A DYNAMIC PRIORITY PROTOCOL FOR REAL-TIME APPLICATIONS USING A TOKEN RING

by

Bakul G. Khanna

B.E. (Electronics and Communications), University of Roorkee, India, 1982

A THESIS SUBMITTED IN PARTIAL FULFILLMENT
OF THE REQUIREMENTS FOR THE DEGREE OF
MASTER OF SCIENCE
in the School
of
Computing Science

APPROVAL

Name:	Bakul G. Khanna
Degree:	Master of Science
Title of thesis:	A Dynamic Priority Protocol for Real-Time Applications Using a Token Ring

Examining Committee: Dr. Tiko Kameda
Chair

Dr. Stella Atkins
Senior Supervisor

Dr. Steve Hardy
Senior Supervisor

Dr. Ramesh Krishnamurti
Committee Member

Dr. Paul K. M. Ho
External Examiner

Date Approved: 23 Feb 89

Title of Thesis/Project/Extended Essay

A Dynamic Priority Protocol for Real-Time Applications using a Token Ring.

Author: ______________________

(signature)

Bakul Khanna

(name)

27 Feb 1989

(date)

Abstract

The IEEE 802.5 token ring protocol is not suited for real-time applications where messages have explicit deadlines and any message which does not meet its deadline is considered lost. This is because the stations are served in a round-robin manner and message deadlines are not taken into account.

This thesis proposes a dynamic priority protocol which exploits the priority mechanism specified in the IEEE 802.5 token ring protocol. This dynamic priority protocol breaks the round-robin mode of service and serves the messages in the order of their closeness to their deadlines. It aims at maximizing the number of messages which meet their deadlines. Its application to the IEEE 802.5 token ring protocol results in a favourable performance improvement over the conventional IEEE 802.5 token ring protocol. This improvement is achieved at the cost of an increase in the message waiting time (*i.e.*,the time between the message arrival and its start of service), which is not important as long as the message is served before its deadline.

This thesis also develops a simulation model for the new dynamic priority protocol and studies it for a variety of cases including different ring configurations and different traffic types.

Acknowledgements

I would like to thank my senior supervisors Stella Atkins and Steve Hardy for their guidance and their encouragement which inspired me and kept me going. I am also thankful to Ramesh Krishnamurti and Paul Ho for their helpful comments on the thesis. Most of this research was financed by Stella's research grants which is gratefully acknowledged.

Thanks are also due to Steve Cumming of the Systems Lab for his help with LATEX and to Chao Cheng of the Engineering Science Lab for his help with system related problems. I am also indebted to my friends and fellow graduate students, in particular Edward Lo for many enlightening discussions, Ranabir Gupta for his thoughtful comments on the thesis and Sanjeev Mahajan for his help with the analytical results. Finally, I would like to thank my husband for his constant support and encouragement throughout this endeavour.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

A conventional token ring protocol like the IEEE 802.5 protocol is not suited for some real-time applications (*i.e.*, systems where messages have explicit deadlines). This is so because the conventional token ring protocol serves the stations in a round-robin manner and does not take message deadlines into account. This results in fair service to all the stations but does not aim towards ensuring that the messages individually meet their deadlines. This thesis develops a dynamic priority protocol which exploits the priority mechanism of the IEEE 802.5 token ring protocol and makes it more suitable for such real-time applications.

This chapter provides some background into the field of Local Area Networks, introduces the subject of the thesis and explains the motivation behind it.

1.1 Local Area Networks

A *Local Area Network* (LAN) is typically defined as an interconnection of computing nodes via a communication network that covers a limited geographical area [Liu84]. LAN's are generally characterized in terms of their topologies; three of the common ones are star, ring and bus.

In the *star* topology, a central switching element is used to connect all the nodes in the network. A station wishing to transmit data sends a request to a central switch for a connection to some destination station, and the central element uses circuit switching to establish a path between the two stations as if they were connected by a dedicated point-to-point link. This topology has the advantage of simplicity but lacks robustness, because if the central switch fails, the entire network is disabled. A *Computerized Branch Exchange* (CBX) is usually implemented using a star topology [Sta84]. The CBX is a digital on-premise branch exchange designed to handle both voice and data connections. Data rates to individual end points are typically low but bandwidth is guaranteed and there is essentially no *network delay*, once a connection has been made. This mode of connection is called circuit switching and is in contrast with message switching where it is not necessary to set up a dedicated path between the stations. Rather, if a station needs to send a message, it appends the destination address to the message. This message is then passed through the network from node to node. At each node the entire message is received, stored briefly and transmitted to the next node. This delay at each station is the network delay. It is caused by waiting for all the bits of the message to arrive at a station and a queuing delay which is caused by waiting for the opportunity to transmit to the next node.

The *ring* topology consists of a closed loop, with each node attached to it by means of repeaters. Data circulates around the ring on a series of point-to-point data links between repeaters. A station wishing to transmit waits for its turn and then sends the data out onto the ring in the form of a packet, which contains both the source and destination address fields as well as the data. As the packet circulates, the destination node copies the data onto a local buffer. Since the ring is a closed loop, the packet will circulate infinitely unless it is removed. The packet may be removed by the destination station. Alternatively, the packet could be removed by the transmitting station after it has made one trip around the loop. The latter approach is more desirable because (1) it permits automatic acknowledgements (the destination station flips a bit in the control sequence) and (2) it permits multicast addressing: one packet sent simultaneously to multiple stations.

The *bus* topology is characterized by means of a multiple-access, broadcast medium. Since all devices share a common transmission medium, only one device can transmit at a time. As in the case of the ring topology, data is transmitted in the form of packets which contain the destination and the source address and the data. Each station monitors the media and copies packets addressed to itself.

The ring and the bus topologies are much more robust than the star but at the same time are also more complex. Various channel access protocols exist for the ring and bus topologies. The IEEE 802 is a family of standards for LAN's which deal with the physical and the data link layer of the *Open Systems Interconnection* (OSI) model. The data link layer is split into two parts, the upper layer is the *Logical Link Control* (LLC) layer and the lower layer is the *Medium Access Control* (MAC) layer. There are three standards at the MAC layer:

- IEEE 802.3 - This is the *Carrier Sensitive Multiple Access Protocol with Collision Detection* (CSMA/CD). It uses the bus topology. The station wishing to transmit listens to the medium to determine if another transmission is in progress. If the medium is in use, the station backs off some period of time and tries again. If the medium is idle the station may transmit. While transmitting the station continues to listen to the medium. If a collision is detected during transmission, the station ceases to transmit immediately and waits for a random amount of time before trying again.

- IEEE 802.4 - This is the token bus protocol. It uses a bus topology in which the stations on the bus form a logical ring (*i.e.*, the stations are assigned positions in an ordered sequence with the last member of the sequence following the first). Each station knows the identity of the station preceding it and following it. A control packet known as the *token* regulates the right of access. When a station receives the token, it is granted control of the medium for a specified time. This station may transmit one or more packets and may poll stations and receive responses. When it is done, it passes the token on to the next station in logical sequence. This station now has permission to transmit. Non-token stations are also allowed on the bus. These stations can only respond to polls or requests for acknowledgements.

- IEEE 802.5 - This is the token ring protocol. It uses the ring topology and is similar to the token bus protocol except that all the stations are connected on a unidirectional ring. As in the case of the token bus, the token regulates the right of access to the medium.

This thesis develops a dynamic priority scheme which is an extension of the token ring protocol and is more suitable for real-time applications. The token ring protocol was chosen for this purpose because it provides for multiple priority operation which the token bus and CSMA/CD do not. The token ring protocol will be described in detail in the following section.

1.2 The IEEE 802.5 Token Ring Protocol

The token ring protocol is the most popular of all ring protocols [Sta88]. The IEEE 802.5 token ring protocol is based on the use of a small token packet that circulates around the ring. Each station on the ring maintains a queue for messages to be transmitted to other stations via the data channel. When a station wishes to transmit a message, it waits until it detects a token passing by. At this point the station modifies the token from a free token to a busy token (also called *Start_of_Frame*) and transmits the queued messages immediately following the busy token. The busy token with the message appended to it is called a *frame*. After the message is transmitted and the busy token returns to the transmitting station, the transmitting station releases a new free token so that the next station downstream can access the channel. The use of a token guarantees that only one station may transmit at a time. Under lightly loaded conditions, there is some inefficiency since a station must wait for a token to come around before transmitting. However, under heavy loads, the ring functions in a round-robin manner, which is both efficient and fair.

The following subsections present the background necessary to understand the token ring protocol and then describe the operation of the protocol.

1.2.1 Background

The number of messages served (*i.e.*, transmitted) at each station depends on the service strategy. There are three common service strategies

- Exhaustive Service - The station queue is emptied whenever that station is served.
- Non-exhaustive Service - Only a prescribed number of messages are served at each station. A special case of this service discipline is in the situation where only one message is served at each station. This is called the Limited-to-one or Ordinary service discipline.
- Gated Service - Only those messages in the queue are served which were present at the time the token arrived at the station.

The amount of time a station may occupy the channel is controlled by a timer called the *Token Holding Timer* (THT). This timer plays a role in the exhaustive and gated service strategies to ensure that a station presenting an abnormal *traffic load* does not hold on to the token forever. The traffic load is the ratio of the message arrival rate on the ring to the message service rate.

Ring latency is the total delay encountered by a bit of data when it traverses around the ring. It is expressed in terms of bit time, where one bit time is the time one bit of data occupies on the ring. The ring latency is present due to the processing delays at each station (*i.e.*, station latency) and the propogation delay of the transmission medium. If the ring latency is shorter than the token length, then the station transmitting a token will start receiving the first few bits of the token before it has finished transmitting. This will not allow the token to circulate continuously because the received bits of the token will have to be buffered at the transmitting station until the station has finished transmitting. To avoid this condition, an artificial delay is introduced into the ring to make the ring latency greater than the length of the token. This function is performed by the latency buffer which is provided by one of the stations.

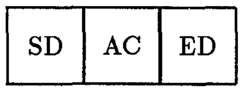
There are three ways in which a transmitting station can generate a new free token. They are the single packet, single token and multiple token operation.

In the *single packet* case, a transmitting station waits until all its transmitted bits have travelled around the ring and have been removed by the transmitting station. The station then releases a new free token. This is the most conservative approach because it ensures that there is only one transmission at any point in time. Very few implementations use this approach because it does not utilize the token efficiently; nevertheless it is the simplest to implement and analyze.

In the *single token* case, a transmitting station waits only until it has received its busy token back again. It then releases a new free token. If, however, the message transmission time is greater than the ring latency, then the transmitting station will receive the busy token before it has finished transmitting. In this case a new token is released right after transmission.

In the *multiple token* case, a station releases a token right after transmission. If the message transmission time is shorter than the ring latency, it is possible to have several busy tokens and one free token on the ring at the same time. If the message transmission time is longer than the ring latency, multiple token operation is the same as the single token operation.

TOKEN FORMAT FOR THE IEEE 802.5 PROTOCOL



SD = Starting Delimiter(1 octet)
AC = Access Control(1 octet)
ED = Ending Delimiter(1 octet)

Figure 1.1: Token Format

Based on the above definitions, the following subsection describes the operation of the IEEE 802.5 token ring protocol.

1.2.2 Operation

The token and frame formats for the IEEE 802.5 token ring protocol, including the bit length for each field, are shown in Figure 1.1 and Figure 1.2 respectively. The bit lengths are expressed in terms of *octets*, where one octet is eight bits.

The token format consists of three fields: The Starting Delimiter (SD), the Access Control (AC) field and the Ending Delimiter(ED). The SD is the means by which a station recognizes that the transmission on the ring is indeed a token or a frame. The AC field contains three priority bits, a token bit, a monitor bit and three reservation bits. The AC field format is shown in Figure 1.3. The three bit reservation and priority fields allow for eight distinct priority levels. The reservation field (R) is used by stations with high priority messages to indicate that the next free token be issued at the priority of the waiting message. The priority field (P) indicates the current priority of the token. Only those stations which have messages of priorities greater than or equal to the priority of the token are allowed to seize the token. The token subfield (T) indicates whether the token is busy or free. The monitor field (M) is modified only by the active monitor on the ring. The monitor is the ring function that is responsible for the recovery of error situations. There is only one monitor on the ring at a time which is referred to as the active monitor. When the IEEE 802.5 token ring protocol operates at a single priority, the priority and reservation fields are not used. The Ending Delimiter (ED) contains a special pattern for ED recognition, an intermediate frame bit and an error bit.

The frame format consists of three fields: the *Start_of_Frame_Sequence* (SFS), the *Frame_Check_Sequence* (FCS) and the *End_of_Frame_Sequence* (EFS). The SFS is made up of the SD and the AC fields which are the same as in the token format. However, when

FRAME FORMAT FOR THE IEEE 802.5 PROTOCOL

SFS		FCS					EFS	
SD	AC	FC	DA	SA	INFO	FCS	ED	FS

SFS = Start of Frame Sequence
FCS = Frame Check Sequence
EFS = End of Frame Sequence
SD = Starting Delimiter(1 octet)
AC = Access Control(1 octet)
FC = Frame Control(1 octet)
DA = Destination Address(2 or 6 octets)
SA = Source Address(2 or 6 octets)
INFO = Information(0 or more octets)
FCS = Frame Check Sequence(4 octets)
ED = Ending Delimiter(1 octet)
FS = Frame Status(1 octet)

Figure 1.2: Frame Format

ACCESS CONTROL FIELD FORMAT

P	P	P	T	M	R	R	R

P = Priority bit
T = Token bit
M = Monitor bit
R = Reservation bit

Figure 1.3: Access Control Field Format

these two octets are part of a frame, the token bit in the AC field is transmitted as a one. The FCS is composed of one *Frame_Control* (FC) octet, either two or six *Destination_Address* (DA) octets, two or six *Source_Address* (SA) octets, an information field of zero or more octets and a *Cyclic_Redundancy_Check* (CRC) field of four octets. The EFS is composed of an ED and a *Frame_Status* (FS) field. The ED is identical to the ED in the token format. The FS field enables the destination station to acknowledge the receipt of the packet to the source station.

The IEEE 802.5 token ring protocol also provides for multiple priority operation with a maximum of eight priority levels. The priority algorithm of the token ring protocol can be summarized as follows:
A station wishing to transmit a message must wait for a free token whose priority is less than or equal to the priority of the message to be transmitted. While waiting, a station may reserve a token at the priority level of the waiting message in the following manner:

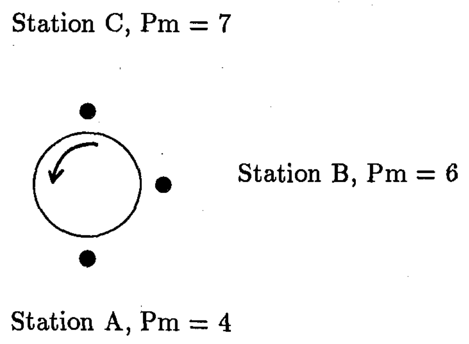
- If a busy token goes by, a station may set the reservation field in the token to the waiting message priority only if the reservation field contains a value less than the message priority.
- If a free token goes by, whose priority is greater than the waiting message priority, the station may set the reservation field in the token to the message priority if the reservation field contains a value less than the message priority.

When a station seizes a token, it marks the token as busy, sets the reservation field to the lowest priority level and leaves the priority field unchanged.

After a station has finished transmitting a message, it releases a new free token with the priority field in the token set to the maximum of the following values - the token priority, the reservation field value and the waiting messsage priority. The reservation field is set to the maximum of the reservation field value and the waiting message priority.

The station that upgraded the priority of the token is responsible for downgrading it to its former level, when all higher priority stations are finished. This is to assure that no token circulates indefinitely because its priority is too high. The priority is downgraded in the following manner:
When the station which upgraded the priority of the token to a certain level detects a free token at that higher priority, it can assume that there is no more higher priority traffic waiting and it downgrades the priority of the token before passing it on to the next station.

The net effect is round-robin within each of the eight priority levels. Starvation of lower priority messages can easily occur. Only the highest priority level provides guaranteed message delivery time. In the worst case, the first high priority waiting message will have to wait for almost two *frame times* plus one round trip of the token before it gains access to the token. This is shown in Figure 1.2.2 where Pm denotes the priority of the waiting message. A frame time is the time taken for a frame to traverse the ring. One frame time is needed due to the fact that the high priority message might arrive at a station C just when the header of a frame has gone past (*i.e.*, a reservation cannot be



made). The second frame time arises from the fact that after the above message has been served (by station A), the next station downstream *i.e.*, B may seize the token (a reservation is made when this frame passes station C which has a high priority waiting message). The token round trip comes from the fact that after the message on the ring returns to the transmitting station B, it removes the message from the the ring and releases a new free token. Depending on the relative positions of the stations, it can take one round trip for the free token to arrive at the requesting station.

1.3 Protocols for Real-Time Applications

Some real-time network applications involve message delivery with strict deadlines, where any message whose service is not started by its deadline is considered lost. This is based on the observation that the message contents of an excessively delayed message lose their value at the receiving end. Therefore the transmitting station may discard such a message when it overshoots its deadline.

In so-called *hard* real-time networks, the network cannot afford to lose any time-critical messages. For example, consider two communicating hard real-time tasks (executing on different nodes), where one is required to precede the other. In this case scheduling decisions concerning a task are affected by the completion time of the task that precedes it and by the delay in communication from that task. The main issue for such applications is whether a certain schedule is feasible or not [Kur84]. In such applications, traffic is classified as synchronous (time-critical) and asynchronous (not critical) traffic. The current practice to schedule such traffic on the IEEE 802.5 token ring is to use round robin scheduling with each station's THT set proportional to the time required to service its synchronous message set. The asynchronous messages are served when there is no synchronous message waiting [Str88].

In certain other, so-called *soft* real-time networks, a small amount of message loss is usually tolerated. An example of such an application is packetized voice where the loss of a small fraction of the packets is usually tolerable since the speech is still intelligible to the receiver. Other applications include distributed sensor networks and some real-time process control systems. Yet another example is that of impatient customers who leave their queues if their service is not started before a certain deadline or a system which has single buffers at each station and a message that arrives at a station before the previous one has been served, overwrites it. The primary aim of such an application running on a real-time LAN is to maximize the percentage of messages which meet their deadlines. This is a requirement quite different from that of a conventional LAN where the primary aim is to minimize the mean message waiting time. This fundamentally different objective suggests that conventional LAN protocols may not be suitable for real-time applications.

Several protocols exist for supporting hard real-time communication applications in ring networks. Message based priorities are used to distinguish between synchronous and asynchronous traffic by assigning synchronous messages a higher priority than the asynchronous messages [Rom81,Str88]. This breaks the round-robin mode of service by ensuring that all the higher priority messages are served before the lower priority messages. It results in round-robin mode of service within each priority level. This strategy may lead to starvation of low priority messages, which is not a problem since the major concern is to ensure that the synchronous messages meet their deadlines.

None of the above schemes is applicable to soft real-time applications where the traffic presented to the ring is of a single priority and the aim is to maximize the percentage of messages which meet their deadlines. A single priority token ring protocol may not be suitable for real-time applications because the stations on the ring are serviced in a round-robin manner and message deadlines are not taken into account. Due to the round-robin mode of service there is an inherent priority assignment imposed on the stations due to the topology of the ring.

A prioritized token ring protocol breaks the round-robin mode of service by ensuring that all high priority messages are served before the lower priority messages. It results in round-robin service within each priority level. The prioritized token ring can be used for hard real-time traffic by assigning higher priorities to time-critical messages and lower priorities to background messages.

The following section introduces the new dynamic priority protocol whose application to the IEEE 802.5 token ring protocol aims at maximizing the number of messages which meet their deadlines and hence makes it suitable for soft real-time applications.

1.4 The New Dynamic Priority Protocol

In this thesis, a dynamic priority protocol is proposed for a token ring network for soft real-time applications. It serves the messages in the order of the closeness to their deadlines. The deadline is assumed to be the deadline at the source station *i.e.*, the time before which the message service should have begun. A message which has overshot its deadline is discarded by the source station.

At any time the priority of a queued message is determined by the amount of time remaining until its deadline. Channel access rights are given to nodes dynamically, depending on the priority of the message at the head of queue at the node at that instant of time.

The new dynamic priority protocol is an extension of IEEE 802.5 token ring protocol. It is based on the principle that as an unserviced message approaches its deadline, its priority is raised as a function of time until it either gets served or it overshoots its deadline. If the deadline is overshot then the message is discarded at the source station.

The dynamic priority protocol overrides the inherent priorities imposed on the stations in a token ring LAN due to the interconnection topology. Round-robin scheduling takes effect only to serve messages which are equally critical. For example, if more than one message on the ring at different stations, at any instant of time, have the same time remaining until their deadlines, then those messages should be served in an order specified by the round-robin scheme.

On the principle that one cannot achieve *something for nothing*, it is clear that an improvement in the number of lost messages is achieved at the cost of some other measure, which in this case is the message *slack time*. The message slack time is the difference between the message's start of service and its deadline. The dynamic priority protocol results in the slack time being reduced which essentially means that on the average, more messages are served closer to their deadlines. This is not a cause for concern since a deadline exists and the message waiting time for a successfully transmitted message is always less than the deadline.

1.5 Issues to be Explored

The following issues related to the dynamic priority protocol will be explored in the thesis.

- The time function according to which the priority of a message is stepped up.
- The improvement of the dynamic priority protocol over the conventional IEEE 802.5 protocol in terms of the number of messages which meet their deadlines.
- The variation of the above improvement with change in number of stations, network load and message deadlines.
- The application of the dynamic priority protocol to different traffic types and distributions, for example: symmetric, asymmetric and avalanche traffic.
- The application of the dynamic priority protocol to hard real-time networks.

1.6 Composition of the Thesis

Chapter 2 deals with an overview of the field. In particular, it presents some schemes based on the token ring protocol for real-time applications and points out the drawbacks of these schemes. Chapter

3 explains the new dynamic priority protocol in detail. Chapter 4 discusses the implementation issues of the protocol including the simulation models used to measure its performance. Chapter 5 presents the results of the simulation runs. Chapter 6 summarizes the results and presents some conclusions based on the simulation results. It also provides some insight into topics for future work.

Chapter 2

Overview

This chapter gives an overview of some earlier protocols based on the IEEE 802.5 token ring protocol. It points out some of the drawbacks inherent with these schemes and shows how our new dynamic priority protocol helps in alleviating some of these problems.

2.1 Earlier Token Ring Protocols for Real-Time Systems

2.1.1 A Static Message Based Priority Scheme

Two message based priority schemes have been proposed in [Rom81] for a token ring network. Both these schemes are forerunners of the IEEE 802.5 token ring protocol and are based on the multiple token operation. These include a preemptive and a non-preemptive scheme. In the previous chapter it was shown that the IEEE 802.5 prioritized token ring protocol can be made suitable for real-time applications by assigning higher priorities to time-critical messages and lower priorities to non critical traffic. This priority assignment leads to the stations not being served in a round-robin manner since the high priority messages are served before the low priority messages. This scheme leads to the starvation of lower priority messages.

In the preemptive scheme, [Rom81] each station with a message to transmit monitors the channel. Each message transmitted is preceded by its priority. Assuming that each station can detect message boundaries, if a station with a high priority message detects a lower priority message going past, it intercepts the message and replaces the message with its own. This substitution implies that the intercepted message has been lost and will have to be retransmitted. This scheme has the potential to distort the transmission order of messages on the ring as will be seen in the following example. If we assume that a small number denotes a low priority message, then consider a sequence of messages going around the ring in the order of priorities 7, 5, 2, 1. If a station has a message of priority 6 to transmit, it waits for the first message in the sequence, i.e. a priority 7 message to go past. It then detects the second message in the sequence of priority 5. It intercepts this message and places its own message instead. So now the sequence of messages going around the ring have priorities in the order 7, 6, 2, 1. We see that a message of priority 5 has been preempted but lower priority messages (*i.e.*, 1 and 2) are allowed to continue on.

In the worst case, the first high priority waiting message may have to wait for one round trip of the token or one frame time to access the channel. The round trip of the token is needed in case there is no message being transmitted on the ring. In that case the station will need to wait for one round trip of the token (worst case) to gain access to the channel. The frame time is needed in case a *Start_of_Frame* has just passed the station. In that case the station will have to wait for the *Start_of_Frame* of the next message because it cannot preempt the current message since the field indicating the priority of the message may have already passed the station. It would definitely not want to preempt a higher priority message. This scheme could lead to starvation of low priority messages.

The non-preemptive scheme in [Rom81] alleviates some of the problems of the above preemptive scheme. In this scheme, at the end of a successful message transmission, ready subscribers commence a reservation period, during which the channel behaves as in the intercepting algorithm above. At the end of this period, exactly one subscriber from the highest.priority class is identified and transmits its message.

In the worst case two round trips of the token plus a frame delay will be required by a station with the highest priority message to access the channel. The frame time is due to the fact that a message may just have begun to get served when the high priority message arrives. After the current message has been served, there is a reservation cycle which is exactly one round trip of the token. The other round trip of the token is due to the distance the token has to travel to get to the station with the high priority message, which, in the worst case can be one round trip of the token. Again, as for any other scheme with static priorities, it could lead to starvation of lower priority messages.

2.1.2 An Adaptive Token Ring Strategy

An adaptive token ring strategy for a real time environment has been proposed in [Kim83]. It combines the advantages of an exhaustive and a non-exhaustive service scheme (refer to Section 1.2.1 for definitions of exhaustive and non-exhaustive service disciplines). If we define the ring scan time as the time taken for a station to receive the next usable token, it is clear that in a non-exhaustive scheme, the ring scan time is better than that in an exhaustive scheme.

An advantage of the exhaustive scheme over the non-exhaustive scheme is clear if we consider the following scenario. When the ring is poorly utilized, a station with a temporarily heavy traffic volume will not degrade the overall ring performance under the exhaustive policy because that station will simply empty out its queue when it seizes the token. The non-exhaustive ring will, however, not be able to adjust to this type of dynamic behavior of the ring operations and may lead to an extended waiting time for packets that cannot be transmitted during the current scan cycle.

The adaptive token ring strategy in [Kim83] involves dynamically adjusting the allowed *Token Holding Timer* (THT) by observing the ring activities at other stations (The THT ensures that a station presenting a heavy load to the ring does not hold on to the token forever). Thus this scheme incorporates the advantage of both the exhaustive and non-exhaustive ring, namely the lower scan time and fairness of a non-exhaustive ring and the lower average waiting time of an exhaustive ring.

This is an adaptive strategy and works well for a real-time system because it adjusts to the message load on each station by providing a dynamic means of assuring resource sharing in a distributed manner. It is, in principle, similar to setting up a global schedule so that under heavy traffic conditions it becomes a non-exhaustive scheme and every station gets a fair share of the bandwidth while under light traffic conditions it becomes an exhaustive scheme. This scheme is in essence similar to the FDDI token ring scheme [Ros86b,Ros86a,Sev87] where a global schedule is set up. In the FDDI protocol time-critical traffic is transmitted at any reception of a free token while non-critical traffic can only be transmitted if the token is running ahead of schedule.

The strategy in [Kim83] allows for selective removal of real-time traffic when congestion develops in a system, to meet the strict delay constraints of real-time messages. This selective removal is achieved in the following manner: after a station has transmitted its messages, (the number determined by the THT timer) the station immediately discards all the messages still waiting in the queue. Therefore messages waiting more than one token cycle time will be discarded.

