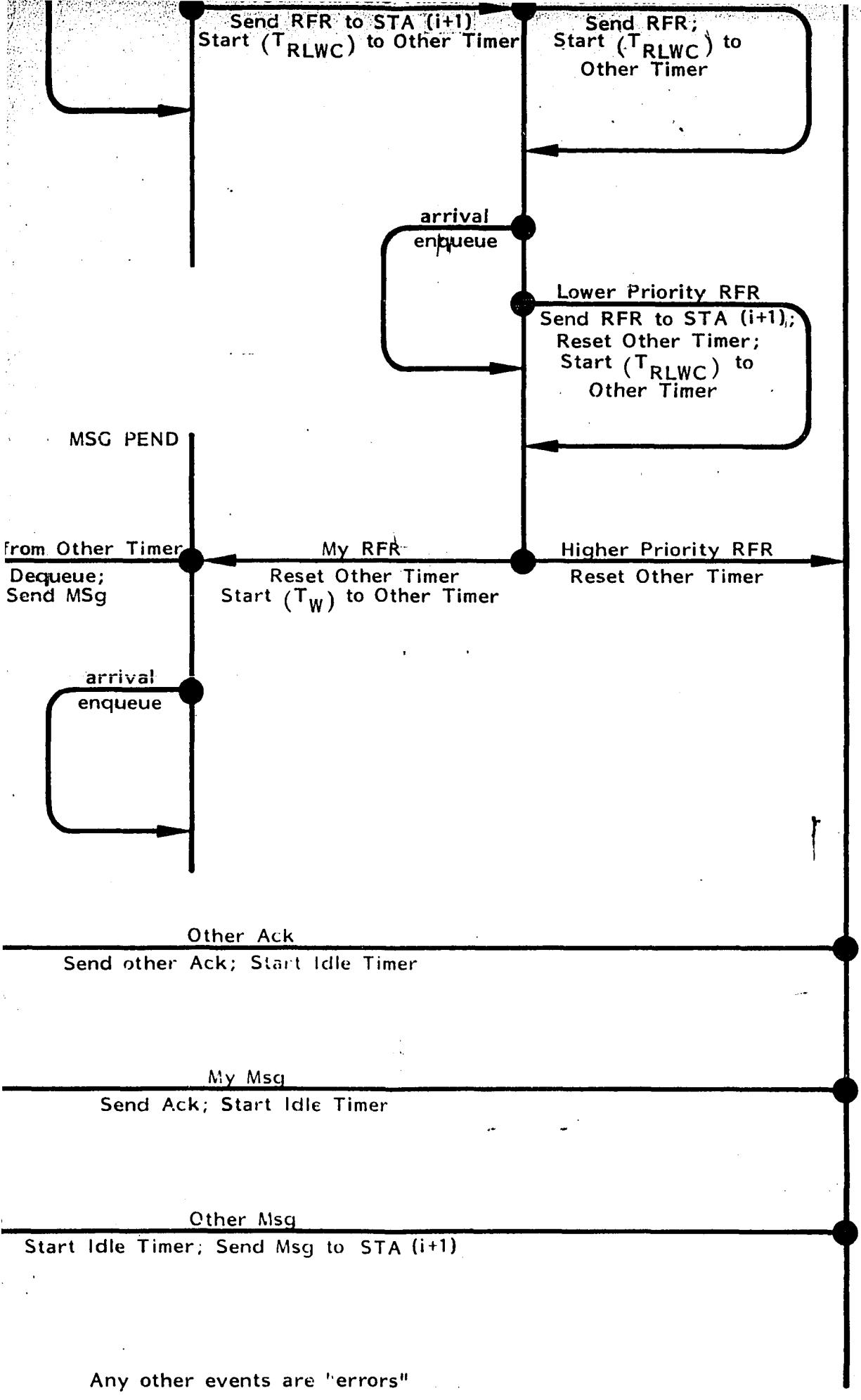
The reason why a finite Station Machine is used to model the DCPR MAC manager is because a state machine is a convenient way of modeling any system which needs to remember certain events. For

Figure 5-8. State specification for the MAC mgr of Figure 5-7


IDLE WAIT
RFR WAIT
arrival
enqueue
T.O. from Idle Timer & Qlen = 0
Other RF
arrival
enqueue;
Start to Idle Timer
RFR PEND
T.O. from Idle-Timer & Qlen > 0
Start to Idle Timer
Other RF
Reset to Idl
arrival
enqueue
RFRSNT
T.O. from Idle Timer
Send RFR to STA (i+1)
Start (T_{RLWC}) to Other Timer
arrival
enqueue
MSGSNT
MSG PEND
My Ack
Start to Idle Timer
T.O. from Other Timer
Dequeue;
Send MSg
My RFR
Reset Other Timer
Start (T_W) to Other Timer


MSG WAIT
Other RFR
RFR
RFR
arrival
enqueue;
Start to Idle Timer
arrival
enqueue
Other RFR
Reset to Idle Timer
RFRSNT
T.O. from Idle Timer
Send RFR to STA (i+1)
Start (T_{RLWC}) to Other Timer
T. O. from Other Timer
Send RFR;
Start (T_{RLWC}) to
Other Timer
arrival
enqueue
Lower Priority RFR
Send RFR to STA (i+1);
Reset Other Timer;
Start (T_{RLWC}) to
Other Timer
My RFR
Reset Other Timer
Start (T_W) to Other Timer
Higher Priority RFR
Reset Other Timer


Send RFR to STA (i+1)
Start (T_{RLWC}) to Other Timer
arrival
enqueue
Lower Pr
Send RFR
Reset Ot
Start (T
Other
MSGSNT
MSG PEND
My Ack
Start to Idle Timer
T.O. from Other Timer
Dequeue;
Send MSg
My RFR
Reset Other Timer
Start (T_W) to Other Timer
Higher P
Reset O
arrival
enqueue
arrival
enqueue
Other Ack
Send other Ack; Start Idle Timer
My Msg
Send Ack; Start Idle Timer
Other Msg
Start Idle Timer; Send Msg to STA (i+1)
Any other events are "errors"


Send RFR to STA (i+1)
Start (T_{RLWC}) to Other Timer
Send RFR;
Start (T_{RLWC}) to
Other Timer
arrival
enqueue
Lower Priority RFR
Send RFR to STA (i+1);
Reset Other Timer;
Start (T_{RLWC}) to
Other Timer
MSG PEND
from Other Timer
Dequeue;
Send MSg
My RFR
Reset Other Timer
Start (T_W) to Other Timer
Higher Priority RFR
Reset Other Timer
arrival
enqueue
Other Ack
Send other Ack; Start Idle Timer
My Msg
Send Ack; Start Idle Timer
Other Msg
Start Idle Timer; Send Msg to STA (i+1)
Any other events are "errors"

example, when a DCPR station sends an RFR it should keep in memory so that when it receives it back it can proceed with transmission of a message.

Figure 5-9 shows the names of states and events in any DCPR station. The figure shows that a DCPR MAC manager can be in any one of the set of seven states {Idle Wait, RFR Wait, RFR Pending, RFR Sent, Msg Wait, Msg Pending, Msg Snt}. Notice that these states represent the vertical lines in Figure 5-8.

It is also seen in Figure 5-9 that there are two types of events: primary and secondary.

Primary Events (PEs) are events external to the DCPR station manager whereas Secondary Events (SEs) are internal events which are triggered by a combination of Primary Events and associated parameters.

As will be seen later events that are handled in the event list for the purposes of simulation belong to the set of five PEs {Arrival, RFR Rev, Msg Rev, Ack Rcv, Timeout}. Also, each Primary Event has associated with it parameters that are passed to the MAC mgr. For example, the PE Arrival has associated with it the parameter, "i," which is the identity (and priority) of the MAC mgr at which the arrival occurs and the parameter, "Time of Arrival," which is the time at which the arrival event occurs at MAC mgr "i."

Each PE has a meaning which can be considered as an interpretation of the event. For example the PE, "RFR Rcv," is to be interpreted as "End of RFR."

Station States:
Idle Wait
RFR Wait
RFR Pending
RFR Sent
Msg Wait
Msg Pending
Msg Snt

Events: These can be grouped in Primary Events which are events external to the station and Secondary Events which are internal events triggered by a combination of Primary Events and associated parameters.

	Event	Parameters	Meaning
Primary Events:	Arrival	i, Time of Arrival	End of Msg
	RFR Rcv	i, Priority of Sending Station	End of RFR
	Msg Rcv	i, Origin Sta, Destination Sta	End of Msg
	Ack Rev	i, Origin Sta, Destination Sta	End of Ack
	Timeout (T.O.)	i, Idle_Timer/ Other_Timer	

	Event	Meaning
Secondary Events:	Lower Priority RFR	RFR Rcv & Priority < i
	Higher Priority RFR	RFR Rcv & Priority > i
	My RFR	RFR Rcv & Priority = i
	My Msg	Msg Rcv & Destination Sta = i
	My Ack	Ack Rcv & Destination Sta = i

Figure 5-9. DCPR station states and events

This means that when an RFR Rcv is received by any MAC mgr the latter assumes that the whole RFR packet has been received. In other words, the receipt of an RFRRcv event does not imply the start of receipt of an RFR packet, say.

Several SEs can be defined. However, only five are defined in Figure 5-9. It can be clearly seen from the figure that an SE is a function of a PE. Notice in Figure 5-8 that events which cause changes of state to occur are either of type PE or SE although it is clear from the figure that most of them are of type SE.

Finally, it should be pointed out that PEs and SEs are notations used in this dissertation to show events emanating from the simulation event list and those which are seen to cause state changes as seen in Figure 5-8. It is also noted that although an SE might explicitly be the event causing state changes and/or outputs, implicitly it is a PE acting on a predicate which is causing those changes.

Software Design

It is not intended to provide detailed software design and code here. The simulation code is in Appendix A. Rather, certain global concepts relating to the simulation software are discussed. Also discussed are the data structures used and the motivations for them and how Figures 3-5, 5-7 to 5-9 can be used to verify the code. Data structures are discussed first. As will soon be seen, the selection of the data structures are crucial to the design.

A valuable feature of every discrete-event simulator is the event list at which simultaneous activities occurring throughout the

simulated network are merged into a serial stream of events. The event list structure is shown in Figure 5-10. The figure shows that a doubly linked circular list is used to implement the event list. The reason for using a doubly linked list is that there would be the need to reset timers in a DCPR event which might be a future event. This requires a removal of an event from the list. A doubly linked list is more efficient than a singly linked list for this purpose. A circular list with a HEAD node is used because it provides an ease of inserting and deleting events anywhere on the list. One final reason for using linked lists instead of an array is because there is no way of knowing the maximum number of events on the list before beginning the simulation. There are two other data structures which are of importance. They are the system state record called STATEREC and the system statistics record called STATSREC.

SIMSTATE (for SIMulation STATE) is a global variable of type STATEREC which can be used to obtain almost any information one would like to know about the system during simulation. It holds such information as a pointer to the head of the event list (CALENDAR) current simulation time (CRNTTIME), an array of pointers to the queues of all the stations (PQ for Priority Queue), the present state of each station (STASTATE for STAtion STATE), an array of priority indices of the various stations (PR for Priority), an array of states of the various timers at each station (TIMERCOMP for TIMER COMPonents), etc.

Similarly STATS (for STATistics) is a global variable of type STATSREC which can be used to obtain information about system

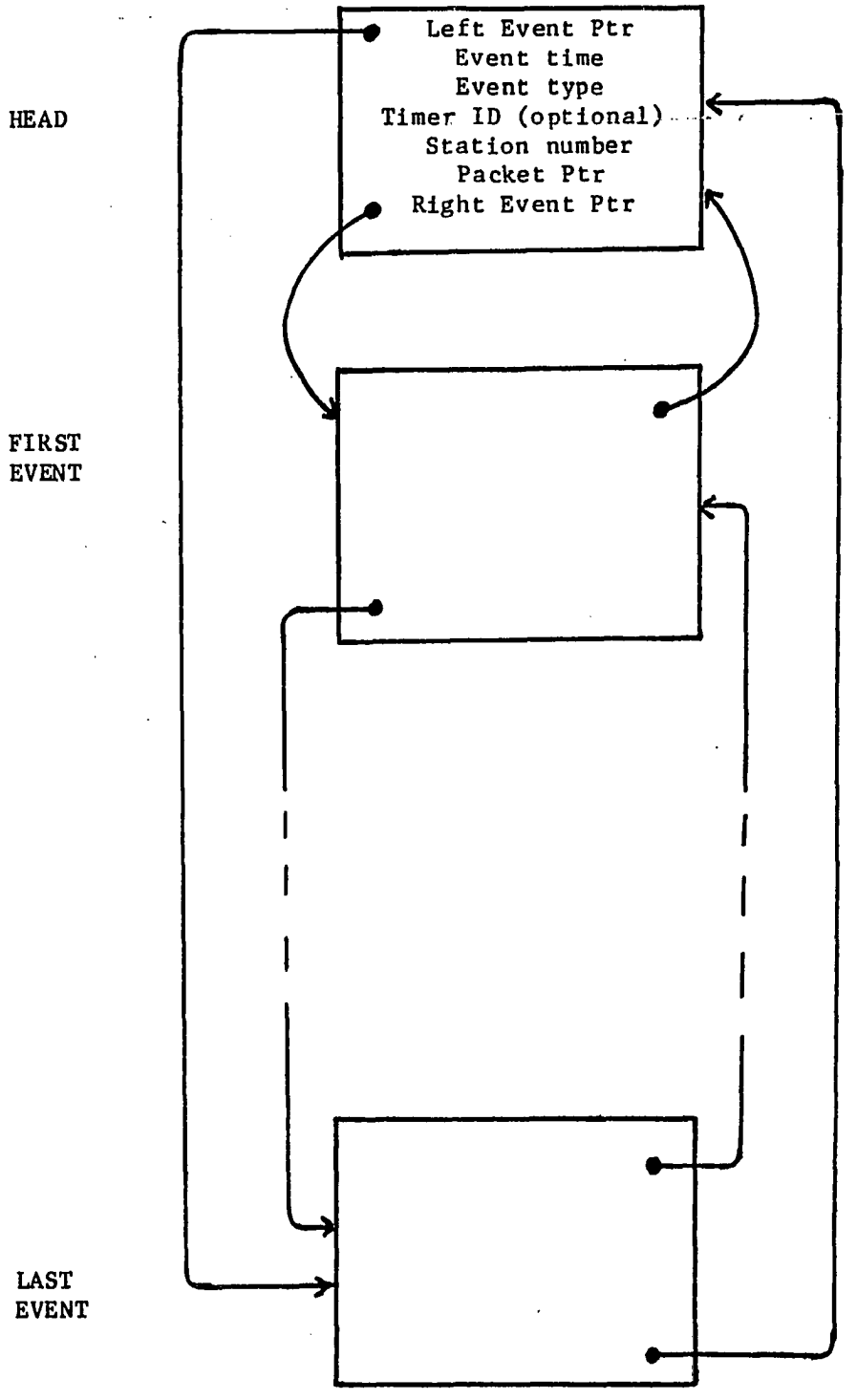


Figure 5-10. Event list structure

statistics. It is the key to such information as present queue length at each station (QLEN for Queue LENgth), number of messages transmitted by each station (NOFMSGSNT), etc. Notice that the reason for keeping track of Queue length is to allow a designer to determine the buffer requirements at each station. That is, how much memory should be provided for a given station to hold queued messages.

It will be seen by looking through the code (see the Appendix) that SIMSTATE and STATS are used by all the Event Processing Routines. It is obvious that it's because of the information they have keys to. Enough about data structures.

We now turn our attention to components of a DCPR station. Recall that other than the MAC manager there exists a queue and two timers in each DCPR station. There are also communication links. How are they handled in the simulation model?

Each queue is modeled as a singly linked list of messages (actually a packet of type "message"). A linked list is used because the maximum length of any queue is unknown a priori. Two pointers (HEAD and TAIL) point to each queue and an array of pointers of all the queues is maintained in the system state record (STATEREC).

The Timers are modelled by providing their states (IDLE or RUNNING) and pointers to timeout events, if any, which have been scheduled (i.e., placed) on the event list to an array. These arrays are maintained in the System State record (STATEREC).

Thus the queues and timers are maintained in a global model. This is done for convenience especially because these components are

very simple compared to the MAC manager.

The communication links, although can be modelled as pulsed delays are so simple that they are also handled globally in the Event Processing routines. In fact, the propagation delays of these communications links are part of STATEREC. It is the array called PROPTIME in STATEREC.

The software design of the Medium Access Control (MAC) Manager among others is discussed next. Here, flow charts are used to show a modular design approach which makes verification of the code easier using Figures 3-5, 5-7 to 5-9.

Let us begin with the main program for the simulator. The Main Control Program for the simulator is in essence derived from Figure 5-6. However, it is modified to look like the model shown in Figure 5-11. The main difference between Figures 5-11 and 5-6 is that the Event Processing routines in Figure 5-6 have been lumped together to be called MACMGR. It should be pointed out that MACMGR is always provided with parameters such as a pointer to the event that has been selected. It is seen from Figure 5-10 that each event has parameters such as Station ID, event type, packet ptr, etc. Station ID is the identity of the station which will process the event, "event type." Notice that the event will belong to the set of Primary Events (see Figure 5-9).

Figure 5-12 is a Logic Diagram of the MACMGR. It shows that the MACMGR is essentially an Event Processor. What the MACMGR does is to determine the type of event it has received and subsequently calls a

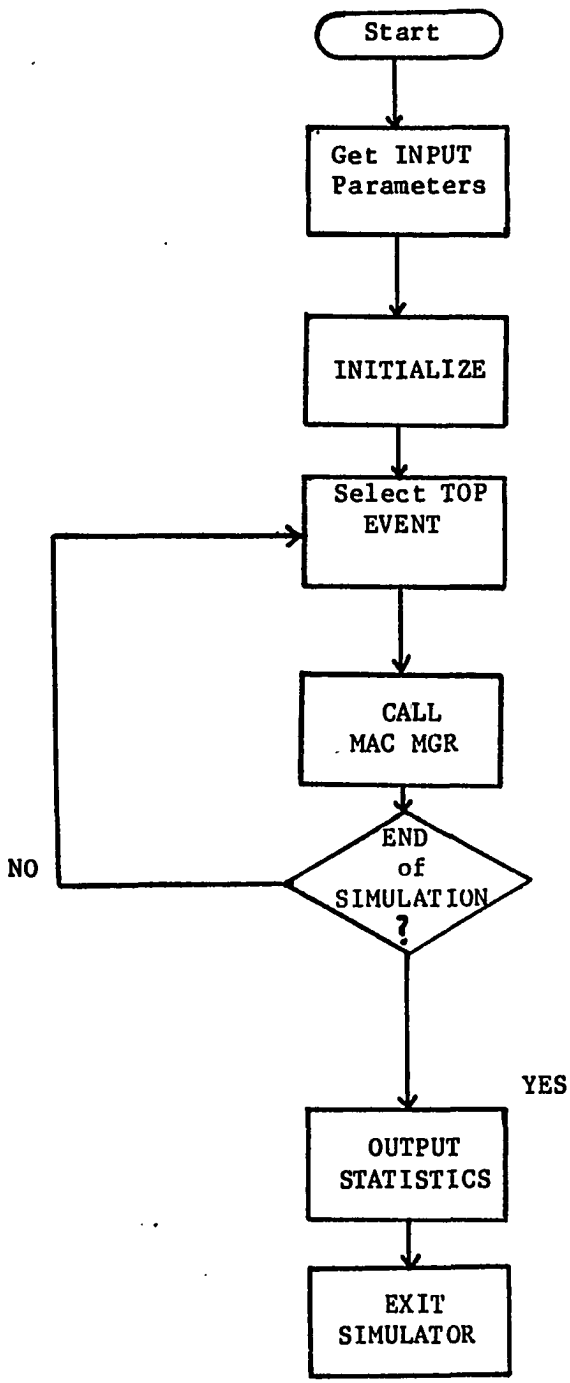


Figure 5-11. Main control program of simulator

Entry variables: P (Pointer to event), Simstate, Stats

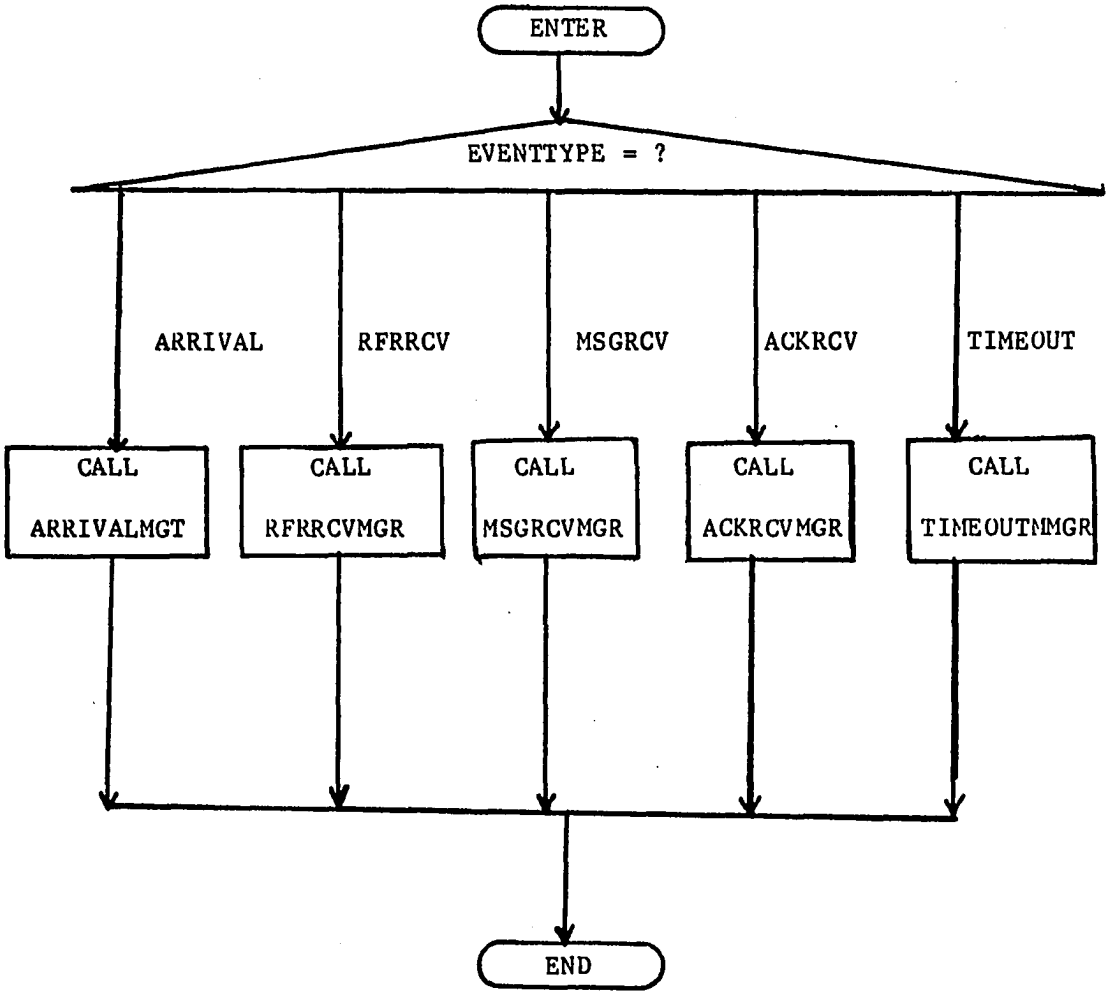


Figure 5-12. MACMGR logic diagram

routine to process that event. For example, if the event is arrival then the Procedure ARRIVALMGR is called.

The ARRIVALMGR logic and in fact all the event processing routines are derived directly from Figure 5-8 (Finite State Specification of the MACmgr). The logic diagram of the simplest of these routines (ARRIVALMGR) is shown in Figure 5-13. It is based on the fact that every message arrival results in the latter being placed in a queue. Furthermore no state changes occur except if the station was in the RFRWAIT state before the message arrival event. Before the state is changed to RERPEND, the IDLETIMER is started. (See Figure 5-7.) Figure 5-14 and 5-15 show the code for MACMGR and ARRIVALMGR, respectively. Compare them with Figures 5-11 and 5-12, respectively and notice how the code contains essentially similar notations as in Figures 5-7, 5-12, and 5-13.

The other event processing routines RFRRCVMGR, MSGRCVMGR, ACKRCVMGR, and TIMEOURTMGR can be similarly derived from Figure 5-8.

Results of Simulation

The simulation model was run using the same set of assumptions used in the analytic model of the previous chapter. For the results in Figures 5-16 to 5-18, the number of stations used was 5. The message length at each station were assumed to exponentially distributed and of average 1000 bits. The arrival rates at the stations were chosen to correspond to total system throughputs of 0.001, 0.01, 0.1, 0.3, 0.5, 0.7, and 0.9. Although the simulation can handle any priority ranking these results are for decreasing priority ordering.