The message based priority schemes in [Rom81] have, like all other static priority schemes, the disadvantage of starvation of lower priority messages. The adaptive scheme in [Kim83] is not a prioritized scheme but suffers from the drawback that the stations are served in a round-robin manner. Thus, messages at the current station being served may jeopardize the chances of transmission of messages at other stations which may be closer to their deadlines. The emphasis seems to be on ensuring fairness as against ensuring that a maximum number of messages meet their deadlines individually.

These drawbacks led to the idea of the dynamic priority scheme where the priority of a waiting message is stepped up over time. The priority of a message at any time is determined by the time remaining until its deadline, which essentially means that messages which are closer to their deadlines have priority over messages which are not as critical.

Chapter 3

The Dynamic Priority Protocol

The preceding chapters gave an overview of the token ring protocol, presented some schemes based on it and discussed the drawbacks of these schemes. The proposed dynamic priority protocol is based on the IEEE 802.5 token ring protocol and attempts to minimize the percentage of lost messages (*i.e.*, messages which do not meet their deadlines and are thus considered lost). The dynamic priority protocol is compared with the standard use of the IEEE 802.5 token ring protocol and is shown to perform better under most cases.

3.1 Dynamic Priority Protocol

The dynamic priority protocol serves messages in the order of their closeness to their deadlines rather than in the order imposed by the topology of the ring. This is in an attempt to maximize the number of messages which meet their deadlines. The deadline is assumed to be the deadline at the source station *i.e.*, the instant of time before which the message service should have begun.

There are many possible message service schedules depending on the various message parameters. For example, messages can be served in the order of message arrival times, message length, message lifetimes (*i.e.*, the time between the message generation time and its deadline) and message deadlines (*i.e.*, the amount of time remaining until the deadline). The service schedule chosen for this thesis is according to message deadlines. There are several reasons for this choice.

If the messages are served in the order of their arrival times, it will lead to *First Come First Serve* scheduling. Whatever the message deadline distribution, the messages will always be served in their order of arrival. Deadline will not have any effect on this scheduling policy. Thus, if a certain message is more critical than others – reflected by its closeness to its deadline, it will not be given any priority over the others.

Messages served in the order of message lengths implements the *Minimum Message Length First* scheduling policy. This thesis assumes constant message lengths which discards this choice as a suitable service schedule.

If the messages are served in the order of their lifetimes it would lead to the ***shortest lifetime first*** scheduling policy which may result in starvation of the longer lifetime messages. There may be applications where the lifetime of a message may be a direct reflection of its priority (*i.e.*, shorter lifetimes would lead to higher priorities). In those cases it may be required to serve the shorter lifetime messages first, even at the cost of starvation of the longer lifetime messages.

In the *message deadline service*, the messages are served in the order of their closeness to their deadlines. For example, a newly generated message with a short lifetime may be placed in the queue behind a message with a longer lifetime, but which has been in the queue for a longer time (*i.e.* it is closer to its deadline). This scheme does not lead to starvation of longer lifetime messages. When a message gets to the head of the queue, it is assigned a priority based on the time remaining until its deadline. This is a truly dynamic scheme because the longer a message waits in the queue, the closer it gets to its deadline and the higher its priority is stepped up.

The message deadline service schedule is found to be the most suitable for this thesis because the aim is to maximize the number of messages which meet their deadlines. This can be accomplished by serving those messages first which are closer to their deadlines rather than those which can wait for some more time, which is the message deadline scheduling policy.

Consider the scenario in Figure 3.1. Assume that a message arrives at the head of the transmit queue Δ time units or more before its deadline (Δ is a protocol parameter). This message is assigned the lowest priority (*i.e.*, one). Its priority stays the same until it reaches Δ time units before its deadline. From then on its priority increases as a function of Δ, its deadline (D) and the current time (t) until it gets served or its priority reaches the maximum level (*i.e.*, eight), since there are a maximum of eight priority levels allowed by the IEEE 802.5 token ring protocol. A message will overshoot its deadline if it stays unserviced at priority level eight.

In Figure 3.2, a message arrives at the head of the queue less than Δ time units before its deadline. It is thus assigned a priority based on the function *f*. As above, the priority is increased with time.

$$priority(t) = f(\Delta, D, t)$$

The parameter Δ and the function *f* are chosen such that only eight priority levels are allowed. A message will overshoot its deadline if it stays unserviced at priority level eight.

The number of priority levels available can affect the performance of the protocol in the following manner. A *small* number of priority levels could result in more than one message being assigned the same priority, though they would have been at different levels if more levels had been available [Leh86]. Since such ties are resolved by the round-robin scheme, it could lead to *priority inversion*, the situation in which a message which an optimal algorithm would assign the highest priority, does not get serviced first because it is assigned the same priority as another message less critical than

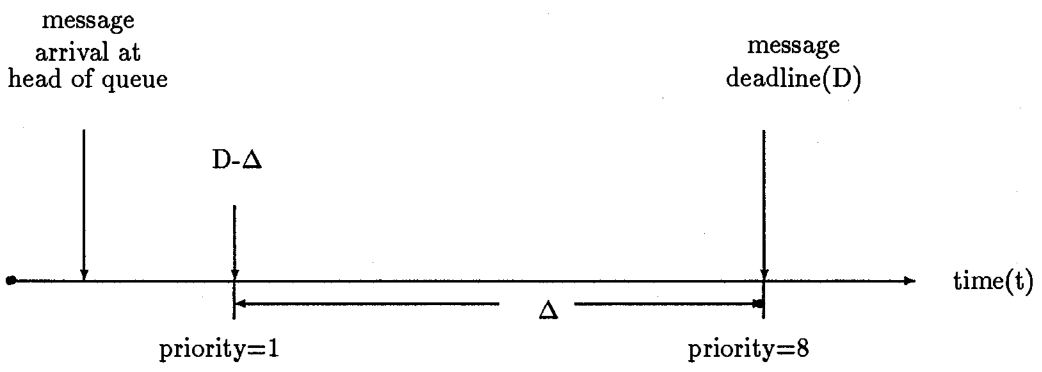


Figure 3.1: Scenario 1

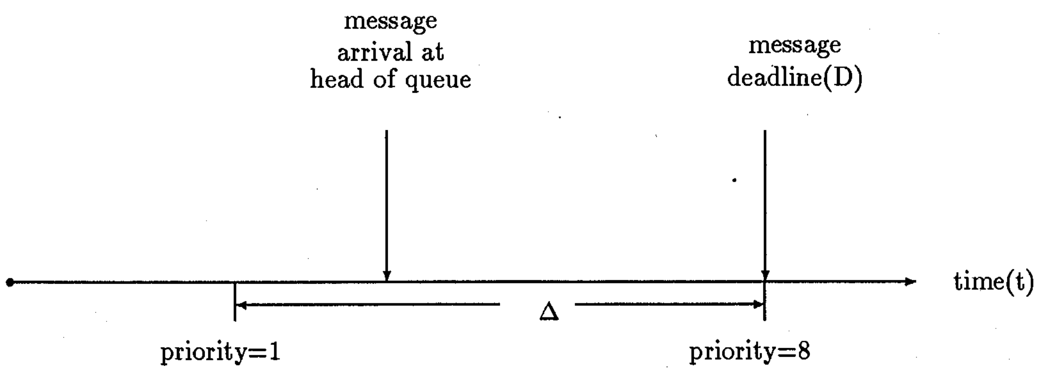


Figure 3.2: Scenario 2

itself. A very *large* number of priority levels will, on the other hand, result in more overhead for the token ring protocol. This overhead arises due to the downgrading of the priority of the token when there is no high priority traffic waiting. In the worst case this process of downgrading the priority of the token to the priority level it was upgraded from, can take one round trip of the token.

In the worst case one round trip of the token plus two frame times will be required by the first station with a high priority message to access the channel as in the IEEE 802.5 protocol.

In the case when $\Delta = 0$, the dynamic priority protocol will behave exactly like the IEEE 802.5 single priority token ring protocol with deadlines. This is because the priority of a waiting message starts to get stepped up 0 time units before its deadline, meaning thereby that the priority of the message is never stepped up, which results in all messages staying at the same priority (*i.e.*, the priority they were generated at). Henceforth, this IEEE 802.5 single priority token ring protocol with deadlines will be referred to as the fixed priority protocol and will be used as a basis for comparison with the dynamic priority protocol.

3.2 The Dynamic Priority protocol for Hard Real-Time Applications

The previous section described the dynamic priority protocol when applied to soft real-time traffic. Hard real-time traffic involves two classes of traffic: the synchronous (*i.e.*, time-critical) traffic and the aynchronous (*i.e.*, non-critical) traffic. Synchronous traffic has hard deadlines (*i.e.*, it should have guaranteed message delivery time). Asynchronous traffic, on the other hand is not time-critical and is associated with soft deadlines. The standard method for scheduling such traffic, as described in Section 1.3, results in the synchronous class messages being provided with guaranteed message delivery time and a possibility of the asynchronous class messages being starved. This strategy is used provided the data rate on the ring is sufficient to accomodate the synchronous traffic message arrival rate.

A *Deferred Server* (DS) scheme has been developed in [Str88] which guarantees synchronous class message delivery and improves the response of the asynchronous class of messages. It is based on the principle that since there is no advantage to the system for the synchronous class messages completing early, the DS algorithm assigns a priority higher than the synchronous traffic to the asynchronous traffic up until the point where the synchronous messages would start to be late. High priority asynchronous service is limited by a Deferrable Server which has a fixed capacity. When the server's capacity is exhausted by asynchronous message arrivals, additional arrivals are assigned background priority. This increases the responsiveness of the asynchronous class messages at the cost of reducing the average slack time for the synchronous messages.

The dynamic priority protocol is not useful when applied to the above real-time traffic. This is so because the dynamic priority protocol will assign priorities to both synchronous and asynchronous messages based on their closeness to their deadlines and will, in no way, favour the synchronous messages over the asynchronous messages. This will result in the asynchronous messages not being starved and the synchronous messages not being provided with guaranteed access.

The dynamic protocol can, however, be adapted to a hard real-time environment in the following manner. Synchronous class messages can be assigned the highest available priority (*i.e.*, eight), since the highest priority messages are provided with guaranteed access in a prioritized token ring scheme. Asynchronous class messages, at their time of generation, are assigned the lowest priority level and their priority is incremented according to a priority function until they reach a level one less than the level assigned to the synchronous set of messages or until they get served.

This scenario guarantees delivery of the synchronous class messages and increases the number of asynchronous class messages which meet their deadlines. This improvement in the number of asynchronous messages which meet their deadlines, is achieved at the cost of a decrease in the average asynchronous message slack time.

Thus, we see that the dynamic priority protocol can be used with both hard and soft real-time applications. The dynamic priority protocol is studied under a variety of different cases to observe its performance. These different cases include different ring configurations, different traffic types and different time functions. An attempt is made to arrive at a set of conditions under which it would be most advantageous to use the dynamic priority protocol.

3.3 Time Functions to Step up Priority

This thesis explores two functions, the linear and the non-linear functions. In the linear case the length Δ is divided into eight equal parts corresponding to the eight priority levels. This time function is shown in Figure 3.3. In the non-linear case the length Δ is divided into eight unequal parts. There can be many possible non-linear functions. An example of a non-linear function is shown in Figure 3.4.

3.4 Traffic Considerations

Real-time traffic is multifarious in its length, arrival process and service requirements [Bou87]. Alarm messages are very different in nature from bulk file transfers. Alarms produce short messages with a high degree of emergency. Bulk file transfers produce large quantities of messages with a low degree of emergency.

In real-time applications the external world is observed through a set of sensors. Most of them produce periodic data of fixed size. These sources are not synchronous and may have many different periods. Exceptionally, alarms are generated which correspond to abnormal physical events. Alarms are subject to produce message avalanches which look like a load peak.

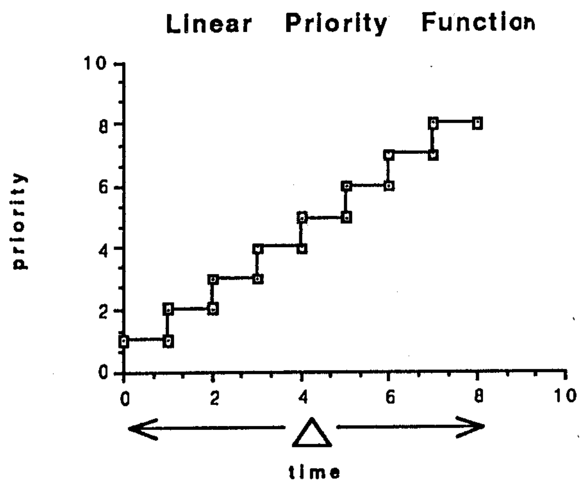


Figure 3.3: Linear priority function

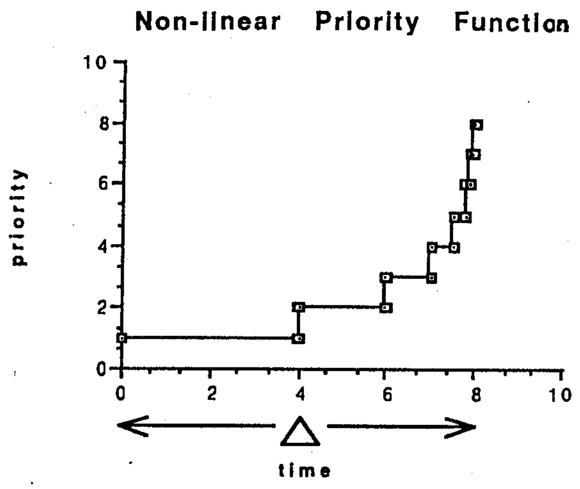


Figure 3.4: Non-linear priority function

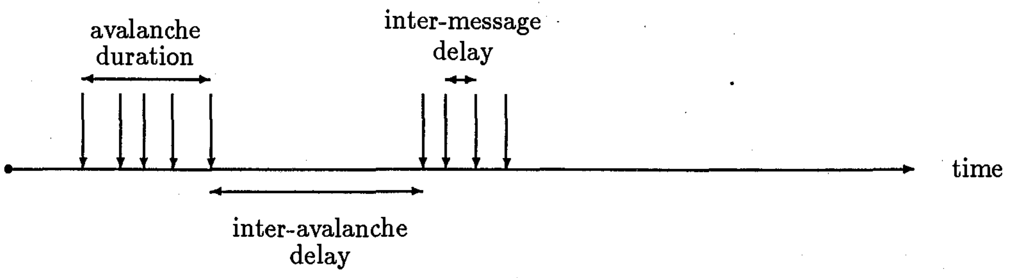


Figure 3.5: Avalanche Traffic

This section describes different types of traffic commonly encountered in real-time applications.

3.4.1 Symmetric Traffic

In this case all the stations on the ring experience the same traffic load. This case does not occur very often practically, but is of interest because it is the easiest to understand.

3.4.2 Asymmetric Traffic

In this case each station may experience a different traffic load. This makes the ring unbalanced. Protocol response to asymmetric traffic is of considerable interest because many practical real-time applications fall into this category. For example, a set of sensors connected to different nodes of the ring may be observing different events at different rates.

3.4.3 Avalanche Traffic

The concept of an avalanche is used to represent the arrival of a large quantity of messages over a short interval. There are three parameters used to define an avalanche. The first is the inter-avalanche delay, the second is the inter-message delay in the avalanche and the third is the avalanche duration. Figure 3.5 shows these parameters clearly.

Alarm traffic in a process control system can be modeled using the avalanche traffic model. The inter-avalanche delay is the time between groups of alarms. The avalanche duration is the alarm condition. The inter-message delay is the time between individual alarms.

Chapter 4

Performance Modeling

4.1 System Models

In order to predict the future performance of the system, an abstract representation of the system is needed which will embody its behavior. This is called a model of the system [Mac85]. A model contains parameters that can be varied to portray different conditions. There are two main approaches to tackle these models: the *analytical* approach and the *simulation* approach.

A model solved by an analytical method represents the system by a set of mathematical equations. These equations are solved to determine the performance of the system.

A model solved by simulation is a computer program that acts like the system. When the simulation is run, the computer program keeps track of the contention for resources represented in the model and calculates the performance measures based on what it has observed.

The analytical approach can be applied to relatively simple problems. The simulation approach, on the other hand, can be applied to all kinds of problems but has the disadvantage of being statistical in nature. This can introduce inaccuracies in the performance measures. Simulation when applied to very detailed problems, also introduces the disadvantage of having to validate the simulation program. The results produced by solving an analytical model are exact for that model. However, the analytical model may not be an accurate representation of the actual system. Therefore, the analytical results may not agree with the system behavior. Since a simulation model may be made as detailed and realistic as we like, the results from the simulation model may be closer to the behavior of the system in spite of the statistical variability.

In this thesis, due to the complexity of the protocol being analyzed, simulation has been used for the performance evaluation of various models.

4.2 Language Used

GPSS was the language used for simulation. There was a choice between using a simulation language as against a general purpose language. A simulation language was chosen because it relieves the programmer of the job of implementing the low level simulation functions. Among the simulation language, there was a choice between using GPSS or SIMSCRIPT (These are the two simulation languages available on the University Computer System). GPSS is a low level language and was chosen because it is designed for relatively easy simulation of queuing systems. It also provides ease in modifying the models for the various cases to be simulated.

GPSS/H was the version of the language which was used [Hen83]. A pre-processor was developed in C to allow convenient entry of various system parameters. This pre-processor generates GPSS/H code for the parameters entered. A post-processor was also written in C which calculates the results based on the statistics gathered during simulation and also calculates the *confidence intervals.* The pre-processor and the post-processor are based on Edward Lo's [Lo88] programs. I converted the post-processor from FORTRAN (in which it was originally written) to C since the SUN version of GPSS/H does not support the linking of FORTRAN programs.

4.3 Statistical Considerations

One of the most difficult problems concerned with using simulation is how to determine the accuracy of the simulation estimates [Mac85]. One of the parameters which affects the statistical estimates of the simulation run is the length of the run. Most tests are concerned with the steady state behavior of the system. So the simulation should be run long enough to allow the system to come to steady state. The question is how long is *long enough.* If a simulation were allowed to run for infinite time, it would give the most accurate results, but this, unfortunately, is practically impossible. Another parameter which affects the simulation estimates is the pseudo-random number sequence used.

There are some estimation techniques available, *e.g.*, methods of obtaining confidence intervals which permit us to make valid statistical inferences about the model based on simulation output. These techniques are also essential to address the tradeoff between simulation run length and the level of precision of the simulation estimates. There are three commonly used methods for estimating confidence intervals.

- Independent replications method - This is the preferred method for estimation of transient conditions. It can be applied to estimation of steady state characteristics, but one of the following two methods will generally be preferable for estimating steady state characteristics.

- Regenerative method - This is the preferred method for estimation of steady-state behavior in models with regenerative characteristics (*i.e.*, the tendency of the system to return to the regenerative state which is similar to the initial state).

- Spectral method - This is the preferred method for estimation of steady-state behavior in models without regenerative characteristics.

Since the model being simulated has regenerative characteristics (*i.e.*, no waiting message in the system and a free token), and since we wish to study the steady state behavior of the system, the regenerative method was chosen to estimate confidence intervals. A 90% confidence interval was sought. This determined the number of regenerative cycles and hence the simulation run length required to obtain the sought confidence interval [Cra77].

Another parameter which affects the simulation estimates is the pseudo-random sequence. To establish the randomness of the GPSS pseudo-random number generator, the chi-square test was used [Knu81]. The chi-square is the most basic and best known of all such statistical tests. It is often used in connection with many other tests. Theoretically, there is no end to the number of tests which can be performed. If a sequence behaves randomly with respect to tests T_1, T_2,....., T_n, we cannot be sure that it will not be a failure when subjected to test T_{n+1}. Yet each test gives us more confidence about the randomness of the sequence.

The GPSS pseudo-random number sequence is a sequence of uniformly distributed numbers between zero and one. Based on this sequence, a GPSS/H function RN_j generates another sequence with uniformly distributed numbers between 0 and 999 (end points included). This sequence is subjected to the chi-square test. It involves generating a large test sequence of these random numbers and tabulating the amount of times each number appeares in the sequence. Based on the observed values of these counts and the expected values assuming a perfect random sequence, the percentage departure of the observed sequence from the expected sequence is calculated. A value greater than 90% represents a significant departure from random behavior. A value less than 10% means that the results are too close to the expected results and hence the sequence cannot be considered random. We now present some notation and then give the expression used for calculating this percentage departure.

V - *A measure of the departure of the observed sequence from the expected sequence.*

k - *A count of the numbers which can appear in the sequence.*

p - *The probability of each number appearing in a k length sequence.*

n - *The experimental sequence length.*

Y_s - *The observed count for the number s where $1 \leq s \leq k$.*

$$p = \frac{1}{k}$$

$$V = \frac{1}{n} \sum_{1 \leq s \leq k} \left(\frac{Y_s^2}{p} \right) - n$$

In an experiment, a 20,000 length sequence was generated (with the numbers distributed uniformly between 0 and 999). Three runs were conducted for the random number sequences with different seeds each time. The value of V for these three cases was found to be as follows: 1002.46, 1002.86, and 986.96. With these values of V, the chi-square distribution table was looked up. It was noticed that one of these values fall in the 25%-50% range and two of these values fall into the 50%-75% range. Since none of these ranges are the *suspect* ranges we assume that the GPSS/H pseudo-random number generators are satisfactory.

4.4 Performance Measure

The usual performance measure used for computer communication networks is the mean message waiting time [Kle76]. The mean message waiting time is defined as the time between the start of a request to transmit and the beginning of its transmission. Thus, a performance measure for the conventional IEEE 802.5 token ring protocol is the mean waiting time.

In the fixed priority protocol or the dynamic priority protocol, the waiting time for each successfully transmitted message will always be less than the deadline. Thus, the message waiting time does not provide a useful performance measure. A useful measure in this case is the percentage of messages lost (*ML*) due to their not having met their deadlines. If we define *TMT* and *TML* as follows:
TML - Total messages lost
TMT - Total messages successfully transmitted.
then,

$$ML = \frac{TML}{TMT + TML}$$

The improvement in performance of the dynamic priority protocol over the fixed priority protocol is the difference in the percentages of messages lost for each case, where:
ML(f) - Percentage of messages lost for the fixed priority protocol.
ML(d) - Percentage of messages lost for the dynamic priority protocol.

$$\textit{Improvement} = ML(f) - ML(d)$$

ML(d) will be different for different values of Δ, *e.g.*, *ML(f)* = *ML(d)* when $\Delta = 0$. Henceforth when *ML(d)* is referred to, it will be for that value of Δ for which *ML(d)* is minimum (*i.e.*, the improvement due to the dynamic priority protocol is maximum).

The performance measure in the case of avalanche traffic is the percentage of avalanche messages lost. There is one avalanche period in each regenerative cycle. The avalanche arrives at the start of each regenerative cycle and the regenerative state is achieved as soon as the system goes empty (*i.e.*, all queues empty and a free token).

4.5 Analytical Model

Analytical results for the IEEE 802.5 multiple priority token ring protocol are not yet available. Since the dynamic priority protocol is based on it, analytical results for the dynamic case are also not available. There is, however, some analysis relevant to the dynamic priority protocol which is presented in Appendix A.

Analytical results are available for the IEEE 802.5 single priority case. Exact analytical results are available for the *symmetric* ring. In the symmetric ring, each station is identical in terms of its service time distribution and switchover time. A survey of these results can be found in [Tak86]. Exact results are also available for the *asymmetric* ring for the exhaustive and gated service discipline. The case of interest in this thesis is the asymmetric ring with limited-to-one service discipline. The ring is asymmetric due to the 27-bit latency buffer at one station which leads to asymmetric switchover time [IEE85]. There are some approximate results available for the asymmetric ring with limited-to-one service discipline. Extensive simulations show that the most accurate of these approximate results are by Boxma and Meister [Box86]. These results were used to validate our simulation model, detailed below.

4.6 Simulation Model

This section describes the simplified GPSS/H simulation models for the fixed priority protocol and the dynamic priority protocol. These models are described using the GPSS/H blocks [Sch74]. These models use the single token strategy and the limited-to-one service discipline. The program is capable of handling exponential and constant message inter-arrival times and message service times. The deadlines can be constant or exponentially distributed. The full code is in Appendix B.

The GPSS/H model segment for the fixed priority protocol is shown in Figure 4.1 and the model segment for the dynamic priority protocol is shown in Figures 4.2 and 4.3. Each station on the ring is represented by a model segment. The model segment for station i is shown in the figures. The GENERATE block represents the arrival of messages (in GPSS/H terminology, a transaction represents a message) at the specified rate at each station. The ASSIGN block assigns a deadline to each transaction. This transaction is then queued onto the *user chain* for that station by the LINK block. It is queued in the order of its lifetime. The usage of a LINK block as against a QUEUE block to queue the transaction makes the transaction scan-inactive and saves computing time dramatically. The transaction collects waiting time statistics while it is queued onto the user chain. Program control is transferred to all stations in a round-robin manner representing the path of the token.

The simulator clock keeps track of the time of events in a simulation run. It is incremented by an ADVANCE block. The statistics gathered are based on this clock. A clock tick of one micro sec was chosen for this program. One micro sec is equal to one bit time (One bit time is the time occupied by one bit on the ring) for a one Mhz ring and hence is the smallest possible unit for this application.

In the fixed priority protocol, each time a model segment is activated, it indicates the arrival of a free token. This is point 1 in Figure 4.1. A transaction is dequeued from the user chain using the UNLINK block. This dequeued transaction proceeds to the TEST block and the transaction that entered the UNLINK block is removed from the system by the TERMINATE block. The TEST block determines whether the dequeued transaction has overshot its deadline. If it has, the transaction is discarded and a lost message is recorded. The user chain is then checked for another transaction. If the transaction has not overshot its deadline, the token (*i.e.*, the facility which is a single server) is seized using the SEIZE block. The waiting time is tabulated using the TABULATE block. The ADVANCE block increments the simulator clock by an amount equal to the message service time. The token is released using the RELEASE block. The TEST block checks for the empty state of the system. If the system is empty, the TERMINATE block is executed which indicates the end of a regenerative cycle and it collects statistics about the previous cycle. Control is then transferred to the next station indicating the arrival of a free token. If the system is not empty, control is transferred to the next station indicating the arrival of a free token.

If there is no message waiting at a station, the UNLINK block does not dequeue any message and control is transferred to point 2 in Figure 4.1. Here the ADVANCE block increments the simulator clock by an amount equivalent to the station latency and the free token is transferred to the next station downstream. We can get an estimate of the token utilization by measuring the amount of time the token is in use.

The GPSS model segment for the dynamic priority protocol is shown in Figures 4.2 and 4.3. This is more complicated because each time a token (free or busy) is received at a station, a transaction must be dequeued from the user chain. This is in an attempt to transmit the transaction (if the token is free and the priority of the transaction is greater than or equal to PR) or to make a reservation (if it cannot be transmitted). This is unlike the case in the fixed priority protocol when the transactions only needed to be dequeued from the user chain at the receipt of a free token. The shaded blocks in these figures are not GPSS primitive blocks. They are, in turn, composed of many GPSS primitive blocks and are used here to hide programming detail.

There are three global variables - PR, RR and BUSY. PR represents the priority of the token, RR indicates the reservation field of the token and BUSY indicates whether the the token is busy or free.

Each time the model segment for a station is activated, it indicates the arrival of a token at that station. This is represented by point 1 in Figure 4.2. The transaction goes through a pair of TEST blocks which establish whether the current station is the transmitting station, in which case the RELEASE block releases the token, the ADVANCE block increments the simulator clock by an amount equal to the message service time and control is transferred to the UNLINK block. In case the current station is not a transmitting station, control is transferred directly to the UNLINK block. The UNLINK block dequeues a transaction from the head of the user chain. The transaction which entered the UNLINK block waits until the dequeued message is either transmitted or is queued back to the user chain. In case it is transmitted, this transaction is terminated by the TERMINATE block. In case it is not transmitted, this transaction goes to the ADVANCE block where the simulator clock is incremented by a time equal to the station latency and the token is transferred to the next station.

The dequeued transaction is transmitted to the TEST block where it is checked whether the transaction has overshot its deadline. If it has, the transaction is discarded, a lost message is recorded by the TABULATE block and the user chain is checked for another message. If the deadline has not been missed and a free token has been received, then PR is downgraded as explained in Section 1.2.2. If the token is busy and station i is not the transmitting station then a reservation is made as explained in Section 1.2.2. The message is then queued onto the head of the user chain. If station i is the transmitting station then the transaction cannot be transmitted, the values of RR and PR are updated as described in Section 1.2.2 and the transaction is queued onto the head of the user chain by the LINK block.

After downgrading PR the priority of the transaction is calculated depending on the time remaining until its deadline and Δ. If this calculated priority is greater or equal to PR a SEIZE block seizes the token , a TABULATE block records the waiting time, an ADVANCE block increments the clock by an amount equal to the station latency and transmits the message to the next station. If the calculated priority is less than PR, then a reservation is made (only if the calculated priority is greater than RR) and the message is queued back onto the head of the user chain.

In case no transaction is dequeued from the user chain, control is transferred to point 2 in Figure 4.3.

For both the above protocols the regenerative method is used to estimate confidence intervals which suggest that the simulation should be run for a certain number of regenerative cycles. The regenerative approach is motivated by the fact that many statistical systems have the property of *starting afresh probabilistically* from time to time (*i.e.*, the time the regenerative state is reached). This enables one to observe independent and identically distributed blocks of data in the course of a single simulation run [Cra77]. If the regenerative state is assumed to be the initial state of the system (*i.e.*, the state when all the queues are empty and the token is free), then each time during the run when such a state is reached, the data from the previous cycle is stored and the program starts to record new data for the next regenerative cycle. The simulation stops after the required number of regenerative cycles is over.

The number of regenerative cycles used in our simulation program was 8000. It was calculated for 90% confidence intervals [Cra77]. The length of the simulation runs took of the order of 40 minutes to 100 minutes when run on a SUN 3/50 workstation. The length of the run increased with the increase in traffic load and number of stations.

The avalanche traffic model can be applied to both the fixed priority and the dynamic priority protocol. Avalanche traffic is modeled as the arrival of a large quantity of messages over a short time interval, superimposed upon *background* traffic. Background traffic is the traffic normally present on the ring and is provided by the model.

4.7 Validation

The only way to prove the correctness of a simulation is to compare it against known analytical results.

The simulation results for the single priority token ring protocol were validated by comparing them against the approximate analytical results in [Box86].

The simulation results for the multiple priority token ring protocol cannot be validated against any analytical results due to lack of any. However, the following check was made: In the multiple priority token ring protocol, instead of entering the number of priority levels as eight, the number of priority levels as one was entered. The results were then compared with the single priority token ring results.

There exist no analytical results for the dynamic priority protocol. Thus, the only way to check the validity of the simulation program is to monitor the program step by step and to make sure it is obeying the protocol. This was done using the interactive debugging facilities in GPSS/H. The program was also monitored by instumenting it with print statements which resulted in it printing out trace information during its run. This information was later studied to justify the correctness of the program.

The above means were used to justify the logical correctness of the simulation program. In order to establish the accuracy of the simulation from the statistical point of view, confidence intervals were calculated to test the accuracy of the simulation output and the chi-square test was conducted to test the GPSS random number sequence.

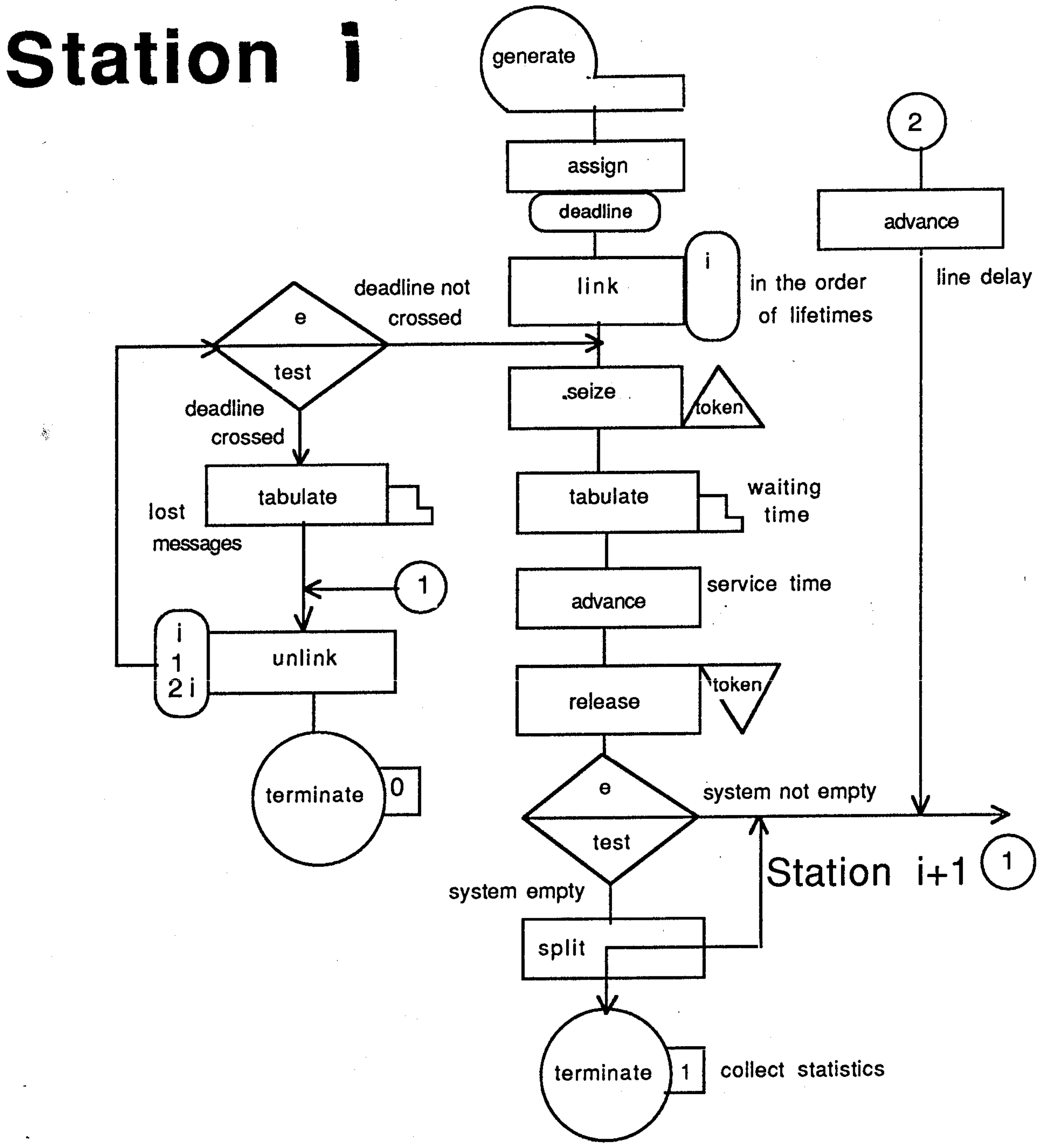


Figure 4.1: GPSS Model for the Fixed Priority Protocol

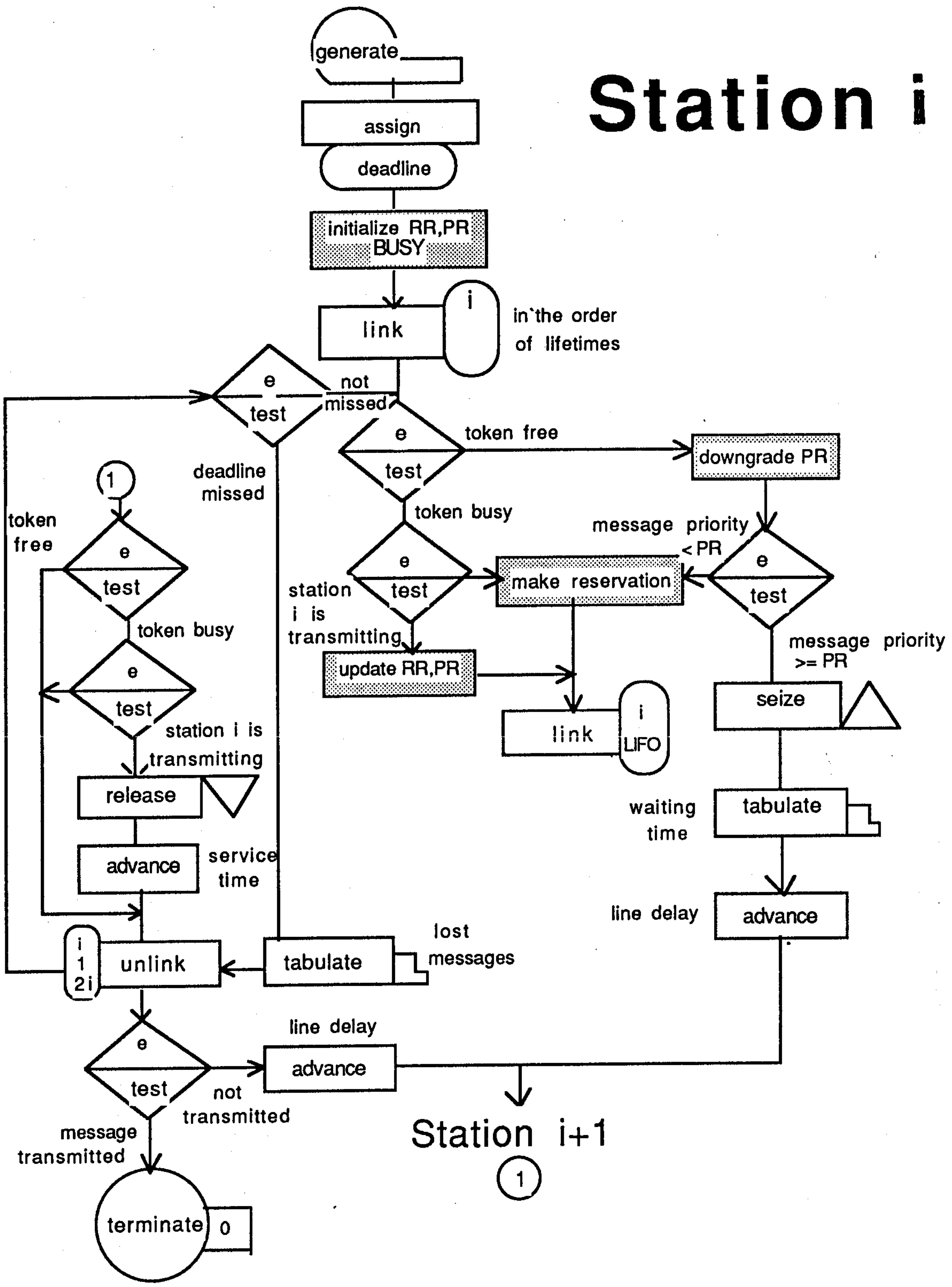


Figure 4.2: GPSS Model for the Dynamic Priority Protocol - Part a

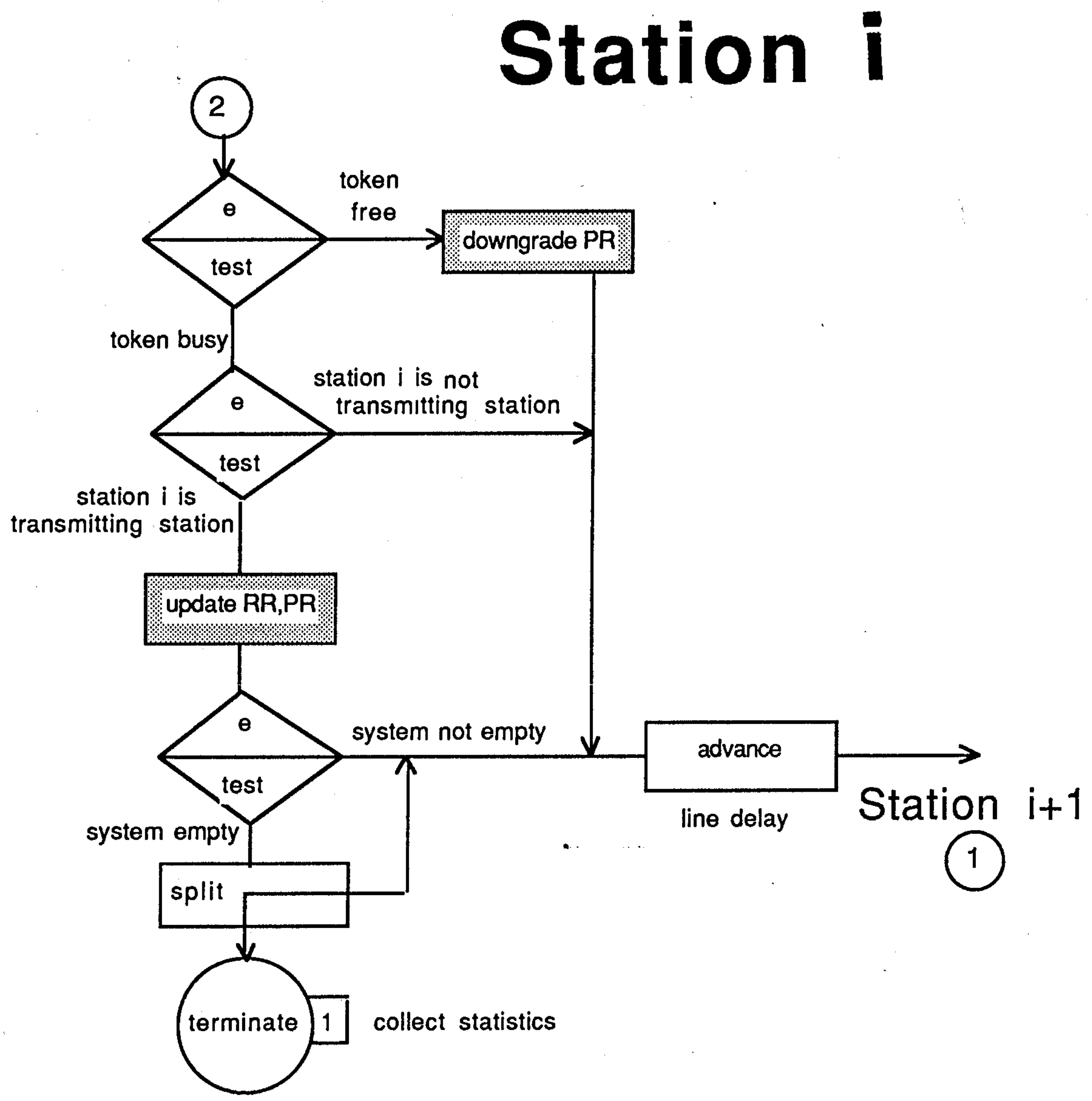


Figure 4.3: GPSS Model for the Dynamic Priority Protocol - Part b

Chapter 5

Results and Evaluation

Simulation runs were conducted for a variety of different cases which include different message arrival rates and distributions, different number of stations and different types of message traffic.

5.1 Ring Configuration

The simulation runs have been conducted under the following conditions.

- Infinite buffer queues at each station
- Single token operation
- Limited-to-one Service
- 1 MHz ring
- Errorless channel
- 256 bits constant service time
- 1 bit latency per station
- 27-bit latency buffer (at station 1)
- Linear priority function for the dynamic case
- 8 priority levels with 1 as the lowest and 8 as the highest priority

The simulation runs have been conducted with exponential message inter-arrival time and exponentially distributed lifetimes.

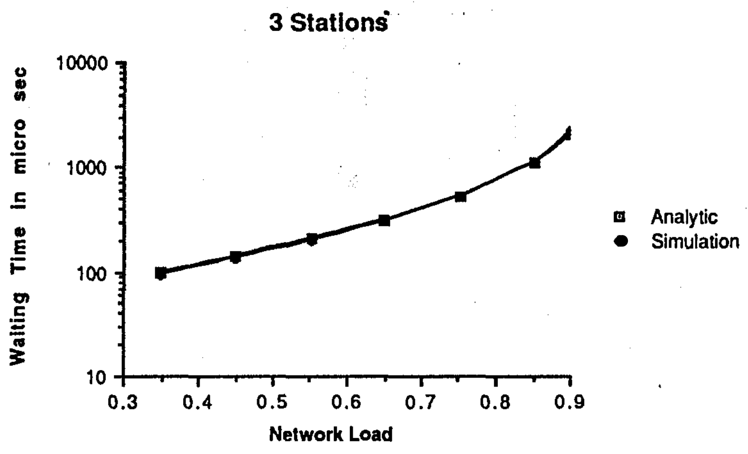


Figure 5.1: Waiting Time versus Load for a 3 Station Single Priority Ring

All time units, unless otherwise specified, are in bit times. One bit time is the time occupied by one bit on the ring. As the ring is a one Mhz ring, one bit time = one micro sec.

The improvement of the dynamic priority protocol over the fixed priority protocol is expressed as a difference of percentages of messages lost for each case.

5.2 Single Priority Case

Simulation runs were conducted for 3, 10, 20, 40 and 80 stations for the IEEE 802.5 single priority token ring. The results were compared with the approximate analytical results based on Boxma and Meister's [Box86] model. There was close conformity between the analytical results based on Boxma and Meister's model and the simulation results produced by this thesis. There was an average discrepency of 1% at low loads and 2.16% at high loads between the analytical and simulation results. Tables 5.1, 5.2, 5.3 and 5.4 show the analytic and simulation results for a 3, 10, 20 and 40 station ring; the data for 3, 10 and 20 stations is graphed in Figures 5.1, 5.2 and 5.3 respectively. It is interesting to note that the waiting time for loads 0.55 to 0.90 decreases as the number of stations is increased from 3 to 10, then increases with further increase in the number of stations. On the other hand, if we consider a ring without the 27 bit latency buffer (*i.e.*, the case where the ring latency is directly proportional to the number of stations), the waiting time increases with the increase in number of stations.

Load	Analytic	Simulation
.35	96.72	94.66
.45	140.58	136.36
.55	205.71	195.89
.65	312.52	303.89
.75	519.92	514.72
.85	1096.34	1099.82
.90	2077.30	2197.60

Table 5.1: Analytical and Simulation Results for Mean Message Waiting Time in micro sec for a 3 Station, Single Priority Ring

Load	Analytic	Simulation
.35	99.15	98.11
.45	141.55	139.47
.55	203.40	196.41
.65	302.11	294.04
.75	484.56	479.42
.85	935.79	931.652
.90	1556.05	1553.08

Table 5.2: Analytical and Simulation Results for Mean Message Waiting Time in micro sec for a 10 Station, Single Priority Ring

Load	Analytic	Simulation
.35	106.23	105.58
.45	149.53	151.37
.55	212.48	195.89
.65	312.36	307.66
.75	495.16	499.39
.85	937.42	916.00
.90	1523.43	1514.70

Table 5.3: Analytical and Simulation Results for Mean Message Waiting Time in micro sec for a 20 Station, Single Priority Ring

Load	Analytic	Simulation
.35	121.33	121.74
.45	167.21	169.26
.55	233.78	233.23
.65	339.10	331.91
.75	530.93	521.47
.85	990.12	944.65
.90	1588.05	1544.79

Table 5.4: Analytical and Simulation Results for Mean Message Waiting Time in micro sec for a 40 Station, Single Priority Ring

Load	Analytic	Simulation
.35	152.01	152.28
.45	203.41	203.76
.55	277.92	277.75
.65	395.65	393.14
.75	609.56	617.05
.85	1118.91	1145.11
.90	1776.55	1811.50

Table 5.5: Analytical and Simulation Results for Mean Message Waiting Time in micro sec for an 80 Station, Single Priority Ring

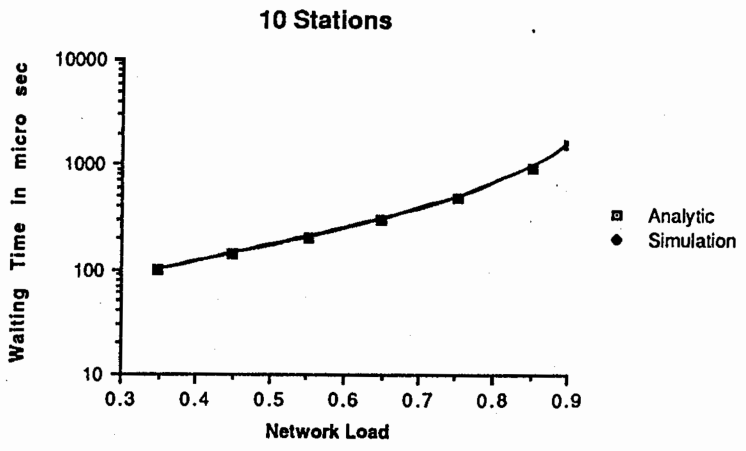


Figure 5.2: Waiting Time versus Load for a 10 Station Single Priority Ring

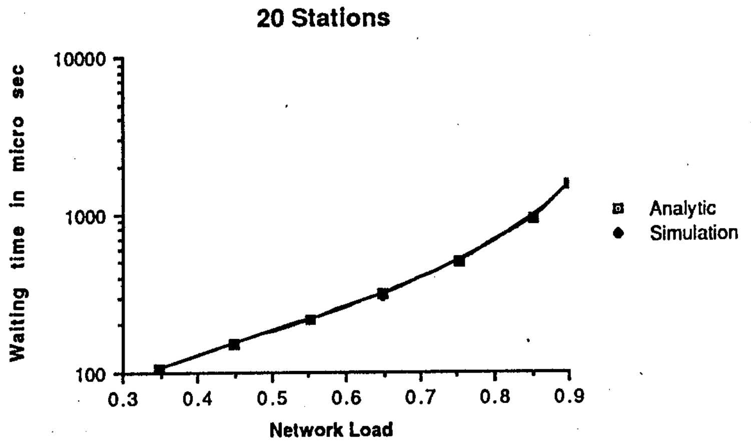


Figure 5.3: Waiting Time versus Load for a 20 Station Single Priority Ring

5.3 Multiple Priority Case

Simulation runs were conducted for the multiple priority case with eight different priority levels. There are no proven analytical results for the IEEE 802.5 multiple priority case as yet, so the simulation results cannot be validated against any analytical results. However, the following check was made. In the multiple priority GPSS/H program, instead of entering the number of priority levels as 8, the number of priority levels 1 was entered. The results were compared with the single priority results and found to be identical.

There are some analytic and simulation results for the multiple priority token ring in [Ped88, Ped87b,Ped87a]. There is no proof of correctness available for the analytical results.

In order to compare our simulation results against those in [Ped88], we used the same set of parameters as in [Ped88]. These parameters include exponential inter-arrival times for loads ranging between .35 and .90. A constant service time of 256 micro sec was used and the ring configuration used was as specified in Section 5.1. The results turned out to be different. In this thesis, the graphs for the multiple priority case show a greater split between the waiting times for priority 1 and 8 than those in [Ped88]. The performance measure used in [Ped88] is the message delivery time. The message delivery time is equal to the message waiting time plus the message service time plus half the ring latency. This difference in performance measures was taken into account when comparing results.

Attempts were made to explain this difference in results. I examined the PASCAL simulator used in [Ped88] and it conformed to the IEEE 802.5 token ring protocol. The only significant difference

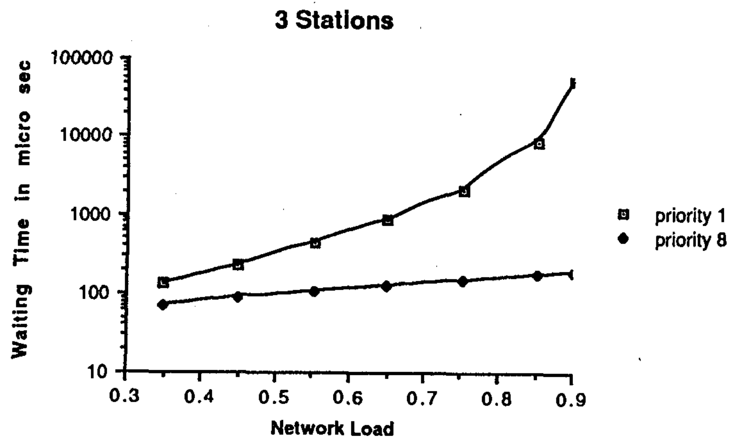


Figure 5.4: Waiting Time for Priority 1 and 8 for a 3 Station Multiple Priority Ring

between Peden's PASCAL simulator and the GPSS/H simulator used in this thesis was the random number generators used in each case. GPSS/H uses its built-in random number generator. In [Ped88] the random function of PASCAL was used to generate random numbers. Our GPSS/H simulation model is assumed to be correct, as the results appear to be intuitively correct; we would expect the highest priority messages to have a much smaller waiting time than the lowest priority messages.

Figures 5.4, 5.5 and 5.6 show the waiting time as achieved by our simulation model, for the highest and lowest priority messages for 3, 10 and 20 stations respectively.

5.4 Dynamic Case

Using thn GPSS/H program for the multiple priority token ring as a basis, a program for the dynamic priority protocol was developed. In this case all messages are generated with the same priority unlike the multiple priority case where the messages are generated with uniformly distributed priorities between one and eight. The priorities of the messages are stepped up according to a priority function. The priority of any message in the queue at any instant of time depends on the time remaining until its deadline. If certain messages (*e.g.*, alarm messages) are generated which need to be given priority over the regular traffic, they can be associated with shorter lifetimes by means of which they will be placed well ahead in the queue. They may not be placed at the head of queue because there may be other longer lifetime messages which have been waiting for sometime and which may be closer to their deadlines than the alarm message.