Entry variables: Simstate, stats

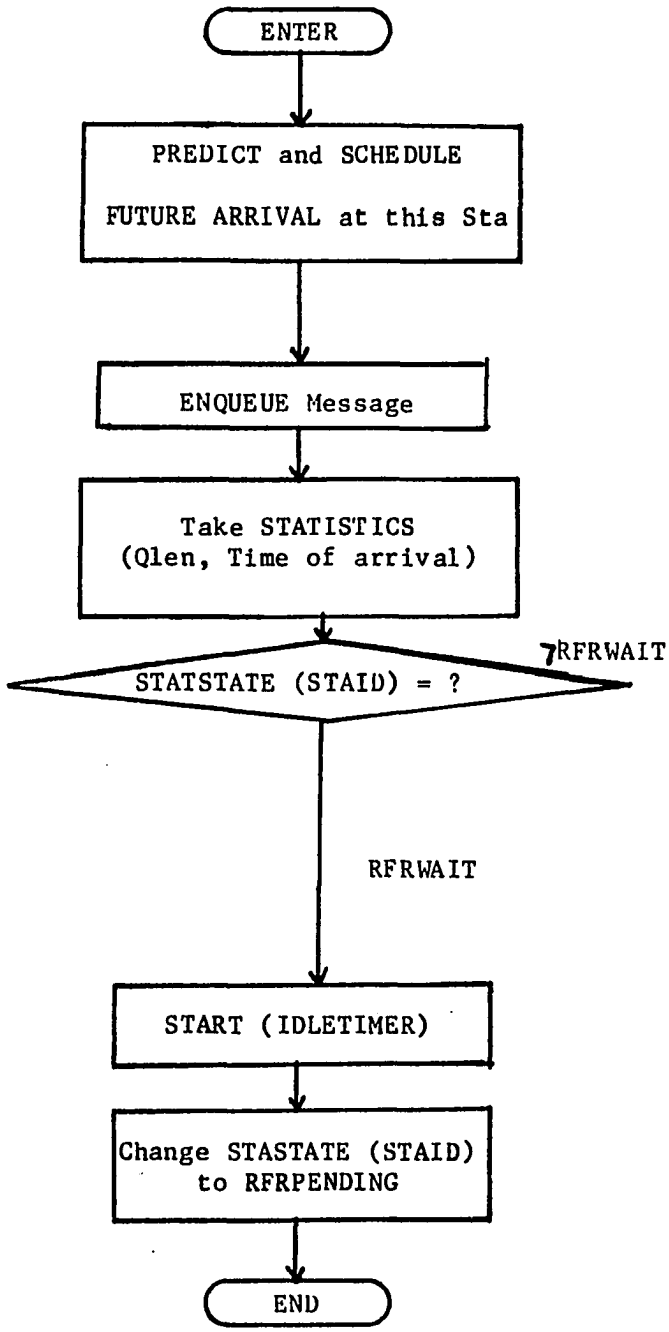


Figure 5-13. ARRIVAL manager logic diagram

```
PROCEDURE DATALINK MGT(VAR P:EVENTPTR; VAR SIMSTATE:STATEREC;
                       VAR STATS:STATSREC);
VAR     PKTP:PACKETPTS;
        WHICHTIMER:TIMERTYPE;
        EVTYPE:EVENTCLASS;
BEGIN
        PKTP:=P@.PKTPTR;                (*GET PERTINENT EVNT PARAMETERS*)
        EVTYPE:=P@.EVENTTYPE;
        CASE EVTYPE OF
            ARRIVAL:ARRIVALMGR(SIMSTATE,STATS);
            RFRRCV :RFRRCVMGR(PKTP,SIMSTATE,STATS);
            MSGRCV :MSGRCVMGR(PKTP,SIMSTATE,STATS);
            ACKRCV :ACKRCVMGR(PKTP,SIMSTATE,STATS);
            TIMEOUT:TIMEOUTMGR(P@.TT,SIMSTATE,STATS)
        END
END;(*PROC DATALINKMGR*)
```

Figure 5-14. PASCAL code for MACMGR of DCPR

```
PROCEDURE ARRIVALMGR(VAR SIMSTATE:STATEREC;VAR STATS:STATSREC);
VAR     NEWP:PACKETPTR;
BEGIN
        FUTUREARRIVAL(SIMSTATE);    (*PRED FUTUR ARIV AT THIS STA*)
        WITH SIMSTATE DO BEGIN
            NEWP:=CREATEMSG(STAID,CRNTTIME,NOMOFSTA);
        ENQUEUE(NEWP,PQ(.SSTAID.));(*PUTMSGINQUEUE*)
        STATS.QLEN(.STAID.):=STATS.QLEN(.STAID.)+1;
        IF STASTATE(.STAID.) = RFRWAIT THEN BEGIN
                START(IDLETIMER,TOPER,SIMSTATE);      (*STRT IDLETIMER*)
                STASTATE(.STAID.):=RFRPEND      (*CHNG STATE TO B.P.*)
        END (*IF*)
        END (*WITH*)
END;(*ARRIVALEVENT*)
```

Figure 5-15. PASCAL code for ARRIVALMGR

Three figures will illustrate some interesting points of the protocol. Figure 5-16 is a plot of system throughput (i.e., normalized effective bit rate) against the average normalized response time of messages originating from the highest priority station. It should be noted that this response time is the same as the best case response time used in the analytical models of the previous chapter. The figure shows that the theoretical values lie below the simulation values. The reason for this is because the minimum reservation time used for the theoretical results is conservative. Actually at low throughputs the reservation time is maximum. As the throughput increases the reservation time changes until it reaches a minimum value. Except for this disparity, the simulation and analytic curves seem to have a similar shape. This means that the use of the analytic models in Chapter 4 does have some merit.

Figure 5-17 is a plot of system throughput against the average reservation time. It can be seen that the average reservation time begins from a maximum of 6 milliseconds and decreases as the throughput increases. Intuitively decreasing priority DCPR should behave in this fashion since higher throughput implies that more stations are ready and hence, lower reservation times on the average.

Figure 5-18 is a plot of the throughput against the average number of ready stations. The figure shows that the average number of ready stations begins from a minimum of 1 and reaches a maximum of 5 as the throughput increases. As seen earlier this is intuitively correct.

Clearly, these figures show the insight gained by using the

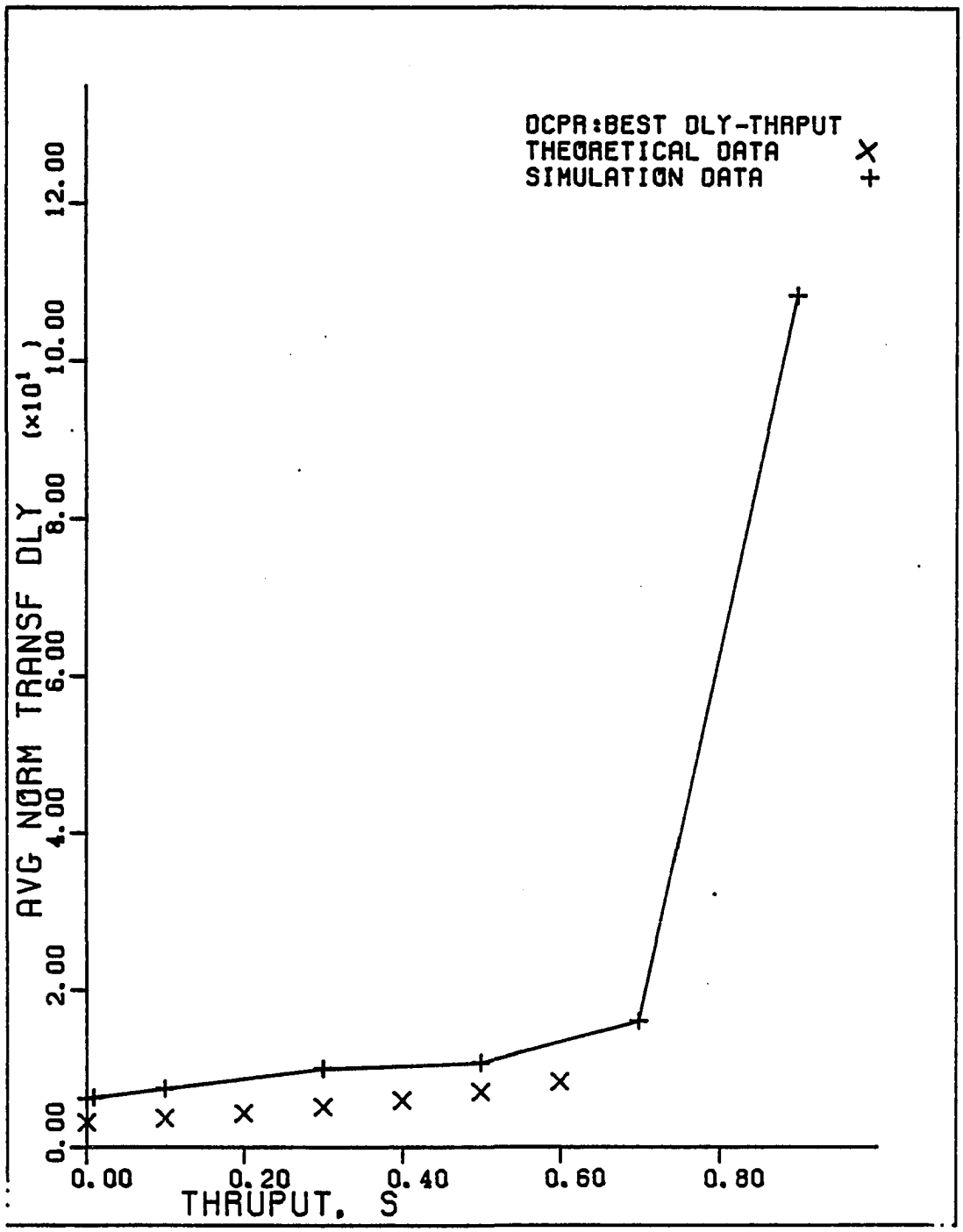


Figure 5-16. Thruput versus average normalized response time for highest priority station, decreasing priority

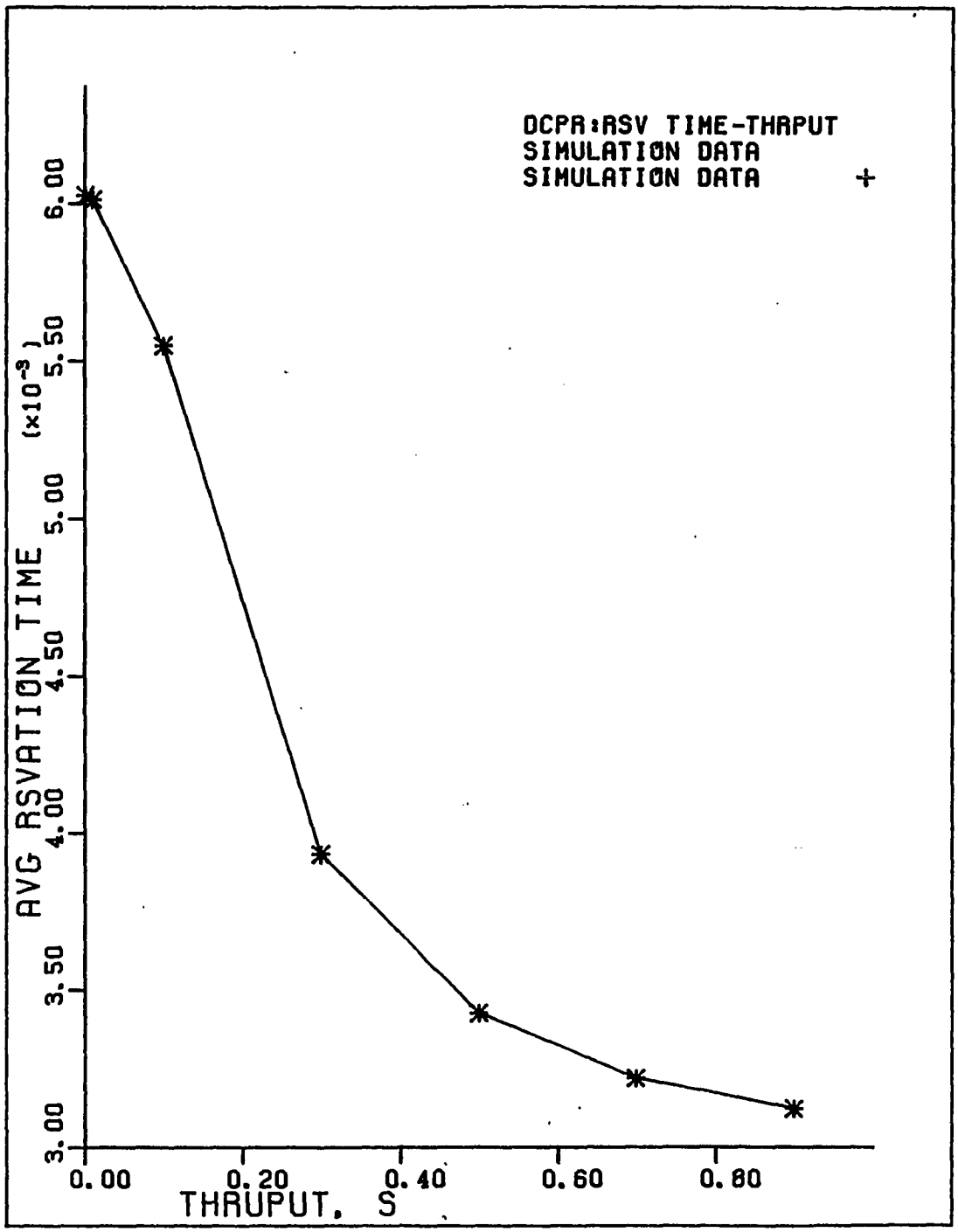


Figure 5-17. Thruput vs average reservation time, decreasing priority ordering

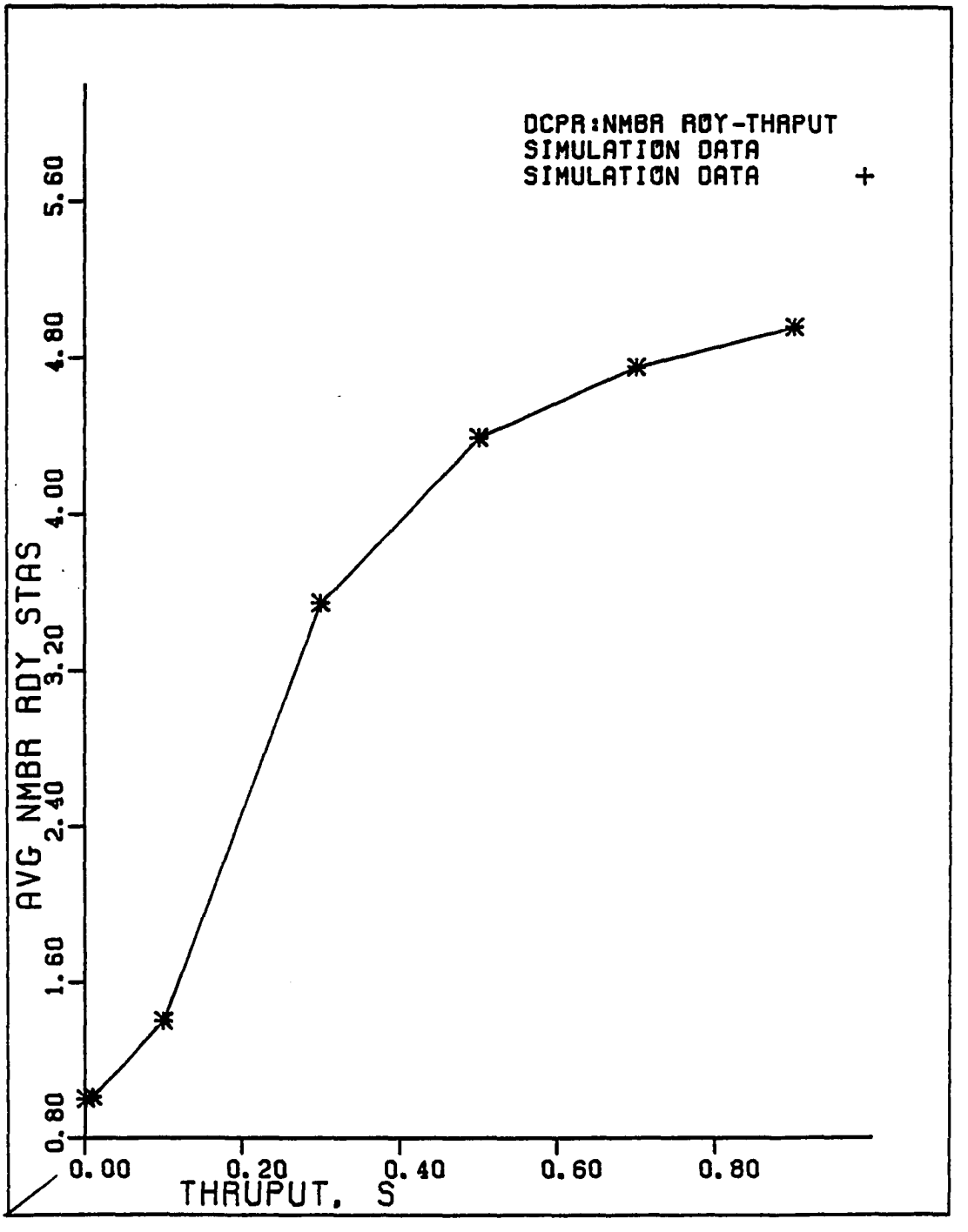


Figure 5-18. Throughput vs average number of ready stations

model to analyze the protocol.

Since throughput is directly proportional to both average message length and arrival rate, increasing any of the two will let the reservation time approach the minimum value. This is likely to increase performance.

CHAPTER 6. THE IMPROVED DCPR PROTOCOL

In the original DCPR protocol the reservation time (i.e., the time it takes for the highest priority ready station to gain access to the channel) can be considered as overhead. As was found in the previous two chapters, this overhead causes the performance of DCPR to deteriorate heavily especially under light load. This was due to the fact that the DCPR protocol by nature allows contention to be resolved by sequential querying.

Moreover, the overhead depends upon the priority ranking of the stations. The decreasing order of priority ranking seemed to provide the best overall performance whereas the increasing order of priority provided the worst overall performance. Imposing a fixed priority ranking is not desirable in practice because it should be possible to place any station anywhere on the ring. To allow the latter and still be able to predict performance effectively, the worst case reservation time should be assumed for all cases.

In this chapter an improved DCPR (IDCPR) protocol is suggested and performance comparison made with the original. This new protocol will be seen to have the following characteristics:

- Contention resolution of ready stations is done in parallel
- Reservation time does not depend on priority ranking
- Reservation time does not depend on load (i.e., arrival rate, etc.)
- Reservation time depends only on the ring latency.

Clearly, all the characteristics are desirable and hence,

advantageous.

The Improved DCPR (IDCPR) Protocol

The original DCPR is modified as follows:

- If a station receives an RFR from a higher priority station it closes its link, moves into MSGWAIT state and sends this higher priority RFR along regardless of whether it has sent its own RFR already or not.
- If a ready station which has already sent its RFR receives a lower priority RFR it does not have to send its RFR again as in the previous DCPR because it will receive it back eventually assuming there are no channel errors or link failures.

A timeout can still be incorporated to deal with the case of channel errors and station failures. Following Chapter 3, the IDCPR algorithm can be written in PASCAL-like notation as:

```
1. REPEAT
        Sense Channel
   UNTIL Channel_is_idle
   Open Link;
2. IF RFR_rcvd THEN Wait-for-Message & Close Link
   ELSE IF Rdy THEN BEGIN
        Send RFR; Start Timer; Wait
        IF higher_priority_RFR_rcvd THEN BEGIN
             Close Link; Transmit higher_priority_RFR;
             Wait_for_Message;
```

```
ELSE IF timeout THEN Go To 2
ELSE IF my_RFR_rcvd THEN BEGIN
     Send_Message
     Wait_for_ACK
     IF ACK_rcvd THEN Go To 1
ENDIF.
```

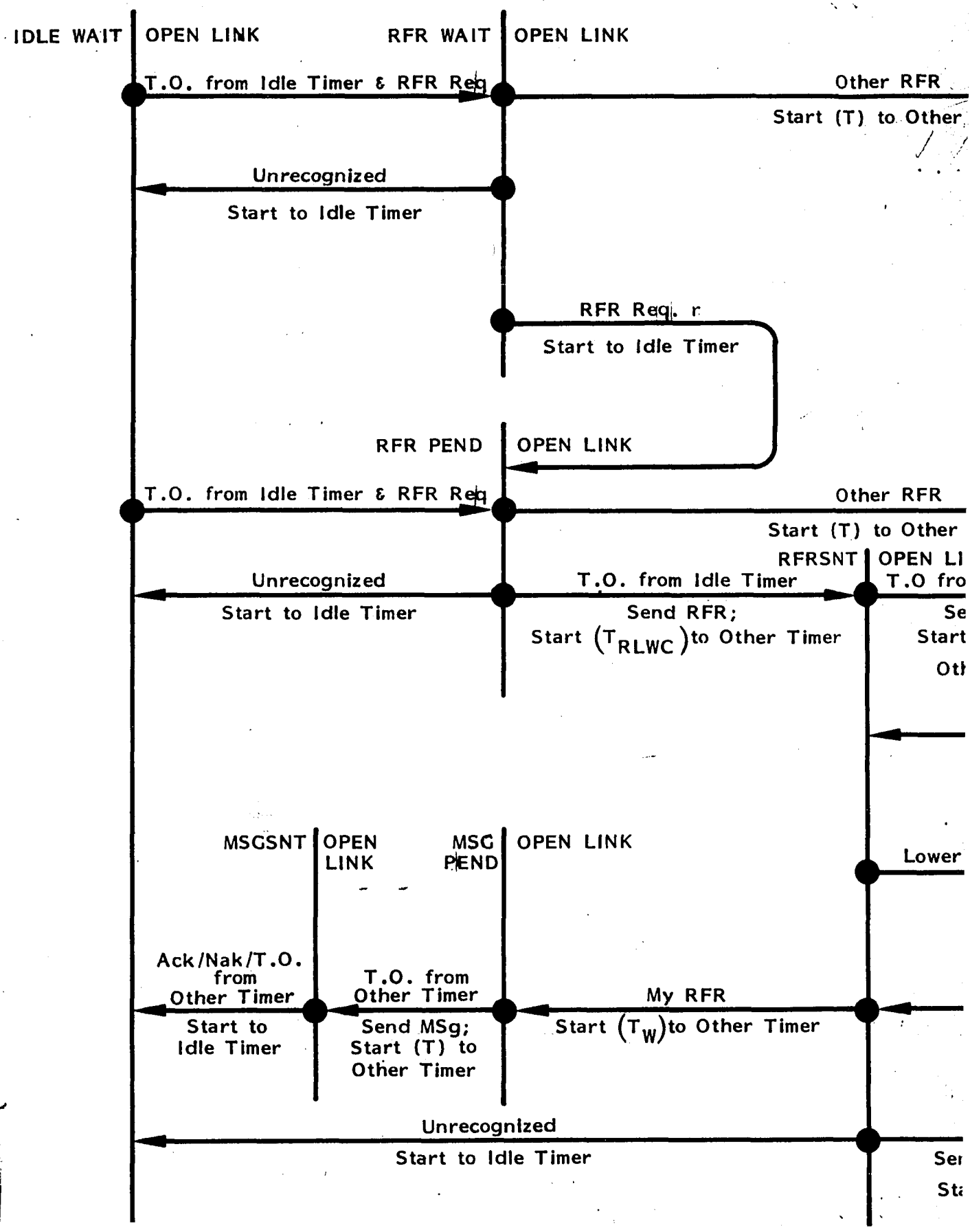
Figure 6.1 shows a SAN description of IDCPR. It may be seen that the only difference between this figure and Figure 3-7 occurs in the RFRSnt State.

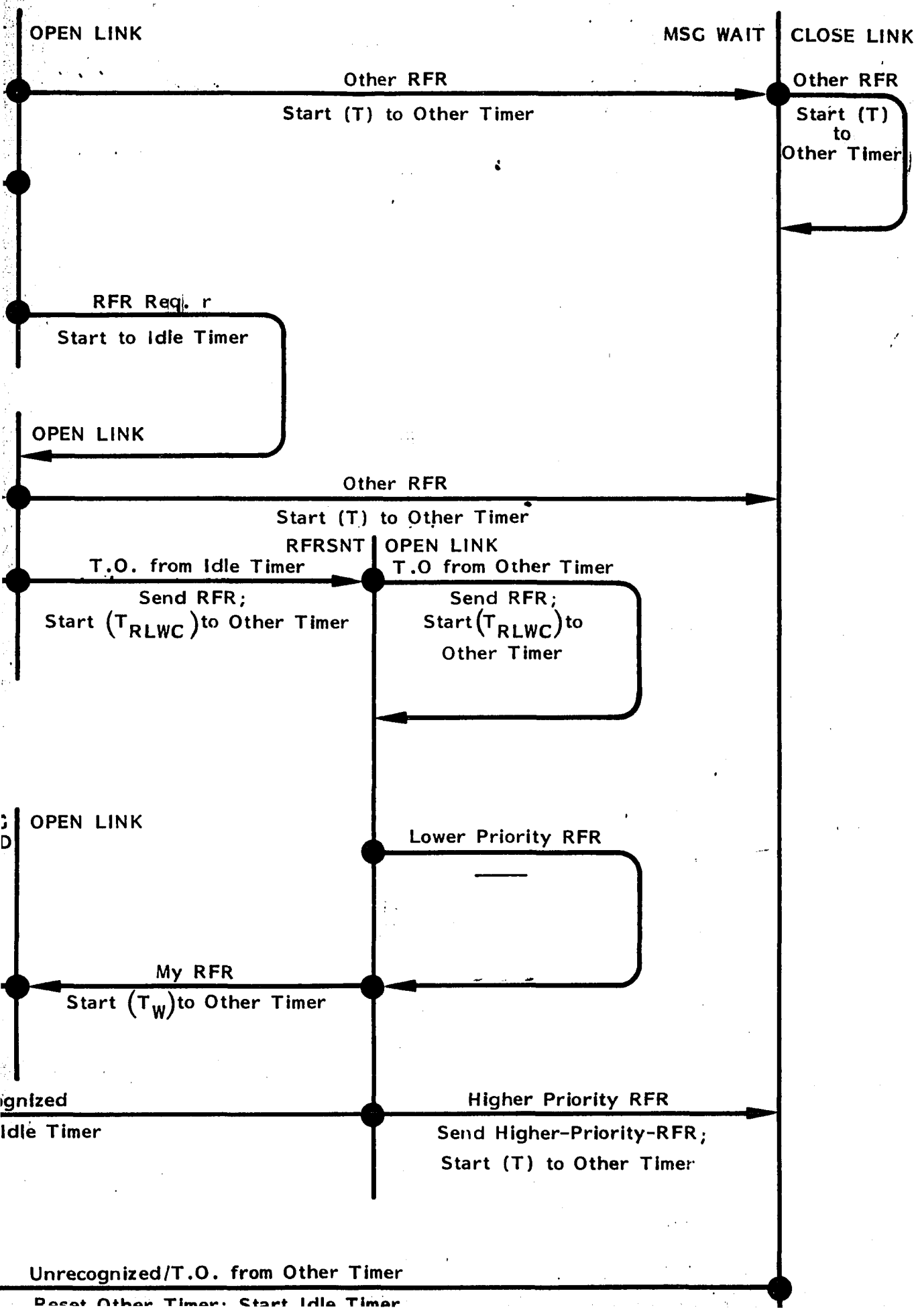
IDCPR Examples

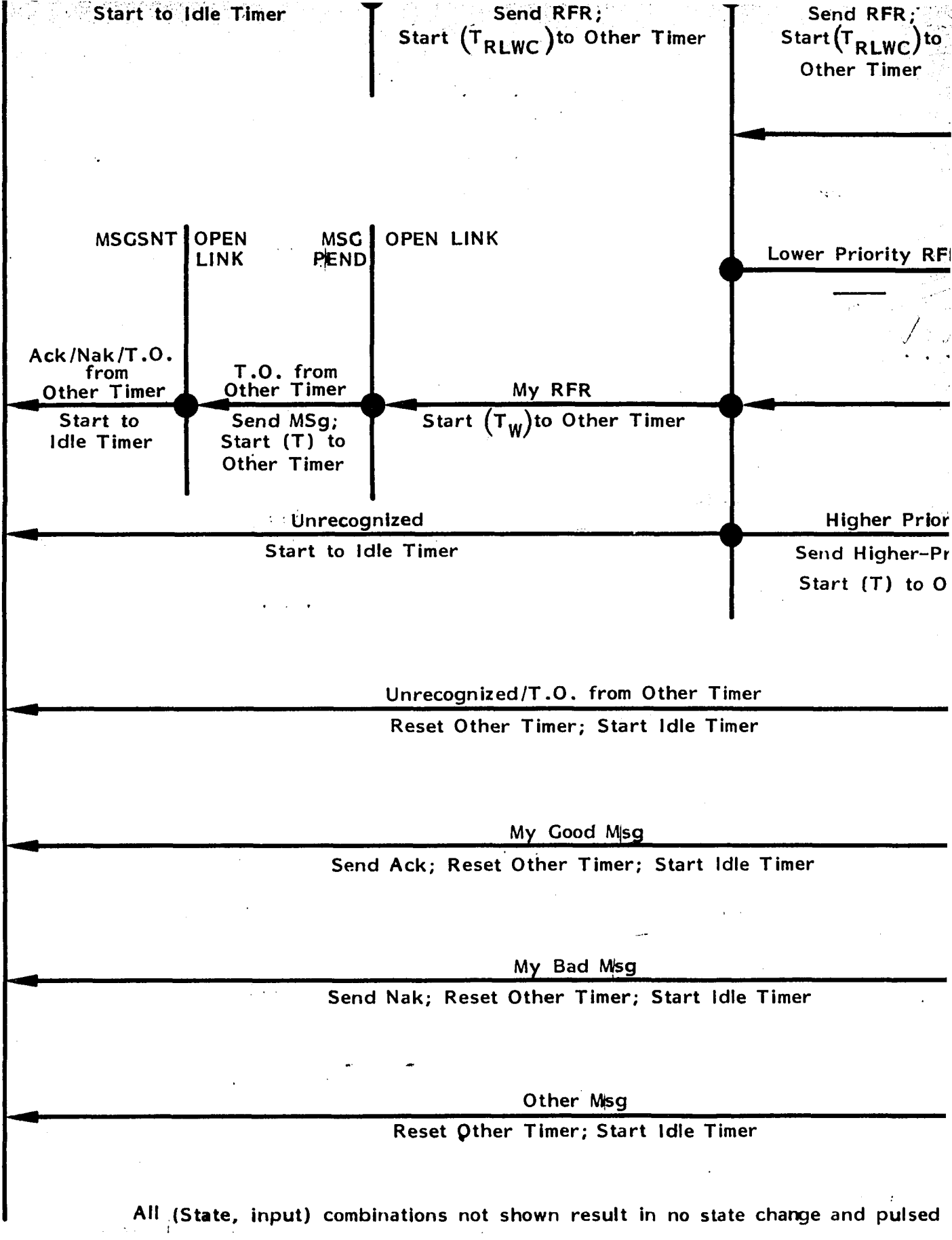
This protocol is now illustrated using the same assumptions and examples that were used for the original DCPR. Hopefully these examples will show why the improved DCPR protocol performs better than the original. As in the latter example we start with the case where the stations have been arranged in decreasing order of priority and only one statin (in the example, T_1) is ready at the start of the reservation process. (See Figures 3-2 and 6-2.)