In this chapter the term *deadline* is used to refer to a time interval as against

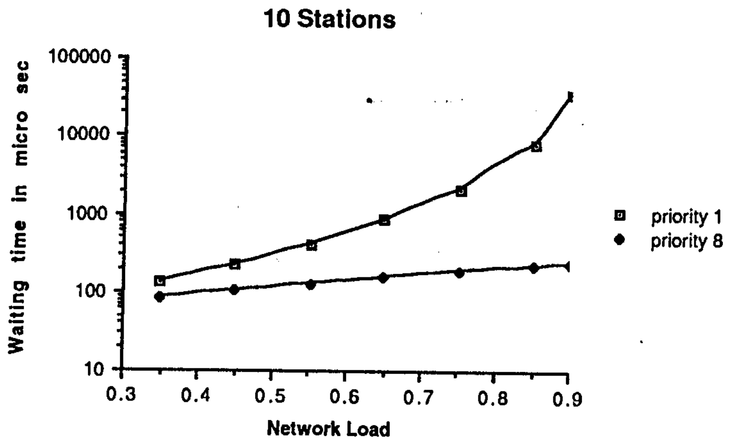


Figure 5.5: Waiting Time for Priority 1 and 8 for a 10 Station Multiple Priority Ring

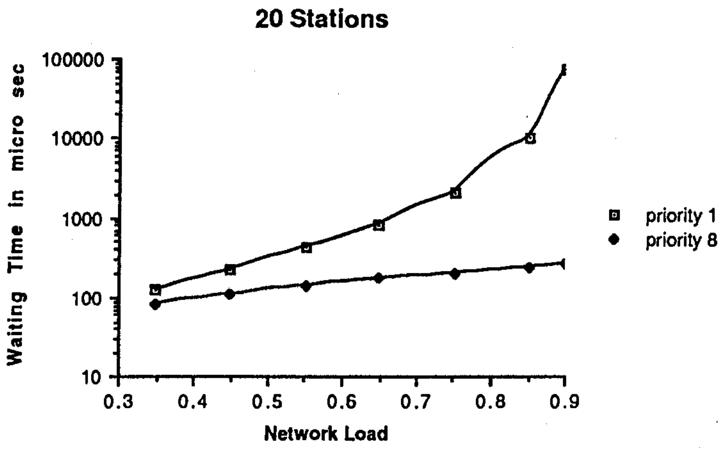


Figure 5.6: Waiting Time for Priority 1 and 8 for a 20 Station Multiple Priority Ring

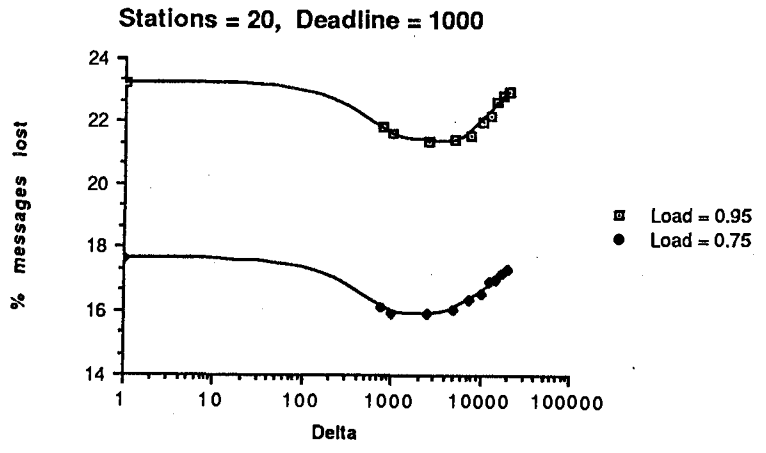


Figure 5.7: Percentage Messages Lost for Symmetric Traffic - 20 Stations, 1000 Deadline

Message Deadline	Improvement in the Fraction of Messages Lost	
	Network Load = 0.75	Network Load = 0.95
1000	1.75	1.86
2000	2.38	3.28

Table 5.6: Improvement due to the Dynamic Priority Protocol for Symmetric traffic for a 20 Station Ring

5.4.1 Symmetrical Traffic

Symmetric traffic implies that every station on the ring experiences the same traffic load. This kind of symmetery is not encountered very often in practical situations, nevertheless, we studied it out of interest. Figures 5.7 and 5.8 show the graphs of the percentage of messages lost versus Δ for a 20 station ring. Table 5.6 shows the improvement due to the dynamic priority protocol.

5.4.2 Asymmetric Traffic

Asymmetric traffic implies that all the stations on the ring do not receive the same traffic load. This is a situation which occurs frequently in the industry when different nodes on the ring are connected to different sensors observing different events at varying rates.

Two experiments were conducted for a 20 station asymmetric ring. In the first experiment, a traffic load of 2.85 was applied to station 1 while the remaining 19 stations were subjected to a traffic load of 0.85 resulting in an overall load of 0.95. In the second experiment, station 1 and 11 were applied a traffic load of 1.65, the remaining 18 stations were applied a traffic load of 0.65 resulting

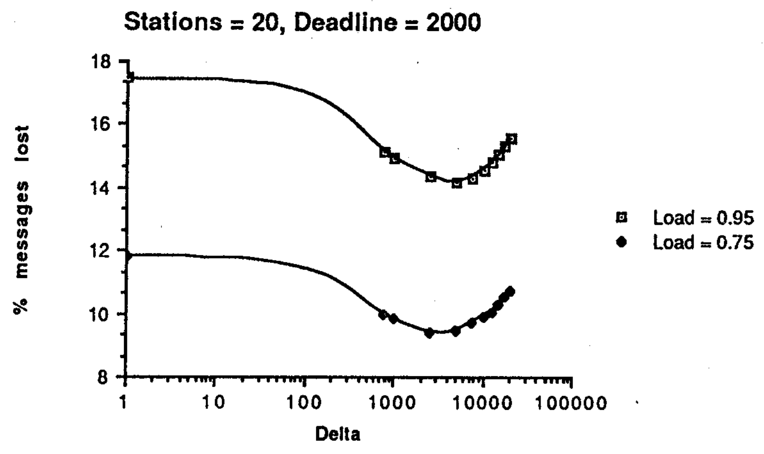


Figure 5.8: Percentage Messages Lost for Symmetric Traffic - 20 Stations, 2000 Deadline

Message Deadline	Improvement in the Fraction of Messages Lost	
	Network Load = 0.75	Network Load = 0.95
1000	1.47	2.02
2000	2.02	3.10

Table 5.7: Improvement due to the Dynamic Priority Protocol for Asymmetric traffic for a 20 Station Ring

in an overall load of 0.75.

Figurs 5.9 and 5.10 show the graphs of the percentage of messages lost versus Δ for the two experiments described above. Table 5.7 shows the improvements due to the dynamic protocol for each of these experiments.

A comparison between Tables 5.6 and 5.7 indicate that the improvements for the symmetric and the asymmetric case are nearly the same, which leads to the conclusion that the improvement depends on the overall load on the ring and not on its distribution.

5.4.3 Avalanche Traffic

Avalanche traffic is modeled as the arrival of a large quantity of messages over a short time interval, superimposed upon normal background traffic. The performance measure is the percentage of avalanche messages lost. Figures 5.11 and 5.12 show the graphs of the percentage of messages lost versus Δ for avalanche traffic in a 3 station ring, where the background traffic load is 0.5, the background traffic deadline is 3000 and the avalanche duration is 5000. Table 5.8 represents the

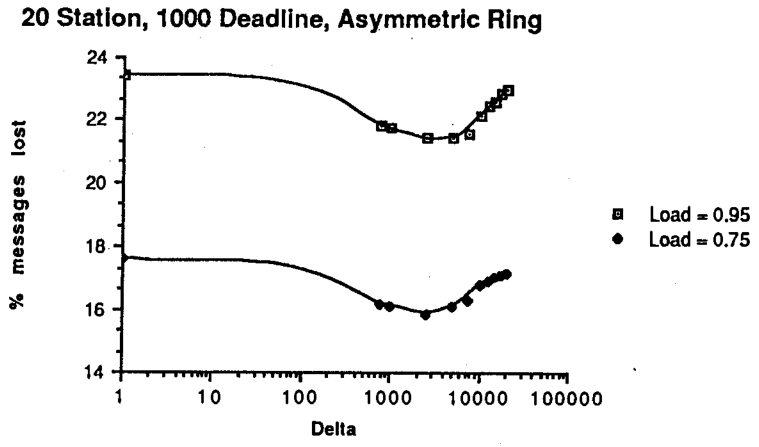


Figure 5.9: Percentage Messages Lost for Asymmetric Traffic - 20 Stations,1000 Deadline

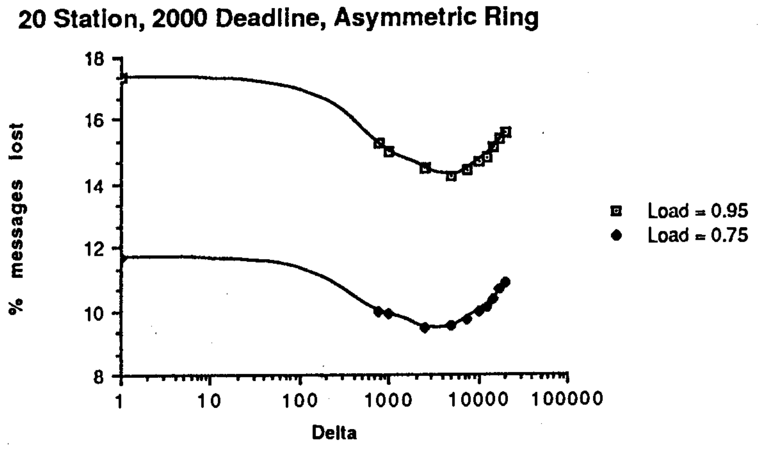


Figure 5.10: Percentage Messages Lost for Asymmetric Traffic - 20 Stations,2000 Deadline

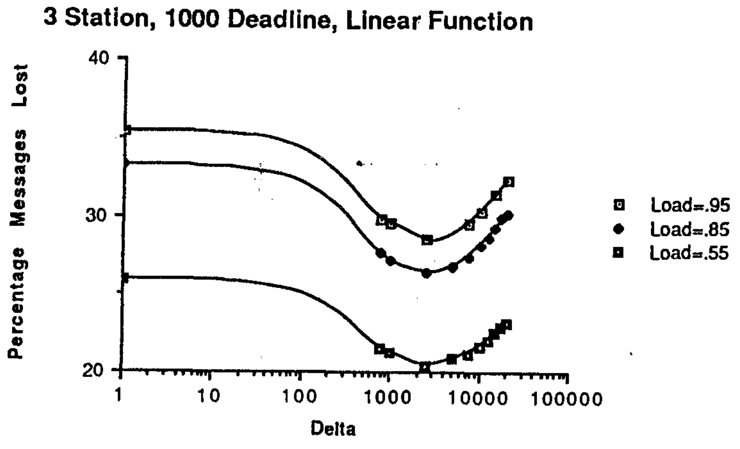


Figure 5.11: Percentage Messages Lost for Avalanche Traffic - 3 Stations, 1000 Deadline

Avalanche Traffic Load	Improvement in the Fraction of Avalanche Messages Lost	
	Avalanche Deadline=1000	Avalanche Deadline=2000
.55	5.50	4.27
.85	6.86	6.00
.95	6.94	6.15

Table 5.8: Improvement due to the Dynamic Priority Protocol for Avalanche traffic for a 20 Station Ring

background traffic deadline is 3000 and the avalanche duration is 5000. Table 5.8 represents the improvement due to the dynamic priority protocol for different avalanche loads and deadlines.

The dynamic priority protocol performs significantly better for the case of avalanche traffic than for the symmetric and asymmetric traffic cases for nearly the same traffic load. (The load referred to in the case of avalanche traffic is only due to the avalanche traffic and not the background traffic.) This is due to the fact that the avalanche of messages is only present for a short duration during which the queues build up. After the avalanche is over, there is only the background traffic which is relatively slow. This queue build-up leads to a high token utilization which leads to a large improvement due to the dynamic priority protocol.

5.4.4 Variation with Load

Simulations were run for a 10 station ring with the traffic load varying from .35 to 1.0. It was seen that the improvement exhibited by the dynamic priority protocol increased with the increase in the traffic load until very high loads when it levelled off.

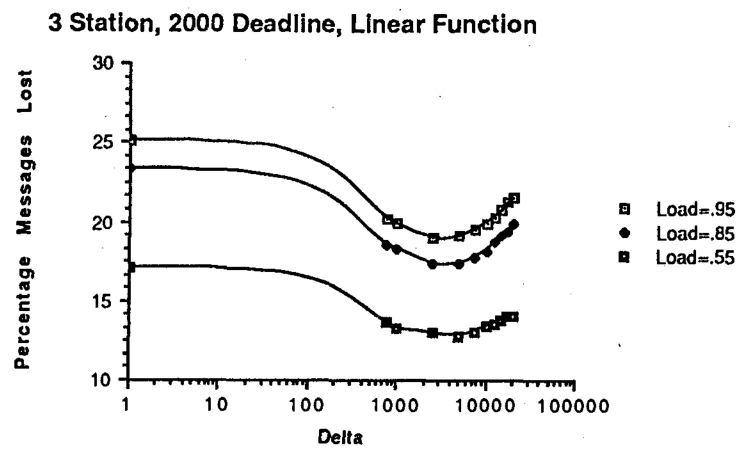


Figure 5.12: Percentage Messages Lost for Avalanche Traffic - 3 Stations, 2000 Deadline

If we define the observed load to be the fraction of the time the token is busy, it is clear that in a system with deadlines, the observed load is always less than or equal to the traffic load.

Figure 5.13 shows the graph between the traffic load and the observed load, for a 10 station ring.

At low loads, since there were few messages getting lost, there was not much room for improvement by the dynamic priority protocol. As the load was increased, more messages started getting lost, and the dynamic priority protocol resulted in greater improvements. As the traffic load was increased to very high values, the number of messages getting lost increased further but the improvement did not increase accordingly. It started to level off.

If we consider the extreme case of an overloaded ring, however much it is overloaded, the observed load will never exceed 1 (*i.e.*, the token cannot do better than to remain busy 100% of the time). Thus the reason the improvement starts to level off at high loads is because the token is busy nearly all the time and cannot do much better. Figures 5.14 and 5.15 show the graphs of the percentage of messages lost versus Δ for two different deadlines. Figure 5.16 shows the improvement due to the dynamic priority protocol.

5.4.5 Variation with Number of Stations

Simulation runs were conducted for 3, 10, 20, 40, 80 and 100 stations at 0.9 load. For a large number of stations, the dynamic priority protocol performs worse than the fixed priority protocol at low values of Δ. A large number of stations results in a large ring latency which leads to a large token cycle time. The combination of the high cycle time with low values of Δ, results in the dynamic

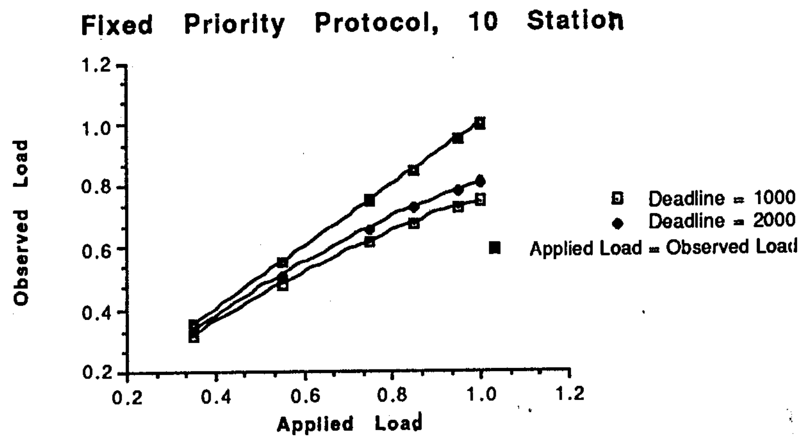


Figure 5.13: Traffic Load versus Observed Load - 10 Stations

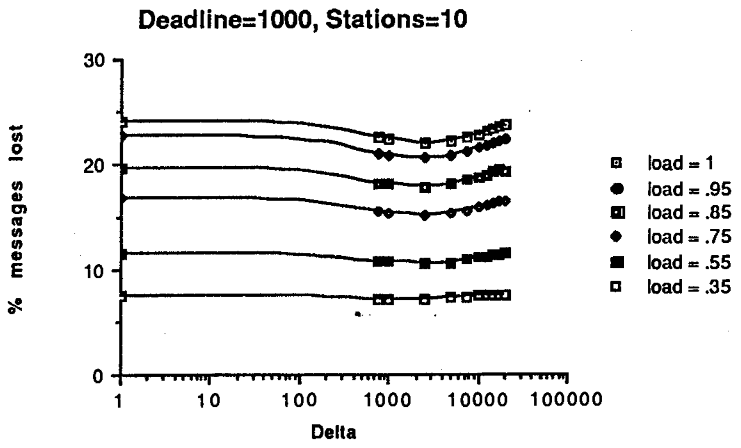


Figure 5.14: Percentage Messages Lost - 10 Stations, 1000 Deadline

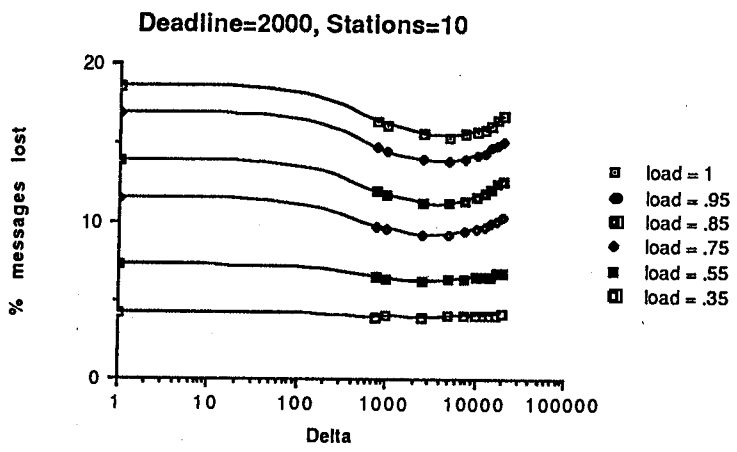


Figure 5.15: Percentage Messages Lost - 10 Stations,2000 Deadline

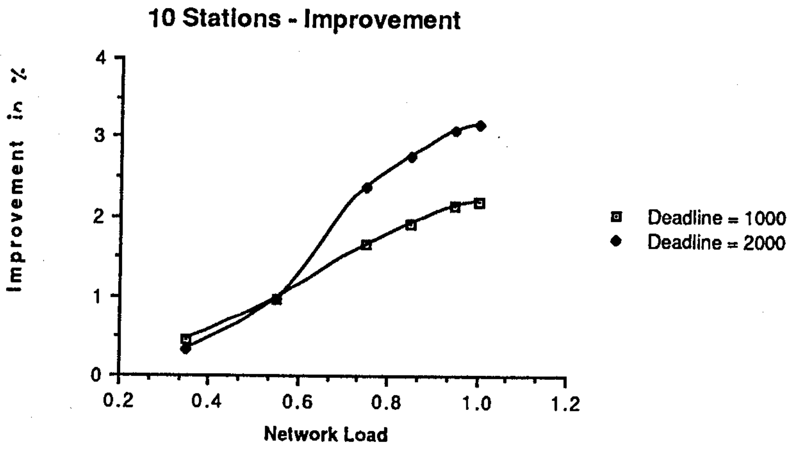


Figure 5.16: Variation with Load - 10 Stations

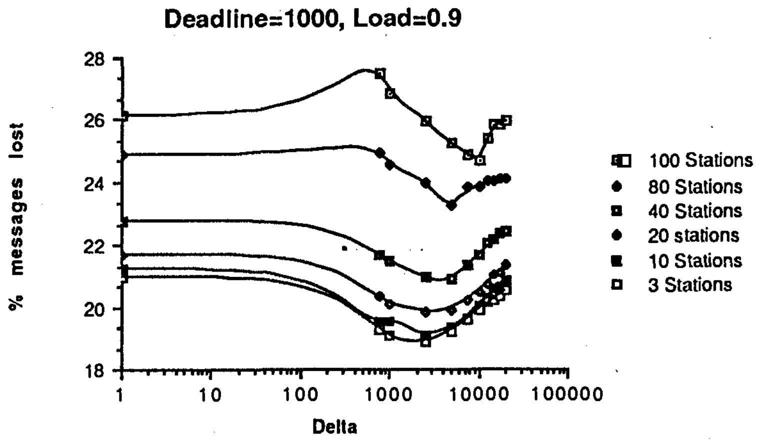


Figure 5.17: Percentage Messages Lost - 0.9 Load, 1000 Deadline

This is so because a certain station may reserve a token at a high priority. Since the value of Δ is small, by the time the token comes around to servicing that station at the requested priortity, that message may have already missed its deadline and the next message in the queue would be a low priority message, which cannot be served. In a ring with a high latency, this overhead is significant, resulting in poor performance.

The improvement due to the dynamic priority protocol first increases with the increase in the number of stations, reaches an optimum and then starts to decrease as the number of stations is increased furthur. Figures 5.17 and 5.18 show the graphs between the percentage of messages lost versus Δ and Figure 5.19 shows the improvement of the dynamic priority protocol.

5.4.6 Variation with Deadline

Simulations were run for 3 stations with traffic loads of .85 and .95. The deadline was varied from 500 to 7000. It was noticed that the improvement exhibited by the dynamic priority protocol increased with the increase in deadline, reached an optimum and then decreased as the deadline was furthur increased.

The improvement was low at short deadlines because there were more messages getting lost and the queues did not build up at the stations. This resulted in the token not being utilized efficiently. In an extreme case, if there is queue build-up, the token will, at each station have a message to carry resulting in higher token utilization and therefore better performance. As the deadline was increased, the number of messages getting lost decreased and the queue build-up increased resulting

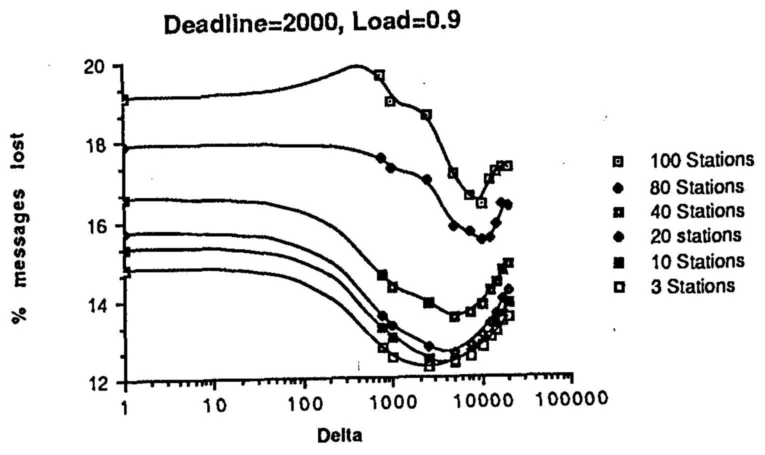


Figure 5.18: Percentage Messages Lost - 0.9 Load,2000 Deadline

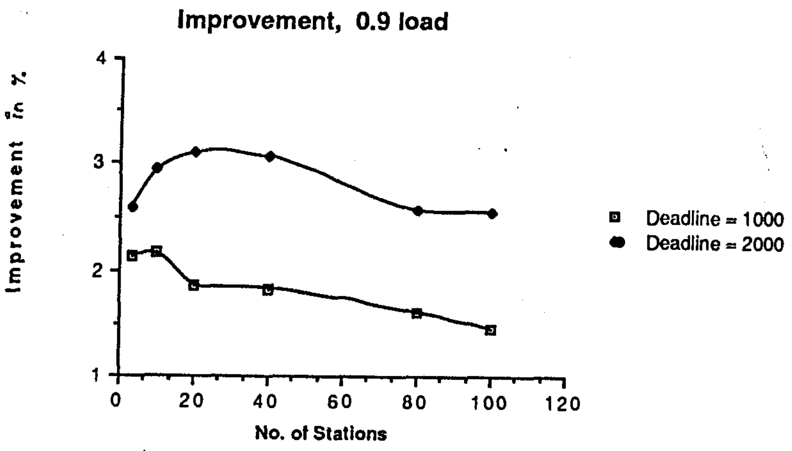


Figure 5.19: Variation with Number of Stations 0.9 Load

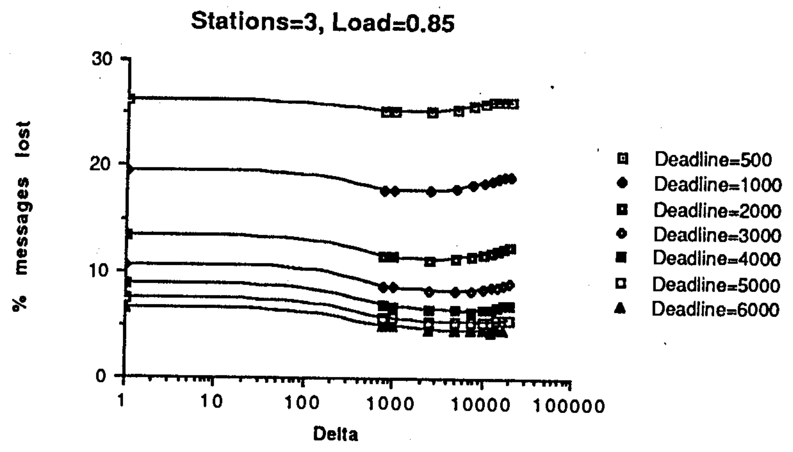


Figure 5.20: Percentage Messages Lost - 3 Stations, 0.85 Load

increased, the number of messages getting lost decreased and the queue build-up increased resulting in greater improvement.

As the deadline was increased further, the number of messages getting lost decreased, queue build-up increased resulting in a lower improvement. At the extremity, if the deadline were infinity, there would be no messages getting lost, thus, there would be no improvement. Figures 5.20 and 5.21 show the graphs of the percentage of messages lost versus Δ for a 3 station ring and Figure 5.22 shows the improvement of the dynamic priority protocol.

5.4.7 Non-linear Priority Function

The dynamic priority protocol with a non-linear priority function was studied for the case of avalanche traffic. The ring configuration and the protocol parameters were exactly the same as in Section 5.4.1. Figures 5.23 and 5.24 show the graphs of the percentage of messages lost versus Δ for the case of a non-linear priority function. Figures 5.11 and 5.12 show the corresponding graphs for the linear priority function.

Table 5.9 shows the improvements achieved by the non-linear priority function when applied to avalanche traffic. Contrast this with Table 5.8 which shows the improvement for the linear priority function when applied to avalanche traffic. The non-linear priority function case exhibits marginally better improvement than the linear priority function.

A point of interest is that the range of Δ for which the dynamic priority protocol shows improvement is much larger for the non-linear priority case than in the linear priority case. Therefore,

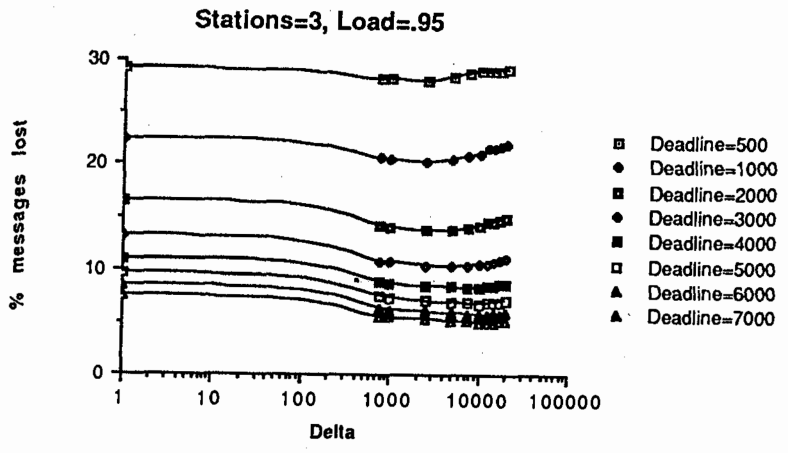


Figure 5.21: Percentage Messages Lost - 3 stations, 0.95 load

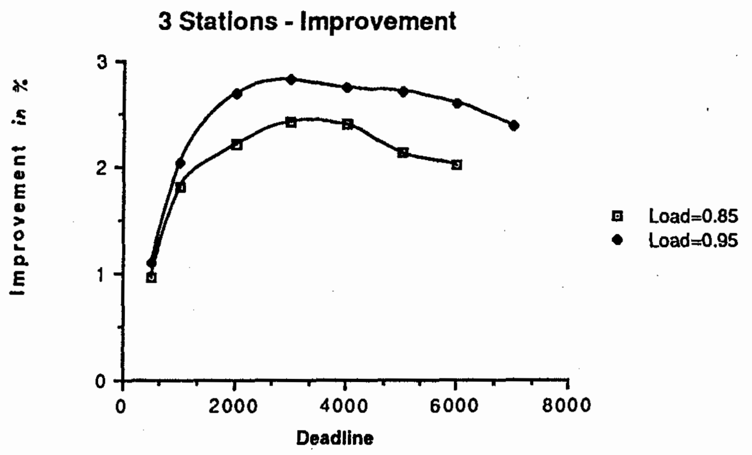


Figure 5.22: Variation with Deadlines - 3 Stations

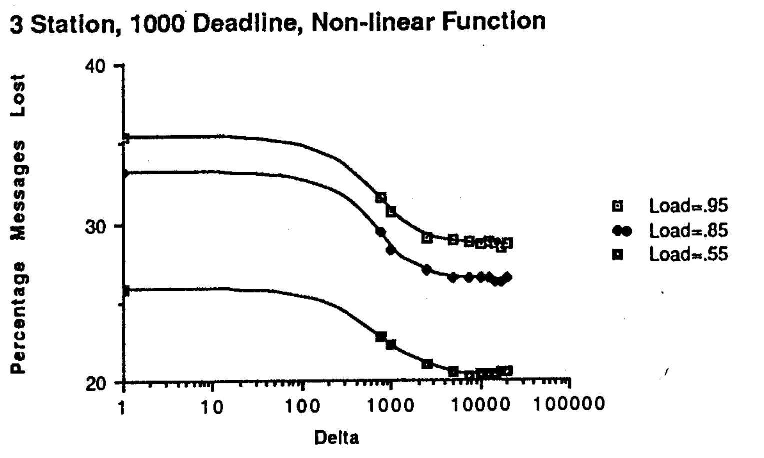


Figure 5.23: Percentage Messages Lost for Non-Linear case - 3 stations, 1000 Deadline

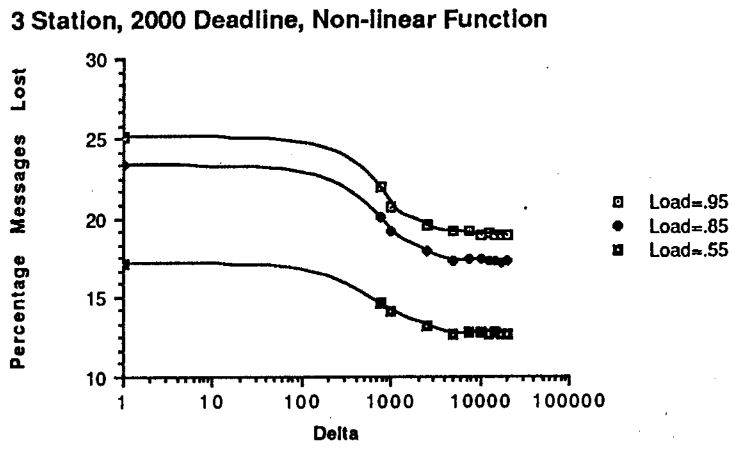


Figure 5.24: Percentage Messages Lost for Non-linear case - 3 Stations, 2000 Deadline

Message Load	Improvement in the Fraction of Avalanche Messages Lost	
	Avalanche Deadline=1000	Avalanche Deadline=2000
.55	5.62	4.43
.85	7.00	6.18
.95	7.06	6.27

Table 5.9: Improvement for the Non-linear Avalanche case for a 3 station Ring

in situations where the traffic distribution is not known, it will be significantly easier to choose a *reasonable* value of Δ for the case of this non-linear priority function.

5.4.8 Optimum Range of Δ

The dynamic priority protocol with Δ=0, performs exactly the same as the fixed priority protocol and hence exhibits no improvement. As the value of Δ is increased, the improvement due to the dynamic priority protocol increases and reaches an optimum. As Δ is increased further, the improvement decreases. This is because for a large value of Δ, when most messages get to the head of the queue, they are closer to the deadline as compared to the point Δ units before the deadline. This situation is similar to Figure 3.2. Thus most messages, when they get to the head of the queue are assigned high priorities. In an extreme case the ring will function as a single priority token ring with all messages at the highest priority. Thus, the dynamic priority protocol shows no improvement for very high values of Δ.

It was noticed that this optimum range of Δ is dependent on the following parameters: number of stations, traffic load and deadline. As the number of stations is increased, the optimum range of operation increases to higher values of Δ. Figures 5.17 and 5.18 confirm this fact. This is because as the number of stations increases, the ring latency increases which means that it takes longer for the token to traverse the ring. Therefore, if a certain station makes a reservation, the message at that station must not have crossed its deadline when it receives a free token at the requested priority, meaning thereby that there should be enough time between when a station makes a reservation and until it receives a usable token. This is possible if the value of Δ is large.

An increase in traffic load results in a long time for a station requesting a token to receive one. It is important that during this time the message at the station requesting a free token does not cross its deadline. This is again possible if Δ is large. Figures 5.14 and 5.15 confirm this fact.

A high ring latency and a high value of the network load result in higher optimum values of Δ. Therefore a high value of the token cycle time will result in higher optimum values of Δ, since the expected token cycle time E(C) is defined as [Sch87]:

$$E(C) = \frac{RingLatency}{1 - Traffic\ Load}$$

Figures 5.20 and 5.21 also indicate that the range of operation of Δ increases with the increase in deadline. For the same traffic load, a longer deadline results in a higher observed load as compared to a shorter deadline. A high observed load results in a larger token cycle time which leads to high operating values of Δ.

The improvement due to the dynamic priority protocol using the non-linear priority function is less sensitive to the choice of Δ than in the case of the linear priority function. This is because the optimum range of Δ is much larger for the non-linear priority function, therefore, it is much easier to choose a *reasonable* value of Δ.

5.4.9 The Dynamic Priority Protocol for Hard Real-Time Applications

Hard real-time application involve synchronous and asynchronous message traffic where synchronous traffic should have guaranteed message delivery time. We consider an example of a 3 station ring, with station 1 generating synchronous class traffic. First we derive an expression for the maximum bandwidth of synchronous traffic capable of being supported by the above ring configuration. If 'N' is the number of stations on the ring, then

Ring Latency(RL) = N*Station Latency + Latency buffer + Propogation delay
= N*1+27
= 30 (for 3 stations and negligible propogation delay)

Max wait Time(W) = Token Time + 2*Message Time + Ring Latency
= 24 + 2*256 + 30
= 566

W has been calculated for a constant message service time of 256 micro sec. In case of a variable message service time, the expression for W would still be valid. The value of the message time used to calculate W in that case should be the worst case message service time. Thus a synchronous message will have to wait for a maximum of 566 micro sec to get served. If the message interarrival time for synchronous traffic for a 3 station ring is greater than 566, then it will be capable of providing guaranteed access. In the following experiment on a 3 station ring, the synchronous traffic load was chosen to be .55 and the asynchronous traffic load was .95. The synchronous (hard) deadline was equal to 566 and the asynchronous (soft) deadline was varied. Tables 5.10, 5.11 and 5.12 give the results of the dynamic priority protocol when applied to the above data for different asynchronous deadlines.

We see that there are no synchronous messages lost confirming the fact that the maximum waiting time for synchronous messages is 566 micro sec. The dynamic priority protocol results in an improvement in terms of the number of asynchronous messages which meet their deadlines when applied to real-time traffic. This improvement is at the cost of increasing the asynchronous message mean waiting time which is not important since the point of interest is whether a message meets its deadline or not.

Traffic Type	Mean message Waiting Time	Percentage Messages Lost
Sync. Traffic Δ=0	125.85	0.00
Async. Traffic Δ=0	287.67	10.42
Sync. Traffic Δ=2500	121.74	0.00
Async. Traffic Δ=2500	381.32	8.85

Table 5.10: Percentage Messages Lost and Waiting Times for 2000 Asynchronous Deadline

Traffic Type	Mean message Waiting Time	Percentage Messages Lost
Sync. Traffic Δ=0	132.81	0.00
Async. Traffic Δ=0	357.16	7.91
Sync. Traffic Δ=2500	130.89	0.00
Async. Traffic Δ=2500	476.33	6.52

Table 5.11: Percentage Messages Lost and Waiting Times for 3000 Asynchronous Deadline

Traffic Type	Mean message Waiting Time	Percentage Messages Lost
Sync. Traffic Δ=0	136.72	0.00
Async. Traffic Δ=0	426.08	6.41
Sync. Traffic Δ=2500	131.35	0.00
Async. Traffic Δ=2500	549.05	5.26

Table 5.12: Percentage Messages Lost and Waiting Times for 4000 Asynchronous Deadline

Chapter 6

Conclusions

6.1 Summary

A new dynamic priority protocol has been proposed in this thesis. It exploits the priority mechanism of the IEEE 802.5 token ring protocol and makes it more suitable for real-time applications.

The performance measure used in this study is the percentage of messages lost due to their having missed their deadlines. This study shows that the dynamic priority protocol performs better than the fixed priority protocol. This improvement is expressed as a difference of the percentage of messages lost for the dynamic priority protocol and that for the fixed priority protocol.

The improvement is maximum for a certain range of Δ (*i.e.*, a protocol parameter as defined in Section 3.1) when Δ is not too large or too small. Too small a value of Δ means that the priority of a waiting message starts being stepped up very shortly before its deadline. Thus the high priority values for the messages are in effect for a very short time only. If a busy token happens to pass by when the priority of a message is at one of the higher values, (this may not be very likely) a reservation is made and the next token is issued at the requested priority. By the time this token reaches the requesting station the message would probably have crossed its deadline. This results in low values of improvements for small values of Δ. For cases when the ring latency is large compared to the lifetime (*i.e.*, the time between the message arrival and the deadline), the dynamic priority protocol performs worse than the fixed priority protocol. This is because the overhead of the dynamic priority protocol becomes significant for large ring latencies.

Very large values of Δ indicate that most messages, when they get to the head of the queue, are much closer to the deadline as against to the point Δ units before the deadline when the priority starts being stepped up. Thus, most of these messages are assigned 'higher' priorities when they get to the head of queue. The situation is similar to Figure 3.2. In an extreme case (*i.e.*, with very large values of Δ), all messages are assigned the highest priority when they get to the head of queue. The ring thus functions as a single priority ring with all the messages at the same highest priority.

The improvement for the linear and the non-linear priority function are nearly the same. In case of the non-linear priority function, we show that the performance is less sensitive to the choice of Δ *i.e.*, we have a wider range of Δ to choose from. If the traffic distribution is not known it will be safer to use the non-linear priority function which will assure us of a good value of Δ.

The degree of improvement exhibited by the dynamic priority protocol depends on various parameters, *e.g.*, the traffic load, number of stations, deadlines traffic type. The improvement is most significant for a small number of stations, higher loads and avalanche traffic.

6.2 Conclusions

The dynamic priority protocol always performs better than the fixed priority protocol. The improvemnts are of the order of 2% - 3% for symmetric and asymmetric traffic and of the order of 5% - 7% for avalanche traffic. There is a certain range of operation of Δ for which the dynamic priority protocol shows maximum improvement. This range of Δ is much larger for the non-linear priority function than for the linear priority function.

The improvement due to the dynamic priority protocol increases with the increase in traffic load and then stablizes as the load approaches the point of overloading. It increases with the increase in deadline, reaches an optimum and then decreases. It increases with the increase in the number of stations, reaches an optimum and then decreases with further increase in stations on the ring

For hard real-time applications the dynamic priority protocol, while guaranteeing synchronous traffic response, also exhibits improvement for asynchronous traffic.

6.3 Future Work

An area of interest is the application of the dynamic priority protocol to the FDDI token ring protocol. Since the FDDI token ring protocol does not have any restriction on the number of priority levels, it would be interesting to determine an optimum number of priority levels.

Analytical results for the dynamic case are not available yet. They could lead to further insight about the performance of the protocol.

An interesting subject of research would be to design a scheme by means of which we could establish an upper bound on the improvement the dynamic priority protocol is capable of. This could be very useful in terms of making sure that the dynamic priority protocol is made to operate at its optimum.

The dynamic protocol could be modeled for a practical situation by the introduction of a certain error rate on the transmission media This will require an acknowledgement and an error recovery mechanism.

Appendix A

Analysis for Dynamic Priority Protocol

A.1 Analysis

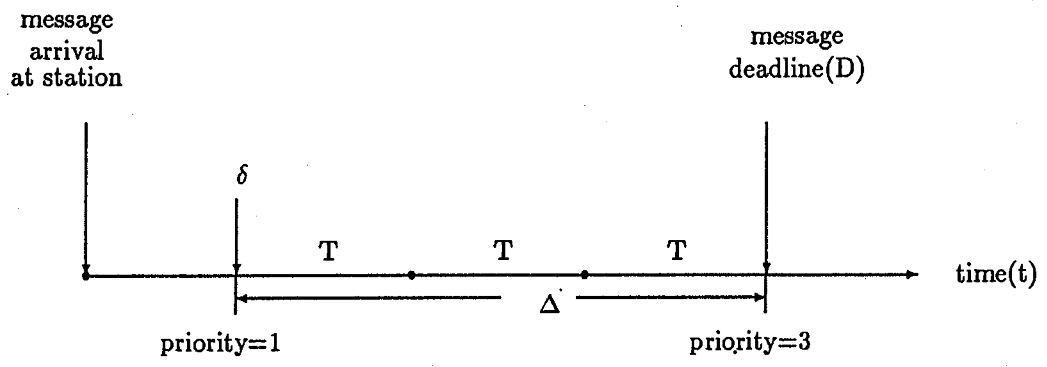
In order to analyze the dynamic priority protocol we require the Probability Distribution Function (PDF) of the waiting time for priority level i $(1 \leq i \leq 8)$ in the IEEE 802.5 protocol. These analytical results for the multiple priority token ring are not available yet. However, some attempts have been made to analyze simplified versions of the IEEE 802.5 token ring protocol [She85].

In this appendix we derive an expression to calculate the probability of message loss (due to their having missed their deadlines) for the dynamic priority protocol assuming the analytical results for the IEEE 802.5 token ring protocol are available. This analysis can be applied to the token bus protocol also.

For the sake of simplicity, we assume that the dynamic priority protocol operates at 3 priority levels. The analysis can be easily extended to 8 priority levels.

Consider the scenario in Figure A.1. The priority of a message stays at one until time δ+T, after which it is stepped up according to the linear priority function.

δ = D-Δ
Δ=3T
W = Waiting time
$F_i(t)$ = PDF of waiting time for a message of priority i in an IEEE 802.5 token ring.



$$Prob(W \leq t) = F_1(t) \qquad 0 \leq t \leq \delta + T$$

$$\begin{aligned}
Prob(\delta + T \leq W \leq t) &= Prob(\delta + T \leq W) \, . \, Prob(W \leq t \mid \delta + T \leq W)^1 \\
&= (1 - Prob(W \leq \delta + T)) \, . \, \frac{Prob(W \leq t \; AND \; \delta + T \leq W)}{Prob(\delta + T \leq W)} \\
&= (1 - F_1(\delta + T)) \, . \, \frac{F_2(t) - F_2(\delta + T)}{1 - F_2(\delta + T)}
\end{aligned}$$

$$\delta + T \leq t \leq \delta + 2T$$

$$\begin{aligned}
Prob(\delta + 2T \leq W \leq t) &= Prob(\delta + 2T \leq W) \, . \, Prob(W \leq t \mid \delta + 2T \leq W) \\
&= (1 - Prob(W \leq \delta + T) - Prob(\delta + T \leq W \leq \delta + 2T)) \, . \, \frac{Prob(W \leq t \; AND \; \delta + 2T \leq W)}{Prob(\delta + 2T \leq W)} \\
&= (1 - F_1(\delta + T) - \frac{1 - F_1(\delta + T)}{1 - F_2(\delta + T)} . F_2(\delta + 2T) - F_2(\delta + T)) \, . \, \frac{F_3(t) - F_3(\delta + 2T)}{1 - F_3(\delta + 2T)}
\end{aligned}$$

$$\delta + 2T \leq t \leq D$$

[1]The first term refers to the time before δ+T, when the priority of the waiting message is one. Thus $F_1(t)$ will be used in that term. The second term refers to the region $\delta + T \leq t \leq \delta + 2T$ in which case $F_2(t)$ will be used.

Those messages will be lost whose waiting times are greater than D.

$$Probability\ of\ message\ loss = Prob(W \geq D)$$

$$= 1 - Prob(W \leq D)$$

$$= 1 - Prob(W \leq \delta + T) - Prob(\delta + T \leq W \leq \delta + 2T) - Prob(\delta + 2T \leq W \leq D)$$

Substituting the values for the probabilities gives us the probability of message loss in a dynamic priority system.

Appendix B

Simulation Code

This appendix contains the source code listings for the pre-processor and the post-processor for the following cases:

- Single priority token ring protocol.
- Multiple priority token ring protocol
- Dynamic priority protocol.