Initially after detecting channel idle all stations open their links. Station T_1 first sends its RFR to station T_2. Station T_2 receives the RFR, closes its link and transmits it to Station T_3. Stations T_3 and T_4 eventually transmit this RFR until Station T_1 receives its RFR back. At this point, T_1 knows that all the links are closed meaning all the other stations are waiting for a message. This is the end of the reservation/scheduling period. T_1 then transmits a message.

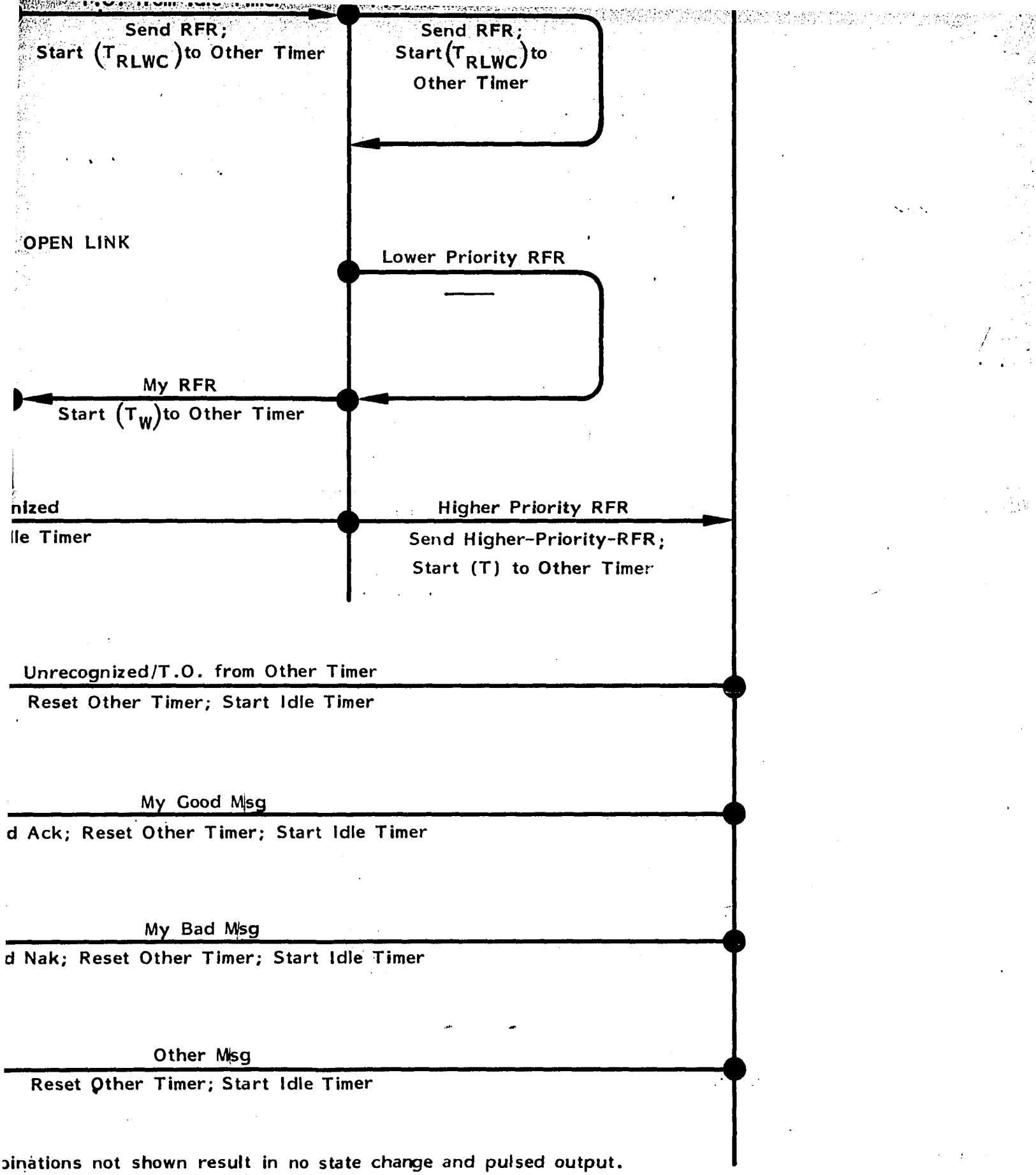
Figure 6-1. Improved DCPR SAN model


IDLE WAIT
OPEN LINK
RFR WAIT
OPEN LINK
T.O. from Idle Timer & RFR Req
Other RFR
Start (T) to Other
Unrecognized
Start to Idle Timer
RFR Req. r
Start to Idle Timer
RFR PEND
OPEN LINK
T.O. from Idle Timer & RFR Req
Other RFR
Start (T) to Other
RFRSNT
OPEN LI
T.O fro
Unrecognized
Start to Idle Timer
T.O. from Idle Timer
Send RFR;
Start (T_RLWC) to Other Timer
Se
Start
Oth
MSGSNT
OPEN LINK
MSG PEND
OPEN LINK
Lower
Ack/Nak/T.O. from Other Timer
Start to Idle Timer
T.O. from Other Timer
Send MSg; Start (T) to Other Timer
My RFR
Start (T_W) to Other Timer
Unrecognized
Start to Idle Timer
Ser
Sta


OPEN LINK
MSG WAIT
CLOSE LINK
Other RFR
Start (T) to Other Timer
Other RFR
Start (T) to Other Timer
RFR Req. r
Start to Idle Timer
OPEN LINK
Other RFR
Start (T) to Other Timer
RFRSNT
T.O. from Idle Timer
Send RFR;
Start (T_RLWC) to Other Timer
OPEN LINK
T.O from Other Timer
Send RFR;
Start (T_RLWC) to Other Timer
OPEN LINK
Lower Priority RFR
My RFR
Start (T_W) to Other Timer
gnized
Idle Timer
Higher Priority RFR
Send Higher-Priority-RFR;
Start (T) to Other Timer
Unrecognized/T.O. from Other Timer



All (State, input) combinations not shown result in no state change and pulsed



binations not shown result in no state change and pulsed output.

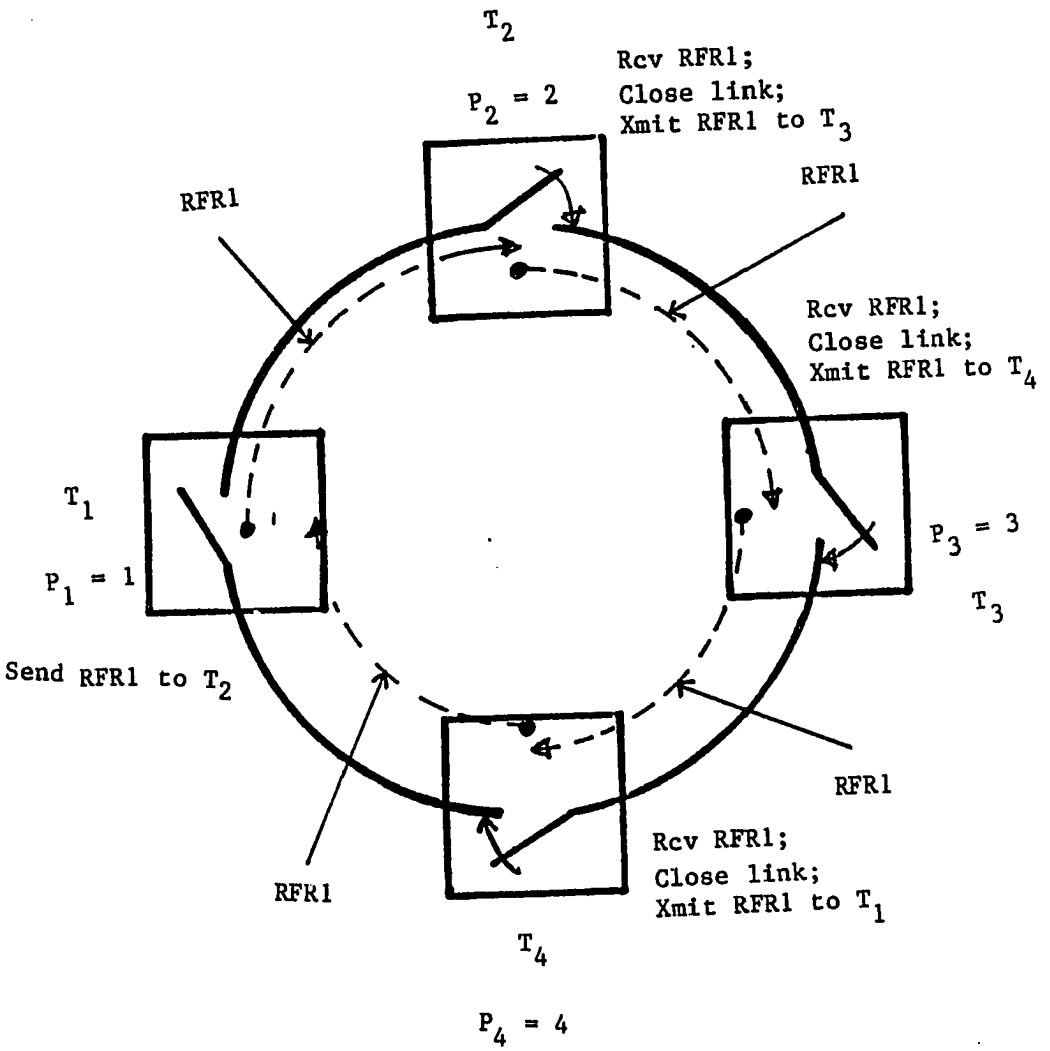


Figure 6-2. Improved DCPR protocol: decreasing order of priority. one station (T_1) ready

Now consider a second case where all stations are ready (see Figures 3-3 and 6-3). After idle_detect each station sends its RFR to the next one. Since all the receiving stations with the exception of T_1 are lower priority stations they all close their links. According to the protocol T_2 then sends T_1's RFR to T_3, T_3 sends T_2's RFR to T_4 and T_4 sends T_3's to T_1. Then T_3 sends T_1's RFR to T_4 and T_4 sends T_2's to T_1. Finally T_4 sends T_1's RFR back to it. (See Figure 6-3.) Notice that T_1 does nothing whenever it receives any RFR because those it receives are of lower priority.

As a third example, consider the case where only stations T_1 and T_3 are ready. Station T_1 sends RFR1 to T_2 and T_3 sends RFR3 to T_1. Finally, T_4 sends RFR1 back to T_1. (See Figure 6-4.)

As a final example consider the case where only stations T1 and T2 are ready (see Figure 6-5).

Reservation Times

Scheduling times associated with the improved DCPR can now be determined having discussed the logic using examples.

The same notation that was used to discuss the reservation time for the original protocol in Chapter 4 will be used here. The timing charts of Figures 6-6 and 6-7 are further used to make the following discussion easier to comprehend. The information in those figures were derived from Figures 6-2 to 6-5.

Let

$$t_{ADJ} \triangleq t_{RFR} + t_R/N \tag{6.1}$$

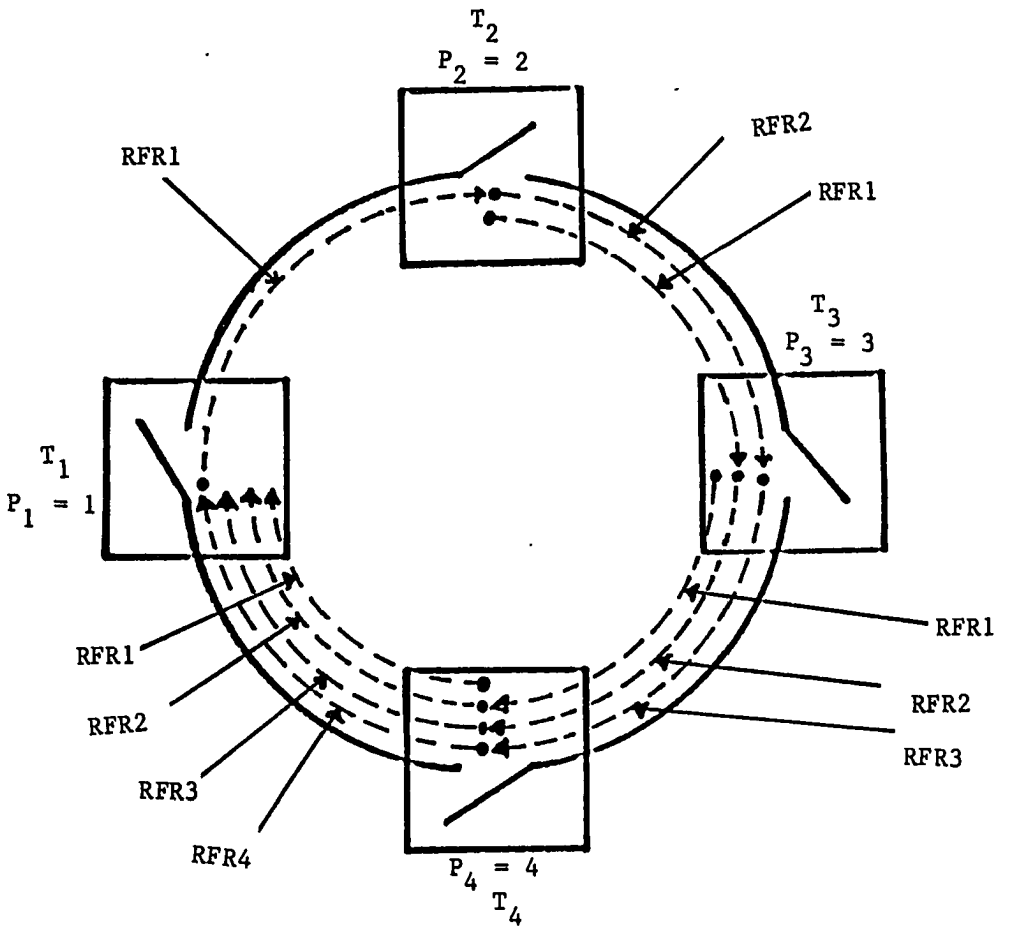


Figure 6-3. Improved DCPR protocol: decreasing order of priority, all stations ready

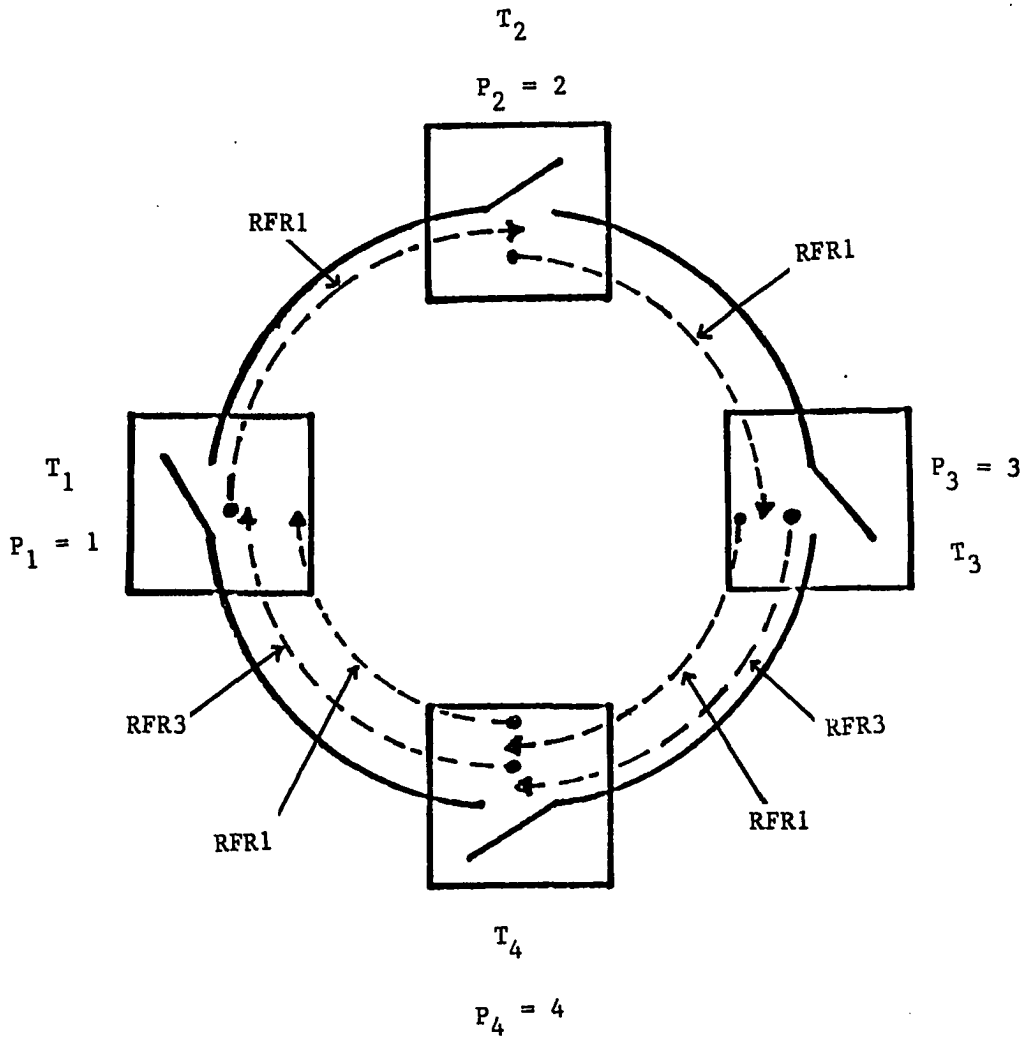


Figure 6-4. Improved DCPR protocol: decreasing order of priority, stations T_1 and T_3 ready

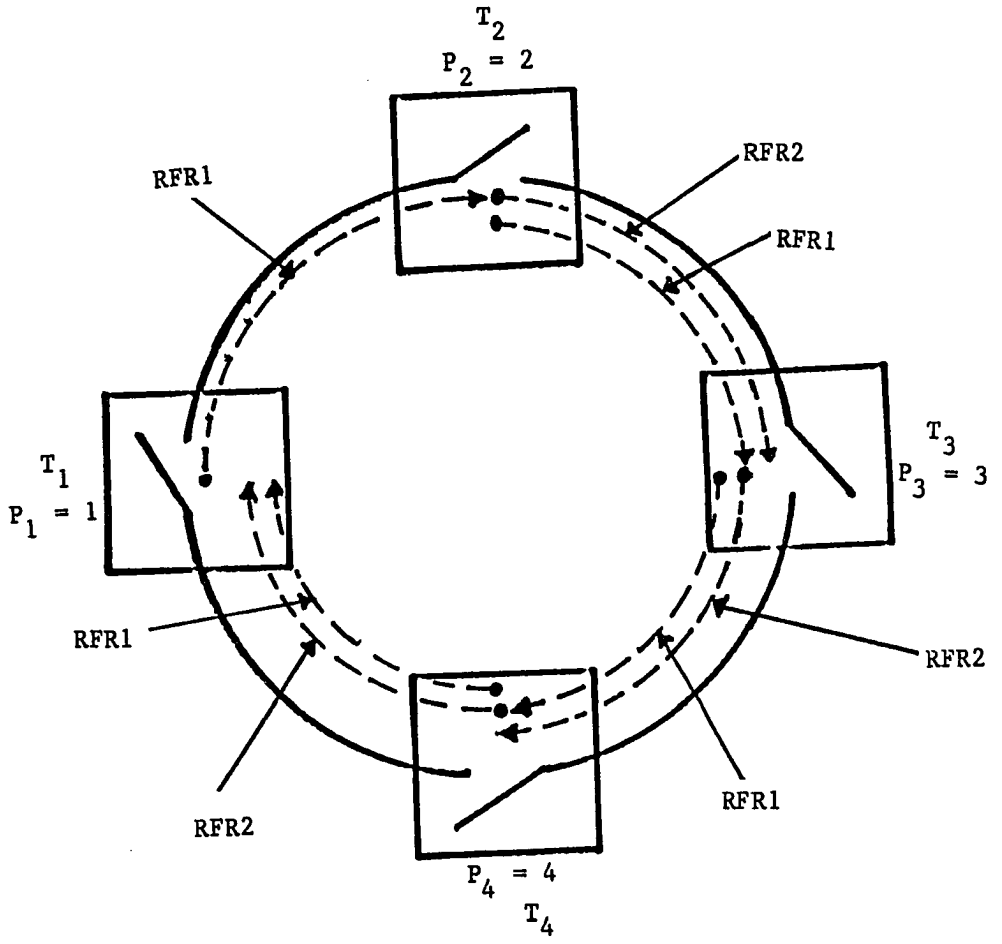


Figure 6-5. Improved DCPR protocol: decreasing order of priority. stations T_1 and T_2 ready

where t_{ADJ} is the time it takes any station to transmit its RFR to the adjacent station (recall that equal propagation delays between adjacent stations are assumed) for a ring consisting of 4 stations.

It is clear from Figures 6-6 and 6-7 that the reservation time for all four cases is given by

$$T_{rsv} = 4t_{ADJ} + 2t_{iw} \tag{6.2}$$

In general, for N stations it is given by

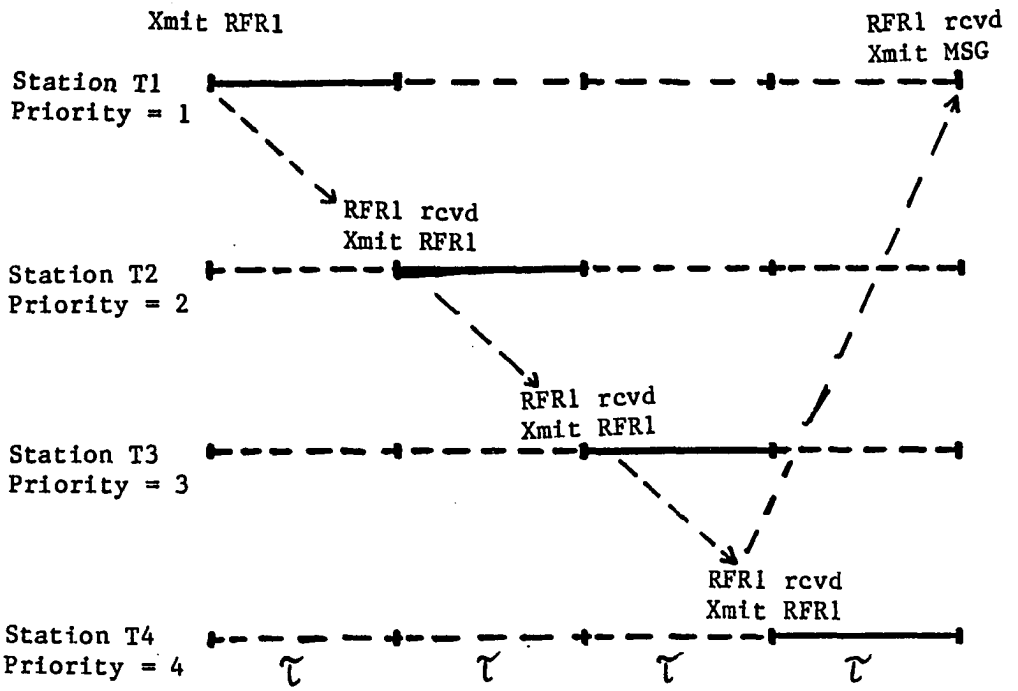
$$\begin{aligned} T_{rsv} &= Nt_{ADJ} + 2t_{iw} \\ &= Nt_{RFR} + t_R + 2t_{iw} \end{aligned} \tag{6.3}$$

Although Eqs. (6.1)-(6.3) are derived for the decreasing priority configuration it can be shown that for any priority ranking the improved DCPR protocol yields the same reservation time as in Eq. (6.3) for N stations. Furthermore, as seen in Eq. (6.3) this reservation time is independent of station arrival rates.

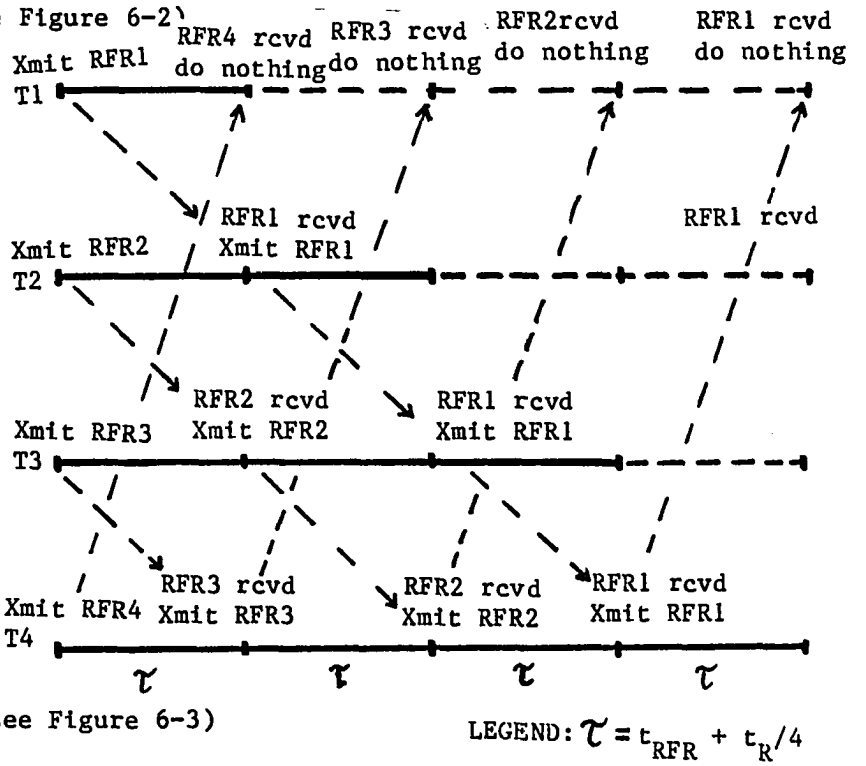
A theorem relating the scheduling time of the improved DCPR will next be stated and proved.

Theorem 6.1

For any priority ranking $(p_1, p_2, \ldots, p_N)$ the time it takes for the highest priority ready station to reserve the channel is $t_R + B_{dRFR}N/C + 2t_{iw}$ where B_{dRFR} is the station latency (in bits), t_R is the round trip propagation delay and $2t_{iw}$ is the idle-detect time (required for synchronization).



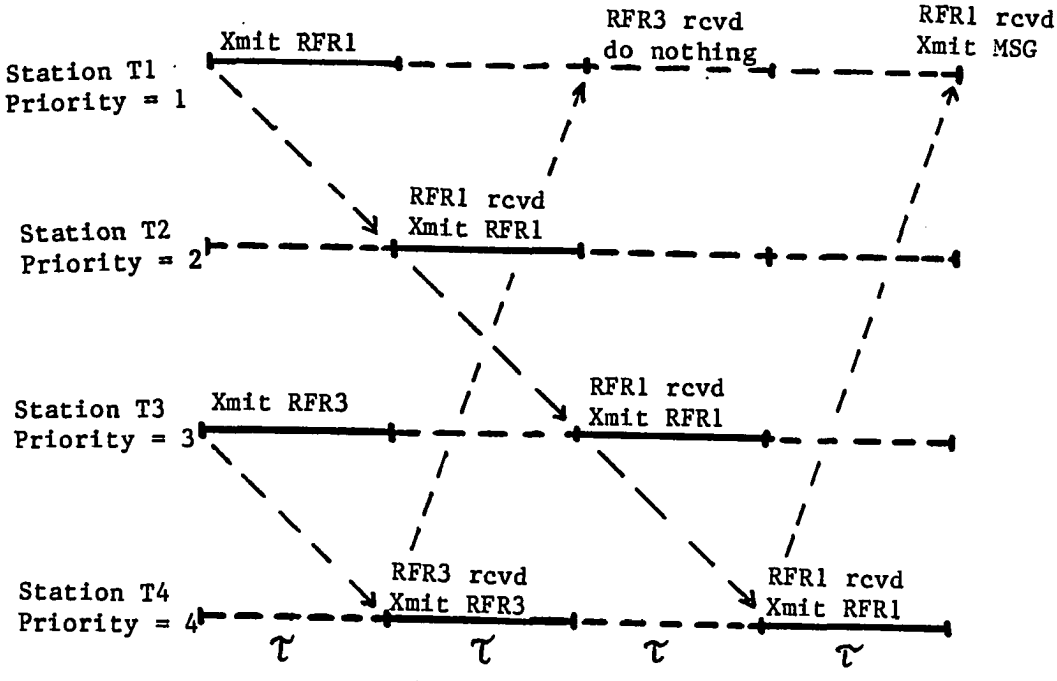
a. Example (see Figure 6-2)



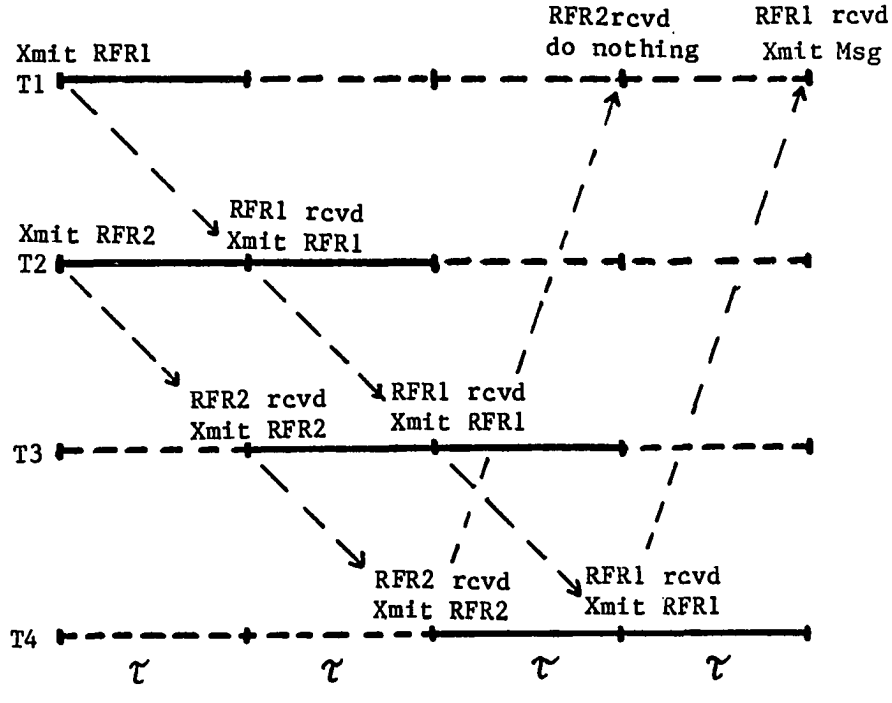
b. Example 2 (see Figure 6-3)

LEGEND: $\tau = t_{RFR} + t_R/4$

Figure 6-6. Improved DCPR timing for Examples 1 and 2



a. Example 3 (see Figure 6-4)



b. Example 4 (see Figure 6-5) LEGEND: $\tau = t_{RFR} + t_R/4$

Figure 6-7. Improved DCPR timing for Examples 3 and 4

Proof

With reference to Figure 6-8, let station T_i be the highest priority (priority p_i) ready station of an N-station ring just before reservation begins.

Initially all links are open.

Station T_i sends RFRi to station T_{i+1}. According to the IDCPR protocol, station T_{i+1} closes its link and passes RFRi along regardless of whether it is ready or not (because it is a lower priority station).

Each station delays an RFR long enough to determine its identity.

As each station sees RFRi it closes its link and passes it along.

On the other hand, any RFR seen by station T_i is removed from the ring (since station T_i has its link open). Thus station T_i receives back RFRi after N transmissions (i.e., of RFRi, each station, transmitting it once). The transmission time for RFRi at each station is B_{dRFR}/C. If the propagation delay between station T_k and $T_{1+k \bmod N}$ is t_k for all k in the set {1, ..., N}, then the reservation time is given by

$$T_{rsv} = \sum_{K=1}^{N} (t_k + B_{dRFR}/C) + 2t_{iw} \qquad (6.4)$$

Therefore,

$$T_{rsv} = t_R + B_{dRFR}N/C + 2t_{iw} \qquad (6.5)$$

This concludes the proof of Theorem 6.1.

The key points conveyed by Eq. (6.5) (i.e., Theorem 6.1) are:

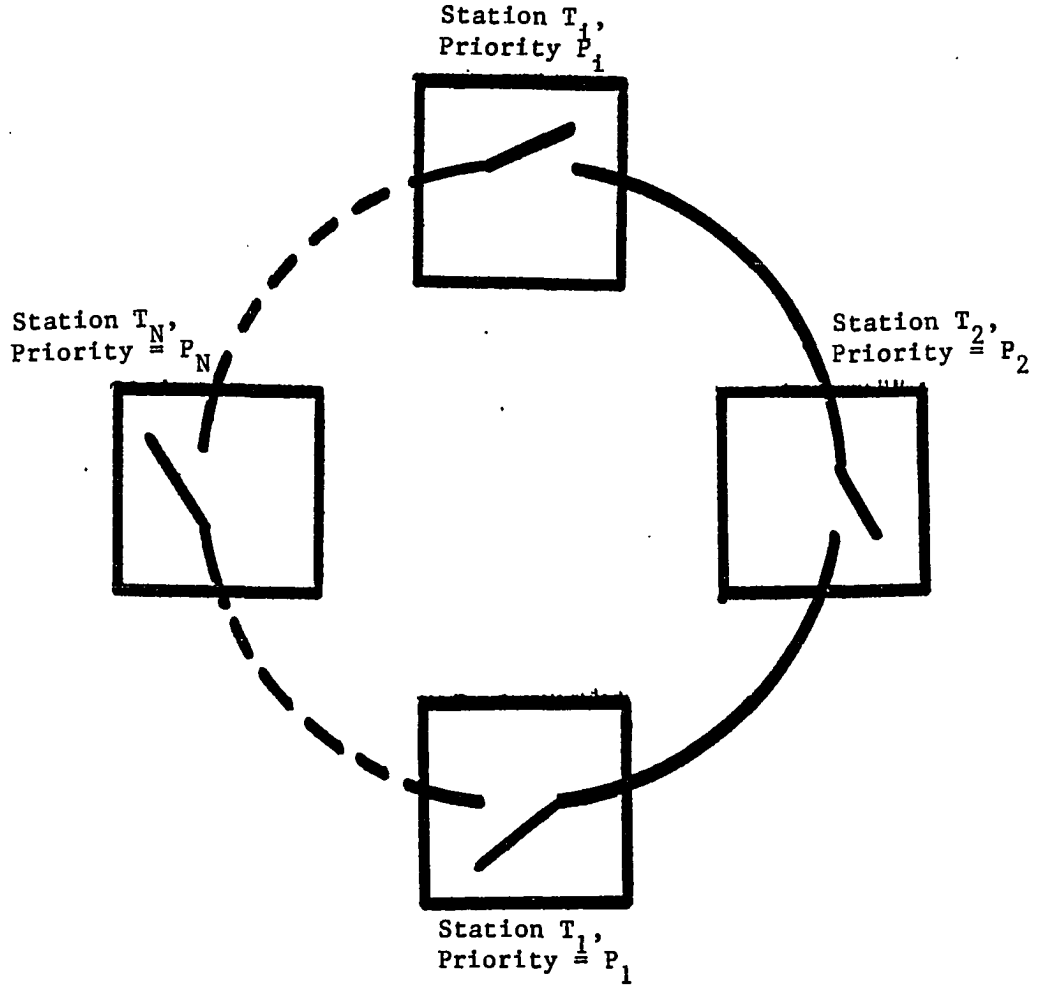


Figure 6-8. An N-Station ring structure

- The reservation time is independent of the priority ranking of the stations.
- The reservation time is independent of the station of the stations (i.e., the readiness of the stations) at any time.
- The reservation time depends only on the total number of active stations on the ring.
- The reservation time is independent of the channel load.

Theorem 6.2. The reservation time of IDCPR is less than the minimum reservation time of DCPR.

Proof:

It was found from the previous chapters that in DCPR, the minimum reservation time occurs when all stations are ready before reservation for the decreasing priority ranking. The theorem can be proved by comparing Eqs. (5.5) and (6.5).

Theorem 6.2 illustrates the most important contribution of the IDCPR protocol, namely insuring that the reservation time is not only constant, but also less than the best reservation time DCPR can offer.

CHAPTER 7. PERFORMANCE COMPARISON WITH KNOWN MAC PROTOCOLS

In this chapter DCPR and IDCPR are compared with other known MAC protocols in terms of performance. Performance measures of concern here are throughput-delay characteristics. Since DCPR is a ring access protocol it will be compared with popular ring access protocols, namely:

1) Token Ring

2) Slotted Ring

3) Register Insertion Ring.

Great care has to be taken in comparing Ring Networks because unlike bus networks, ring access protocol performance depends heavily on the protocol parameters (Stallings [40]). For example, the performance of register insertion rings depend upon the length of the insertion register. Performance may also depend on which station is responsible for packet removal (source station or destination station).

In order to make a favorable comparison, a consistent set of assumptions will have to be made. The assumptions to be used are the same as those used in the previous chapters. That is: 1) Poisson arrivals of same rate at stations; 2) stations equally spaced apart in distance; 3) i.i.d. message lengths of same average and variance.

These are general assumptions. There are particular assumptions which will be made for each of the ring access protocols.

1. Token Ring

As seen in Chapter 2, there are three ways of implementing the token-passing access protocol on ring topology. They are:

a) Multiple-token (multi-token, multi-packet): a new free token is generated after data transmission.

b) Single-token (single-token, multi-packet): a new free token is generated after the transmitting station has erased its own busy token.

c) Single-packet (single-token, single-packet): a new free token is generated after the transmitted data has been completely erased.

It is noted that the single-token scheme is the same as the multiple-token scheme if the packet length is longer than the ring latency

The assumptions to be used are:

- Single-token scheme (This is the scheme recommended in the IEEE 802.5 standard [41]).
- packets are removed by the source station.
- messages have fixed lengths.
- exhaustive service discipline (i.e., a station which possesses the token transmits all messages waiting in its queue). According to Bux [42] and Hammond and O'Reilly [6] this discipline is essentially equivalent to the non-exhaustive discipline (where a fixed length of bits are sent by all stations) when the channel load is less that 90% for operation under the given general model assumptions given above.

Based on all of the above conditions the normalized average response time is given by [6],

$$T_p = 1 + \frac{a}{2} + \frac{a(1 - S/N)}{2(1 - S)} + \frac{S}{2(1 - S)} \quad , a \leq 1 \qquad (7.1)$$

$$T_p = 1 + \frac{a}{2} + \frac{a(1 - Sa/N)}{2(1 - Sa)} + \frac{Sa^2}{2(1 - Sa)} \quad , a > 1$$

- $a \leq 1$ where a is the normalized ring latency defined by $a = t_{RL}C/E[L_m]$.

2. Slotted Ring

Slotted Rings have been analyzed by Bux [42] and Hammond and O'Reilly [6]. Bux's model will be used here because it was obtained via a more rigorous analysis.

Based on the assumptions that:

- more than one station has data to transmit (i.e., the ring is never completely idle)
- there are no gaps in the ring
- the minipacket length is far less than the message length

The average transfer time is given by

$$E[T] = \frac{2}{1 - \rho} E[T_p^*] + \frac{t_{RL}}{2} \qquad (7.2)$$

where

$$\rho = N\lambda E[T_p^*] \qquad (7.3)$$

and

$$E[T_p^*] = \frac{L_h + L_d}{L_d} \cdot \frac{E[L_p]}{C} \qquad (7.4)$$

and the following quantities are defined

$E[T_p^*]$ = average packet service time

L_h = length (in bits) of minipacket header

L_d = length (in bits) of minipacket

L_p = length (in bits) of packet.

Substituting Eq. (7.4) in Eq. (7.2) and, after some manipulation, the normalized average response time can be determined as:

$$T_p = \frac{2(1 + h)}{1 - S(1 + h)} + \frac{a}{2} \qquad (7.5)$$

where,

h = Lh/Ld is the overhead factor.

3. Register Insertion Ring

Hammond and O'Reilly [6] have shown that based on the assumptions:

- fixed message lengths
- destination station removes messages
- symmetric traffic where stations have equally likely destination probabilities, i.e., $\alpha = (N/2) - 1$ where α is the average number of insertion buffers traversed by a message.

The normalized average transfer time is given by

$$T_p = 1 + \frac{a}{2} + \frac{NS}{8(1 - S/2)} \qquad (7.6)$$

4. DCPR

This has already been analyzed in Chapter 4. Thus, the M/G/1 queueing model will be used. However, as opposed to Chapter 4, the average normalized response time for the total system is derived from analysis as a FIFO (First_In, First_Out) queue.

The average response time is given by Kleinrock [9],

$$E[T] = E[X] + \frac{(1 + C_b^2)}{2(1 - \rho)} E[X] \tag{7.7}$$

where X is the service time and $C_b^2 = \sigma_b^2/(E[X])^2$ is the coefficient of variation.

After some algebra, the average normalized response time for fixed message length is given by

$$T_p = (h' + 1)\ [1 + \frac{S(h' + 1)}{2(1 - S(h' + 1))}\ ] \tag{7.8}$$

where S and h' are as defined in Chapter 4 (see Eq. 4.52).

It can be seen that if the overhead factor $h' << 1$,

$$T_p = 1 + \frac{S}{2(1 - S)} \tag{7.9}$$

which is the ideal response time for any system (M/D/1 queue).

5. Improved DCPR (IDCPR)

Equation (7.8) is used for this model with a different reservation time (see Eq. 6.5) substituted. This will reflect in the value of h'.

Numerical Results

Throughput-delay equations have just been developed for the candidate ring access protocols: token, slotted, register insertion, DCPR, and IDCPR. The main objective of this development is to compare the two ring access protocols (DCPR and IDCPR) discussed in this dissertation with the well-known token, slotted, and register insertion ring protocols.

The comparison is not of the ring networks but rather their media access control protocol layers. Moreover, only one performance aspect will be the issue here namely, throughput-delay. Other equally important practical performance issues such as [43] cost, reliability, availability, higher-layer protocols, etc. are not addressed in this discussion.

Comparative performance will be made with the help of Figures 7-1 through 7-8.

The figures show plots of throughput (S) versus normalized average transfer delay (T_p) for fixed values of channel capacity (C), number of stations (N), and average message length (L_m). Fixed message lengths are assumed in all the figures.

Slotted ring is plotted for two values of the overhead factor: the ideal case (which never occurs), h = 0 and the practical case, h = 1. DCPR is also plotted for two values of the reservation time: the minimum, T_{rsvmin} (which occurs under heavy load) and the maximum, T_{rsvmax} (which occurs under light load).

The range of variables used in the figures are: C = (1 Mbps, 10

Mbps), L_m = (1 kbits, 50 kbits), and N = (10, 50). The figures (7-1 to 7-8) show that the token ring operates the best under light load whereas the register insertion ring operates the best under heavy load. The figures also show that the register insertion ring and IDCPR are sensitive to the number of stations on the ring whereas the slotted and token rings are relatively in sensitive.

Figures 7-1 to 7-4 show that the DCPR and IDCPR perform poorly when the average message length is small (1000 bits) but perform well when the average message length is large (50 kbits).

Figures 7-5 to 7-8 show that increasing the channel capacity does not necessarily provide a better performance. In fact the performance of IDCPR deteriorates under these conditions. This and the fact that IDCPR performs very well with large average message lengths shows that the factor affecting the performance of IDCPR is the normalized ring latency. Note that either a low average message length or a high channel capacity will give a high normalized ring latency.

In conclusion, it can be said that when the average message length is large (50 kbits, say), IDCPR is as good as the token ring but far better that the slotted ring with h = 1 (note that most practical slotted rings such as the Cambridge ring use h = 1).

LEGEND: DCPR min = DCPR with min. rsv. time; SR = slotted ring
DCPR max = DCPR with max. rsv. time; TR = token ring
IDCPR = improved DCPR; RI = register insertion ring

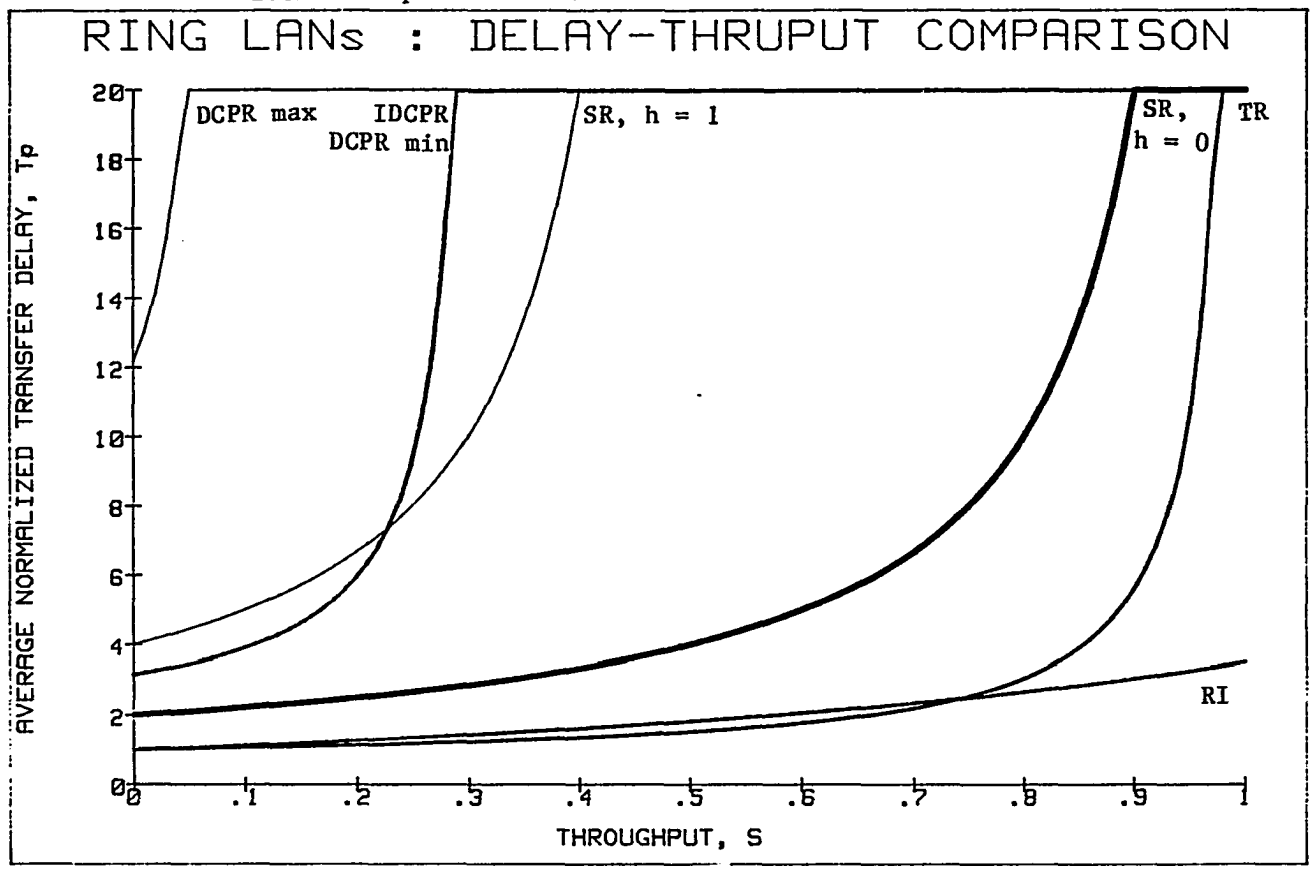


Figure 7-1. Normalized averaged transfer delay as a function of throughput. Channel capacity C = 1 Mbps, average message length Lm = 1k bits, number of stations N = 10

LEGEND: DCPR min = DCPR with min. rsv. time; SR = slotted ring
DCPR max = DCPR with max. rsv. time; TR = token ring
IDCPR = improved DCPR; RI = register insertion ring