Both the pre-processor (preproc.c) and the post-processor (CONFID.c) are written in C.

```
/*******************************************************
* Module name:          single_priority/preproc.c     *
*                                                     *
* Date last modified:   5 Feb 1989                    *
*                                                     *
* Author:               Bakul Khanna                  *
*                                                     *
* Description:          This program generates GPSS/H *
* code for the single priority token ring protocol.   *
* It prompts the user for parameter entry which include *
* ring configuration, message inter-arrival times,    *
* message service times and message deadlines.        *
* It calls a routine CONFID which calculates the      *
* confidence intervals based on the statistics gathered *
* during simulation.                                  *
*******************************************************/

#include <stdio.h>


#define HEADER  56
#define LAT_BUF 27


int nodes,stn_lat,ser_time,arr_time,deadline,delta;
int arr_ind,sym,ring_ind,grp,ser_ind;
int ring[100][2];
float rho;
FILE *fp;



int range_check(lower,upper)
int lower,upper;
/*
* This routine makes sure that the parameter entered
* lies between its lower and upper limits.
*/
        {
        int entry;
        scanf("%d",&entry);
        while ((entry>upper) || (entry<lower))
                {
                printf("entry not within range. try again\n");
                scanf("%d",&entry);
                }
        return(entry);
        }



expand(num,ind)
int num,ind;
/*
* This routine is part of the declaration for the
* variable REGEN
*/
        {
        int k;
        for (k=1;k<num+1;k++)
                {
                fprintf(fp,"Q%d",(ind*10+k));
                if (k==num)
                        fprintf(fp,">0\n");
                else
                        fprintf(fp,">0+");
                }
        }


header()
/*
* This module generates the header and the declarations
* for the GPSS/H program.
*/
        {
        int i,j;
        fprintf(fp,"*************************************\n");
        fprintf(fp,"*                                   *\n");
        fprintf(fp,"*    %d",nodes);
        fprintf(fp,"  FIXED  LATENCY STATIONS      *\n");
        fprintf(fp,"*                                   *\n");
        fprintf(fp,"*                                   *\n");
        fprintf(fp,"*************************************\n");
        fprintf(fp,"*\n");
        fprintf(fp,"* Single priority Token Ring\n");
        fprintf(fp,"* Single token operation\n");
        fprintf(fp,"* Limited-to-one Service Discipline\n");
        fprintf(fp,"* Regenerative method to calculate confidence intervals\n");
        fprintf(fp,"*\n");
        fprintf(fp,"        SIMULATE   10000S,SAVE      \n");
        fprintf(fp,"        RMULT      ,111111111,333333333,555555555\n");
        fprintf(fp,"        OPERCOL    60\n");
        fprintf(fp,"        REALLOCATE COM,40000\n");
        fprintf(fp,"*\n");
        fprintf(fp,"*\n");
        fprintf(fp,"*       INITIALIZATIONS OF EXPONENTIAL FUNCTIONS AND VARIABLES\n");
        fprintf(fp,"*\n");
        fprintf(fp,"*\n");
        fprintf(fp," EXPO1 FUNCTION   RN2,C24   FOR INTERARRIVAL TIMES\n");
        fprintf(fp,"0.0,0.00/.1,.104/.2,.222/.3,.355/.4,.509/.5,.69/.6,.915\n");
        fprintf(fp,".7,1.2/.75,1.38/.8,1.6/.84,1.83/.88,2.12/.9,2.3\n");
        fprintf(fp,".92,2.52/.94,2.81/.95,2.99/.96,3.2/.97,3.5\n");
        fprintf(fp,".98,3.9/.99,4.6/.995,5.3/.998,6.2/.999,7.0/.9997,8.0\n");
        fprintf(fp,"*\n");

        if (ser_ind == 1)
                {
                fprintf(fp," EXPO2 FUNCTION   RN3,C24   FOR MESSAGE SERVICE TIMES\n");
                fprintf(fp,"0.0,0.00/.1,.104/.2,.222/.3,.355/.4,.509/.5,.69/.6,.915\n");
                fprintf(fp,".7,1.2/.75,1.38/.8,1.6/.84,1.83/.88,2.12/.9,2.3\n");
                fprintf(fp,".92,2.52/.94,2.81/.95,2.99/.96,3.2/.97,3.5\n");
                fprintf(fp,".98,3.9/.99,4.6/.995,5.3/.998,6.2/.999,7.0/.9997,8.0\n");
                }
        fprintf(fp,"*\n");
        if (nodes<=10)
                {
                fprintf(fp," REGEN BVARIABLE  ");
                expand(nodes,0);
                fprintf(fp,"*\n");
                }
        else
                {
                for (j=1;j<(nodes/10)+1;j++)
                        {
                        fprintf(fp," %d",j);
                        fprintf(fp,"     BVARIABLE  ");
                        expand(10,j-1);
                        }
                if (nodes%10 != 0)
                        {
```

```
                fprintf(fp," %d",(nodes/10)+1);
                fprintf(fp,"      BVARIABLE   ");
                expand(nodes%10,nodes/10);
                }
        fprintf(fp," REGEN BVARIABLE   ");
        if (nodes%10 == 0)
                j = nodes/10;
        else
                j = nodes/10 + 1;
        for(i=1;i<j+1;i++)
                {
                fprintf(fp,"BV");
                fprintf(fp,"%d",i);
                if (i != j)
                        fprintf(fp,"+");
                else
                        fprintf(fp,"\n");
                }
        }

for(i=1;i<nodes+1;i++)
        {
        fprintf(fp," %d",i);
        fprintf(fp,"     TABLE      M1,200,100,20\n");
        }

fprintf(fp,"*\n");

if (ser_ind == 1)
        {
        fprintf(fp," SERS  FVARIABLE  %d",ser_time);
        fprintf(fp,"*FN$EXPO2\n");
        }
else
        {
        fprintf(fp," SERS  FVARIABLE  %d\n",ser_time);
        }

fprintf(fp,"*\n");

if (sym == 1)
        {
        if (arr_ind == 1)
                {
                fprintf(fp," ARRS  FVARIABLE  %d",arr_time);
                fprintf(fp,"*FN$EXPO1\n");
                }
        else
                {
                fprintf(fp," ARRS  FVARIABLE  %d\n",arr_time);
                }
        }
else
        {
        for (i=1; i<nodes+1; i++)
                {
                fprintf(fp," ARR%d",i);
                fprintf(fp,"  FVARIABLE  %d",ring[i][2]);
                if (ring[i][1] == 1)
                        fprintf(fp,"*FN$EXPO1\n");
                else
                        fprintf(fp,"\n");
                }
        }
fprintf(fp,"*\n");
}
```

```
setup()
/*
* This routine starts the token rolling and
* directs it to station 1
*/
        {
        fprintf(fp,"*\n");
        fprintf(fp,"*         MAKE THE MODEL ACTIVE\n");
        fprintf(fp,"*\n");
        fprintf(fp,"          GENERATE    1,,,1,1\n");
        fprintf(fp,"          SAVEVALUE   ROUND,%d\n",nodes);
        fprintf(fp,"          SPLIT       1,USE1 \n");
        fprintf(fp,"          TERMINATE   \n");
        }

macro()
/*
* This routine prints out the body of the main macro
*/
        {
        fprintf(fp,"*\n");
        fprintf(fp,"*         MACRO  BEGINS\n");
        fprintf(fp,"*\n");
        fprintf(fp," MAIN     STARTMACRO\n");
        fprintf(fp,"          GENERATE    #F\n");
        fprintf(fp,"          QUEUE       #A\n");
        fprintf(fp,"          LINK        #A,FIFO\n");
        fprintf(fp," CAP#A    SEIZE       TOKEN\n");
        fprintf(fp,"          DEPART      #A\n");
        fprintf(fp,"          TABULATE    #A\n");
        fprintf(fp," NOD#A    SAVEVALUE   STATION,#A\n");
        fprintf(fp,"          TEST NE     X$ROUND,0,NEX#A\n");
        fprintf(fp,"          SAVEVALUE   ROUND-,1\n");
        fprintf(fp,"          ADVANCE     #E\n");
        fprintf(fp,"          TRANSFER    ,NOD#G    \n");
        fprintf(fp," NEX#A    ADVANCE     %d",HEADER);
        fprintf(fp,"\n");
        fprintf(fp,"          SAVEVALUE   HEAD,%d",HEADER);
        fprintf(fp,"+%d",stn_lat);
        fprintf(fp,"*%d\n",nodes);
        fprintf(fp,"          TEST G      V$SERS,X$HEAD,REE#A\n");
        fprintf(fp,"          ADVANCE     V$SERS-X$HEAD\n");
        fprintf(fp," REE#A    SAVEVALUE   ROUND,%d\n",nodes);
        fprintf(fp,"          RELEASE     TOKEN\n");
        fprintf(fp,"          ADVANCE     #E\n");
        fprintf(fp,"          TEST E      BV$REGEN,0,USE#G\n");
        fprintf(fp,"          SPLIT       1,USE#G\n");
        fprintf(fp,"          TERMINATE   1\n");
        fprintf(fp," USE#A    SAVEVALUE   STATION,#A\n");
        fprintf(fp,"          TEST E      1,#A,MEE#A\n");
        fprintf(fp,"          ADVANCE     %d",LAT_BUF);
        fprintf(fp,"\n");
        fprintf(fp," MEE#A    UNLINK      #A,CAP#A,1,,,HEC#A\n");
        fprintf(fp,"          TERMINATE\n");
        fprintf(fp," HEC#A    ADVANCE     #E        \n");
        fprintf(fp,"          TRANSFER    ,USE#G\n");
        fprintf(fp,"          ENDMACRO\n");
        fprintf(fp,"*\n");
        fprintf(fp,"*         MACRO  ENDS\n");
        fprintf(fp,"*\n");
        fprintf(fp,"*\n");
        }
```

```
output()
/*
* This routine is the control program.
* It starts off the GPSS/H program, control
* is transferred to it after each regenerative
* cycle is over when the statistics for that
* cycle are collected and the routine CONFID is called
* after the required number of regenerative
* cycles are over
*/
        {
        fprintf(fp,"*\n");
        fprintf(fp,"*          CONTROL CARDS\n");
        fprintf(fp,"*\n");
        fprintf(fp,"           INTEGER  &I,&J        DUMMY VARIABLE\n");
        fprintf(fp,"           INTEGER  &CYCLE       # OF REGENERATIVE CYCLES\n");
        fprintf(fp,"           INTEGER  &NODE        # OF NODES\n");
        fprintf(fp,"           INTEGER  &TOTAL       TOTAL # OF MESSAGES RUN\n");
        fprintf(fp,"           INTEGER  &MESS        MAX. # OF MESSAGE\n");
        fprintf(fp,"           INTEGER  &N(%d",nodes);
        fprintf(fp,")          NUMBER OF CYCLES RUN FOR EACH NODE\n");
        fprintf(fp,"           REAL     &SUMY(%d",nodes);
        fprintf(fp,")   SUM OF Y\n");
        fprintf(fp,"           REAL     &SUMY2(%d",nodes);
        fprintf(fp,") SUM OF SQUARE OF Y\n");
        fprintf(fp,"           REAL     &SUMA(%d",nodes);
        fprintf(fp,")   SUM OF ALPHA\n");
        fprintf(fp,"           REAL     &SUMA2(%d",nodes);
        fprintf(fp,") SUM OF SQUARE OF ALPHA\n");
        fprintf(fp,"           REAL     &SUMYA(%d",nodes);
        fprintf(fp,") SUM OF ALPHA*Y\n");
        fprintf(fp,"           EXTERNAL &CONFID    C SUBROUTINE\n");
        fprintf(fp,"*\n");
        fprintf(fp,"           LET  &CYCLE=8000\n");
        fprintf(fp,"           LET  &MESS=100000\n");
        fprintf(fp,"           LET  &NODE=%d\n",nodes);
        fprintf(fp,"           START   1,NP\n");
        fprintf(fp,"           UNLIST  CSECHO\n");
        fprintf(fp,"*\n");
        fprintf(fp,"* START  RUNNING\n");
        fprintf(fp,"*\n");
        fprintf(fp,"           LET   &I=1\n");
        fprintf(fp,"*          GIVE A FULL REPORT FOR THE LAST CYCLE\n");
        fprintf(fp," AGAIN IF  (&I'L'&CYCLE)\n");
        fprintf(fp,"             START  1,NP\n");
        fprintf(fp,"           ELSE\n");
        fprintf(fp,"             START  1\n");
        fprintf(fp,"           ENDIF\n");
        fprintf(fp,"*\n");
        fprintf(fp,"*    RECORD THE IMPORTANT STATISTICS\n");
        fprintf(fp,"*\n");
        fprintf(fp,"           DO    &J=1,&NODE\n");
        fprintf(fp,"            IF  (TC&J>0)\n");
        fprintf(fp,"              LET &N(&J)=&N(&J)+1\n");
        fprintf(fp,"              LET &SUMY(&J)=&SUMY(&J)+TB&J*TC&J\n");
        fprintf(fp,"              LET &SUMY2(&J)=&SUMY2(&J)+(TB&J*TC&J)*(TB&J*TC&J)\n");
        fprintf(fp,"              LET &SUMA(&J)=&SUMA(&J)+TC&J\n");
        fprintf(fp,"              LET &SUMA2(&J)=&SUMA2(&J)+TC&J*TC&J\n");
        fprintf(fp,"              LET &SUMYA(&J)=&SUMYA(&J)+TC&J*(TC&J*TB&J)\n");
        fprintf(fp,"              LET &TOTAL=&TOTAL+TC&J\n");
        fprintf(fp,"            ENDIF\n");
        fprintf(fp,"           ENDDO\n");
        fprintf(fp,"*\n");
        fprintf(fp,"*\n");
        fprintf(fp,"        RESET   F$TOKEN\n");
        fprintf(fp,"        IF (&TOTAL>&MESS)\n");
        fprintf(fp,"          GOTO  FIN     FINISH THE RUN\n");
        fprintf(fp,"        ENDIF\n");
        fprintf(fp,"        IF (&I<&CYCLE)\n");
        fprintf(fp,"          LET  &I=&I+1\n");
        fprintf(fp,"          GOTO AGAIN\n");
        fprintf(fp,"        ENDIF\n");
        fprintf(fp,"*\n");
        fprintf(fp,"* C SUBROUTINE TO CALCULATE CONFIDENCE INTERVALS\n");
        fprintf(fp,"*\n");

        fprintf(fp," FIN   CALL   &CONFID(&NODE,&N(1),&SUMY(1),&SUMY2(1),_\n");
        fprintf(fp,"&SUMA(1),&SUMA2(1),&SUMYA(1),FR$TOKEN)\n");
        fprintf(fp,"*\n");
        fprintf(fp,"*\n");
        fprintf(fp,"        END\n");
        }


main()
/*
* This module prompts the user to enter various parameters
*/
{
int i,j;
printf("Number of stations ?\n");
nodes = range_check(2,80);
printf("Station Latency ?\n");
stn_lat = range_check(1,20);
printf("Service Time: 1.Exponential 2.Constant\n");
ser_ind = range_check(1,2);
if (ser_ind == 1)
        printf("Exponential service time ?\n");
else
        printf("Constant service time ?\n");
ser_time = range_check(1, 500);
printf("1.Symmetric ring 2.Assymetric ring\n");
sym = range_check(1,2);
if (sym ==1)
        {
        printf("Inter-arrival time distribution: 1.Exponential 2.Constant\n");
        arr_ind = range_check(1,2);
        if (arr_ind == 1)
                {
                printf("Exponential interarrival time ?\n");
                arr_time = range_check(1,500000);
                }
        else
                {
                printf("Constant interarrival time ?\n");
                arr_time = range_check(1,500000);
                }
        }
else
        {
        ring_ind = 1;
        while (ring_ind < nodes+1)
        {
        printf("1.Individual entry 2.Group entry ?\n");
        grp = range_check(1,2);
        if (grp ==1)
                {
                printf("Node %d",ring_ind);
                printf(": Inter-arrival time distribution: 1.Exponential 2.Constant ?\n");
```

```
            arr_ind = range_check(1,2);
            ring[ring_ind][1] = arr_ind;
            if (arr_ind == 1)
                    {
                    printf("Node %d",ring_ind);
                    printf(": Exponential interarrival time ?\n");
                    arr_time = range_check(1,500000);
                    ring[ring_ind][2] = arr_time;
                    }
            else
                    {
                    printf("Node %d",ring_ind);
                    printf(": Constant interarrival time ?\n");
                    arr_time = range_check(1,500000);
                    ring[ring_ind][2] = arr_time;
                    }
            ring_ind = ring_ind + 1;
            }
    else
            {
            printf("group size ?\n");
            grp = range_check(1,nodes-ring_ind+1);
            printf("Nodes %d",ring_ind);
            printf("-%d",ring_ind+grp-1);
            printf(": Inter-arrival time distribution: 1.Exponential 2.Constant ?\n");
            arr_ind = range_check(1,2);
            for (j=ring_ind; j<ring_ind+grp; j++)
                    ring[j][1] = arr_ind;
            if (arr_ind == 1)
                    {
                    printf("Node %d",ring_ind);
                    printf("-%d",ring_ind+grp-1);
                    printf(": Exponential interarrival time ?\n");
                    arr_time = range_check(1,500000);
                    for (j=ring_ind; j<ring_ind+grp; j++)
                            ring[j][2] = arr_time;
                    }
            else
                    {
                    printf("Node %d",ring_ind);
                    printf("-%d",ring_ind+grp-1);
                    printf(": Constant interarrival time ?\n");
                    arr_time = range_check(1,500000);
                    for (j=ring_ind; j<ring_ind+grp; j++)
                            ring[j][2] = arr_time;
                    }
            ring_ind = ring_ind + grp;
            }
            }
    }


/* calculate value of rho */
if (sym == 1)
        {
        rho = (float)nodes*ser_time/arr_time;
        }
else
        {
        rho = 0.0;
        for (i=1; i<nodes+1; i++)
                {
                rho = rho + (float)ser_time/ring[i][2];
                }
        }
printf("rho is  %f\n",rho);
```

```
if (rho>1.00)
        {
        printf("WARNING:queues will build up!\n");
        }

/* print gpss code */

if ((fp = fopen("gpss.gps","w"))<0)
        {
        perror("fopen");
        exit(1);
        }

header();
setup();
macro();

fprintf(fp,"*\n");
fprintf(fp,"*      CALL  MACRO\n");
fprintf(fp,"*\n");

/* generate macro calls. One for each station on the ring
* The following are the parameters of the macro call
* #A - Station number
* #B - not used
* #C - Message service time
* #D - not used
* #E - Station latency
* #F - Message inter-arrival time
* #G - Next station on the ring
*/

for (i=1;i<nodes+1;i++)
        {
        fprintf(fp," MAIN  MACRO      %d",i);
        fprintf(fp,",  ,V$SERS");
        fprintf(fp,",  ,%d",stn_lat);
        if (sym == 1)
                fprintf(fp,",V$ARRS");
        else
                fprintf(fp,",V$ARR%d",i);
        if (i==nodes)
                fprintf(fp,",1");
        else
                fprintf(fp,",%d",i+1);
        fprintf(fp,"     STATION %d\n",i);
        if (i==1)
                fprintf(fp,"       UNLIST     MACX\n");
        }
fprintf(fp,"       PAGE\n");

output();
}
```

```
/* This module is meant for the single priority case */

#include <stdio.h>

void CONFID(NODE,N,SUMY,SUMY2,SUMA,SUMA2,SUMYA,UTIL)

/*
*CCCCCCCCCCCCCCCCCCCCCCCCC
*                        C
*      Declarations      C
*                        C
*CCCCCCCCCCCCCCCCCCCCCCCCC
*
*      I,J,F        DUMMY VARIABLES
*      NODE         # OF NODES
*      NN1          N()*(N()-1)
*      TOTAL        TOTAL # OF MESSAGES TO THE SYSTEM
*      UTIL         UTILIZATION OF SERVER
*      Z            A FACTOR NEEDED TO CALCULATE CONFIDENCE INTERVALS
*
*      When  variables below start with a T, they become the
*      variables for all nodes combined.
*
*      N(NODE)      NUMBER OF CYCLES RUN FOR EACH NODE.
*      R()          MEAN WAITING TIME
*      RVAL()       +-INTERVAL FOR CONFIDENCE INTERVALS
*      S2()         VARIANCE OF WAITING TIME
*      S11()        VARIANCE OF Y
*      S22()        VARIANCE OF ALPHA
*      S12()        COVARIANCE OF ALPHA AND Y
*      SUMY(NODE)   SUM OF Y
*      SUMY2(NODE)  SUM OF SQUARE OF Y
*      SUMA(NODE)   SUM OF ALPHA
*      SUMA2(NODE)  SUM OF SQUARE OF ALPHA
*      SUMYA(NODE)  SUM OF ALPHA*Y
*      MEANY()      MEAN OF Y
*      VARIY()      VARIANCE OF Y
*      MEANA()      MEAN OF ALPHA
*      VARIA()      VARIANCE OF ALPHA
*
*
*      NOTE: 1. The factor Z needed to calculate the 90% confidence
*            intervals is  1.645. But if different percentage is
*            used, you could just change the value for Z in the
*            initialization section.
*
*            2. To compile this module, perform the following steps
*               i)   cc -r -c CONFID.c
*               ii)  ld -r CONFID.o -lm -o CONFID.o
*/

int *NODE, *N;
double *SUMY,*SUMY2,*SUMA,*SUMA2,*SUMYA,*UTIL;

        {
        int i, j;

        double tn, tsumy, tsumy2, tsuma, tsuma2;
        double tsumya;

        double z,total;

        int node, n[100];
        double sumy[100],sumy2[100],suma[100],suma2[100],sumya[100],util;

        double tr, trval, ts2, ts11, ts22, ts12;
        double tmeany, tvariy, tmeana, tvaria;

        double nn1;

        double r[100], rval[100], s2[100], s11[100], s22[100], s12[100];
        double meany[100], variy[100], meana[100], varia[100];

        FILE *fp;

/*
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C                                 C
C  INITIALIZATION  OF  VARIABLES  C
C                                 C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
*/

        node = *NODE;
        util = *UTIL;
        for (i=1; i<node+1; i++)
                {
                n[i]     = *(N+i-1);
                sumy[i]  = *(SUMY+i-1);
                sumy2[i] = *(SUMY2+i-1);
                suma[i]  = *(SUMA+i-1);
                suma2[i] = *(SUMA2+i-1);
                sumya[i] = *(SUMYA+i-1);
                }

        if ((fp = fopen("stat","w"))<0)
                {
                perror("fopen");
                exit(1);
                }
        z = 1.645;
        util = util/1000;
        total = 0.0;
        for (i=1; i<node+1; i++)
                {
                r[i] = 0.0;
                rval[i] = 0.0;
                s2[i]=0.0;
                s12[i]=0.0;
                s11[i]=0.0;
                s22[i]=0.0;
                meany[i] = 0.0;
                variy[i] = 0.0;
                meana[i] = 0.0;
                varia[i] = 0.0;
                }


/*
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C                                      C
C  CALCULATE THE IMPORTANT PARAMETERS  C
C                                      C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
*/
        for (i=1; i<node+1; i++)
                {
                total = total + suma[i];
                if (n[i] >= 2)
                        {
                        meany[i] = sumy[i]/n[i];
                        variy[i] = sumy2[i]/n[i] - meany[i]*meany[i];
                        meana[i] = suma[i]/n[i];
                        varia[i] = suma2[i]/n[i] - meana[i]*meana[i];
                        nn1 = n[i] * (n[i]-1);
                        s11[i] = sumy2[i]/(n[i]-1) - sumy[i]*sumy[i]/nn1;
                        s22[i] = suma2[i]/(n[i]-1) - suma[i]*suma[i]/nn1;
                        s12[i] = sumya[i]/(n[i]-1) - sumy[i]*suma[i]/nn1;
                        r[i] = meany[i]/meana[i];
                        s2[i] = s11[i] - 2*r[i]*s12[i] + r[i]*r[i]*s22[i];
                        if ((s2[i]>=0) && (n[i]>=0))
                               rval[i] = z*sqrt(s2[i])/(meana[i]*sqrt(n[i]));
                        }
                }
/*
cccccccccccccccccccccccccccccccccccccccccccccc
C                                            C
C      OUTPUT STATISTICS IN A NEAT FORMAT    C
C                                            C
cccccccccccccccccccccccccccccccccccccccccccccc
*/
        fprintf(fp,"****************************\n");
        fprintf(fp,"*                          *\n");
        fprintf(fp,"*                          *\n");
        fprintf(fp,"*   SUMMARY OF STATISTICS  *\n");
        fprintf(fp,"*   UTILIZATION = %f",util);
        fprintf(fp,"  *\n");
        fprintf(fp,"*                          *\n");
        fprintf(fp,"*                          *\n");
        fprintf(fp,"****************************\n");
        fprintf(fp,"TOTAL # OF MESSAGES %f\n",total);
        fprintf(fp,"Z USED %f\n",z);
        fprintf(fp,"_____________________________________________\n");

        for (i=1; i<node+1; i++)
                {
                printf("STATION # %d\n",i);
                printf("TOTAL # of CYCLES %d\n",n[i]);
                printf("S2 = %f\n",s2[i]);
                printf("S11 = %f\n",s11[i]);
                printf("S22 = %f\n",s22[i]);
                printf("S12 = %f\n",s12[i]);
                printf("SUM OF CYCLE WAITING TIME =  %f\n", sumy[i]);
                printf("SUM OF SQUARES = %f\n",sumy2[i]);
                printf("SUM OF # OF MESSAGES =  %f\n", suma[i]);
                printf("SUM OF SQUARES = %f\n",suma2[i]);
                printf("SUM OF PROD. OF # MESS. AND WAITING TIME =  %f\n", sumya[i]);
                printf("MEAN WAITING TIME / CYCLE = %f\n", meany[i]);
                printf("VARIANCE = %f\n", variy[i]);
                printf("MEAN # OF MESSAGES / CYCLE = %f\n", meana[i]);
                printf("VARIANCE = %f\n", varia[i]);
                printf("MEAN WAITING TIME = %f",r[i]);
                printf("+-%f\n",rval[i]);
                printf("_____________________________________________\n");
                }

/*
C
C      CALCULATE  THE  STATISTICS FOR  IDENTICAL STATIONS
C
*/
        tn     = 0.0;
        tsumy  = 0.0;
        tsumy2 = 0.0;
        tsuma  = 0.0;
        tsuma2 = 0.0;
        tsumya = 0.0;
```

```
tr     = 0.0;
trval  = 0.0;
ts2    = 0.0;
ts11   = 0.0;
ts22   = 0.0;
ts12   = 0.0;
tmeany = 0.0;
tvariy = 0.0;
tmeana = 0.0;
tvaria = 0.0;

for (i=1; i<node+1; i++)
        {
        tn = tn + n[i];
        tsumy = tsumy + sumy[i];
        tsumy2 = tsumy2 + sumy2[i];
        tsuma = tsuma + suma[i];
        tsuma2 = tsuma2 + suma2[i];
        tsumya = tsumya + sumya[i];
        }

if (tn >= 2)
        {
        tmeany = tsumy/tn;
        tvariy = tsumy2/tn - tmeany*tmeany;
        tmeana = tsuma/tn;
        tvaria = tsuma2/tn - tmeana*tmeana;
        nn1 = tn*(tn-1);
        ts11 = tsumy2/(tn-1) - tsumy*tsumy/nn1;
        ts22 = tsuma2/(tn-1) - tsuma*tsuma/nn1;
        ts12 = tsumya/(tn-1) - tsumy*tsuma/nn1;
        tr = tmeany/tmeana;
        ts2 = ts11 - 2*tr*ts12 + tr*tr*ts22;
        if ((ts2>=0) && (tn>=0))
                trval = z*sqrt(ts2)/(tmeana*sqrt(tn));
        }
fprintf(fp,"FOR IDENTICAL STATIONS \n");
fprintf(fp,"TOTAL # of CYCLES %f\n",tn);
fprintf(fp,"S2 = %f\n",ts2);
fprintf(fp,"S11 = %f\n",ts11);
fprintf(fp,"S22 = %f\n",ts22);
fprintf(fp,"S12 = %f\n",ts12);
fprintf(fp,"SUM OF CYCLE WAITING TIME =  %f\n", tsumy);
fprintf(fp,"SUM OF SQUARES = %f\n",tsumy2);
fprintf(fp,"SUM OF # OF MESSAGES =  %f\n",tsuma);
fprintf(fp,"SUM OF SQUARES = %f\n",tsuma2);
fprintf(fp,"SUM OF PROD. OF # MESS. AND WAITING TIME =  %f\n", tsumya);
fprintf(fp,"MEAN WAITING TIME / CYCLE = %f\n", tmeany);
fprintf(fp,"VARIANCE = %f\n", tvariy);
fprintf(fp,"MEAN # OF MESSAGES / CYCLE = %f\n", tmeana);
fprintf(fp,"VARIANCE = %f\n", tvaria);
fprintf(fp,"MEAN WAITING TIME = %f",tr);
fprintf(fp," +-%f\n",trval);
fprintf(fp,"____________________________________________\n");
fclose(fp);
}
```

```
/*****************************************************
* Module name:          priority/mult/preproc.c       *
*                                                     *
* Date last modified:   5 Feb 1989                    *
*                                                     *
* Author:               Bakul Khanna                  *
*                                                     *
* Description:          This program generates GPSS/H *
* code for the multiple priority token ring protocol  *
* with eight levels of priority.                      *
* It prompts the user for parameter entry which include *
* ring configuration, message inter-arrival times,    *
* message service times and message deadlines.        *
* It calls a routine CONFID which calculates the      *
* confidence intervals based on the statistics gathered *
* during simulation.                                  *
*****************************************************/

#include <stdio.h>

#define HEADER   56
#define LAT_BUF  27

int nodes,stn_lat,ser_time,arr_time,deadline,delta;
int arr_ind,sym,ring_ind,grp,que_op,ser_ind;
int ring[100][2];
float rho;
FILE *fp;


int range_check(lower,upper)
int lower,upper;
/*
* This routine makes sure that the parameter entered lies
* between its lower and upper limits.
*/
        {
        int entry;
        scanf("%d",&entry);
        while ((entry>upper) || (entry<lower))
                {
                printf("entry not within range. try again\n");
                scanf("%d",&entry);
                }
        return(entry);
        }


expand(num,ind)
int num,ind;
/*
* This routine is part of the declaration for the
* variable REGEN
*/
        {
        int k;
        for (k=1;k<num+1;k++)
                {
                fprintf(fp,"Q%d",(ind*10+k));
                if (k==num)
                        fprintf(fp,">0\n");
                else
                        fprintf(fp,">0+");
                }
        }
```

```
header()
/*
* This routine generates the header and the declarations
* for the GPSS/H program.
*/
        {
        int i,j;
        fprintf(fp,"***************************************\n");
        fprintf(fp,"*                                     *\n");
        fprintf(fp,"*    %d",nodes);
        fprintf(fp,"  FIXED  LATENCY STATIONS      *\n");
        fprintf(fp,"*                                     *\n");
        fprintf(fp,"*                                     *\n");
        fprintf(fp,"***************************************\n");
        fprintf(fp,"*\n");
        fprintf(fp,"* Multiple priority token ring\n");
        fprintf(fp,"* Single token operation\n");
        fprintf(fp,"* Limited-to-one service Discipline\n");
        fprintf(fp,"* Regenerative method to calculate cofidance intervals\n");
        fprintf(fp,"*\n");
        fprintf(fp,"        SIMULATE   100000S,SAVE      \n");
        fprintf(fp,"        RMULT      ,111111111,333333333,555555555\n");
        fprintf(fp,"        OPERCOL    60\n");
        fprintf(fp,"        REALLOCATE COM,400000\n");
        fprintf(fp,"*\n");
        fprintf(fp,"*\n");
        fprintf(fp,"*       INITIALIZATIONS OF EXPONENTIAL FUNCTIONS AND VARIABLES\n");
        fprintf(fp,"*\n");
        fprintf(fp,"*\n");
        fprintf(fp," EXPO1 FUNCTION   RN2,C24   FOR INTERARRIVAL TIMES\n");
        fprintf(fp,"0.0,0.00/.1,.104/.2,.222/.3,.355/.4,.509/.5,.69/.6,.915\n");
        fprintf(fp,".7,1.2/.75,1.38/.8,1.6/.84,1.83/.88,2.12/.9,2.3\n");
        fprintf(fp,".92,2.52/.94,2.81/.95,2.99/.96,3.2/.97,3.5\n");
        fprintf(fp,".98,3.9/.99,4.6/.995,5.3/.998,6.2/.999,7.0/.9997,8.0\n");
        if (ser_ind == 1)
                {
                fprintf(fp," EXPO2 FUNCTION   RN3,C24   FOR INTERARRIVAL TIMES\n");
                fprintf(fp,"0.0,0.00/.1,.104/.2,.222/.3,.355/.4,.509/.5,.69/.6,.915\n");
                fprintf(fp,".7,1.2/.75,1.38/.8,1.6/.84,1.83/.88,2.12/.9,2.3\n");
                fprintf(fp,".92,2.52/.94,2.81/.95,2.99/.96,3.2/.97,3.5\n");
                fprintf(fp,".98,3.9/.99,4.6/.995,5.3/.998,6.2/.999,7.0/.9997,8.0\n");
                }
        fprintf(fp,"*\n");

        fprintf(fp," PRIORITY FUNCTION RN4,D8\n");
        fprintf(fp,".125,1/.25,2/.375,3/.5,4/.625,5/.75,6/.875,7/1,8\n");

        fprintf(fp,"*\n");
        fprintf(fp,"*\n");
        if (que_op == 1)
                {
                for (j=1;j<nodes+1;j++)
                        {
                        fprintf(fp," %d",j);
                        fprintf(fp,"     BVARIABLE  ");
                        expand(8,j);
                        }
                fprintf(fp," REGEN BVARIABLE  ");
                for(i=1;i<nodes+1;i++)
                        {
                        fprintf(fp,"BV");
                        fprintf(fp,"%d",i);
                        if (i != nodes)
```

```
                                        fprintf(fp,"+");
                                else
                                        fprintf(fp,"\n");
                                }
                }
        else
                {
                if (nodes<=10)
                        {
                        fprintf(fp," REGEN BVARIABLE  ");
                        expand(nodes,0);
                        }
                else
                        {
                        for (j=1;j<(nodes/10)+1;j++)
                                {
                                fprintf(fp," %d",j);
                                fprintf(fp,"     BVARIABLE  ");
                                expand(10,j-1);
                                }
                        if (nodes%10 != 0)
                                {
                                fprintf(fp," %d",(nodes/10)+1);
                                fprintf(fp,"     BVARIABLE  ");
                                expand(nodes%10,nodes/10);
                                }
                        fprintf(fp," REGEN BVARIABLE  ");
                        if (nodes%10 == 0)
                                j = nodes/10;
                        else
                                j = nodes/10 + 1;
                        for(i=1;i<j+1;i++)
                                {
                                fprintf(fp,"BV");
                                fprintf(fp,"%d",i);
                                if (i != j)
                                        fprintf(fp,"+");
                                else
                                        fprintf(fp,"\n");
                                }
                        }
                }

        fprintf(fp,"*\n");

        for(i=1;i<nodes+1;i++)
                {
                for (j=1; j<9; j++)
                        {
                        fprintf(fp," %d",i);
                        fprintf(fp,"%d",j);
                        fprintf(fp,"    TABLE      M1,200,100,20\n");
                        }
                }
        fprintf(fp,"*\n");
        if (ser_ind == 1)
                {
                fprintf(fp," SERS  FVARIABLE  %d",ser_time);
                fprintf(fp,"*FN$EXPO2\n");
                }
        else
                {
                fprintf(fp," SERS  FVARIABLE  %d\n",ser_time);
                }

        fprintf(fp,"*\n");
        if (sym == 1)
                {
                if (arr_ind == 1)
                        {
                        fprintf(fp," ARRS  FVARIABLE  %d",arr_time);
                        fprintf(fp,"*FN$EXPO1\n");
                        }
                else
                        {
                        fprintf(fp," ARRS  FVARIABLE  %d\n",arr_time);
                        }
                }
        else
                {
                for (i=1; i<nodes+1; i++)
                        {
                        fprintf(fp," ARR%d",i);
                        fprintf(fp,"  FVARIABLE  %d",ring[i][2]);
                        if (ring[i][1] == 1)
                                fprintf(fp,"*FN$EXPO1\n");
                        else
                                fprintf(fp,"\n");
                        }
                }
        fprintf(fp,"*\n");
        }


setup()
/*
* This routine contains the initialization macros and
* also contains the code to start the token rolling.
* The token is directed to station 1
*/
        {
        int i;
        fprintf(fp,"*\n");
        fprintf(fp,"*  INIT  MACRO  BEGINS\n");
        fprintf(fp,"*\n");

        fprintf(fp," INIT  STARTMACRO\n");
        fprintf(fp," ZP#A  MATRIX     H,100,2\n");
        fprintf(fp,"       SAVEVALUE  POINTP#A,2,H\n");
        fprintf(fp,"       ENDMACRO\n");

        if (que_op == 1)
                {
                fprintf(fp,"*\n");
                fprintf(fp,"*  QUE  MACRO  BEGINS\n");
                fprintf(fp,"*\n");
                fprintf(fp," QUE   STARTMACRO\n");
                fprintf(fp,"       QUEUE      #A#B\n");
                fprintf(fp,"       LINK       #A#B,FIFO\n");
                fprintf(fp,"       ENDMACRO\n");
                }


        fprintf(fp,"*\n");
        fprintf(fp,"*      MAKE THE MODEL ACTIVE\n");
        fprintf(fp,"*\n");
        fprintf(fp,"       UNLIST     ABS\n");
        fprintf(fp,"       GENERATE   1,,,1,1\n");
        fprintf(fp,"       SAVEVALUE  STATION,1,H\n");
        fprintf(fp,"       SAVEVALUE  TX,0,H\n");
        fprintf(fp,"       SAVEVALUE  PRTY,1,H\n");
        fprintf(fp,"       SAVEVALUE  RR,1,H\n");
        fprintf(fp,"        SAVEVALUE  BUSY,0,H\n");
        fprintf(fp,"        SAVEVALUE  FLAG,0,H\n");
        fprintf(fp,"        UNLIST     MACX\n");
        for(i=1;i<nodes+1;i++)
                {
                fprintf(fp," INIT  MACRO      %d\n",i);
                }
        fprintf(fp,"        SPLIT      1,USE1 \n");
        fprintf(fp,"        TERMINATE  1\n");
        }


macro()
/*
* This routine prints out the body of the main macro
*/
        {
        int i;
        fprintf(fp,"*\n");
        fprintf(fp,"*  MAIN  MACRO  BEGINS\n");
        fprintf(fp,"*\n");

        fprintf(fp," MAIN  STARTMACRO\n");
        fprintf(fp,"       UNLIST     ABS\n");
        fprintf(fp,"       GENERATE   #F\n");
        fprintf(fp,"       ASSIGN     1,FN$PRIORITY\n");

        if (que_op == 1)
                {
                fprintf(fp,"*\n");
                fprintf(fp,"* QUEUE TRANSACTION IN APPROPRIATE QUEUE\n");
                fprintf(fp,"*\n");

                fprintf(fp,"       TEST E     1,P1,AG1#A\n");
                fprintf(fp," QUE   MACRO      #A,1\n");
                fprintf(fp," AG1#A TEST E     2,P1,AG2#A\n");
                fprintf(fp," QUE   MACRO      #A,2\n");
                fprintf(fp," AG2#A TEST E     3,P1,AG3#A\n");
                fprintf(fp," QUE   MACRO      #A,3\n");
                fprintf(fp," AG3#A TEST E     4,P1,AG4#A\n");
                fprintf(fp," QUE   MACRO      #A,4\n");
                fprintf(fp," AG4#A TEST E     5,P1,AG5#A\n");
                fprintf(fp," QUE   MACRO      #A,5\n");
                fprintf(fp," AG5#A TEST E     6,P1,AG6#A\n");
                fprintf(fp," QUE   MACRO      #A,6\n");
                fprintf(fp," AG6#A TEST E     7,P1,AG7#A\n");
                fprintf(fp," QUE   MACRO      #A,7\n");
                fprintf(fp," AG7#A QUEUE      #A8\n");
                fprintf(fp,"       LINK       #A8,FIFO\n");
                }
        else
                {
                fprintf(fp,"       QUEUE      #A\n");
                fprintf(fp,"       LINK       #A,FIFO\n");
                }

        fprintf(fp,"*\n");
        fprintf(fp,"* A TRANSACTION IS READY TO BE TRANSMITTED\n");
        fprintf(fp,"*\n");

        fprintf(fp," CAP#A SEIZE      TOKEN\n");

        fprintf(fp,"*\n");
        fprintf(fp,"* DEPART FROM APPROPRIATE QUEUE\n");
        fprintf(fp,"*\n");
```

```
if (que_op == 1)
        {
        fprintf(fp,"        TEST E      P1,1,AB1#A\n");
        fprintf(fp,"        DEPART      #A1\n");
        fprintf(fp,"        TRANSFER    ,SON#A\n");
        fprintf(fp," AB1#A TEST E      P1,2,AB2#A\n");
        fprintf(fp,"        DEPART      #A2\n");
        fprintf(fp,"        TRANSFER    ,SON#A\n");
        fprintf(fp," AB2#A TEST E      P1,3,AB3#A\n");
        fprintf(fp,"        DEPART      #A3\n");
        fprintf(fp,"        TRANSFER    ,SON#A\n");
        fprintf(fp," AB3#A TEST E      P1,4,AB4#A\n");
        fprintf(fp,"        DEPART      #A4\n");
        fprintf(fp,"        TRANSFER    ,SON#A\n");
        fprintf(fp," AB4#A TEST E      P1,5,AB5#A\n");
        fprintf(fp,"        DEPART      #A5\n");
        fprintf(fp,"        TRANSFER    ,SON#A\n");
        fprintf(fp," AB5#A TEST E      P1,6,AB6#A\n");
        fprintf(fp,"        DEPART      #A6\n");
        fprintf(fp,"        TRANSFER    ,SON#A\n");
        fprintf(fp," AB6#A TEST E      P1,7,AB7#A\n");
        fprintf(fp,"        DEPART      #A7\n");
        fprintf(fp,"        TRANSFER    ,SON#A\n");
        fprintf(fp," AB7#A DEPART      #A8\n");
        }
else
        fprintf(fp,"        DEPART      #A\n");

fprintf(fp,"*\n");
fprintf(fp,"* TABULATE WAITING TIMES\n");
fprintf(fp,"*\n");

fprintf(fp," SON#A TEST E      P1,1,BX1#A\n");
fprintf(fp,"        TABULATE    #A1 \n");
fprintf(fp,"        TRANSFER    ,SIN#A\n");
fprintf(fp," BX1#A TEST E      P1,2,BX2#A\n");
fprintf(fp,"        TABULATE    #A2\n");
fprintf(fp,"        TRANSFER    ,SIN#A\n");
fprintf(fp," BX2#A TEST E      P1,3,BX3#A\n");
fprintf(fp,"        TABULATE    #A3\n");
fprintf(fp,"        TRANSFER    ,SIN#A\n");
fprintf(fp," BX3#A TEST E      P1,4,BX4#A\n");
fprintf(fp,"        TABULATE    #A4\n");
fprintf(fp,"        TRANSFER    ,SIN#A\n");
fprintf(fp," BX4#A TEST E      P1,5,BX5#A\n");
fprintf(fp,"        TABULATE    #A5\n");
fprintf(fp,"        TRANSFER    ,SIN#A\n");
fprintf(fp," BX5#A TEST E      P1,6,BX6#A\n");
fprintf(fp,"        TABULATE    #A6\n");
fprintf(fp,"        TRANSFER    ,SIN#A\n");
fprintf(fp," BX6#A TEST E      P1,7,BX7#A\n");
fprintf(fp,"        TABULATE    #A7\n");
fprintf(fp,"        TRANSFER    ,SIN#A\n");
fprintf(fp," BX7#A TABULATE    #A8    \n");
fprintf(fp," SIN#A SAVEVALUE   BUSY,1,H   \n");
fprintf(fp,"        SAVEVALUE   RR,1,H     \n");
fprintf(fp,"        SAVEVALUE   TX,#A,H    \n");
fprintf(fp,"        ADVANCE     #E\n");
fprintf(fp,"        TRANSFER    ,USE#G\n");

fprintf(fp,"*\n");

fprintf(fp," USE#A TEST E      1,#A,MEE#A\n");
fprintf(fp,"        ADVANCE     %d",LAT_BUF);
fprintf(fp,"\n");
fprintf(fp," MEE#A TEST E      XH$BUSY,1,MON#A\n");
fprintf(fp,"        TEST E      XH$TX,#A,MON#A\n");
fprintf(fp,"        ADVANCE     %d",HEADER);
fprintf(fp,"\n");
fprintf(fp,"        SAVEVALUE   HEAD,%d",HEADER);
fprintf(fp,"+%d",LAT_BUF);
fprintf(fp,"+%d",stn_lat);
fprintf(fp,"*%d\n",nodes);
fprintf(fp,"        TEST GE     V$SERS,X$HEAD,REE#A\n");
fprintf(fp,"        ADVANCE     V$SERS-X$HEAD\n");
fprintf(fp," REE#A RELEASE     TOKEN\n");

fprintf(fp," MON#A TEST NE     BV$REGEN,0,HAP#A\n");
fprintf(fp,"*       SAVEVALUE   STATION,#A,H\n");
fprintf(fp,"*       PRINT       ,,XH\n");
fprintf(fp,"*       PRINT       ,,C\n");
fprintf(fp,"*       PRINT       ,,Q\n");
if (que_op == 1)
        {
        fprintf(fp," HAP#A UNLINK      #A8,NEX#A,1,,,AH1#A\n");
        fprintf(fp,"       TRANSFER    ,BLO#A\n");
        fprintf(fp," AH1#A UNLINK      #A7,NEX#A,1,,,AH2#A\n");
        fprintf(fp,"       TRANSFER    ,BLO#A\n");
        fprintf(fp," AH2#A UNLINK      #A6,NEX#A,1,,,AH3#A\n");
        fprintf(fp,"       TRANSFER    ,BLO#A\n");
        fprintf(fp," AH3#A UNLINK      #A5,NEX#A,1,,,AH4#A\n");
        fprintf(fp,"       TRANSFER    ,BLO#A\n");
        fprintf(fp," AH4#A UNLINK      #A4,NEX#A,1,,,AH5#A\n");
        fprintf(fp,"       TRANSFER    ,BLO#A\n");
        fprintf(fp," AH5#A UNLINK      #A3,NEX#A,1,,,AH6#A\n");
        fprintf(fp,"       TRANSFER    ,BLO#A\n");
        fprintf(fp," AH6#A UNLINK      #A2,NEX#A,1,,,AH7#A\n");
        fprintf(fp,"       TRANSFER    ,BLO#A\n");
        fprintf(fp," AH7#A UNLINK      #A1,NEX#A,1,,,HEC#A\n");
        }
else
        fprintf(fp," HAP#A UNLINK      #A,NEX#A,1,,,HEC#A\n");

fprintf(fp," BLO#A TEST NE     XH$FLAG,0\n");

fprintf(fp,"*\n");
fprintf(fp,"* ABOVE STATEMENT BLOCKS WHILE FLAG IS 0\n");
fprintf(fp,"*\n");

fprintf(fp,"        TEST E      XH$FLAG,1,AGA#A\n");
fprintf(fp,"        SAVEVALUE   FLAG,0,H\n");
fprintf(fp,"        TERMINATE\n");
fprintf(fp," AGA#A SAVEVALUE   FLAG,0,H\n");
fprintf(fp,"        TRANSFER    ,CON#A\n");

fprintf(fp,"*\n");
fprintf(fp,"* NO TRANSACTIONS UNLINKED FROM USER CHAIN\n");
fprintf(fp,"*\n");

fprintf(fp," HEC#A TEST E      XH$BUSY,1,DOW#A\n");
fprintf(fp,"        TEST E      XH$TX,#A,CON#A\n");
fprintf(fp,"        SAVEVALUE   BUSY,0,H\n");
fprintf(fp,"        TEST L      XH$PRTY,XH$RR,PON#A         \n");
fprintf(fp,"        TEST G      XH$PRTY,MH$ZP#A(XH$POINTP#A-1,1),PIK#A\n");
fprintf(fp,"        MSAVEVALUE ZP#A,XH$POINTP#A,1,XH$RR,H\n");
fprintf(fp,"        MSAVEVALUE ZP#A,XH$POINTP#A,2,XH$PRTY,H\n");
fprintf(fp,"        SAVEVALUE   POINTP#A+,1,H\n");
fprintf(fp,"        SAVEVALUE   PRTY,XH$RR,H\n");
fprintf(fp,"        SAVEVALUE   RR,1,H\n");
fprintf(fp,"        TRANSFER    ,PON#A\n");
fprintf(fp," PIK#A SAVEVALUE   PRTY,XH$RR,H\n");
fprintf(fp,"        SAVEVALUE   RR,1,H\n");
fprintf(fp,"        MSAVEVALUE ZP#A,(XH$POINTP#A-1),1,XH$PRTY,H\n");
fprintf(fp," PON#A SAVEVALUE  TX,0,H\n");
fprintf(fp,"        ADVANCE    #E\n");
fprintf(fp,"        TEST E     BV$REGEN,0,USE#G\n");
fprintf(fp,"        SPLIT      1,USE#G\n");
fprintf(fp,"        TERMINATE  1\n");

fprintf(fp,"*\n");
fprintf(fp,"* UNLINKED TRANSACTIONS FROM USER CHAIN COME HERE\n");
fprintf(fp,"*\n");

fprintf(fp," NEX#A TEST E     XH$BUSY,1,PRE#A\n");
fprintf(fp,"        TEST E     XH$TX,#A,SHO#A\n");
fprintf(fp,"        SAVEVALUE  BUSY,0,H\n");
fprintf(fp,"        TEST GE    XH$PRTY,P1,GIN#A\n");
fprintf(fp,"        TEST GE    XH$PRTY,XH$RR,GIN#A\n");
fprintf(fp,"        TEST L     XH$RR,P1,PIN#A\n");
fprintf(fp,"        SAVEVALUE  RR,P1,H\n");
fprintf(fp,"        TRANSFER   ,PIN#A\n");
fprintf(fp," GIN#A TEST L     XH$RR,P1,HOP#A        \n");
fprintf(fp,"        SAVEVALUE  TEMP,P1,H\n");
fprintf(fp,"        TRANSFER   ,HON#A\n");
fprintf(fp," HOP#A SAVEVALUE  TEMP,XH$RR,H\n");
fprintf(fp," HON#A TEST G     XH$PRTY,MH$ZP#A(XH$POINTP#A-1,1),KIK#A\n");
fprintf(fp,"        MSAVEVALUE ZP#A,XH$POINTP#A,1,XH$TEMP,H\n");
fprintf(fp,"        MSAVEVALUE ZP#A,XH$POINTP#A,2,XH$PRTY,H\n");
fprintf(fp,"        SAVEVALUE  POINTP#A+,1,H\n");
fprintf(fp,"        SAVEVALUE  RR,1,H\n");
fprintf(fp,"        SAVEVALUE  PRTY,XH$TEMP,H\n");
fprintf(fp,"        TRANSFER   ,PIN#A\n");
fprintf(fp," KIK#A MSAVEVALUE ZP#A,(XH$POINTP#A-1),1,XH$TEMP,H\n");
fprintf(fp,"        SAVEVALUE  PRTY,XH$TEMP,H\n");
fprintf(fp,"        SAVEVALUE  RR,1,H\n");
fprintf(fp," PIN#A SAVEVALUE  TX,0,H\n");


fprintf(fp," ON#A  SAVEVALUE  FLAG,2,H\n");
fprintf(fp,"        BUFFER\n");
if (que_op == 1)
        {
        fprintf(fp,"        TEST E     1,P1,AD1#A\n");
        fprintf(fp,"        LINK       #A1,LIFO\n");
        fprintf(fp," AD1#A TEST E     2,P1,AD2#A\n");
        fprintf(fp,"        LINK       #A2,LIFO\n");
        fprintf(fp," AD2#A TEST E     3,P1,AD3#A\n");
        fprintf(fp,"        LINK       #A3,LIFO\n");
        fprintf(fp," AD3#A TEST E     4,P1,AD4#A\n");
        fprintf(fp,"        LINK       #A4,LIFO\n");
        fprintf(fp," AD4#A TEST E     5,P1,AD5#A\n");
        fprintf(fp,"        LINK       #A5,LIFO\n");
        fprintf(fp," AD5#A TEST E     6,P1,AD6#A\n");
        fprintf(fp,"        LINK       #A6,LIFO\n");
        fprintf(fp," AD6#A TEST E     7,P1,AD7#A\n");
        fprintf(fp,"        LINK       #A7,LIFO\n");
        fprintf(fp," AD7#A LINK       #A8,LIFO\n");
        fprintf(fp,"        TRANSFER   ,CON#A\n");
        }
else
        fprintf(fp,"        LINK       #A,LIFO\n");
fprintf(fp,"\n");
fprintf(fp,"* DOWNGRADE PRIORITY\n");
fprintf(fp,"\n");

fprintf(fp," DOW#A TEST E     XH$PRTY,MH$ZP#A(XH$POINTP#A-1,1),CON#A\n");
fprintf(fp,"        TEST G     XH$RR,MH$ZP#A(XH$POINTP#A-1,2),LUG#A\n");
fprintf(fp,"        SAVEVALUE  PRTY,XH$RR,H\n");
```

```
        fprintf(fp,"       MSAVEVALUE ZP#A,(XH$POINTP#A-1),1,XH$PRTY,H\n");
        fprintf(fp,"       SAVEVALUE  RR,1,H\n");
        fprintf(fp,"       TRANSFER   ,CON#A\n");
        fprintf(fp," LUG#A SAVEVALUE  PRTY,MH$ZP#A(XH$POINTP#A-1,2),H\n");
        fprintf(fp,"       SAVEVALUE  POINTP#A-,1,H\n");
        fprintf(fp," CON#A ADVANCE    #E\n");
        fprintf(fp,"       TRANSFER   ,USE#G\n");

        fprintf(fp,"\n");
        fprintf(fp,"* FREE TOKEN\n");
        fprintf(fp,"\n");

        fprintf(fp," PRE#A TEST E     XH$PRTY,MH$ZP#A(XH$POINTP#A-1,1),HUG#A\n")

        fprintf(fp,"       TEST L     P1,MH$ZP#A(XH$POINTP#A-1,1),HUG#A\n");
        fprintf(fp,"       TEST GE    XH$RR,P1,HOH#A\n");
        fprintf(fp,"       SAVEVALUE  BIG,XH$RR,H\n");
        fprintf(fp,"       TRANSFER   ,DOH#A\n");
        fprintf(fp," HOH#A SAVEVALUE  BIG,P1,H\n");
        fprintf(fp," DOH#A TEST G     XH$BIG,MH$ZP#A(XH$POINTP#A-1,2),LOG#A\n");
        fprintf(fp,"       SAVEVALUE  PRTY,XH$BIG,H\n");
        fprintf(fp,"       MSAVEVALUE ZP#A,(XH$POINTP#A-1),1,XH$BIG,H\n");
        fprintf(fp,"       SAVEVALUE  RR,1,H\n");
        fprintf(fp,"       TRANSFER   ,HUG#A\n");
        fprintf(fp," LOG#A SAVEVALUE  PRTY,MH$ZP#A(XH$POINTP#A-1,2),H\n");
        fprintf(fp,"       SAVEVALUE  POINTP#A-,1,H\n");
        fprintf(fp,"       SAVEVALUE  RR,XH$BIG,H\n");
        fprintf(fp," HUG#A TEST GE    P1,XH$PRTY,PHO#A\n");
        fprintf(fp,"       SAVEVALUE  FLAG,1,H\n");
        fprintf(fp,"       BUFFER\n");
        fprintf(fp,"       TRANSFER   ,CAP#A\n");
        fprintf(fp," PHO#A TEST G     XH$PRTY,MH$ZP#A(XH$POINTP#A-1,1),LNK#A\n");
        fprintf(fp," SHO#A TEST L     XH$RR,P1,LNK#A\n");
        fprintf(fp,"       SAVEVALUE  RR,P1,H\n");
        fprintf(fp," LNK#A TRANSFER   ,ON#A\n");
        fprintf(fp,"       ENDMACRO\n");
        fprintf(fp,"*\n");
        fprintf(fp,"*      MACRO  ENDS\n");
        fprintf(fp,"*\n");
        }


output()
/*
* This routine is the control program.
* It starts off the GPSS/H program, control it
* transferred to it after each regenerative
* cycle is over when the statistics for that cycle
* are collected. The routine CONFID is called
* after the required number of regenerative cycles
* are over
*/
        {
        int i;
        fprintf(fp,"*\n");
        fprintf(fp,"*      CONTROL CARDS\n");
        fprintf(fp,"*\n");
        fprintf(fp,"*\n");
        fprintf(fp,"       INTEGER  &I,&J,&K,&L\n");
        fprintf(fp,"       INTEGER  &CYCLE\n");
        fprintf(fp,"       INTEGER  &NODE \n");
        fprintf(fp,"       INTEGER  &TOTAL\n");
        fprintf(fp,"       INTEGER  &MESS  \n");
        i = nodes*10 + 10;
        fprintf(fp,"       INTEGER  &N(%d",i);
        fprintf(fp,")\n");
        fprintf(fp,"        REAL     &SUMY(%d",i);
        fprintf(fp,")\n");
        fprintf(fp,"        REAL     &SUMY2(%d",i);
        fprintf(fp,")\n");
        fprintf(fp,"        REAL     &SUMA(%d",i);
        fprintf(fp,")\n");
        fprintf(fp,"        REAL     &SUMA2(%d",i);
        fprintf(fp,")\n");
        fprintf(fp,"        REAL     &SUMYA(%d",i);
        fprintf(fp,")\n");
        fprintf(fp,"        REAL     &LINDELAY\n");
        fprintf(fp,"        EXTERNAL &CONFID\n");

        fprintf(fp,"        LET &CYCLE=8000\n");
        fprintf(fp,"        LET &MESS=100000\n");
        fprintf(fp,"        LET &NODE=%d\n",nodes);
        fprintf(fp,"        START   1,NP\n");
        fprintf(fp,"        UNLIST  CSECHO\n");

        fprintf(fp,"        LET     &I=1\n");
        fprintf(fp," AGAIN IF (&I'L'&CYCLE)\n");
        fprintf(fp,"             START  1,NP\n");
        fprintf(fp,"        ELSE\n");
        fprintf(fp,"             START  1\n");
        fprintf(fp,"        ENDIF\n");

        fprintf(fp,"*\n");
        fprintf(fp,"* RECORD THE IMPORTANT STATISTICS\n");
        fprintf(fp,"* \n");
        fprintf(fp,"        DO      &K=1,8\n");
        fprintf(fp,"           DO      &J=1,&NODE\n");
        fprintf(fp,"              LET      &L=&J*10+&K\n");
        fprintf(fp,"              IF (TC&L>0)\n");
        fprintf(fp,"              LET &N(&L)=&N(&L)+1\n");
        fprintf(fp,"              LET &SUMY(&L)=&SUMY(&L)+TB&L*TC&L\n");
        fprintf(fp,"              LET &SUMY2(&L)=&SUMY2(&L)+(TB&L*TC&L)*(TB&L*TC&L)\n");
        fprintf(fp,"              LET &SUMA(&L)=&SUMA(&L)+TC&L\n");
        fprintf(fp,"              LET &SUMA2(&L)=&SUMA2(&L)+TC&L*TC&L\n");
        fprintf(fp,"              LET &SUMYA(&L)=&SUMYA(&L)+TC&L*(TB&L*TC&L)\n");
        fprintf(fp,"              LET &TOTAL=&TOTAL+TC&L\n");
        fprintf(fp,"              ENDIF\n");
        fprintf(fp,"           ENDDO\n");
        fprintf(fp,"        ENDDO\n");

        fprintf(fp,"        RESET      F$TOKEN\n");

        fprintf(fp,"        IF (&TOTAL>&MESS)\n");
        fprintf(fp,"           GOTO  FIN\n");
        fprintf(fp,"           ENDIF\n");
        fprintf(fp,"        IF (&I<&CYCLE)\n");
        fprintf(fp,"           LET &I=&I+1\n");
        fprintf(fp,"           GOTO  AGAIN\n");
        fprintf(fp,"           ENDIF\n");
        fprintf(fp,"* C ROUTINE TO CALCULATE CONFIDENCE INTERVALS\n");
        fprintf(fp," FIN   DO          &K=1,8\n");
        fprintf(fp,"              LET      &L=10+&K\n");
        fprintf(fp,"              CALL       &CONFID(&NODE,&N(&L),&SUMY(&L),&SUMY2(&L),_\n");
        fprintf(fp,"&SUMA(&L),&SUMA2(&L),&SUMYA(&L),FR$TOKEN)\n");
        fprintf(fp,"        ENDDO\n");

        fprintf(fp,"        END\n");
        }

main()
/*
```

```
*This module prompts the user to enter various paramaters
*/
{
int i,j;
printf("Number of stations ?\n");
nodes = range_check(2,80);
printf("Station latency ?\n");
stn_lat = range_check(1,20);
printf("Service Time: 1.Exponential 2.Constant\n");
ser_ind = range_check(1,2);
if (ser_ind == 1)
        printf("Exponential service time ?\n");
else
        printf("Constant service time ?\n");
ser_time = range_check(1, 500);

printf("1.Symmetric ring 2.Assymetric ring\n");
sym = range_check(1,2);
if (sym ==1)
        {
        printf("Inter-arrival time distribution: 1.Exponential 2.Constant\n");
        arr_ind = range_check(1,2);
        if (arr_ind == 1)
                {
                printf("Exponential interarrival time ?\n");