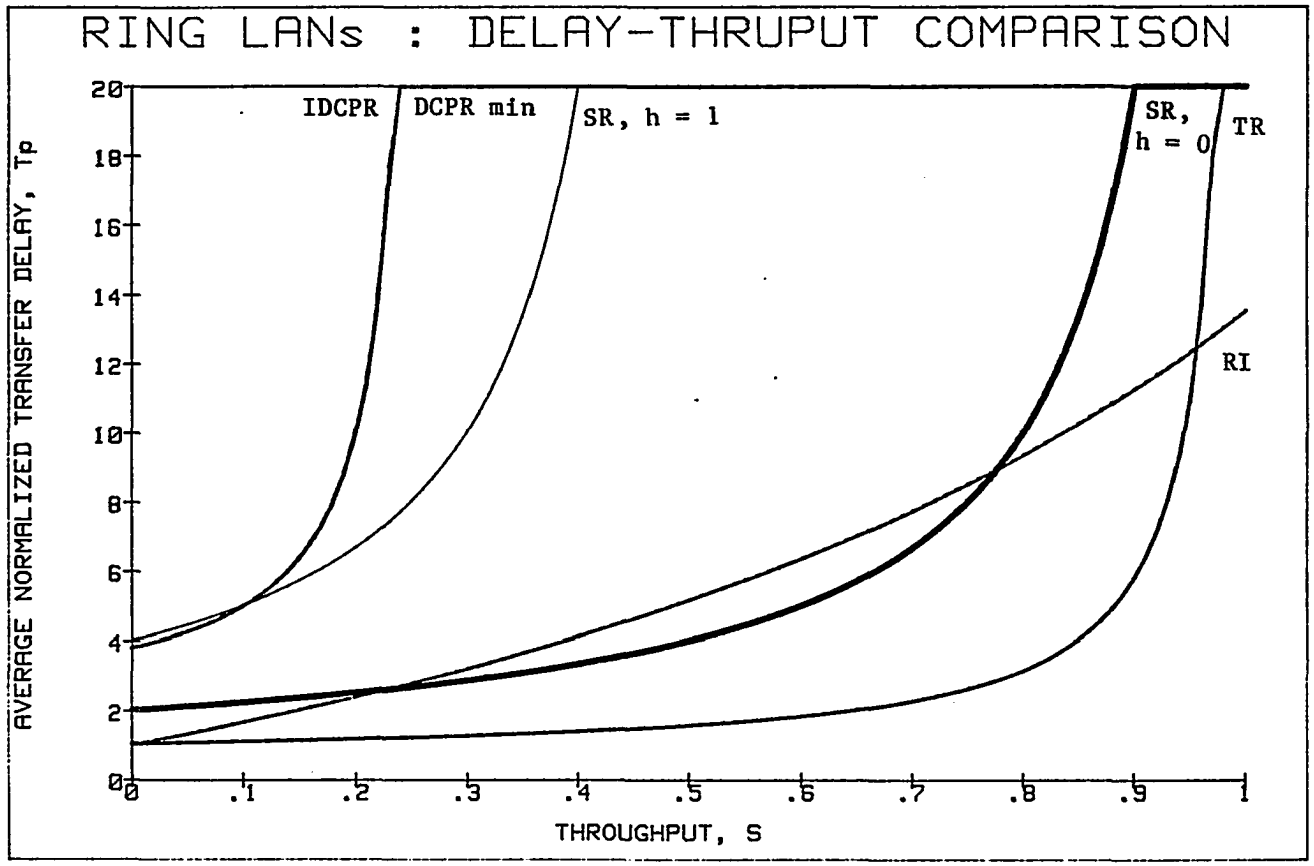


Figure 7-2. Normalized averaged transfer delay as a function of throughput. Channel capacity C = 1 Mbps, average message length Lm = 1000 bits, number of stations N = 50

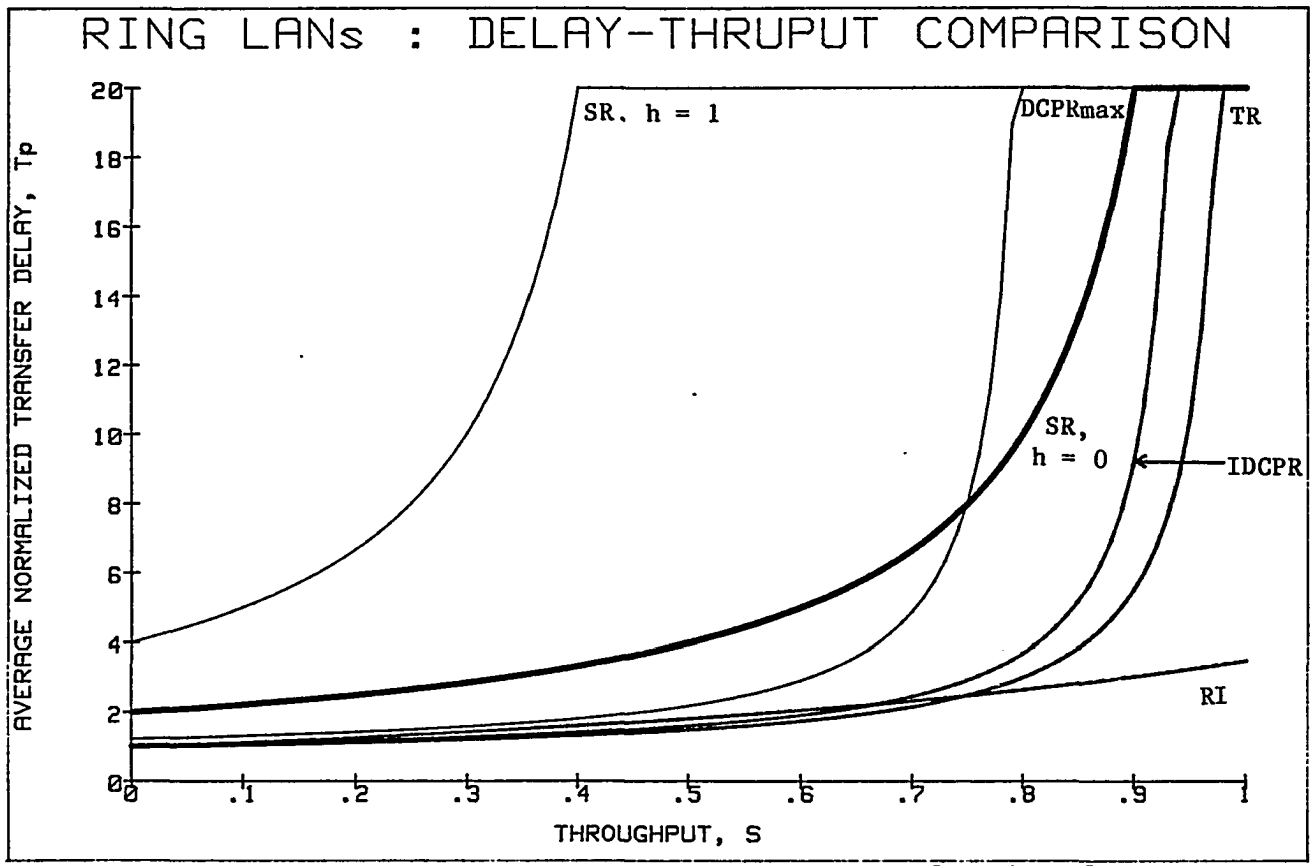


Figure 7-3. Normalized averaged transfer delay as a function of throughput. Channel capacity C = 1 Mbps, average message length Lm = 50 bits, number of stations N = 10

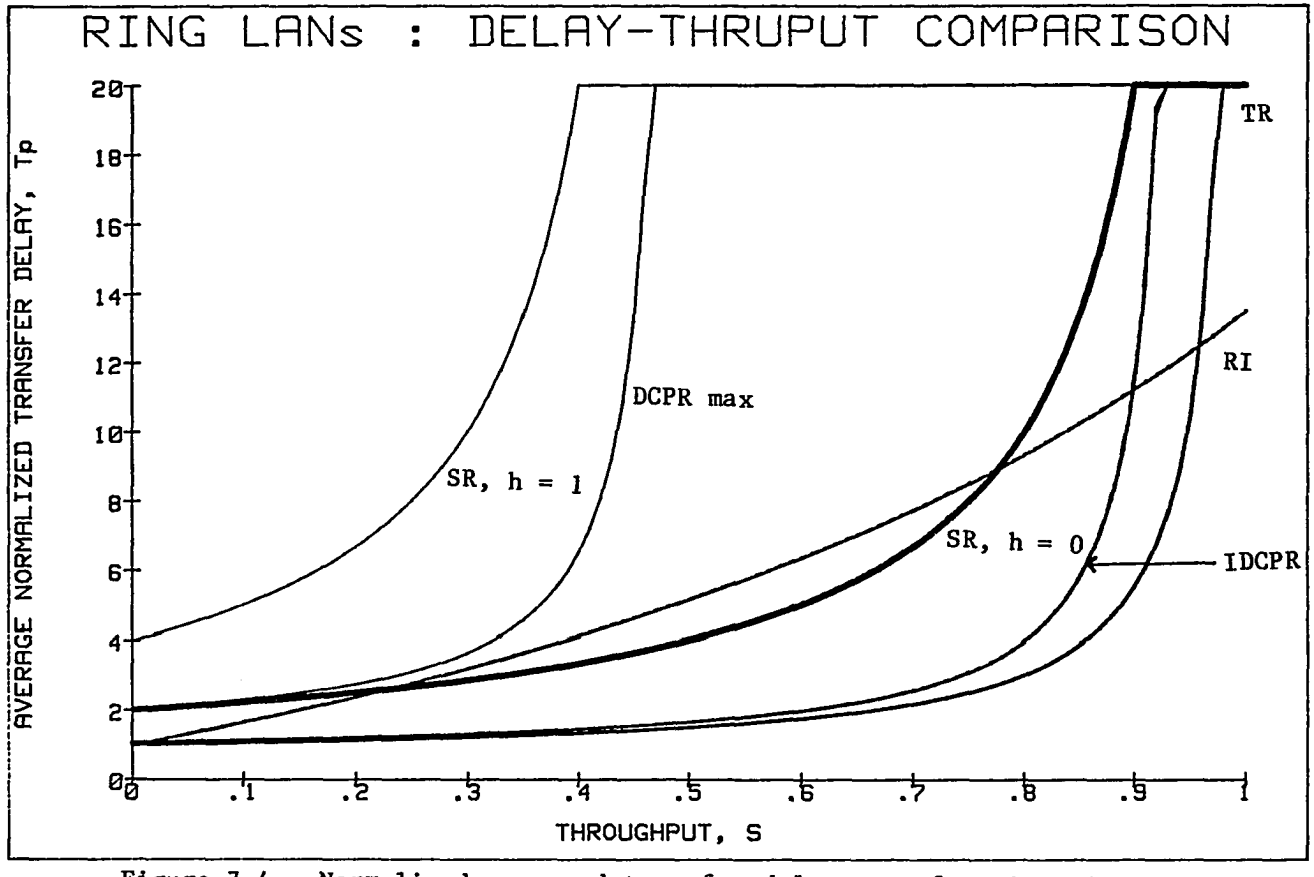


Figure 7-4. Normalized averaged transfer delay as a function of throughput. Channel capacity C = 1 Mbps, average message length Lm = 50k bits, number of stations N = 50

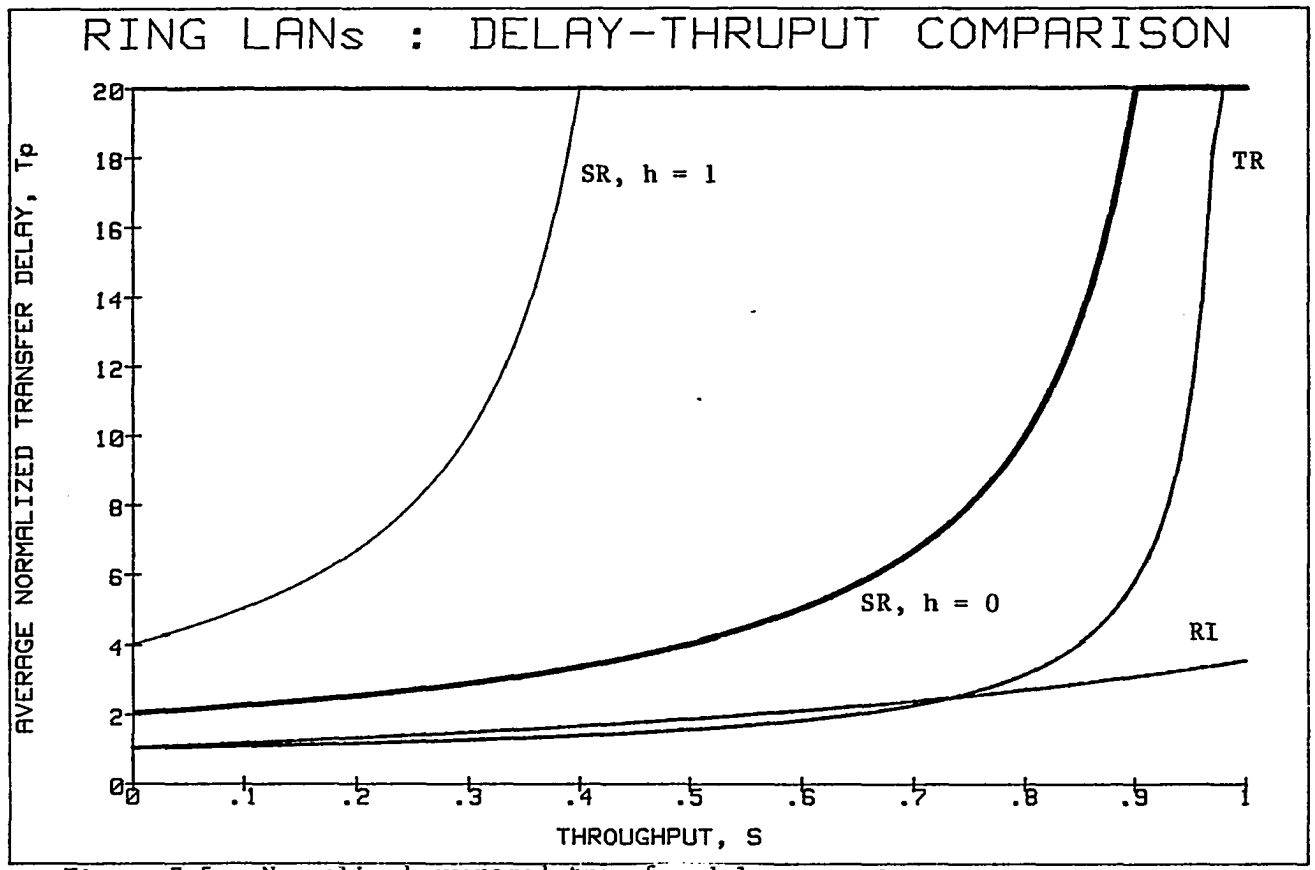


Figure 7-5. Normalized averaged transfer delay as a function of throughput. Channel capacity C = 10 Mbps, average message length Lm = 1k bits, number of stations N = 10

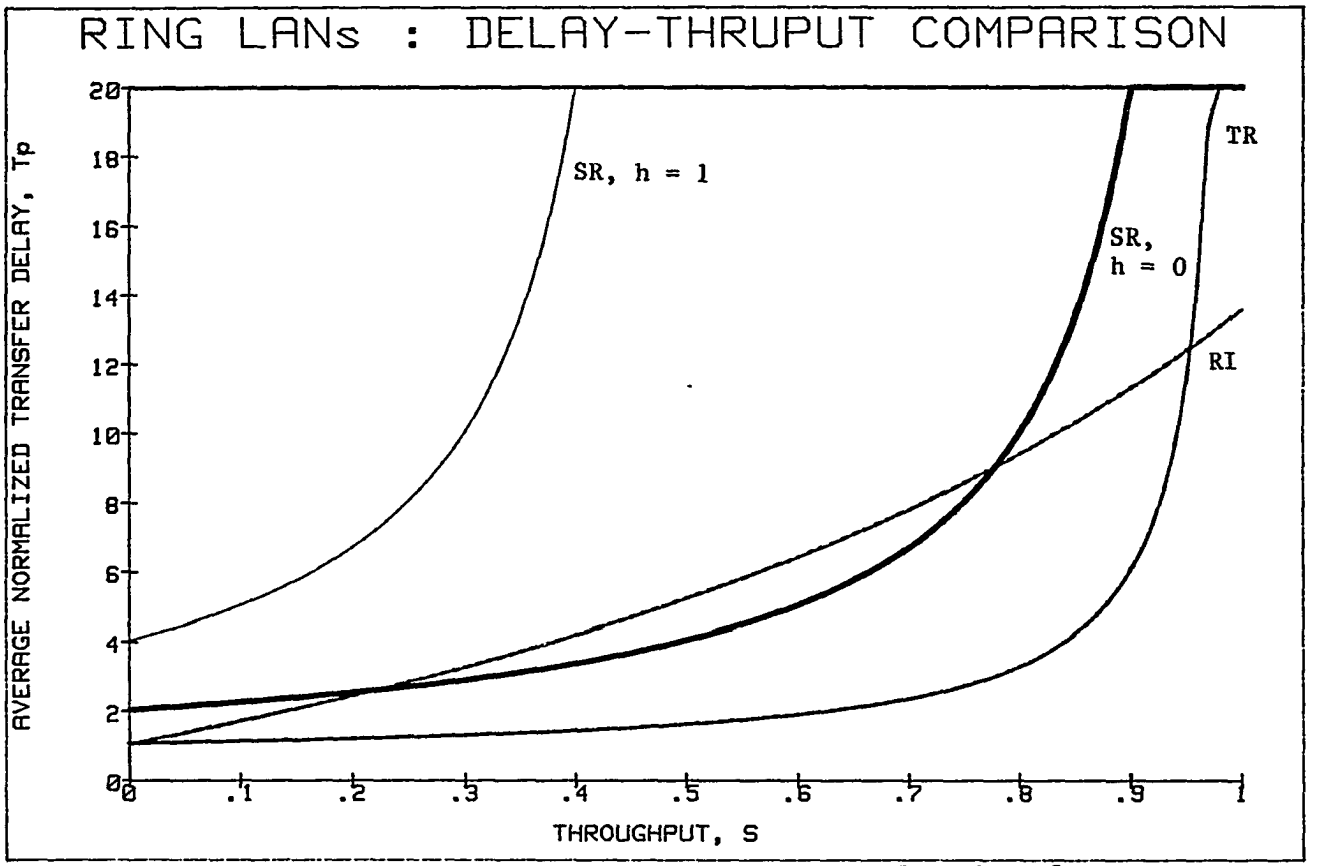


Figure 7-6. Normalized averaged transfer delay as a function of throughput. Channel capacity C = 10 Mbps, average message length Lm = 1k bits, number of stations N = 50

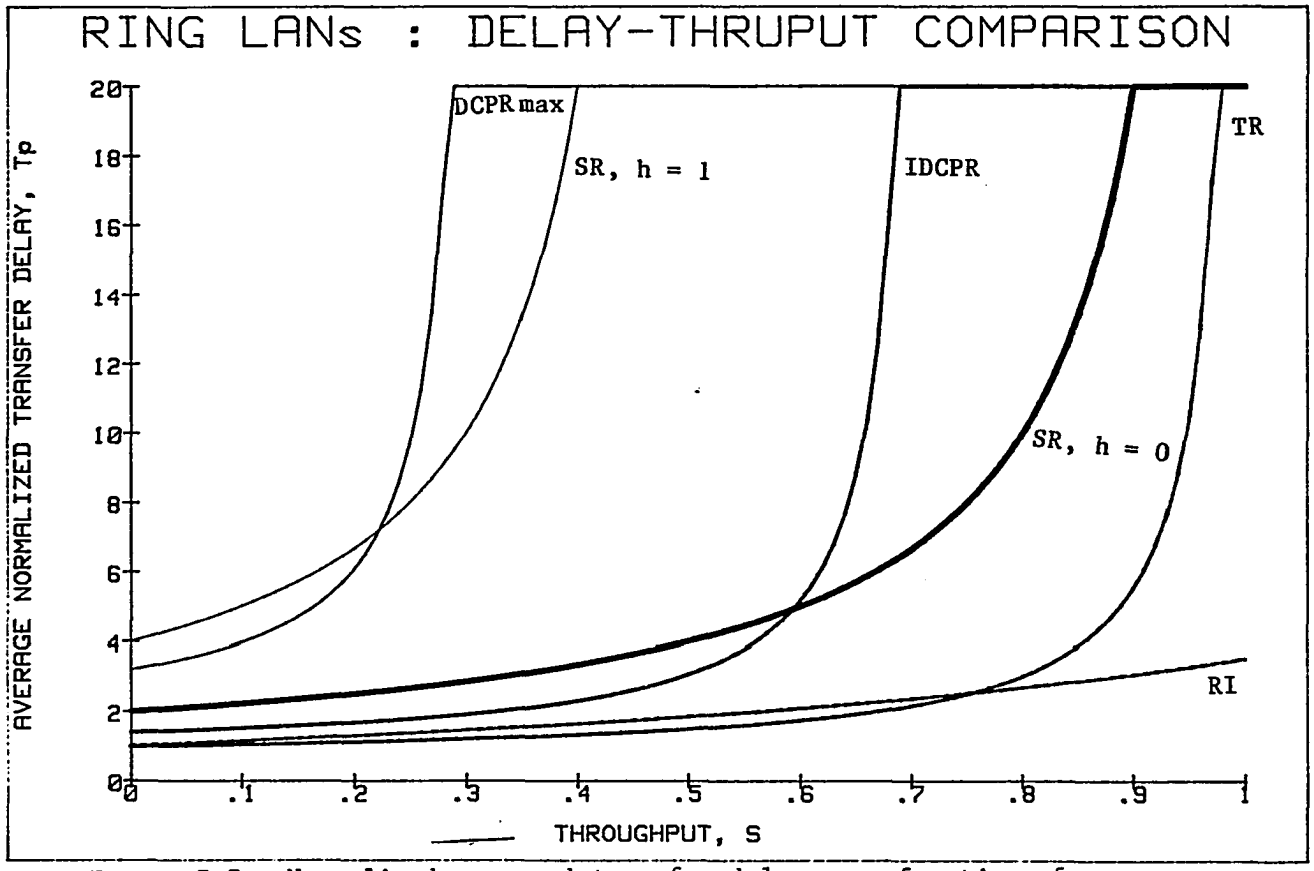


Figure 7-7. Normalized averaged transfer delay as a function of throughput. Channel capacity C = 10 Mbps, average message length Lm = 50k bits, number of stations N = 10

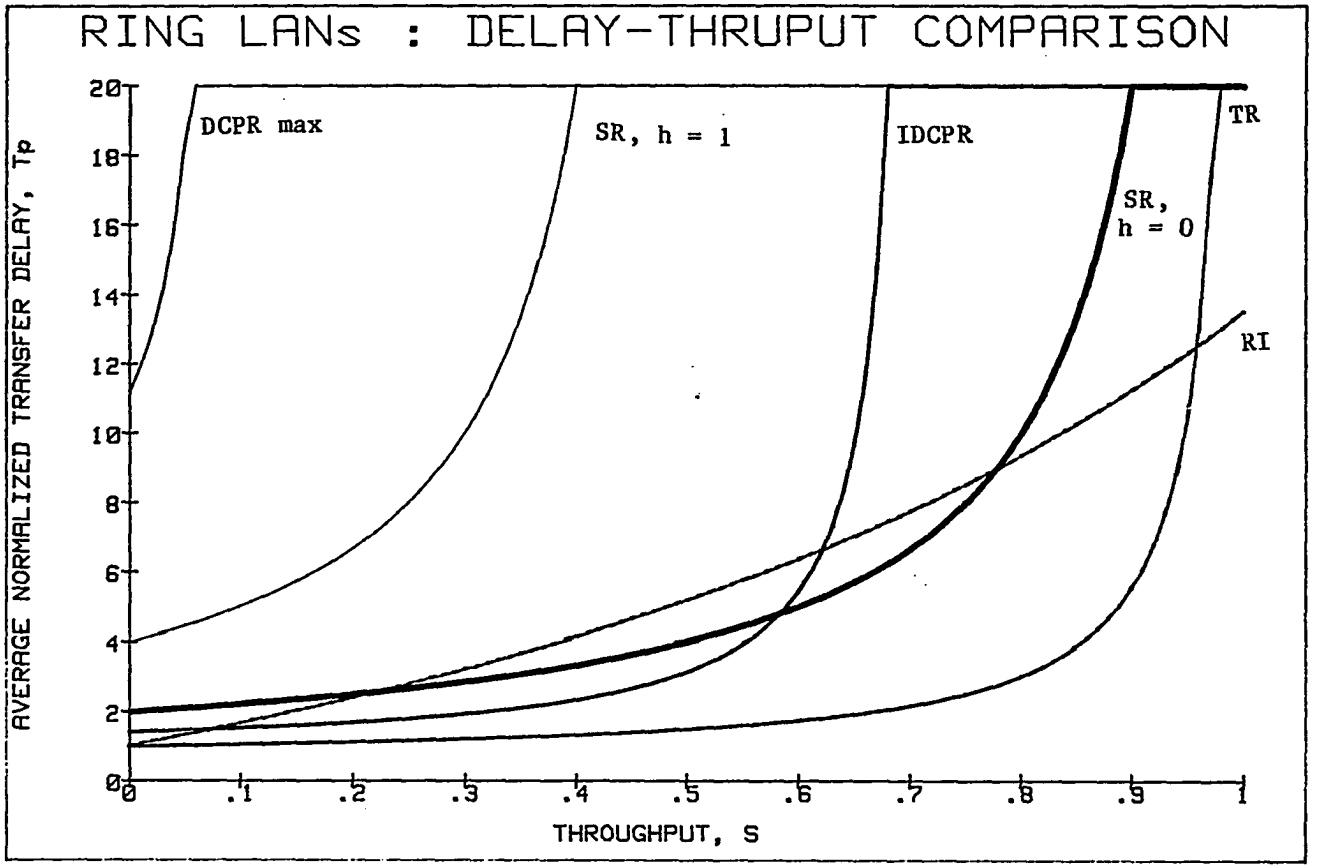


Figure 7-8. Normalized averaged transfer delay as a function of throughput. Channel capacity C = 10 Mbps, average message length Lm = 50k bits, number of stations N = 50

CHAPTER 8. CONCLUSIONS

This dissertation introduced a medium access control protocol designed for a local area computer network with a ring topology.

Priorities are used to ensure that only one ready station gains access to the channel. The priorities are embedded in a "request for reservation" (RFR) packet. During the reservation period each ready station transmits its RFR to the next station (unidirectional channel is assumed). A station which receives an RFR closes its link if it is not ready or if the received RFR is of higher priority. Otherwise, it leaves its link open. The only way a ready station knows it has won the bidding for the channel is if it receives its RFR back. This in turn means all links are closed and all stations are waiting for a message.

Two protocols were described. The first one called the Distributed Channel-Sense Priority Ring (DCPR) was described in Chapter 3. This protocol used sequential querying to cause stations to close their links. Thus, the highest priority station sends its RFR repeatedly until all stations close their link. As found in Chapter 4, the reservation time for DCPR affected its performance. Under light load, it reached its maximum. Under heavy load it was minimum. It was also found that the reservation time depends on the priority ranking of the stations along the ring, arrival rate of messages to the stations along the ring, arrival rate of messages to the stations and the number of stations.

Although analytic models were developed to obtain performance measures of DCPR, they were based on independent assumption for

reservation times. As proved in Chapter 5, the analytic models were not very accurate because the reservation times were dependent. This led to the use of simulation. The simulation results agreed with the intuitive results, namely that the reservation times depend on the readiness state of the stations. In spite of its inaccuracy, the M/G/1 queueing model gave some insight in DCPR performance.

The second protocol, an "improved" version of DCPR (called IDCPR) was motivated by the fact that too much time was spent to reserve the channel under light load. As seen in Chapter 6, the key idea behind IDCPR is to let all stations be involved in transmitting the higher priority RFRs. As a result, the higher priority station did not have to wait to time out and send its RFR repeatedly until it eventually receives it back. Rather it transmits it once and expect the lower priority stations to pass it along as they see it. This new protocol was seen to have several advantages.

First, whereas DCPR used "sequential querying" IDCPR used "parallel querying" to force lower priority stations to close their links. Second, whereas in DCPR the reservation time depends on such factors as arrival rate of messages, message length, priority ordering of stations on the ring and number of stations, IDCPR reservation time is dependent only on the number of stations. Third, as a result of the latter, a network designer, or manager does not have to worry about where to place a station of a certain priority to achieve a certain performance. Thus IDCPR has the property that any station of any priority can be placed anywhere on the ring without degrading performance. Fourth and most

important, the reservation time is the minimum of the reservation time in DCPR and constant at all times for the same number of stations.

A performance comparison of IDCPR and well-known ring access protocols such as token ring, slotted ring, and register insertion ring was made in Chapter 7. It was found that in terms of throughput-delay performance, IDCPR is not better than the token ring or register insertion ring. Whereas the token ring, for example, performed very well under a wide range of system parameters, IDCPR was sensitive to the number of stations and especially, message lengths. Indeed, it performed the best when the message length was very large (e.g., 50000 bits). The performance of the register insertion ring deteriorated as the number of stations increased. In spite of this, the register insertion ring was, in general, found to have a very high utilization. This is a well-known fact [3,6,18,19]. Intuitively, this is not unexpected because a station does not wait for any signal before sending its message. The channels are kept busy as long as there are messages to transmit. Moreover, only actual messages are transmitted. There are no tokens or RFRs. The slotted ring seemed to have the worst performance of the ring networks, the main reason being the large amount of overhead bits per minipacket. Note that in practical slotted ring networks such as the Cambridge Ring the overhead factor (number of overhead bits per actual data bits in a minipacket) is 24/16 [6,44]. In spite of such a poor performance, slotted rings are very popular in Europe. The reason may well be, as alluded to at the beginning of Chapter 7, that other factors such as cost, reliability, higher layer protocols, ease of implementation, etc.

have to be considered before making general statements about which LAN protocol is better. In fact, Blair and Shepherd [45] have shown that the Cambridge Ring is more cost effective than Ethernet, although the latter is very popular in the United States.

Since it has been demonstrated that IDCPR performs very well when the average message length is large then based on the analysis in this dissertation it belongs to a class of channel access control protocols used for implementing High Speed Local Networks (HSLNs) [3,46]. These networks are also called by such names as "Back-End Networks" [47] and "Computer Room Networks." They are used to connect computers to storage devices and peripherals. The storage devices such as high speed, high volume disk systems need to transfer large amounts of data as quickly as possible. Interest in these kinds of networks has prompted ANSI (American National Standards Institute) to form the X3T9.5 subcommittee to define a standard called LDDI (Local Distributed Data Interface) [48]. Two candidates for the standard are HYPER-channel [47] which uses a prioritized CSMA MAC protocol on a bus topology and DEC's CI-network [49] which uses a CSMA protocol with prioritized access delay on a coaxial star cable. A second standard under consideration called FDDI (Fiber Distributed Data Interface) is being defined for fiber optic networks. The multiple-token, multiple-packet approach is being considered for a ring topology [50,51].

IDCPR can be a candidate for either of the two standards depending on the transmission medium-coaxial or fiber. Note that although it has been demonstrated that in terms of throughput-delay, the token-ring

approach is better than IDCPR the former has its own problems namely complexity, loss of tokens, multiple tokens, etc.

The research reported in this dissertation provides several contributions to the field of Local Area Computer Networks. First, the IDCPR protocol illustrates an application of multiaccess protocol using a generalized reservation scheme on a ring topology LAN. Reservation schemes described in the literature are typically for bus networks. Second, the State Architecture Notation, a formal protocol description technique, has been used to specify and simulate the protocol. Third, IDCPR can be considered as an addition to the medium access control protocols that can be used in networks interconnecting computers and mass storage devices.

Finally, IDCPR will be a continuing research topic in the following directions:

1) Analyze its performance using fair access dynamic priority schemes such as round-robin (RR) and shortest length message first (SLMF). Both of these schemes will not be hard to implement. In the case of RR, each station will update its priority at the end of each message transmission. In the case of SLMF, the length of messages will be transmitted as RFRs.
2) Incorporate management functions to deal with such practical problems as multiple RFR's of the same priority or lost RFRs.
3) Do a more comprehensive comparison with other MAC protocols in cluding contention ring, token bus, and CSMA/CD bus.

BIBLIOGRAPHY

1. L. G. Roberts. "The Evolution of Packet Switching." Proceedings of the IEEE, 66, No. 11 (Nov. 1978):1307-1313.

2. M. Schwartz. Telecommunication Networks: Protocols, Modeling and Analysis. Reading, Mass.: Addison Wesley, 1987.

3. W. Stallings. "Local Networks." Computing Surveys, 16, No. 1 (March 1984):3-41.

4. R. M. Metcalfe and D. R. Boggs. "Ethernet: Distributed Packet Switching for Local Computer Networks." Communications of the ACM, 19, No. 7 (July 1976):395-404.

5. J. F. Shoch, Y. K. Dabl, D. D. Redell, and R. C. Crane. "Evolution of the Ethernet Local Computer Network." Computer, 15 (August 1982):10-26.

6. J. L. Hammond and P. J. P. O'Reilly. Performance Analysis of Local Computer Networks. Reading, Mass.: Addison Wesley, 1986

7. F. A. Tobagi and V. B. Hunt. "Performance Analysis of Carrier Sense Multiple Access with Collision Detection." Computer Networks 4 (1980):245-259.

8. I. Rubin. "Synchronous and Channel-Sense Asynchronous Dynamic Group-Random-Access Schemes for Multiple-Access Communications." IEEE Trans. on Communications, Com-31, No. 9 (Sept. 1983):1063-1076.

9. L. Kleinrock. Queueing Systems. Vol. 2: Computer Applications. New York: John Wiley and Sons, 1976.

10. A. G. Konheim and B. Meister. "Waiting Lines and Times in a System with Polling." Journal of ACM, 21 No. 3 (July 1974):470-490.

11. J. F. Hayes and D. N. Sherman. "A Study of Data Multiplexing Techniques and Delay Performance." Bell System Technical Journal, 51 No. 9:1983-2011.

12. J. A. Murphy. "Arcnet: Design and Implementation of a Local Area Network." Technical Report. Datapoint Corporation, San Antonio, Texas, October 1982.

13. D. W. Andrews and G. D. Schultz. "A Token-Ring Architecture for Local-Area Networks: An Update." The Proceedings of COMPCON F82 (1982):615-624.

14. E. G. Rawson. "Application of Fibre Optics to Local Area Networks." Proc. Local Area Networks Symposium, Boston (May 1979):155-167.

15. I. N. Dallas and E. B. Spratt. Proceedings of the IFIP WG6.4/ University of Kent Workshop on Ring Technology Based Local Area Networks, Kent, U.K., September 1983.

16. J. H. Saltzer and K. T. Pogram. "A Star-Shaped Ring Network with High Maintain Ability." Proc. Local Area Networks Symposium, Boston (May 1979):179-189.

17. M. V. Wilkes and D. J. Wheeler. "The Cambridge Digital Communication Ring." Proceedings of LACN Symposium, Boston (May 1979):47-61.

18. M. T. Liu. "Distributed Loop Computer Networks." In Advances in Computers, pp. 163-221. Edited by M. C. Yovits. New York: Academic Press, 1978.

19. W. Stallings. Data and Computer Communications. New York: Macmillan, 1985.

20. A. S. Tanenbaum. Computer Networks. Englewood Cliffs, New Jersey: Prentice-Hall, Inc., 1981.

21. M. T. Liu. "A Study of Ring Networks." In Ring Technology Local Area Networks, pp. 1-39. Edited by I. N. Dallas and E. B. Spratt. Amsterdam: North-Holland Publishing Co., 1984.

22. I. Rubin. "Message Delays in FDMA and TDMA Communication Channels." IEEE Trans. on Communication, COM-27 (May 1979):769-777.

23. I. Rubin. "Message Delays for a TDMA Scheme under a Nonpreemptive Priority Discipline." IEEE Trans. on Communications, COM-3, No. 5 (May 1984):583-588.

24. L. G. Roberts. "Dynamic Allocation of Satellite Capacity through Packet Reservation." Proc. AFIPS National Computer Conference, 42 (1973):711-716.

25. G. Pujolle (ed.). Performance of Data Communications Systems and Their Applications. Amsterdam: North Holland Publishing Co., 1981.

26. J. W. Mark. "Distributed Scheduling Conflict-Free Multiple Access for Local Area Communications Networks." IEEE Trans. on Communications, COM-28, No. 12 (Dec. 1980):1968-1976.

27. J. F. Hayes. "An Adaptive Technique for Local Distribution." IEEE Trans. on Communications, COM-26, No. 8 (August 1978):1178-1186.

28. E. H. Rothhauser and D. Wild. "MLMA - A Collision-Free Multi-Access Method." Pages 431-436. In Proceedings IFIP Congress 77. Amsterdam: North-Holland Publishing Co., 1977.

29. J. I. Capetanakis. "Tree Algorithms for Packet Broadcast Channels." IEEE Trans. on Information Theory, IT-25, No. 5 (1979):505-515.

30. S. S. Lam. Tutorial: Principle of Communication and Networking Protocols. Silver Spring, Maryland: IEEE Computer Society Press, 1984.

31. J. Seidler. Principles of Computer Communication Network Design. New York: Halsted Press, Wiley and Sons, 1983.

32. L. Kleinrock. "Performance of Distributed Multi-access Computer Communication Systems." Pages 547-552. In Proc. IFIP Congress 77. Amsterdam: North-Holland Publishing Co., 1977.

33. F. A. Tobagi. "Multiaccess Protocols in Packet Communication Systems." IEEE Trans. on Communications, COM-28, No. 4 (April 1980):468-488.

34. F. A. Tobagi. "Carrier Sense Multiple Access with Message-based Priority Functions." IEEE Trans. on Communications, COM-30 (1982):185-200.

35. M. G. Hluchyj, C. D. Tsao, and R. R. Boorstyn. "Performance Analysis of a Preemptive Priority Queue with Applications to Packet Communication Systems." The Bell System Technical Journal, 62, No. 10 (December 1983):3225-3245.

36. C. Sunshine. Communication Protocol Modeling. Dedham, Mass.: Artech House, Inc., 1981.

37. T. F. Piatkowski. "Elements of the Total Design Process for Large Discrete Control Systems." Class Notes for EE581. Iowa State University, Ames, Iowa, Spring 1981.

38. D. P. Heyman and M. J. Sobel. Stochastic Models in Operations Research. Vol. I. New York: McGraw-Hill Publishing Co., 1982.

39. H. Kobayaski and A. G. Konheim. "Queueing Models for Computer Communications System Analysis." IEEE Trans. on Communications, COM-25, No. 1 (1977):2-28.

40. W. Stallings. "Local Network Performance." IEEE Communications Magazine 22, No. 2 (Feb. 1984):27-36.

41. IEEE Standards Board. IEEE Standards for Local Area Networks: Token-Ring Access Method and Physical Layer Specifications. New York: IEEE, 1985.

42. W. Bux. "Local Area Subnetworks: A Performance Comparison." IEEE Trans. on Communications, COM-29, No. 10 (October 1981):1465-1473.

43. L. Kleinrock. "A Decade of Network Development." Journal of Telecommunication Networks 1 (Spring 1982):1-11.

44. F. Halsall. Introduction to Data Communications and Computer Networks. Reading, Mass.: Addison Wesley, 1985.

45. G. D. Blair and D. Shepherd. "Performance Comparison of Ethernet and the Cambridge Digital Communication Ring." Computer Networks 6 (1982):105-113.

46. W. Stallings. Tutorial: Local Network Technology. Silver Spring, Maryland: IEEE Computer Society Press, 1985.

47. J. E. Thornton. "Back-End Network Approaches." Computer, 13 (February 1980):10-17.

48. W. E. Burr. "An Overview of the Proposed American National Standard for Local Distributed Data Interfaces." Communications of the ACM 26 (August 1983):554-561.

49. W. D. Strecker. "CI: A High Speed Interconnect for Computers and Mass Storage Controllers." Proceedings of the Eighth International Fiber Optics Communications and Local Area Networks Exposition. Information Gatekeepers, Inc. (1984):246-250.

50. S. Joshi and V. Iyer. "New standards for local networks push upper limits for lightwave data." Data Communications 3 (July 1984):127-138.

51. F. E. Ross. "FDDI - a Tutorial." IEEE Communications Magazine 24, No. 5 (May 1986):10-17.

ACKNOWLEDGEMENTS

Words cannot tell how thankful I am to Dr. Arthur V. Pohm. His constant encouragement, personal kindness, support, concern, and advice throughout my graduate career are gratefully appreciated. This work could not have been completed without the help of Dr. Douglas W. Jacobson who provided valuable suggestions and guidance during its development.

My appreciation is also extended to Drs. Terry A. Smay, Chester S. Comstock, Charles T. Wright, Harry W. Hale and Dale D. Grosvenor for serving on my graduate committee and Rebecca Shivvers for typing this dissertation.

My graduate program was supported financially by teaching and research assistantships through the efforts of Dr. J. O. Kopplin, chairman of the department of Electrical Engineering and Computer Engineering. These efforts are gratefully acknowledged. Norand Corporation of Cedar Rapids, Iowa, provided partial research funding for this work.

I will be remiss if I do not extend my sincere gratitude and appreciation to Dr. Thomas F. Piatkowski, currently chairman of the Department of Computer Science, State University of New York at Binghamton, for introducing me to the field of Computer Networking and Data Communications.

Deserving special recognition are my wife, Rebecca, and son. Frederick, for having patiently endured many hours alone, while I was grappling with the subject matter in this dissertation. Becky's love,

assistance, guidance, encouragement and companionship were instrumental in my ability to successfully accomplish the tasks expected of me during my entire graduate program. My late mother, Comfort Tanuweh (Day) Yalley worked hard and provided the love encouragement and support which laid the foundation to make this work possible. May she rest in peace.

Last, but not the least, thanks be to the Almighty God for His abundant mercy and grace through Jesus Christ without whom my life will be drifting like a ship without a sail.

APPENDIX: PASCAL LISTING OF DCPR SIMULATION PROGRAM

```
1.  PROGRAM DCPR(INPUT,DCPRIN,OUTPUT,DCPROUT,SYSFILE,ERRFILE,PLTFILE1,
2.              PLTFILE2,PLTFILE3);
3.
4.  (* THIS MODEL IS A PROGRAM USED FOR SIMULATING THE DISTRIBUTED *)
5.  (* CHANNEL-SENSE PRIORITY RING UNDER ERROR-FREE CONDITIONS*)
6.
7.  CONST  LHDR=56;          (* PKT HDR LENGTH IN BITS *)
8.         LACK=24;          (* ACK/NACK/DONE LEN IN BITS *)
9.         LRFR=16;          (* RFR LENGTH IN BITS *)
10.       RINGLEN=1.0; (*TOTAL RING LENGTH IN KILOMETERS*)
11.       RINGDLY=5.0E-6;(*CABLE PGTN DLY IN SEC/KILOMETER*)
12.       MAXSTA=10;
13.        INFINITY=10000;
14.       DEBUG=FALSE;(*PROGRAM DEBUG ENABLER *)
15.        INTERACT=FALSE; (*INTERACTIVE MODE OF INPUTTING DATA*)
16.        PSCDEBUG=FALSE; (*PASCALVS(IBM) DEBUGGER ENABLER*)
17.        NOMOFEVENTS=15; (*FIXED # OF EVENTS USED WHEN PSCDEBG=TRUE*)
18.        NUMBER=10;  (*NUMBER OF FUTURE EVNTS TO BE PRINTED *)
19.   TYPE  STARANG=1..MAXSTA;
20.        ARANGE=ARRAY(.1..50.) OF REAL;(*ARRAY FOR INDEP BATCHES*)
21.       PARTYPE=(ARRATE,STAPRIOR,MSGD);(*THIS TELLS PAR TO READ*)
22.       TYPESTASTATE=(IDLEWAIT,RFRWAIT,RFRPEND,RFRSNT,MSGWAIT,MSGPEND,
23.                     MSGSNT);
24.       RECSTASTATE=ARRAY(.STARANG.) OF TYPESTASTATE;
25.       EVENTCLASS=(ARRIVAL,RFRRCV,MSGRCV,ACKRCV,TIMEOUT);
26.        NONTOEVNTS=ARRIVAL..ACKRCV;
27.        PACKETTYPE=(RFR,MSG,ACK);
28.        PACKETPTR=@PACKET;
29.        PACKET=RECORD                  (*VARIANT RECORD FOR PACKET*)
30.                     ORIG:STARANG;
31.                     DEST:STARANG;
32.                     CASE PKTTYPE:PACKETTYPE OF
33.                          RFR,ACK :();
34.                          MSG     :(TIMEOFARRIV:REAL;NEXT:PACKETPTR)
35.               END;
36.        MSGDISTTYPE=(FIXED,EXPONENTIAL);
37.        TIMERTYPE=(IDLETIMER,OTHERTIMER);
38.        EVENTPTR=@EVENT;
39.        EVENT=RECORD                   (*VARIANT RECORD FOR EVENTS *)
40.               EVENTTIME:REAL;
41.               PKTPTR:PACKETPTR;
42.               SID:STARANG;
43.               LLINK:EVENTPTR;                  (*PTR TO LEFT EVENT*)
44.               RLINK:EVENTPTR;                  (*PTR TO RIGHT EVNT*)
45.               CASE EVENTTYPE:EVENTCLASS OF
46.                       TIMEOUT:(TT:TIMERTYPE);
47.                ARRIVAL,RFRRCV:();
48.                MSGRCV,ACKRCV:()
49.        END;
50.
51.        RATES=ARRAY(.STARANG.) OF REAL;(* ARRAY OF REAL NUMBERS *)
52.        QUEUE=RECORD
53.               HEAD:PACKETPTR;
54.               TAIL:PACKETPTR;
55.             END;
56.        PRIORQUEUE=ARRAY(.STARANG.) OF QUEUE;(* ARRAY OF q pointers*)
57.        PRIOR=ARRAY(.STARANG.) OF INTEGER;
58.        CHSTTYPE=(CHIDL,SYNC,RSV,XMIT); (*CHAN STATE TYPE *)
59.        TIMERSTATETYPE=(IDLE,RUNNING,EXPIRED);
60.        TIMERREC=RECORD
```

```
                        TMRSTATE:ARRAY(.TIMERTYPE.) OF TIMERSTATETYPE;
                        TOEVPTR :ARRAY(.TIMERTYPE.) OF EVENTPTR
                    END;
          TCOMPTYPE=ARRAY(.STARANG.) OF TIMERREC;(*ARRAYOFTIMERCOMPONENTS*)
          STATEREC=RECORD (* SYSTEM STATE RECORD TYPE *)
                        CALENDAR:EVENTPTR;
                        CRNTTIME:REAL;
                        PQ:PRIORQUEUE;
                        STASTATE:RECSTASTATE;(*prsnt state of each sta*)
                        STAID:STARANG;(*id of sta in present xsaction*)
                        TLASTDEP:REAL; (* TIMEOFLAST DEPARTURE *)
                        PT:PRIOR; (* PRIOR INDEX OF EACH STAID*)
                        PROPTIME:RATES; (*prop times btwn adj stations*)
                        MDT:MSGDISTTYPE;
                        MSGDIST:RATES;
                        ARR:RATES;(* ARRIVALRATES*)
                        NI:RATES;(*IMPL. DEPENDENT T.O.'S IN MSGWAIT *)
                        TIMERCOMP:TCOMPTYPE;
                        CHSTATE:CHSTTYPE;(* CHANNEL STATE VARIABLE *)
                        NOMOFSTA:INTEGER;(* # OF STAIDS ON RING *)
                        SIMTIME:INTEGER;(*MAX SIMULATION TIME *)
                        TBEGRSV:REAL;(*TIM TO BEG RSVATION OF CHAN. *)
                        TBEGXMIT:REAL;(*TIM TO BEG XMISSION OF MSG  *)
                        TARRIV:REAL;(*ARRIVTIM FOR MSG IN XMITION *)
                    END;
          STATSREC=RECORD (* RECORD OF SYSTEM STATISTICS *)
                        NOFMSGSNT:PRIOR;(* msgs xmitted by each sta*)
                        QLEN:PRIOR;(* QUEUE LENGTH AT EACH STAID*)
                        XMITTIME:REAL;(*TOT TIME CHANEL XMITS MSGS *)
                        TOTRSVTIME:REAL;(*TOTTIME CHAN XMITS Rsv PKTS*)
                         SYNCTIME:REAL;(*TOTTIME CH IS IDL FOR SYNC*)
                                  (*THIS TIM IS TWICE IDL T.O.=2MS*)
                        RESPTIME:RATES;(*TOT RSPNS TIME ARRAY *)
                        RDYSTAS:INTEGER (*TOT # OF RDY STATIONS*)
        (*NOTE! RESPONSE TIME IS DEFINED AS TIME FROM MSG ARRIVAL*)
        (* TIME THE TIME THAT MSG IS SUCCESSFULLY RCVD AT ULT DEST *)
                    END;
VAR     SIMSTATE:STATEREC;
        STATS:STATSREC;
        P:EVENTPTR;
        NODE:STARANG;
        ETYP:EVENTCLASS;
        EVENTTIME:REAL;
        PRIORQ:PRIORQUEUE;
        DONE:BOOLEAN;
        RESPONSE:CHAR;
        REINIT:BOOLEAN;
        STRTARR:BOOLEAN;
        STRTDEP:BOOLEAN;
        ARANDSEED:REAL;
        DRANDSEED:REAL;
        TRFR:REAL;
        RTPDLY:REAL;
        RANDSEED:REAL;
        ERR1:INTEGER;
        EVENTCNTR:INTEGER;(*STORES # OF EVENTS IN DEBUG MODE*)
        DCPRIN:TEXT;(* FILE FOR STORING INPUT DATA *)
        DCPROUT:TEXT;(* FILE FOR  "     OUTPUT STATISTICS*)
        SYSFILE:TEXT;(* "     "    "   SYSTEM STATE AND EVENTS*)
        ERRFILE:TEXT;(* "     "    "   RECORDING ERRORS       *)
```

```
121.          PLTFILE1:TEXT;(* FILE FOR THRPUT-DLY VARSS TO BE PLOTTED*)
122.          PLTFILE2:TEXT;(* FILE FOR THRPUT-RSVTIM VARSS TO BE PLOTTED*)
123.          PLTFILE3:TEXT;(* FILE FOR THRPUT-# RDY STATIONS TO BE PLOTTED*)
124.          TACK:REAL;       (*XMISION TIME FOR ACK                *)
125.          THDR:REAL;       (*XMISION TIME FOR HEADER             *)
126.          TMSG:REAL;       (*XMISSION TIME FOR MESSAGE           *)
127.         STADLY:INTEGER;(*DELAY AT STATIONS IN BITS*)
128.          LINKDLY:REAL;(*PROP DLY BTWN ADJ STATIONS*)
129.       STALAT:REAL; (*DELAY AT STATIONS IN SECS*)
130.          MSGLEN:INTEGER;(*FOR STORAGE OF AVG MSG LENGTH*)
131.          N_THRPUT:REAL;(*"     "      " NORMALIZED THRUPUT, S *)
132.         TOPER:REAL;  (*WORST CASE ROUND TRIP RFR DELAY *)
133.         TWAITMSG:REAL; (*TIME THAT MSG TRANSMISSION PENDS*)
134.          TGAP:REAL;     (* GAP PULSE PERIOD IN SECS. *)
135.          CHCAP:REAL;  (* CHAN CAP IN BITS PER SEC *)
136.    FUNCTION RANDOM(VAR DSEED:REAL):REAL;FORTRAN;
137.    PROCEDURE PRNTTMRSTATE(WHICHTIMER:TIMERTYPE;
138.                                  VAR SIMSTATE:STATEREC);
139.    (*THIS PROC PRINTS STATES OF TIMER SPECIFIED BY  *)
140.    (*WHICHTIMER OF ALL STATIONS                     *)
141.    VAR I:INTEGER;
142.    BEGIN
143.       WITH SIMSTATE,TIMERCOMP(.I.) DO BEGIN
144.           FOR I:=1 TO NOMOFSTA DO BEGIN
145.              WRITE(SYSFILE,'STATE OF ');
146.              IF WHICHTIMER=IDLETIMER THEN BEGIN
147.                WRITE(SYSFILE,'IDLETIMER')
148.              END ELSE BEGIN
149.                WRITE(SYSFILE,'OTHERTIMER')
150.              END; (* IF *)
151.              WRITE(SYSFILE,I:3,'=');
152.              CASE TMRSTATE(.WHICHTIMER.) OF
153.                  IDLE:      WRITELN(SYSFILE,'IDLE');
154.                  RUNNING:   WRITELN(SYSFILE,'RUNNING');
155.                  EXPIRED:   WRITELN(SYSFILE,'EXPIRED')
156.              END;(*CASE*)
157.          END(*FOR*)
158.       END(*WITH*)
159.    END;(*PRNTTMRSTATES*)
160.    PROCEDURE PRNTSTASTATES(VAR SIMSTATE:STATEREC);
161.    VAR I:INTEGER;
162.    BEGIN
163.           WITH SIMSTATE DO BEGIN.
164.              FOR I:=1 TO NOMOFSTA DO BEGIN
165.                  WRITE(SYSFILE,'STATE OF STATION ',I:3,'=');
166.                  CASE STASTATE(.I.) OF
167.                   IDLEWAIT:WRITELN(SYSFILE,'IDLEWAIT');
168.                   RFRWAIT :WRITELN(SYSFILE,'RFRWAIT');
169.                   RFRPEND :WRITELN(SYSFILE,'RFRPEND');
170.                   RFRSNT  :WRITELN(SYSFILE,'RFRSNT');
171.                   MSGWAIT :WRITELN(SYSFILE,'MSGWAIT');
172.                   MSGPEND :WRITELN(SYSFILE,'MSGPEND');
173.                   MSGSNT  :WRITELN(SYSFILE,'MSGSNT')
174.                  END (*CASE*)
175.           END (*FOR*)
176.         END(* WITH*)
177.    END;(*PRNTSTATSTATES*)
178.    PROCEDURE PRNTEVENT(EVP:EVENTPTR);
179.    BEGIN
180.           WRITE(SYSFILE,'EVENT ');
```

```
181.        WITH EVP@ DO BEGIN
182.                CASE EVENTTYPE OF
183.                        ARRIVAL :WRITE(SYSFILE,'ARRIVAL');
184.                        MSGRCV  :WRITE(SYSFILE,'MSGRCVD');
185.                        RFRRCV  :WRITE(SYSFILE,'RFRRCVD');
186.                        TIMEOUT :BEGIN
187.                                   WRITE(SYSFILE,'TIMEOUT FROM ');
188.                                 IF TT=IDLETIMER THEN BEGIN
189.                                   WRITE(SYSFILE,'IDLETIMER ')
190.                                 END ELSE BEGIN
191.                                   WRITE(SYSFILE,'OTHERTIMER ')
192.                                 END;
193.                               END;(*TIMEOUT*)
194.                        ACKRCV  :WRITE(SYSFILE,'ACKRCVD')
195.                END;(*CASE*)
196.          WRITELN(SYSFILE,' AT TIME ',EVENTTIME,' AT STA ',SID:3)
197.          END (*WITH*)
198.    END;(*PRNTEVENT*)
199.    PROCEDURE PRNTSTAT(VAR ST:STATSREC);
200.    VAR I:INTEGER;
201.    BEGIN
202.     WITH ST DO BEGIN
203.       WRITELN(SYSFILE);
204.       WRITELN(SYSFILE,'...STATISTICS...');
205.       WRITELN(SYSFILE);
206.       WRITELN(SYSFILE,'USEFULCHTIME=',XMITTIME,'TOTRSVTIME=',
207.                       TOTRSVTIME,'SYNCTIME=',SYNCTIME);
208.        WRITELN(SYSFILE,'TOT # OF ROY STATIONS= ',RDYSTAS);
209.        FOR I:=1 TO SIMSTATE.NOMOFSTA DO BEGIN
210.          WRITELN(SYSFILE,NOFMSGSNT(.I.),' MSGS SNT BY STA ',I);
211.          WRITELN(SYSFILE,RESPTIME(.I.),' SECS TOTRSPTIM AT STA ',I);
212.       WRITELN(SYSFILE,'QLEN AT STA ',I,'=',QLEN(.I.));
213.        END;(*FOR*)
214.     END (*WITH*)
215.    END; (*PRNTSTAT*)
216.    PROCEDURE PRNTSYSSTATE(VAR SS:STATEREC);
217.    BEGIN
218.          WRITELN(SYSFILE);
219.          WRITELN(SYSFILE,'...MORE SYSTEM STATES...');
220.          WITH SS DO BEGIN
221.       WRITELN(SYSFILE,'CRNTTIME=',CALENDAR@.RLINK@.EVENTTIME,'TLASTDEP=',
222.                                                              TLASTDEP)
223.          END (*WITH*)
224.    END;
225.    FUNCTION RANDINT(LOW,HIGH:INTEGER;VAR SEED:REAL):INTEGER;
226.    (* RETURNS A RANDOM INTEGER BETWEEN LOW AND HIGH INCLUSIVE *)
227.    CONST DEBUGRAND=FALSE ;
228.    VAR     RANREAL:REAL;
229.          RANG :INTEGER;
230.          ANSWER:INTEGER;
231.    BEGIN
232.          RANG :=HIGH-LOW+1;
233.          RANREAL:=RANDOM(SEED);
234.          RANREAL:=RANREAL*RANG;
235.          ANSWER:=TRUNC(RANREAL);
236.          ANSWER:=ANSWER+LOW;
237.          IF DEBUGRAND THEN BEGIN
238.                WRITELN('IN RANDOMINT,ANSWER<=',ANSWER,'>')
239.          END;
240.          RANDINT:=ANSWER
```

```
241.    END(*RANDINT*);
242.    FUNCTION RANDREAL(LO,HI:REAL;VAR SEED:REAL):REAL;
243.    (*RETURNS A REAL NUMBER BETWEEN LO AND HI *)
244.    BEGIN
245.          RANDREAL:=RANDOM(SEED)*(HI-LO)+LO
246.    END;(*RANDREAL*)
247.    PROCEDURE REPORTERROR(ERRORTYPE:INTEGER);
248.    VAR    I:STARANG;
249.    BEGIN
250.           CASE ERRORTYPE OF
251.              1:BEGIN
252.                   WRITELN(OUTPUT,'ERROR1. EMPTY CALENDAR ENCOUNTERED!!');
253.                END;
254.              2:BEGIN
255.              WRITELN(OUTPUT,'ERROR2.PKT XMITION MUST BE IN PKTSNT STATE');
256.                   PRNTSTASTATES(SIMSTATE)
257.                END;
258.              3:BEGIN
259.              WRITELN(OUTPUT,'ERROR3.RFR MUST BE RCVD IN RFRSNT/IDLE ST!');
260.                  PRNTSTASTATES(SIMSTATE)
261.                END;
262.              4:BEGIN
263.              WRITELN(OUTPUT,'ERROR4.TOO MANY LOOPS IN PROCEDURE SENDRFR');
264.                   PRNTSTASTATES(SIMSTATE)
265.               END
266.           END;(* CASE *)
267.           HALT
268.    END;
269.    PROCEDURE INSRTEVNT(VAR OLDP:EVENTPTR;VAR NEWP:EVENTPTR);
270.
271.    (*THIS PROCEDURE PLACES AN EVENT NEWP TO THE RIGHT OF EVNT OLDP*)
272.    (*AND RETURNS A POINTER TO THIS NEW EVENT                      *)
273.    BEGIN
274.           NEWP@.LLINK:=OLDP;              (*RESET LEFT AND RIGHT POINTERS*)
275.           NEWP@.RLINK:=OLDP@.RLINK;       (*FROM NEW EVNT PTR NEWP       *)
276.           OLDP@.RLINK@.LLINK:=NEWP;       (*RESET LEFT AND RIGHT POINTERS*)
277.           OLDP@.RLINK:=NEWP               (*TO NEW EVNT POINTD TO BY NEWP*)
278.    END;
279.    PROCEDURE REMOVEVNT(VAR HEAD:EVENTPTR;VAR PTR:EVENTPTR);
280.
281.    (*THIS PROCEDURE REMOVES EVNT PNTD TO BY PTR FROM THE EVENTLIST*)
282.    (*WHOSE TOP IS PNTED TO BY HEAD(OR CALENDAR)           *)
283.
284.    BEGIN
285.           IF PTR <> HEAD THEN BEGIN
286.                  PTR@.LLINK@.RLINK:=PTR@.RLINK;
287.                  PTR@.RLINK@.LLINK:=PTR@.LLINK
288.           END ELSE BEGIN
289.                REPORTERROR(1)   (*EMPTY CALENDAR*)
290.           END
291.    END;
292.
293.    PROCEDURE SCHEDULE(VAR CALENDAR:EVENTPTR;VAR NEWEVP:EVENTPTR);
294.
295.    (* THIS PROCEDURE PUTS NEW EVENTS ON THE CALENDAR chronologically*)
296.    VAR    DONE:BOOLEAN;
297.           OLDP:EVENTPTR;
298.           P:EVENTPTR;
299.    BEGIN
300.           P:=CALENDAR@.RLINK;     (*GET PTR TO 1ST ITEM ON EVNTLIST*)
```

```
301.            IF P<>CALENDAR THEN BEGIN       (*LIST NOT EMPTY!         *)
302.                    IF P@.EVENTTIME <= NEWEVP@.EVENTTIME THEN BEGIN
303.                            REPEAT          (* MUST SEARCH LIST        *)
304.                                    OLDP:=P;
305.                                    P:=P@.RLINK;
306.                              IF P<>CALENDAR THEN BEGIN (*INMIDDLE?*)
307.                                DONE:=(P@.EVENTTIME > NEWEVP@.EVENTTIME)
308.                              END ELSE BEGIN
309.                                  DONE:=TRUE
310.                              END (* IF*)
311.                            UNTIL DONE;     (*INSRT NEW ITEM ON RT OF OLDP*)
312.                            INSRTEVNT(OLDP,NEWEVP)
313.                    END ELSE BEGIN          (* IT BECOMES 1ST ITEM     *)
314.                            INSRTEVNT(CALENDAR,NEWEVP)
315.                    END (*IF*)
316.            END ELSE BEGIN          (*EVENTLIST(CALENDAR) IS EMPTY  *)
317.                    INSRTEVNT(CALENDAR,NEWEVP)
318.            END(* IF *)
319.    END;(*SCHEDULE*)
320.    FUNCTION BUILTEVP(EVTYPE:EVENTCLASS;PPTR:PACKETPTR;EVTIME:REAL;
321.                                            S:STARANG):EVENTPTR;
322.    VAR     EVENTP:EVENTPTR;
323.    BEGIN
324.            NEW(EVENTP);
325.            WITH EVENTP@ DO BEGIN
326.                    EVENTTYPE:=EVTYPE;
327.                    EVENTTIME:=EVTIME;
328.                    PKTPTR:=PPTR;
329.                    SID:=S
330.            END; (*WITH*)
331.            BUILTEVP:=EVENTP
332.    END;(*BUILTEVP*)
333.    FUNCTION QUEUEEMPTY(Q:QUEUE):BOOLEAN;
334.    BEGIN
335.            QUEUEEMPTY:=(Q.HEAD=NIL)
336.    END;
337.    PROCEDURE ENQUEUE(VAR P:PACKETPTR;VAR Q:QUEUE);
338.    BEGIN
339.            IF Q.TAIL <> NIL THEN BEGIN
340.                    Q.TAIL@.NEXT:=P;
341.                    P@.NEXT:=NIL;
342.                    Q.TAIL:=P
343.            END ELSE BEGIN
344.                    Q.HEAD:=P;
345.                    Q.TAIL:=P;
346.                    P@.NEXT:=NIL
347.            END(*IF*)
348.    END;(*ENQUEUE*)
349.    PROCEDURE DEQUEUE(VAR P:PACKETPTR;VAR Q:QUEUE);
350.    BEGIN
351.            IF NOT QUEUEEMPTY(Q) THEN BEGIN
352.                    P:=Q.HEAD;
353.                    IF P@.NEXT <> NIL THEN BEGIN
354.                            Q.HEAD:=P@.NEXT
355.                    END ELSE BEGIN
356.                            Q.HEAD:=NIL;
357.                            Q.TAIL:=NIL
358.                    END
359.            END
360.    END;(*DEQUEUE*)
```

```
361.    FUNCTION ALLQUSEMPTY(ARRAYQ:PRIORQUEUE;N:STARANG):BOOLEAN;
362.    VAR     I:INTEGER;
363.            STI:BOOLEAN;
364.    BEGIN
365.            I:=1;
366.            STI:=TRUE;
367.           LOOP UNTIL MAXSTA:
368.                    STI:=QUEUEEMPTY(ARRAYQ(.I.))AND STI;
369.                    I:=I+1;
370.                    IF (STI=FALSE) THEN BEGIN
371.                            EXIT
372.                    END;
373.             IF I > N THEN MAXSTA
374.            POSTLUDE
375.                    MAXSTA:ALLQUSEMPTY:=STI
376.            END;(*END OF LOOP STATEMENT      *)
377.            ALLQUSEMPTY:=STI
378.    END;
379.
380.    FUNCTION EXPON(RATE:REAL;VAR SEED:REAL):REAL;
381.    (* RETURNS A RANDOM VARIABLE OF AN EXPONENTIAL DISTRIBUTION GIVEN RATE*)
382.    BEGIN
383.            EXPON:= - (1/RATE)*LN(RANDOM(SEED))
384.    END;
385.    FUNCTION NXTARRIVAL(ARATE:REAL;VAR SEED:REAL):REAL;
386.    VAR     N:INTEGER;
387.    BEGIN
388.            NXTARRIVAL:=EXPON(ARATE,SEED)
389.    END;
390.    FUNCTION XMITLEN(DRATE:REAL;M:MSGDISTTYPE):REAL;
391.    VAR     N:INTEGER;
392.    BEGIN
393.            CASE M OF
394.                    EXPONENTIAL:XMITLEN:=EXPON(DRATE,DRANDSEED);
395.                          FIXED:XMITLEN:=DRATE
396.            END
397.    END;
398.    FUNCTION SUMINT(A:PRIOR;L:INTEGER):REAL;
399.    VAR     I:INTEGER;
400.            S:REAL;
401.    BEGIN
402.            S:=0.0;
403.            FOR I:=1 TO L DO BEGIN
404.                    S:=S+A(.I.)
405.            END;
406.            SUMINT:=S
407.    END;(*SUMINT*)
408.    FUNCTION SUMREAL(VAR A:ARANGE;L:INTEGER):REAL;
409.    VAR I:INTEGER;
410.        S:REAL;
411.    BEGIN
412.      S:=0.0;
413.      FOR I:=1 TO L DO BEGIN
414.        S:=S+A(.I.)
415.      END;
416.      SUMREAL:=S;
417.    END;(*SUMREAL*)
418.    FUNCTION HIGHERPRIORRFR(MYADDR:INTEGER;OTHERADDR:INTEGER):BOOLEAN;
419.    BEGIN
420.            HIGHERPRIORRFR:=(MYADDR > OTHERADDR)
```

```
421.    END;
422.    FUNCTION LOWERPRIORRFR(MYADDR:INTEGER;OTHERADDR:INTEGER):BOOLEAN;
423.    BEGIN
424.          LOWERPRIORRFR:=(MYADDR < OTHERADDR)
425.    END;
426.    FUNCTION MYRFR(MYADDR:INTEGER;OTHERADDR:INTEGER):BOOLEAN;
427.    BEGIN
428.          MYRFR:=(MYADDR = OTHERADDR)
429.    END;
430.    FUNCTION MYMSG(MYID:STARANG;OTHERID:STARANG):BOOLEAN;
431.    BEGIN
432.          MYMSG:=(MYID = OTHERID)
433.    END;
434.    FUNCTION MYACK(MYID:STARANG;OTHERID:STARANG):BOOLEAN;
435.    BEGIN
436.          MYACK:=(MYID = OTHERID)
437.    END;
438.    FUNCTION CREATERFR(SOURCE:STARANG):PACKETPTR;
439.    VAR    NEWP:PACKETPTR;
440.    BEGIN
441.          NEW(NEWP);
442.          NEWP@.ORIG:=SOURCE;
443.        NEWP@.PKTTYPE:=RFR;
444.          CREATERFR:=NEWP
445.    END;
446.    FUNCTION CREATEACK(SOURCE:STARANG;SINK:STARANG):PACKETPTR;
447.    VAR    NEWP:PACKETPTR;
448.    BEGIN
449.          NEW(NEWP);
450.          NEWP@.ORIG:=SOURCE;
451.          NEWP@.DEST:=SINK;
452.        NEWP@.PKTTYPE:=ACK;
453.          CREATEACK:=NEWP
454.    END;
455.    FUNCTION NEWMSGDEST(SOURCE:STARANG;MINID:STARANG;MAXID:STARANG):STARANG;
456.    VAR    J:STARANG;
457.    BEGIN
458.          REPEAT
459.                J:=RANDINT(MINID,MAXID,RANDSEED)
460.          UNTIL J <>  SOURCE;
461.          NEWMSGDEST:=J
462.    END;
463.    FUNCTION CREATEMSG(SOURCE:STARANG;ARRIVTIM:REAL;NOFSTA:STARANG):
464.                                                              PACKETPTR;
465.    VAR    NEWP:PACKETPTR;
466.    BEGIN
467.          NEW(NEWP);
468.          NEWP@.ORIG:=SOURCE;
469.          NEWP@.DEST:=NEWMSGDEST(SOURCE,1,NOFSTA);
470.        NEWP@.PKTTYPE:=MSG;
471.          NEWP@.TIMEOFARRIV:=ARRIVTIM;
472.          NEWP@.NEXT:=NIL;
473.          CREATEMSG:=NEWP
474.    END;
475.    FUNCTION SUMRDYSTA:INTEGER;
476.    VAR I,N:INTEGER;
477.    BEGIN
478.        N:=0;
479.        WITH SIMSTATE DO BEGIN
480.          FOR I:=1 TO NOMOFSTA DO BEGIN
```

```
481.            IF NOT QUEUEEMPTY(PQ(.I.)) THEN BEGIN
482.               N:=N+1;
483.            END;(*IF*)
484.         END;(*FOR*)
485.      END;
486.      SUMRDYSTA:=N;
487.   END;
488.   PROCEDURE FUTUREARRIVAL(VAR SIMSTATE:STATEREC);
489.
490.   (*THIS ROUTINE PREDICTS FUTURE ARRIVAL FOR CURR STA AND SCHEDULES IT*)
491.
492.   VAR    T:REAL;
493.          EVNTP:EVENTPTR;
494.   BEGIN
495.          WITH SIMSTATE DO BEGIN
496.                 T:=NXTARRIVAL(ARR(.STAID.),ARANDSEED);(*PREDNXTARRTIM*)
497.                 EVNTP:=BUILTEVP(ARRIVAL,NIL,CRNTTIME+T,STAID);
498.                 SCHEDULE(CALENDAR,EVNTP)
499.          END
500.   END;
501.   PROCEDURE START(TIMERID:TIMERTYPE;T:REAL;VAR SIMSTATE:STATEREC);
502.   VAR    EVNTP:EVENTPTR;
503.   BEGIN
504.          WITH SIMSTATE DO BEGIN
505.              EVNTP:=BUILTEVP(TIMEOUT,NIL,CRNTTIME+T,STAID);
506.              EVNTP@.TT:=TIMERID; (*INSRT TIMERTYPE IN T.O. EVNT NODE*)
507.              SCHEDULE(CALENDAR,EVNTP);(*PLACE ON EVNTLIST        *)
508.              WITH TIMERCOMP(.STAID.) DO BEGIN
509.                  TOEVPTR(.TIMERID.):=EVNTP;(*STORE PTR TO T.O.EVNT*)
510.                  TMRSTATE(.TIMERID.):=RUNNING (*CHNG TIMER STATE*)
511.              END (*WITH*)
512.          END (*WITH*)
513.   END;(*PROC*)
514.   PROCEDURE RST(TIMERID:TIMERTYPE;VAR SIMSTATE:STATEREC);
515.   BEGIN
516.          WITH SIMSTATE DO BEGIN
517.              WITH TIMERCOMP(.STAID.) DO BEGIN
518.                  REMOVEVNT(CALENDAR,TOEVPTR(.TIMERID.));(*RMOV T.O.EVNT*)
519.                  TMRSTATE(.TIMERID.):=IDLE;  (*RESET TIMER TO IDL STATE*)
520.              END
521.          END
522.   END;(*PROC*)
523.   PROCEDURE SEND(TYPEOFPKT:PACKETTYPE;PPTR:PACKETPTR;T:REAL;
524.                                          VAR SIMSTATE:STATEREC);
525.   VAR EVNTP:EVENTPTR;
526.       NXTSTAID:STARANG;
527.   BEGIN
528.          WITH SIMSTATE DO BEGIN
529.              NXTSTAID:=1+STAID MOD NOMOFSTA;(*FIND FWD ADJ STA ON RING*)
530.              CASE TYPEOFPKT OF
531.                  RFR:EVNTP:=BUILTEVP(RFRRCV,PPTR,CRNTTIME+T,NXTSTAID);
532.                  MSG:EVNTP:=BUILTEVP(MSGRCV,PPTR,CRNTTIME+T,NXTSTAID);
533.                  ACK:EVNTP:=BUILTEVP(ACKRCV,PPTR,CRNTTIME+T,NXTSTAID)
534.              END;(*CASE*)
535.              SCHEDULE(CALENDAR,EVNTP)(*PUT EVNT ON EVNTLIST*)
536.          END
537.   END;(*PROC SEND*)
538.   PROCEDURE MONITOR(VAR SS:STATEREC;VAR ST:STATSREC);
539.   BEGIN
540.          PRNTSTASTATES(SS);
```

```
541.            PRNTEVENT(SS.CALENDAR@.RLINK);(*PRINT IMMINENT EVNT*)
542.            PRNTSYSSTATE(SS);
543.            PRNTSTAT(ST)
544.        END;
545.        PROCEDURE SENDRFR(VAR SIMSTATE:STATEREC;S:STARANG;TIME:REAL);
546.        VAR     N:INTEGER;
547.                NEWP:PACKETPTR;
548.                J:STARANG;
549.                STATE:TYPESTASTATE;
550.                RFRTIME:REAL;
551.                CNT:INTEGER; (*COUNTER TO COUNT REPEAT LOOP *)
552.                ERR4:INTEGER;
553.        BEGIN
554.                RFRTIME:=TRFR+TIME;
555.                J:=S;
556.                N:=SIMSTATE.NOMOFSTA;
557.                CNT:=0;
558.                REPEAT
559.                        RFRTIME:=RFRTIME+SIMSTATE.PROPTIME(.J.);
560.                        J:=J MOD N + 1;
561.                        STATE:=SIMSTATE.STASTATE(.J.);
562.                        CNT:=CNT+1
563.                UNTIL(STATE <> MSGWAIT) OR (CNT > N);
564.                IF CNT > N    THEN BEGIN
565.                        REPORTERROR(4)
566.                END;
567.                NEWP:=CREATERFR(S);
568.                SEND(RFR,NEWP,RFRTIME,SIMSTATE)    (*NOTE! MAY BE WRONG!!!!*)
569.        END(*SENDRFR*);
570.        PROCEDURE SENDMSG(VAR P:PACKETPTR;VAR SIMSTATE:STATEREC);
571.        VAR     MSGTIME:REAL;    (*TEMP. LOC. FOR MSG XMISSION TIME *)
572.        BEGIN
573.                WITH SIMSTATE DO BEGIN
574.                    MSGTIME:=XMITLEN(MSGDIST(.STAID.),MDT);
575.               SEND(MSG,P,THDR+MSGTIME+STALAT+PROPTIME(.STAID.),SIMSTATE);
576.                END
577.        END;
578.        PROCEDURE ARRIVALMGR(VAR SIMSTATE:STATEREC;VAR STATS:STATSREC);
579.        VAR     NEWP:PACKETPTR;
580.        BEGIN
581.                FUTUREARRIVAL(SIMSTATE);    (*PRED FUTUR ARIV AT THIS STA*)
582.                WITH SIMSTATE DO BEGIN
583.                    NEWP:=CREATEMSG(STAID,CRNTTIME,NOMOFSTA);
584.                ENQUEUE(NEWP,PQ(.STAID.));(*PUTMSGINQUEUE*)
585.                STATS.QLEN(.STAID.):=STATS.QLEN(.STAID.)+1;
586.                IF STASTATE(.STAID.) = RFRWAIT THEN BEGIN
587.                        START(IDLETIMER,TOPER,SIMSTATE);    (*STRT IDLETIMER*)
588.                        STASTATE(.STAID.):=RFRPEND    (*CHNG STATE TO B.P.*)
589.                END (*IF*)
590.                END (*WITH*)
591.        END;(*ARRIVALEVENT*)
592.        PROCEDURE RFRRCVMGR(VAR P:PACKETPTR;VAR SIMSTATE:STATEREC;
593.                                VAR STATS:STATSREC);
594.
595.        (* A STAID HAS RECEIVED A RFR.IT HAS TO CHANGE ITS STATE BASED ON
596.           its relative priority with the station whose RFR it rcvd  *)
597.        CONST   DBGRFRRCV=FALSE; (*DEBUG ENABLER OF THIS MODULE*)
598.        VAR     NEWRFRP:PACKETPTR;
599.                ERR3:INTEGER;(*ERROR VARIABLE*)
600.        BEGIN
```

```
601.          WITH SIMSTATE DO BEGIN
602.          CASE STASTATE(.STAID.) OF
603.              RFRWAIT      :IF NOT MYRFR(PT(.STAID.),PT(.P@.ORIG.)) THEN
604.                           BEGIN    (*  RFR RECEIVED NOT MINE*)
605.                             IF DBGRFRRCV THEN BEGIN
606.                                WRITE('IN RFRRCVMGR SID= ',STAID:3);
607.                                WRITE('RFR SOURCE= ',P@.ORIG:3);
608.                                WRITE('PRIORITIES ARE',PT(.STAID.):3);
609.                                WRITELN('AND',PT(.P@.ORIG.):3);
610.                                HALT
611.                             END; (*IF*)
612.                               DISPOSE(P);   (*THROW AWAY THE RFR   *)
613.                            STASTATE(.STAID.):=MSGWAIT
614.                           END ELSE BEGIN
615.                             WRITELN(ERRFILE,'MY RFR RCVD IN RFRWAIT ST');
616.                             HALT;
617.                           END;
618.              RFRPEND      :IF NOT MYRFR(PT(.STAID.),PT(.P@.ORIG.)) THEN
619.                              BEGIN (*RFR RCVD NOT MINE*)
620.                                RST(IDLETIMER,SIMSTATE);
621.                                STASTATE(.STAID.):=MSGWAIT;
622.                              END ELSE BEGIN
623.                               WRITELN(ERRFILE,'MY RFRRCVD IN RFRPEND ST');
624.                               HALT;
625.                            END; (*IF*)
626.              RFRSNT       :IF MYRFR(PT(.STAID.),PT(.P@.ORIG.)) THEN BEGIN
627.                               DISPOSE(P);
628.                               RST(OTHERTIMER,SIMSTATE);
629.                               START(OTHERTIMER,TWAITMSG,SIMSTATE);
630.                               STASTATE(.STAID.):=MSGPEND
631.                  END ELSE IF HIGHERPRIORRFR(PT(.STAID.),PT(.P@.ORIG.))
632.                      THEN BEGIN
633.                          DISPOSE(P);
634.                          RST(OTHERTIMER,SIMSTATE);
635.                          STASTATE(.STAID.):=MSGWAIT
636.                  END ELSE BEGIN (*LOWER  PRIOR RFR(HIGHERADDR) RCVD  *)
637.                          P@.ORIG:=STAID;(*SEND NEW RFR*)
638.                    SEND(RFR,P,TRFR+STALAT+PROPTIME(.STAID.),SIMSTATE);
639.                          RST(OTHERTIMER,SIMSTATE);
640.                          START(OTHERTIMER,TOPER,SIMSTATE)
641.                  END;
642.              MSGWAIT   :SEND(RFR,P,STALAT+PROPTIME(.STAID.),SIMSTATE);
643.           OTHERWISE BEGIN
644.                          WRITELN(ERRFILE,'RFR RCVD IN WRONG STATE!!!!');
645.                          HALT;
646.                     END (*OTHERWISE*)
647.           END(*CASE*)
648.          END (*WITH*)
649.     END;(*RFRRCVEVENT*)
650.     PROCEDURE MSGRCVMGR(VAR P:PACKETPTR;VAR SIMSTATE:STATEREC;
651.                         VAR STATS:STATSREC);
652.     VAR    NEWACKP:PACKETPTR;
653.     BEGIN
654.          WITH SIMSTATE DO BEGIN
655.              IF STASTATE(.STAID.)=MSGWAIT THEN BEGIN
656.                  IF MYMSG(STAID,P@.DEST) THEN BEGIN
657.                     NEWACKP:=CREATEACK(STAID,P@.ORIG);(*CREATE AN ACK*)
658.            SEND(ACK,NEWACKP,THDR+TACK+STALAT+PROPTIME(.STAID.),SIMSTATE);
659.                     START(IDLETIMER,TOPER,SIMSTATE);
660.                    TARRIV:=P@.TIMEOFARRIV;(*STORE ARRIVTIME B4 DSCRD*)
```

```
661.                    DISPOSE(P)   (*SEND MSG TO HIGHER LAYER      *)
662.                END ELSE BEGIN   (* NOT MY MESSAGE               *)
663.                  SEND(MSG,P,STALAT+PROPTIME(.STAID.),SIMSTATE);
664.                   START(IDLETIMER,TOPER,SIMSTATE)
665.                END;
666.                STASTATE(.STAID.):=IDLEWAIT
667.            END ELSE BEGIN
668.               WRITELN(ERRFILE,'MSG MUST BE RCVD IN MSGWAIT ST.!!!');
669.               HALT;
670.            END
671.        END
672.    END;(*PROC MSGRCVEVNT*)
673.    PROCEDURE ACKRCVMGR(VAR P:PACKETPTR;VAR SIMSTATE:STATEREC;
674.                        VAR STATS:STATSREC);
675.    BEGIN
676.        WITH SIMSTATE DO BEGIN
677.            CASE STASTATE(.STAID.) OF
678.                MSGWAIT :IF NOT MYACK(STAID,P@.DEST) THEN BEGIN
679.                           SEND(ACK,P,STALAT+PROPTIME(.STAID.),SIMSTATE);
680.                             START(IDLETIMER,TOPER,SIMSTATE);
681.                             STASTATE(.STAID.):=IDLEWAIT;
682.                          END ELSE BEGIN
683.                           WRITELN(ERRFILE,'CANT RCV MYACK IN M.W.');
684.                           HALT;
685.                          END;
686.                MSGSNT  :IF MYACK(STAID,P@.DEST) THEN BEGIN
687.                             START(IDLETIMER,TOPER,SIMSTATE);
688.    (*CHNG CHAN STATE TO CHIDLE      *)
689.                           IF CHSTATE = XMIT THEN BEGIN
690.                              CHSTATE:=CHIDL
691.                           END; (* IF *)
692.                            WITH STATS DO BEGIN
693.                             XMITTIME:=XMITTIME+CRNTTIME-TBEGXMIT;
694.             NOFMSGSNT(.STAID.):=NOFMSGSNT(.STAID.)+1;
695.             RESPTIME(.STAID.):=RESPTIME(.STAID.)+CRNTTIME-TARRIV;
696.                            END; (* WITH *)
697.                           STASTATE(.STAID.):=IDLEWAIT
698.                          END ELSE BEGIN
699.                           WRITELN(ERRFILE,'OTHRACK RCVD IN M.S. ST');
700.                           HALT;
701.                          END;
702.                OTHERWISE BEGIN
703.                        WRITELN(ERRFILE,'ACK RCVD IN WRONG STATE!!!');
704.                        HALT;
705.                          END (*OTHERWISE*)
706.            END (*CASE*)
707.        END     (*WITH*)
708.    END;(*PROC ACKRCVEVNT*)
709.    PROCEDURE TIMEOUTMGR(TIMRNAME:TIMERTYPE;VAR SIMSTATE:STATEREC;
710.                         VAR STATS:STATSREC);
711.    VAR    RFRP,MSGP:PACKETPTR;
712.    BEGIN
713.        WITH SIMSTATE DO BEGIN
714.            CASE STASTATE(.STAID.) OF
715.                IDLEWAIT:IF TIMRNAME=IDLETIMER THEN BEGIN
716.                           IF QUEUEEMPTY(PQ(.STAID.)) THEN BEGIN
717.                               STASTATE(.STAID.):=RFRWAIT
718.                           END ELSE BEGIN      (*Q. NOTEMPTY   *)
719.                               START(IDLETIMER,TOPER,SIMSTATE);
720.                               STASTATE(.STAID.):=RFRPEND
```

```
                                END (*IF*)
                             END ELSE BEGIN
                                WRITELN(ERRFILE,'OTHR_T T.O. NOT VALID');
                                HALT;
                             END;
                    RFRPEND :IF TIMRNAME=IDLETIMER THEN BEGIN
                                IF NOT QUEUEEMPTY(PQ(.STAID.)) THEN BEGIN
                                    RFRP:=CREATERFR(STAID);
                               SEND(RFR,RFRP,TRFR+STALAT+PROPTIME(.STAID.),
                                                              SIMSTATE);
          (*CHK IF CHAN STATE BE CHANGED TO RSV. STORE RSV TIME IF SO*)
                                    IF CHSTATE = CHIDL THEN BEGIN
                                        CHSTATE:=RSV;
                                      STATS.RDYSTAS:=STATS.RDYSTAS+SUMRDYSTA;
                                        TBEGRSV:=CRNTTIME
                                    END; (* IF *)
                                    START(OTHERTIMER,TOPER,SIMSTATE);
                                    STASTATE(.STAID.):=RFRSNT
                                END ELSE BEGIN
                                    WRITELN(ERRFILE,'QLEN=O NOT VALID!!');
                                    HALT;
                                END
                             END ELSE BEGIN
                                WRITELN(ERRFILE,'OTHR_T T.O.INVALID IN RP');
                                HALT;
                             END;
                    RFRSNT  :IF TIMRNAME=OTHERTIMER THEN BEGIN
                                RFRP:=CREATERFR(STAID);
                      SEND(RFR,RFRP,TRFR+STALAT+PROPTIME(.STAID.),SIMSTATE);
                                START(OTHERTIMER,TOPER,SIMSTATE)
                             END ELSE BEGIN
                                WRITELN(ERRFILE,'IDL_T T.O.INVALD IN R.S.');
                                HALT;
                             END;
                    MSGPEND :IF TIMRNAME=OTHERTIMER THEN BEGIN
                                DEQUEUE(MSGP,PQ(.STAID.));
                                SENDMSG(MSGP,SIMSTATE);
          (* BEGIN TAKING STATISTICS *)

                           STATS.QLEN(.STAID.):=STATS.QLEN(.STAID.)-1;
       (*CHANGE STATE OF CHANNEL TO XMIT STATE. STORE TIME THIS BEGINS*)
                                IF CHSTATE = RSV THEN BEGIN
                                    CHSTATE:=XMIT;
                                    TBEGXMIT:=CRNTTIME
                                END; (* IF *)
                             WITH STATS DO BEGIN
                              TOTRSVTIME:=TOTRSVTIME+CRNTTIME-TBEGRSV;
                             SYNCTIME:=SYNCTIME+2*TOPER
                              END; (* WITH *)
                                STASTATE(.STAID.):=MSGSNT
                             END ELSE BEGIN
                                WRITELN(ERRFILE,'IDL_T T.O.INVALD IN M.P.');
                                 HALT;
                             END;
                OTHERWISE BEGIN
                             WRITELN(ERRFILE,'TIMEOUT IN WRONG STATE!!!!!');
                             HALT;
                          END (*OTHERWISE*)
                END (*CASE*)
            END (*WITH*)
```

```
781.    END;(*TIMEOUTEVENT*)
782.    PROCEDURE DATALINKMGR(VAR P:EVENTPTR;VAR SIMSTATE:STATEREC;
783.                          VAR STATS:STATSREC);
784.    VAR     PKTP:PACKETPTR;
785.            WHICHTIMER:TIMERTYPE;
786.            EVTYPE:EVENTCLASS;
787.    BEGIN
788.            PKTP:=P@.PKTPTR;          (*GET PERTINENT EVNT PARAMETERS*)
789.            EVTYPE:=P@.EVENTTYPE;
790.            CASE EVTYPE OF
791.                ARRIVAL:ARRIVALMGR(SIMSTATE,STATS);
792.                RFRRCV :RFRRCVMGR(PKTP,SIMSTATE,STATS);
793.                MSGRCV :MSGRCVMGR(PKTP,SIMSTATE,STATS);
794.                ACKRCV :ACKRCVMGR(PKTP,SIMSTATE,STATS);
795.                TIMEOUT:TIMEOUTMGR(P@.TT,SIMSTATE,STATS)
796.            END
797.    END;(*PROC DATALINKMGR*)
798.    PROCEDURE GETTOPEVENT(VAR CALENDAR:EVENTPTR;VAR EP:EVENTPTR);
799.    BEGIN
800.         EP:=CALENDAR@.RLINK;  (*GET PTR TO TOP EVNT ON LIST  *)
801.         REMOVEVNT(CALENDAR,EP) (*REMOVE THIS EVENT          *)
802.    END;
803.    PROCEDURE PRNTFUTEVNT(VAR CALENDAR:EVENTPTR;N:INTEGER);
804.    VAR I:INTEGER;
805.        PTR:EVENTPTR;
806.    BEGIN
807.        WRITELN(SYSFILE,'**THESE ARE FUTURE EVENTS**');
808.        FOR I:=1 TO N DO BEGIN
809.             GETTOPEVENT(CALENDAR,PTR);
810.             PRNTEVENT(PTR)
811.       END (*FOR*)
812.    END;(*PRNTFUTEVNT*)
813.    PROCEDURE INITIALIZE(VAR SIMSTATE:STATEREC;VAR STATS:STATSREC);
814.    VAR     I:INTEGER;
815.            T:REAL;
816.            EVNTP:EVENTPTR;
817.    BEGIN
818.         NEW(EVNTP);
819.         WITH SIMSTATE DO BEGIN
820.           CALENDAR:=EVNTP;
821.           CALENDAR@.RLINK:=CALENDAR;
822.           CALENDAR@.LLINK:=CALENDAR;
823.           CRNTTIME:=0.0;
824.           TLASTDEP:=0.0;
825.           IF NOT REINIT THEN BEGIN
826.           FOR I:=1 TO NOMOFSTA DO BEGIN
827.                   PQ(.I.).HEAD:=NIL;
828.                   PQ(.I.).TAIL:=NIL;
829.                   STASTATE(.I.):=IDLEWAIT;
830.                 STAID:=I;
831.           FUTUREARRIVAL(SIMSTATE);
832.           START(IDLETIMER,TOPER,SIMSTATE);
833.           WITH TIMERCOMP(.I.) DO BEGIN
834.             TOEVPTR(.OTHERTIMER.):=NIL;
835.             TMRSTATE(.OTHERTIMER.):=IDLE
836.           END;  (*WITH*)
837.             IF DEBUG THEN BEGIN
838.             END (*IF*)
839.           END;(*FOR*)
840.          CHSTATE:=CHIDL; (*SET CHANNEL STATE TO IDLE=CHIDL*)
```

```
841.            END (*IF NOT REINIT*)
842.          END;(*WITH*)
843.          WITH STATS DO BEGIN
844.                FOR I:=1 TO SIMSTATE.NOMOFSTA DO BEGIN
845.                   NOFMSGSNT(.I.):=0;
846.                   QLEN(.I.):=0;
847.                   RESPTIME(.I.):=0.0
848.                END(*FOR*);
849.                   XMITTIME:=0.0;
850.                   SYNCTIME:=0.0;
851.                   TOTRSVTIME:=0.0;
852.                   RDYSTAS:=0    (*INITIALIZE # RDY STATIONS*)
853.          END(*WITH*)
854.     END;(*INITIALIZE*)
855.     FUNCTION AVGRSVTIM:REAL;
856.     VAR TEMPSUM:REAL;
857.     BEGIN
858.        WITH STATS DO BEGIN
859.          TEMPSUM:=SUMINT(NOFMSGSNT,SIMSTATE.NOMOFSTA);
860.          AVGRSVTIM:=TOTRSVTIM/TEMPSUM;
861.        END;
862.     END;(*AVGRSVTIM*)
863.     PROCEDURE OUTPUTSTATS(STATS:STATSREC;NOMSTA:INTEGER;DURATION:REAL);
864.     VAR     N:INTEGER;
865.             AMD:RATES;
866.              TEMP:REAL;
867.     BEGIN
868.             WRITELN(DCPROUT,NOMSTA:8,'STATIONS WERE ON THE RING');
869.             WRITELN(DCPROUT,DURATION:10:2,'SECONDS SIMULATION TIME');
870.             WITH STATS DO BEGIN
871.                     FOR N:=1 TO NOMSTA  do begin
872.                             IF NOFMSGSNT(.N.)=0 THEN BEGIN
873.                                     AMD(.N.):=INFINITY
874.                             END ELSE BEGIN
875.                                     AMD(.N.):=RESPTIME(.N.)/NOFMSGSNT(.N.)
876.                             END;
877.                             IF(N MOD 5 = 0)THEN BEGIN
878.                             WRITELN(DCPROUT);
879.                             WRITELN
880.                             END;(*IF*)
881.                             WRITE(DCPROUT,'D',N:3,'=',AMD(.N.):10:2,',')
882.                     END;(*FOR*)
883.
884.                     FOR N:=1 TO NOMSTA  do begin
885.                             IF(N MOD 5 = 0) THEN BEGIN
886.                                     WRITELN(DCPROUT);
887.                                     WRITELN
888.                             END;
889.                             WRITE(DCPROUT,'Q',N:3,'=',QLEN(.N.):10,',')
890.                     END;(*FOR*)
891.                     WRITELN(DCPROUT);
892.                IF DURATION=0 THEN BEGIN
893.                  WRITELN(DCPROUT,'SIMULATION TIME IS ZERO!!!')
894.                END ELSE BEGIN
895.                     WRITELN(DCPROUT,'CHANNEL UTILIZATION=',(XMITTIME+
896.                                TOTRSVTIME)/DURATION:10:5);
897.                END; (* IF*)
898.              IF (XMITTIME=0) AND (TOTRSVTIME=0) THEN BEGIN
899.            WRITELN(DCPROUT,'NO XMISSION AND NO RESERVATION!!!')
900.          END ELSE BEGIN
```

```
                WRITELN(DCPROUT,'PROTOCOL EFFICIENCY=',XMITTIME/
                      (XMITTIME +  TOTRSVTIME+SYNCTIME));
    END; (*IF*)
                 TEMP:=SUMINT(NOFMSGSNT,NOMSTA);
                WRITELN(DCPROUT,'FRACTION OF MSG SNT=',
              SUMINT(NOFMSGSNT,NOMSTA)/(SUMINT(NOFMSGSNT,NOMSTA)+
                                        SUMINT(QLEN,NOMSTA))
                                      :8:2);
                 WRITELN(DCPROUT,'AVG RSVTIM=',AVGRSVTIM);
                 WRITELN(DCPRQUT,'AVGSYNCTIM BY SIM=',
                              SYNCTIME/SUMINT(NOFMSGSNT,NOMSTA));
               WRITELN(DCPROUT,'AVG # RDY STATIONS= ',RDYSTAS/TEMP);
    WRITELN(PLTFILE3,MSGLEN,RDYSTAS/TEMP);(*PUT ON PLTFILE3*)
       END;(*WITH*)
       PRNTSTAT(STATS);(*PRINT MORE STATISTICS ONTO SYSFILE*)
END;(*OUTPUTSTATS*)
FUNCTION LENTOTIME(LEN:INTEGER;PDLY:REAL):REAL;
BEGIN
       LENTOTIME:=PDLY + LEN/CHCAP
END;
PROCEDURE READSYSPARS(VAR SS:STATEREC);
VAR    I:INTEGER;
       CH:CHAR;        (*TEMPORARY LOC TO STORE MSGDISTYP FROM TERM*)
BEGIN
       WITH SS DO BEGIN
              IF INTERACT THEN BEGIN
           WRITELN(OUTPUT,'ENTER # OF STAIDS AND SIMULATION TIME');
              READLN(INPUT,NOMOFSTA,SIMTIME);
            WRITELN(OUTPUT,'ENTER TYPE OF MSG DISTR. (F)IXED OR(E)XP');
              READLN(INPUT,CH);
              END ELSE BEGIN  (*INPUT DATA COMING FROM FILE DCPRIN*)
                     READLN(DCPRIN,NOMOFSTA,SIMTIME);
            WRITELN(DCPROUT,'# OF STATIONS= ',NOMOFSTA);
            IF SIMTIME=0 THEN BEGIN
              WRITELN(DCPROUT,'SIMULATION TIME NOT USR-GIVEN');
            END ELSE BEGIN
              WRITELN(DCPROUT,'SIMULATION TIME= ',SIMTIME);
             END;(*IF*)
                     READLN(DCPRIN,CH);
                     READLN(DCPRIN,STADLY);(*READ DLY AT STATION*)
                  STALAT:=STADLY/CHCAP;
              LINKDLY:=RINGLEN*RINGDLY/NOMOFSTA;(*PGDLY BTWN ADJ ST*)
           WRITELN(DCPROUT,'STATION DELAY= ',STADLY:3,' BIT(S)')
              END;(*IF INTERACT *)
              IF CH='E' THEN BEGIN
             WRITELN(DCPROUT,'MSG LENGTH DIST IS EXPONENTIAL');
                     MDT:=EXPONENTIAL
              END ELSE IF CH='F' THEN BEGIN
             WRITELN(DCPROUT,'MSG LENGTH DIST IS FIXED');
                     MDT:=FIXED
              END;
            IF INTERACT THEN BEGIN (*INTERACTIVE DATA INPUT?*)
              WRITELN(OUTPUT,'ENTER STA PRIOR INDICES,');
              WRITELN(OUTPUT,'THEN ARRIVAL RATE OF PACKETS,');
              WRITELN(OUTPUT,'THEN AVERAGE PACKET LENGTH IN BITS');
            END; (*IF INTERACT *)
       WRITE(DCPROUT,'STAID':15,'PRIORITY':15,'MSGARRIVRATE':15);
           WRITELN(DCPROUT,'AVGMSGLENGTH':15);
       WRITELN(DCPROUT);
              FOR I:=1 TO NOMOFSTA DO BEGIN
```

```
961.                        IF INTERACT THEN BEGIN
962.                            READ(INPUT,PT(.I.),N_THRPUT,MSGLEN)
963.                         END ELSE BEGIN
964.                            READLN(DCPRIN,PT(.I.),N_THRPUT,MSGLEN)
965.                         END;(*IF  INTERACT *)
966.                      ARR(.I.):=N_THRPUT*CHCAP/(NOMOFSTA*MSGLEN);
967.                        MSGDIST(.I.):=LENTOTIME(MSGLEN,0.0);(*AVMSGXMIT*)
968.                             IF MDT=EXPONENTIAL THEN
969.                               MSGDIST(.I.):=1/MSGDIST(.I.);(*MSGSVCRATE*)
970.                             PROPTIME(.I.):=LINKDLY;(*PD BTWN STAS*)
971.              WRITELN(DCPROUT,I:15,PT(.I.):15,ARR(.I.):15,MSGLEN:15);
972.                        END (*FOR*)
973.           END (*WITH*)
974.     END;(*READSYSPARS*)
975.     PROCEDURE GETSTAPARS(PAR:PARTYPE);
976.
977.     (*THIS PROC OBTAINS ARRIVAL RATE,PRIORITY OR AVERAGE*)
978.     (*MESSAGE LENGTH OF EACH STATION  AS GIVEN BY PAR .  *)
979.
980.     VAR I:INTEGER;
981.     BEGIN
982.           WRITELN(DCPROUT);
983.            WRITELN(DCPROUT,'*MODEL RUN AGAIN WITH THESE NEW VARIABLES*');
984.           IF PAR = ARRATE THEN BEGIN
985.               READLN(DCPRIN,N_THRPUT);
986.             WRITELN(DCPROUT,'N.E.B.R.=',N_THRPUT);
987.            END;(*IF*)
988.           IF PAR = MSGD  THEN BEGIN
989.             READLN(DCPRIN,MSGLEN);
990.              WRITELN(DCPROUT,'AVERAGE MESSAGE LENGTH=',MSGLEN);
991.           END;(*IF*)
992.         WITH SIMSTATE DO BEGIN
993.              FOR I:=1 TO NOMOFSTA DO BEGIN
994.                CASE PAR OF
995.                   ARRATE:BEGIN
996.                            ARR(.I.):=N_THRPUT*CHCAP/(NOMOFSTA*MSGLEN);
997.                          END;
998.                   STAPRIOR:PT(.I.):=I;
999.                   MSGD:BEGIN
1000.                         MSGDIST(.I.):=LENTOTIME(MSGLEN,0.0);
1001.                         IF MDT = EXPONENTIAL THEN BEGIN
1002.                             MSGDIST(.I.):=1/MSGDIST(.I.)
1003.                         END (*IF*)
1004.                       END (*MSGD*)
1005.               END (*CASE*)
1006.           END (*FOR*)
1007.        END (*WITH*)
1008.    END; (*PROC GETSTAPARS*)
1009.    FUNCTION ENDSIM(VALUE:INTEGER):BOOLEAN;
1010.    BEGIN
1011.      CASE VALUE OF
1012.         1:ENDSIM:=(SIMSTATE.SIMTIME <= SIMSTATE.CRNTTIME);
1013.         2:ENDSIM:=(STATS.NOFMSGSNT(.1.) = 1000)
1014.      END (*CASE*)
1015.    END; (*ENDSIM*)
1016.
1017.    (* MAIN PROGRAM BEGINS HERE *)
1018.
1019.    BEGIN
1020.    RESET(DCPRIN);(*RESET(DCPRIN)*)
```

```
1021.            REWRITE(DCPROUT);
1022.            REWRITE(SYSFILE);
1023.            REWRITE(ERRFILE);
1024.            REWRITE(PLTFILE1);(*MAY BE DELETED IF PLOT NOT NEEDED*)
1025.            REWRITE(PLTFILE2);(*MAY BE DELETED IF PLOT NOT NEEDED*)
1026.            REWRITE(PLTFILE3);(*MAY BE DELETED IF PLOT NOT NEEDED*)
1027.           TOPER:=1.OE-3;  (*WORST CASE ROUND TRIP RFR DELAY *)
1028.           TWAITMSG:=O.O; (*TIME THAT MSG TRANSMISSION PENDS*)
1029.            TGAP:=1.OE-3;     (* GAP PULSE PERIOD IN SECS. *)
1030.            CHCAP:=1.OOE+6;  (* CHAN CAP IN BITS PER SEC *)
1031.            TRFR:=LRFR/CHCAP;        (* RFR XMIT TIME *)
1032.            TACK:=LACK/CHCAP;        (* ACK XMIT TIME *)
1033.            THDR:=LHDR/CHCAP;        (* HDR XMIT TIME *)
1034.                   READSYSPARS(SIMSTATE);
1035.                WRITELN(OUTPUT,'THIS IS THE DCPR PERFORMANCE SIMULATOR');
1036.                   WRITELN(OUTPUT);
1037.            REPEAT
1038.                RANDSEED:=123457.O;
1039.                   ARANDSEED:=36548.O;
1040.                   DRANDSEED:=7453219.O;
1041.                   REINIT:=FALSE;  (* ENABLE FIRST INITIALIZATION*)
1042.                   INITIALIZE(SIMSTATE,STATS);
1043.                   WHILE NOT ENDSIM(2) DO BEGIN
1044.                IF DEBUG THEN BEGIN  (*WE'RE DEBUGGING WITH MONITOR*)
1045.                   MONITOR(SIMSTATE,STATS);
1046.                END;(*IF DEBUG*)
1047.                            GETTOPEVENT(SIMSTATE.CALENDAR,P);
1048.                            SIMSTATE.CRNTTIME:=P@.EVENTTIME;
1049.                            SIMSTATE.STAID:=P@.SID;
1050.                            DATALINKMGR(P,SIMSTATE,STATS);
1051.                            DISPOSE(P)
1052.                   END;(*WHILE*)
1053.                OUTPUTSTATS(STATS,SIMSTATE.NOMOFSTA,SIMSTATE.CRNTTIME);
1054.              WITH STATS DO BEGIN
1055.          WRITELN(PLTFILE1,MSGLEN,RESPTIME(.1.)*CHCAP/
1056.                         (MSGLEN*NOFMSGSNT(.1.)));(*DATA FOR PLOTTING*)
1057.          WRITELN(PLTFILE2,MSGLEN,AVGRSVTIM+2*TGAP);(*DATA FOR PLT*)
1058.               END;(*WITH*)
1059.                   IF INTERACT THEN BEGIN
1060.                      WRITELN('WISH DO RUN AGAIN FOR OTHER VALUES?Y OR N');
1061.                      READLN(INPUT,RESPONSE)
1062.                   END ELSE BEGIN (*INPUT COMING FROM FILE CALLED DCPRIN*)
1063.                      READLN(DCPRIN,RESPONSE)
1064.                   END;(*IF INTERACTIVE MODE*)
1065.                      DONE:=(RESPONSE='N');
1066.             IF RESPONSE <> 'N' THEN BEGIN
1067.                 GETSTAPARS(MSGD);(*RUN AGAIN FOR DIFF MSG LENGTH*)
1068.             END; (*IF*)
1069.              UNTIL DONE;
1070.              WRITELN(DCPROUT);
1071.              WRITELN(DCPROUT,'CPU VIRTUAL TIME=',CLOCK,'MICROSECONDS');
1072.              WRITELN(OUTPUT);
1073.              WRITELN(OUTPUT,'**YOU HAVE EXITED FROM THE DCPR SIMULATOR**')
1074.     END.
1075.
```