                arr_time = range_check(1,500000);
                }
        else
                {
                printf("Constant interarrival time ?\n");
                arr_time = range_check(1,500000);
                }
        }
else
        {
        ring_ind = 1;
        while (ring_ind < nodes+1)
        {
        printf("1.Individual entry 2.Group entry ?\n");
        grp = range_check(1,2);
        if (grp ==1)
                {
                printf("Node %d",ring_ind);
                printf(": Inter-arrival time distribution: 1.Exponential 2.Constant ?\n");
                arr_ind = range_check(1,2);
                ring[ring_ind][1] = arr_ind;
                if (arr_ind == 1)
                        {
                        printf("Node %d",ring_ind);
                        printf(": Exponential interarrival time ?\n");
                        arr_time = range_check(1,500000);
                        ring[ring_ind][2] = arr_time;
                        }
                else
                        {
                        printf("Node %d",ring_ind);
                        printf(": Constant interarrival time ?\n");
                        arr_time = range_check(1,500000);
                        ring[ring_ind][2] = arr_time;
                        }
                ring_ind = ring_ind + 1;
                }
        else
                {
                printf("group size ?\n");
                grp = range_check(1,nodes-ring_ind+1);
                printf("Nodes %d",ring_ind);
                printf("-%d",ring_ind+grp-1);
                printf(": Inter-arrival time distribution: 1.Exponential 2.Constant ?\n");
                arr_ind = range_check(1,2);
                for (j=ring_ind; j<ring_ind+grp; j++)
                        ring[j][1] = arr_ind;
                if (arr_ind == 1)
                        {
                        printf("Node %d",ring_ind);
                        printf("-%d",ring_ind+grp-1);
                        printf(": Exponential interarrival time ?\n");
                        arr_time = range_check(1,500000);
                        for (j=ring_ind; j<ring_ind+grp; j++)
                                ring[j][2] = arr_time;
                        }
                else
                        {
                        printf("Node %d",ring_ind);
                        printf("-%d",ring_ind+grp-1);
                        printf(": Constant interarrival time ?\n");
                        arr_time = range_check(1,500000);
                        for (j=ring_ind; j<ring_ind+grp; j++)
                                ring[j][2] = arr_time;
                        }
                ring_ind = ring_ind + grp;
                }
                }
        }

printf("Queue operation: 1.Multi-queue 2.Single-queue ?\n");
que_op = range_check(1,2);

/* calculate value of rho */
if (sym == 1)
        {
        rho = (float)nodes*ser_time/arr_time;
        }
else
        {
        rho = 0.0;
        for (i=1; i<nodes+1; i++)
                {
                rho = rho + (float)ser_time/ring[i][2];
                }
        }
printf("rho is  %f\n",rho);
if (rho>1.00)
        {
        printf("WARNING:queues will build up!\n");
        }

/* print gpss code */

if ((fp = fopen("gpss.gps","w"))<0)
        {
        perror("fopen");
        exit(1);
        }
header();
setup();
macro();

fprintf(fp,"*\n");
fprintf(fp,"*      CALL  MACRO\n");
fprintf(fp,"*\n");
```

```
/* Generate macro calls. One for each station on the ring.
*  The following are the parameters of the macro call
* #A - Station number
* #B - not used
* #C - Message service time
* #D - not used
* #E - Station latency
* #F - Meggase inter-arrival time
* #G - Next station on the ring
*/
for (i=1;i<nodes+1;i++)
        {
        fprintf(fp," MAIN  MACRO      %d",i);
        fprintf(fp,", ,V$SERS");
        fprintf(fp,", ,%d",stn_lat);
        if (sym == 1)
                fprintf(fp,",V$ARRS");
        else
                printf(fp,",V$ARR%d",i);
        if (i==nodes)
                fprintf(fp,",1");
        else
                fprintf(fp,",%d",i+1);
        fprintf(fp,"     STATION %d\n",i);
        if (i==1)
                fprintf(fp,"       UNLIST     MACX\n");
        }
fprintf(fp,"       PAGE\n");

output();

}
```

```
/* This module is meant for the multiple priority case */

#include <stdio.h>

void CONFID(NODE,N,SUMY,SUMY2,SUMA,SUMA2,SUMYA,UTIL)

/*
*CCCCCCCCCCCCCCCCCCCCCCCCC
*                        C
*       Declarations     C
*                        C
*CCCCCCCCCCCCCCCCCCCCCCCCC
*
*       I,J,F        DUMMY VARIABLES
*       NODE         # OF NODES
*       NN1          N()*(N()-1)
*       TOTAL        TOTAL # OF MESSAGES TO THE SYSTEM
*       UTIL         UTILIZATION OF SERVER
*       Z            A FACTOR NEEDED TO CALCULATE CONFIDENCE INTERVALS
*
*       When  variables below start with a T, they become the
*       variables for all nodes combined.
*
*       N(NODE)      NUMBER OF CYCLES RUN FOR EACH NODE
*       R()          MEAN WAITING TIME
*       RVAL()       +-INTERVAL FOR CONFIDENCE INTERVALS
*       S2()         VARIANCE OF WAITING TIME
*       S11()        VARIANCE OF Y
*       S22()        VARIANCE OF ALPHA
*       S12()        COVARIANCE OF ALPHA AND Y
*       SUMY(NODE)   SUM OF Y
*       SUMY2(NODE)  SUM OF SQUARE OF Y
*       SUMA(NODE)   SUM OF ALPHA
*       SUMA2(NODE)  SUM OF SQUARE OF ALPHA
*       SUMYA(NODE)  SUM OF ALPHA*Y
*       MEANY()      MEAN OF Y
*       VARIY()      VARIANCE OF Y
*       MEANA()      MEAN OF ALPHA
*       VARIA()      VARIANCE OF ALPHA
*
*
*       NOTE: 1. The factor Z needed to calculate the 90% confidence
*             intervals is  1.645. But if different percentage is
*             used, you could just change the value for Z in the
*             initialization section.
*
*             2. Perform the following steps to compile this program
*                i)  cc -r -c CONFID.c
*                ii) ld -r CONFID.o -lm -o CONFID.o
*
*/

int *NODE, *N;
double *SUMY,*SUMY2,*SUMA,*SUMA2,*SUMYA,*UTIL;

        {
        int i, j;

        double tn, tsumy, tsumy2, tsuma, tsuma2;
        double tsumya;

        double z,total;

        int node, n[200];
        double sumy[200],sumy2[200],suma[200],suma2[200],sumya[200],util;
        double tr, trval, ts2, ts11, ts22, ts12;
        double tmeany, tvariy, tmeana, tvaria;

        double nn1;

        double r[200], rval[200], s2[200], s11[200], s22[200], s12[200];
        double meany[200], variy[200], meana[200], varia[200];

        FILE *fp;

/*
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C                                 C
C  INITIALIZATION  OF  VARIABLES  C
C                                 C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
*/

        printf("confid called\n");
        node = *NODE;
        util = *UTIL;
        for (i=1; i<node+1; i++)
                {
                n[i]      = *(N+(i-1)*10);
                sumy[i]   = *(SUMY+(i-1)*10);
                sumy2[i]  = *(SUMY2+(i-1)*10);
                suma[i]   = *(SUMA+(i-1)*10);
                suma2[i]  = *(SUMA2+(i-1)*10);
                sumya[i]  = *(SUMYA+(i-1)*10);
                }
        if ((fp = fopen("stat","a"))<0)
                {
                perror("fopen");
                exit(1);
                }
        z = 1.645;
        util = util/1000;
        total = 0.0;
        for (i=1; i<node+1; i++)
                {
                r[i] = 0.0;
                rval[i] = 0.0;
                s2[i]=0.0;
                s12[i]=0.0;
                s11[i]=0.0;
                s22[i]=0.0;
                meany[i] = 0.0;
                variy[i] = 0.0;
                meana[i] = 0.0;
                varia[i] = 0.0;
                }


/*
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C                                      C
C  CALCULATE THE IMPORTANT PARAMETERS  C
C                                      C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
*/
        for (i=1; i<node+1; i++)
                {
                total = total + suma[i];
                if (n[i] >= 2)
                        {
                        meany[i] = sumy[i]/n[i];
                        variy[i] = sumy2[i]/n[i] - meany[i]*meany[i];
                        meana[i] = suma[i]/n[i];
                        varia[i] = suma2[i]/n[i] - meana[i]*meana[i];
                        nn1 = n[i] * (n[i]-1);
                        s11[i] = sumy2[i]/(n[i]-1) - sumy[i]*sumy[i]/nn1;
                        s22[i] = suma2[i]/(n[i]-1) - suma[i]*suma[i]/nn1;
                        s12[i] = sumya[i]/(n[i]-1) - sumy[i]*suma[i]/nn1;
                        r[i] = meany[i]/meana[i];
                        s2[i] = s11[i] - 2*r[i]*s12[i] + r[i]*r[i]*s22[i];
                        if ((s2[i]>=0) && (n[i]>=0))
                                rval[i] = z*sqrt(s2[i])/(meana[i]*sqrt(n[i]));
                        }
                }
/*
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C                                            C
C       OUTPUT STATISTICS IN A NEAT FORMAT   C
C                                            C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
*/
        fprintf(fp,"****************************\n");
        fprintf(fp,"*                          *\n");
        fprintf(fp,"*                          *\n");
        fprintf(fp,"*   SUMMARY OF STATISTICS  *\n");
        fprintf(fp,"*   UTILIZATION = %f",util);
        fprintf(fp,"  *\n");
        fprintf(fp,"*                          *\n");
        fprintf(fp,"*                          *\n");
        fprintf(fp,"****************************\n");
        fprintf(fp,"TOTAL # OF MESSAGES %f\n",total);
        fprintf(fp,"Z USED %f\n",z);
        fprintf(fp,"____________________________________________\n");

        for (i=1; i<node+1; i++)
                {
                fprintf(fp,"STATION # %d\n",i);
                fprintf(fp,"TOTAL # of CYCLES %d\n",n[i]);
                fprintf(fp,"S2 = %f\n",s2[i]);
                fprintf(fp,"S11 = %f\n",s11[i]);
                fprintf(fp,"S22 = %f\n",s22[i]);
                fprintf(fp,"S12 = %f\n",s12[i]);
                fprintf(fp,"SUM OF CYCLE WAITING TIME =  %f\n", sumy[i]);
                fprintf(fp,"SUM OF SQUARES = %f\n",sumy2[i]);
                fprintf(fp,"SUM OF # OF MESSAGES =  %f\n", suma[i]);
                fprintf(fp,"SUM OF SQUARES = %f\n",suma2[i]);
                fprintf(fp,"SUM OF PROD. OF # MESS. AND WAITING TIME =  %f\n", sumya[i]);
                fprintf(fp,"MEAN WAITING TIME / CYCLE = %f\n", meany[i]);
                fprintf(fp,"VARIANCE = %f\n", variy[i]);
                fprintf(fp,"MEAN # OF MESSAGES / CYCLE = %f\n", meana[i]);
                fprintf(fp,"VARIANCE = %f\n", varia[i]);
                fprintf(fp,"MEAN WAITING TIME = %f",r[i]);
                fprintf(fp,"+-%f\n",rval[i]);
                fprintf(fp,"____________________________________________\n");
                }

/*
C
C       CALCULATE  THE  STATISTICS FOR  IDENTICAL STATIONS
C
*/
        tn     = 0.0;
        tsumy  = 0.0;
        tsumy2 = 0.0;
        tsuma  = 0.0;
        tsuma2 = 0.0;
        tsumya = 0.0;
```

```
tr     = 0.0;
trval  = 0.0;
ts2    = 0.0;
ts11   = 0.0;
ts22   = 0.0;
ts12   = 0.0;
tmeany = 0.0;
tvariy = 0.0;
tmeana = 0.0;
tvaria = 0.0;

for (i=1; i<node+1; i++)
        {
        tn = tn + n[i];
        tsumy = tsumy + sumy[i];
        tsumy2 = tsumy2 + sumy2[i];
        tsuma = tsuma + suma[i];
        tsuma2 = tsuma2 + suma2[i];
        tsumya = tsumya + sumya[i];
        }

if (tn >= 2)
        {
        tmeany = tsumy/tn;
        tvariy = tsumy2/tn - tmeany*tmeany;
        tmeana = tsuma/tn;
        tvaria = tsuma2/tn - tmeana*tmeana;
        nn1 = tn*(tn-1);
        ts11 = tsumy2/(tn-1) - tsumy*tsumy/nn1;
        ts22 = tsuma2/(tn-1) - tsuma*tsuma/nn1;
        ts12 = tsumya/(tn-1) - tsumy*tsuma/nn1;
        tr = tmeany/tmeana;
        ts2 = ts11 - 2*tr*ts12 + tr*tr*ts22;
        if ((ts2>=0) && (tn>=0))
                trval = z*sqrt(ts2)/(tmeana*sqrt(tn));
        }
fprintf(fp,"FOR IDENTICAL STATIONS \n");
fprintf(fp,"TOTAL # of CYCLES %f\n",tn);
fprintf(fp,"S2 = %f\n",ts2);
fprintf(fp,"S11 = %f\n",ts11);
fprintf(fp,"S22 = %f\n",ts22);
fprintf(fp,"S12 = %f\n",ts12);
fprintf(fp,"SUM OF CYCLE WAITING TIME =  %f\n", tsumy);
fprintf(fp,"SUM OF SQUARES = %f\n",tsumy2);
fprintf(fp,"SUM OF # OF MESSAGES =  %f\n",tsuma);
fprintf(fp,"SUM OF SQUARES = %f\n",tsuma2);
fprintf(fp,"SUM OF PROD. OF # MESS. AND WAITING TIME =  %f\n", tsumya);
fprintf(fp,"MEAN WAITING TIME / CYCLE = %f\n", tmeany);
fprintf(fp,"VARIANCE = %f\n", tvariy);
fprintf(fp,"MEAN # OF MESSAGES / CYCLE = %f\n", tmeana);
fprintf(fp,"VARIANCE = %f\n", tvaria);
fprintf(fp,"MEAN WAITING TIME = %f",tr);
fprintf(fp," +-%f\n",trval);
fprintf(fp,"______________________________________________\n");
fclose(fp);
}
```

```
/*********************************************************
* Module name:          dynamic/preproc.c               *
*                                                       *
* Date last modified:   5 Feb 1989                      *
*                                                       *
* Author:               Bakul Khanna                    *
*                                                       *
* Description:          This program generates GPSS/H.  *
* code for the dynamic priority protocol.               *
* It prompts the user for parameter entry which include *
* ring configuration, message inter-arrival times,      *
* message service times and message deadlines.          *
* It calls a routine CONFID which calculates the        *
* confidence intervals based on the statistics gathered *
* during simulation.                                    *
*********************************************************/

#include <stdio.h>

#define HEADER  56
#define LAT_BUF 27

int nodes,stn_lat,ser_time,arr_time,delta,deadline;
int arr_ind,sym,ring_ind,grp,func,ser_ind,dead_ind;
int ring[100][2];
float rho;
FILE *fp;



int range_check(lower,upper)
int lower,upper;
/*
* This routine makes sure that the parameter entered
* lies between its lower and upper limits.
*/
        {
        int entry;
        scanf("%d",&entry);
        while ((entry>upper) || (entry<lower))
                {
                printf("entry not within range. try again\n");
                scanf("%d",&entry);
                }
        return(entry);
        }


expand(num,ind)
int num,ind;
/*
* This routine is part of the declaration for the
* variable REGEN
*/
        {
        int k;
        for (k=1;k<num+1;k++)
                {
                fprintf(fp,"CH%d",(ind*10+k));
                if (k==num)
                        fprintf(fp,">0\n");
                else
                        fprintf(fp,">0+");
                }
        }
```

```
header()
/*
* This routine generates the header and the declarations
* for the GPSS/H program.
*/
        {
        int i,j;
        fprintf(fp,"**************************************\n");
        fprintf(fp,"*                                    *\n");
        fprintf(fp,"*    %d",nodes);
        fprintf(fp,"  FIXED  LATENCY STATIONS      *\n");
        fprintf(fp,"*                                    *\n");
        fprintf(fp,"*                                    *\n");
        fprintf(fp,"**************************************\n");
        fprintf(fp,"*\n");
        fprintf(fp,"* Dynamic priority protocol\n");
        fprintf(fp,"* Single token operation\n");
        fprintf(fp,"* Limited-to-one service distribution\n");
        fprintf(fp,"* Regenerative method to calculate confidence intervals\n");
        fprintf(fp,"*\n");
        fprintf(fp,"       SIMULATE   100000S,SAVE      \n");
        fprintf(fp,"       RMULT      ,111111111,333333333,555555555\n");
        fprintf(fp,"       OPERCOL    60\n");
        fprintf(fp,"       REALLOCATE COM,40000\n");
        fprintf(fp,"*\n");
        fprintf(fp,"*       INITIALIZATIONS OF EXPONENTIAL FUNCTIONS AND VARIABLES\n");
        fprintf(fp,"*\n");
        fprintf(fp," EXPO1 FUNCTION   RN2,C24   FOR INTERARRIVAL TIMES\n");
        fprintf(fp,"0.0,0.00/.1,.104/.2,.222/.3,.355/.4,.509/.5,.69/.6,.915\n");
        fprintf(fp,".7,1.2/.75,1.38/.8,1.6/.84,1.83/.88,2.12/.9,2.3\n");
        fprintf(fp,".92,2.52/.94,2.81/.95,2.99/.96,3.2/.97,3.5\n");
        fprintf(fp,".98,3.9/.99,4.6/.995,5.3/.998,6.2/.999,7.0/.9997,8.0\n");

        fprintf(fp,"*\n");

        if (dead_ind == 1)
                {
                fprintf(fp," DL    FUNCTION   RN3,C24   FOR DEADLINES\n");
                fprintf(fp,"0.0,0.00/.1,.104/.2,.222/.3,.355/.4,.509/.5,.69/.6,.915\n");
                fprintf(fp,".7,1.2/.75,1.38/.8,1.6/.84,1.83/.88,2.12/.9,2.3\n");
                fprintf(fp,".92,2.52/.94,2.81/.95,2.99/.96,3.2/.97,3.5\n");
                fprintf(fp,".98,3.9/.99,4.6/.995,5.3/.998,6.2/.999,7.0/.9997,8.0\n");
                }

        fprintf(fp,"*\n");

        if (ser_ind == 1)
                {
                fprintf(fp," EXPO2 FUNCTION   RN3,C24   FOR MESSAGE SERVICE TIMES\n");
                fprintf(fp,"0.0,0.00/.1,.104/.2,.222/.3,.355/.4,.509/.5,.69/.6,.915\n");
                fprintf(fp,".7,1.2/.75,1.38/.8,1.6/.84,1.83/.88,2.12/.9,2.3\n");
                fprintf(fp,".92,2.52/.94,2.81/.95,2.99/.96,3.2/.97,3.5\n");
                fprintf(fp,".98,3.9/.99,4.6/.995,5.3/.998,6.2/.999,7.0/.9997,8.0\n");
                }

        fprintf(fp,"*\n");
        fprintf(fp,"*\n");
        if (nodes<=10)
                {
                fprintf(fp," REGEN BVARIABLE  ");
                expand(nodes,0);
                }
        else
                {
```

```
        for (j=1;j<(nodes/10)+1;j++)
                {
                fprintf(fp," %d",j);
                fprintf(fp,"     BVARIABLE  ");
                expand(10,j-1);
                }
        if (nodes%10 != 0)
                {
                fprintf(fp," %d",(nodes/10)+1);
                fprintf(fp,"     BVARIABLE  ");
                expand(nodes%10,nodes/10);
                }
        fprintf(fp," REGEN BVARIABLE  ");
        if (nodes%10 == 0)
                j = nodes/10;
        else
                j = nodes/10 + 1;
        for(i=1;i<j+1;i++)
                {
                fprintf(fp,"BV");
                fprintf(fp,"%d",i);
                if (i != j)
                        fprintf(fp,"+");
                else
                        fprintf(fp,"\n");
                }
        }
fprintf(fp,"*\n");
fprintf(fp,"*\n");

for (i=1;i<nodes+1;i++)
        {
        fprintf(fp," %d",i);
        fprintf(fp,"     TABLE      M1,300,100,20\n");
        }

fprintf(fp,"*\n");

if (ser_ind == 1)
        {
        fprintf(fp," SERS  FVARIABLE  %d",ser_time);
        fprintf(fp,"*FN$EXPO2\n");
        }
else
        {
        fprintf(fp," SERS  FVARIABLE  %d\n",ser_time);
        }

fprintf(fp,"*\n");

if (sym == 1)
        {
        if (arr_ind == 1)
                {
                fprintf(fp," ARRS  FVARIABLE  %d",arr_time);
                fprintf(fp,"*FN$EXPO1\n");
                }
        else
                {
                fprintf(fp," ARRS  FVARIABLE  %d\n",arr_time);
                }
        }
else
        {
        for (i=1; i<nodes+1; i++)
                {
                        fprintf(fp," ARR%d",i);
                        fprintf(fp,"  FVARIABLE  %d",ring[i][2]);
                        if (ring[i][1] == 1)
                                fprintf(fp,"*FN$EXPO1\n");
                        else
                                fprintf(fp,"\n");
                        }
                }
        fprintf(fp,"*\n");
        }


setup()
        {
/*
* This routine performs the necessary initializations,
* starts the token rolling and directs it to
* station 1
*/
        int i;
        fprintf(fp,"*\n");
        fprintf(fp,"*  INIT  MACRO  BEGINS\n");
        fprintf(fp,"*\n");

        fprintf(fp," INIT  STARTMACRO\n");
        fprintf(fp," ZP#A  MATRIX     H,100,2\n");
        fprintf(fp,"       SAVEVALUE  POINTP#A,2,H\n");
        fprintf(fp,"       ENDMACRO\n");

        fprintf(fp,"*\n");
        fprintf(fp,"*      MAKE THE MODEL ACTIVE\n");
        fprintf(fp,"*\n");
        fprintf(fp,"       UNLIST     ABS\n");
        fprintf(fp,"       GENERATE   1,,,1,1\n");
        fprintf(fp,"       SAVEVALUE  TX,0,H\n");
        fprintf(fp,"       SAVEVALUE  PRTY,1,H\n");
        fprintf(fp,"       SAVEVALUE  RR,1,H\n");
        fprintf(fp,"       SAVEVALUE  BUSY,0,H\n");
        fprintf(fp,"       SAVEVALUE  FLAG,0,H\n");
        fprintf(fp,"       SAVEVALUE  DELTA,%d",delta);
        fprintf(fp,",H\n");
        fprintf(fp,"       SAVEVALUE  1,0,H\n");
        fprintf(fp,"       UNLIST     MACX\n");
        for(i=1;i<nodes+1;i++)
                {
                fprintf(fp," INIT  MACRO      %d\n",i);
                }
        fprintf(fp,"       SPLIT      1,USE1 \n");
        fprintf(fp,"       TERMINATE  1\n");
        }


macro()
/*
* This routine prints out the body of the main macro
*/
        {
        int i;
        fprintf(fp,"*\n");
        fprintf(fp,"*  MAIN  MACRO  BEGINS\n");
        fprintf(fp,"*\n");

        fprintf(fp," MAIN  STARTMACRO\n");
        fprintf(fp,"       UNLIST     ABS\n");
        fprintf(fp,"       GENERATE   #F\n");
```

```
if (dead_ind == 1)
        fprintf(fp,"        ASSIGN     2,#B*FN$DL\n");
else
        fprintf(fp,"        ASSIGN     2,#B\n");
fprintf(fp,"        ASSIGN     3,(P2-M1)\n");
fprintf(fp,"        ASSIGN     4,(CH#A+1)\n");
fprintf(fp,"        SAVEVALUE  GO,0\n");
fprintf(fp," OOK#A LOOP       4,OOP#A\n");
fprintf(fp,"        LINK       #A,3\n");
fprintf(fp," OOP#A UNLINK     #A,OOM#A,1\n");
fprintf(fp,"        TEST NE    X$GO,0\n");
fprintf(fp,"        SAVEVALUE  GO,0\n");
fprintf(fp,"        TRANSFER   ,OOK#A\n");
fprintf(fp," OOM#A ASSIGN     3,(P2-M1)\n");
fprintf(fp,"        SAVEVALUE  GO,1\n");
fprintf(fp,"        BUFFER\n");
fprintf(fp,"        LINK       #A,FIFO\n");

fprintf(fp,"* A TRANSACTION IS READY TO BE TRANSMITTED\n");

fprintf(fp," CAP#A SEIZE      TOKEN\n");

fprintf(fp,"* TABULATE WAITING TIMES\n");

fprintf(fp,"        TABULATE   #A\n");
fprintf(fp,"        SAVEVALUE  BUSY,1,H    \n");
fprintf(fp,"        SAVEVALUE  RR,1,H      \n");
fprintf(fp,"        SAVEVALUE  TX,#A,H     \n");
fprintf(fp,"        ADVANCE    #E\n");
fprintf(fp,"        TRANSFER   ,USE#G\n");
fprintf(fp," USE#A TEST E     #A,1,MEE#A\n");
fprintf(fp,"        ADVANCE    %d",LAT_BUF);
fprintf(fp,"\n");
fprintf(fp," MEE#A TEST E     XH$BUSY,1,MON#A\n");
fprintf(fp,"        TEST E     XH$TX,#A,MON#A\n");
fprintf(fp,"        ADVANCE    %d",HEADER);
fprintf(fp,"\n");
fprintf(fp,"        SAVEVALUE  HEAD,%d",HEADER);
fprintf(fp,"+%d",LAT_BUF);
fprintf(fp,"+1*%d\n",nodes);
fprintf(fp,"        TEST GE    V$SERS,X$HEAD,REE#A\n");
fprintf(fp,"        ADVANCE    V$SERS-X$HEAD\n");
fprintf(fp," REE#A RELEASE    TOKEN\n");
fprintf(fp," MON#A UNLINK     #A,NEX#A,1,,,HEC#A\n");
fprintf(fp,"        TEST NE    XH$FLAG,0\n");

fprintf(fp,"* ABOVE STATEMENT BLOCKS WHILE FLAG IS 0\n");

fprintf(fp,"        TEST E     XH$FLAG,1,AGA#A\n");
fprintf(fp,"        SAVEVALUE  FLAG,0,H\n");
fprintf(fp,"        TERMINATE\n");
fprintf(fp," AGA#A TEST E     XH$FLAG,2,AGB#A\n");
fprintf(fp,"        SAVEVALUE  FLAG,0,H\n");
fprintf(fp,"        TRANSFER   ,CON#A\n");
fprintf(fp," AGB#A SAVEVALUE  FLAG,0,H\n");
fprintf(fp,"        TRANSFER   ,HEC#A\n");

fprintf(fp,"* NO TRANSACTIONS UNLINKED FROM USER CHAIN\n");

fprintf(fp," HEC#A TEST E     XH$BUSY,1,DOW#A\n");
fprintf(fp,"        TEST E     XH$TX,#A,CON#A\n");
fprintf(fp,"        SAVEVALUE  BUSY,0,H\n");
fprintf(fp,"        TEST L     XH$PRTY,XH$RR,PON#A         \n");
fprintf(fp,"        TEST G     XH$PRTY,MH$ZP#A(XH$POINTP#A-1,1),PIK#A\n");
fprintf(fp,"        MSAVEVALUE ZP#A,XH$POINTP#A,1,XH$RR,H\n");
fprintf(fp,"        MSAVEVALUE ZP#A,XH$POINTP#A,2,XH$PRTY,H\n");
fprintf(fp,"        SAVEVALUE  POINTP#A+,1,H\n");
fprintf(fp,"        SAVEVALUE  PRTY,XH$RR,H\n");
fprintf(fp,"        SAVEVALUE  RR,1,H\n");
fprintf(fp,"        TRANSFER   ,PON#A\n");
fprintf(fp," PIK#A SAVEVALUE  PRTY,XH$RR,H\n");
fprintf(fp,"        SAVEVALUE  RR,1,H\n");
fprintf(fp,"        MSAVEVALUE ZP#A,(XH$POINTP#A-1),1,XH$PRTY,H\n");
fprintf(fp," PON#A SAVEVALUE  TX,0,H\n");
fprintf(fp,"        ADVANCE    #E\n");
fprintf(fp,"        TEST E     BV$REGEN,0,USE#G\n");
fprintf(fp,"        SPLIT      1,USE#G\n");
fprintf(fp,"        TERMINATE  1\n");

fprintf(fp,"* UNLINKED TRANSACTIONS FROM USER CHAIN COME HERE\n");

fprintf(fp," NEX#A TEST E     XH$BUSY,1,PRE#A\n");
fprintf(fp,"        TEST G     M1,P2,OK#A\n");
fprintf(fp,"        SAVEVALUE  1+,1,H\n");
fprintf(fp,"        UNLINK     #A,NEX#A,1,,,PEC#A\n");
fprintf(fp,"        TERMINATE\n");
fprintf(fp," PEC#A SAVEVALUE  FLAG,3,H\n");
fprintf(fp,"        BUFFER\n");
fprintf(fp,"        TERMINATE\n");
fprintf(fp," BAC#A TEST GE    XH$PRTY,P1,GIN#A\n");
fprintf(fp,"        TEST GE    XH$PRTY,XH$RR,GIN#A\n");
fprintf(fp,"        TEST L     XH$RR,P1,PIN#A        \n");
fprintf(fp,"        SAVEVALUE  RR,P1,H\n");
fprintf(fp,"        TRANSFER   ,PIN#A\n");
fprintf(fp," GIN#A TEST L     XH$RR,P1,HOP#A        \n");
fprintf(fp,"        SAVEVALUE  TEMP,P1,H\n");
fprintf(fp,"        TRANSFER   ,HON#A\n");
fprintf(fp," HOP#A SAVEVALUE  TEMP,XH$RR,H\n");
fprintf(fp," HON#A TEST G     XH$PRTY,MH$ZP#A(XH$POINTP#A-1,1),KIK#A\n");
fprintf(fp,"        MSAVEVALUE ZP#A,XH$POINTP#A,1,XH$TEMP,H\n");
fprintf(fp,"        MSAVEVALUE ZP#A,XH$POINTP#A,2,XH$PRTY,H\n");
fprintf(fp,"        SAVEVALUE  POINTP#A+,1,H\n");
fprintf(fp,"        SAVEVALUE  RR,1,H\n");
fprintf(fp,"        SAVEVALUE  PRTY,XH$TEMP,H\n");
fprintf(fp,"        TRANSFER   ,PIN#A\n");
fprintf(fp," KIK#A MSAVEVALUE ZP#A,(XH$POINTP#A-1),1,XH$TEMP,H\n");
fprintf(fp,"        SAVEVALUE  RR,1,H\n");
fprintf(fp,"        SAVEVALUE  PRTY,XH$TEMP,H\n");
fprintf(fp," PIN#A SAVEVALUE  TX,0,H\n");
fprintf(fp,"        SAVEVALUE  BUSY,0,H\n");

fprintf(fp," ON#A  SAVEVALUE  FLAG,2,H\n");
fprintf(fp,"        BUFFER\n");

fprintf(fp,"        LINK       #A,LIFO\n");

fprintf(fp,"* DOWNGRADE PRIORITY\n");

fprintf(fp," DOW#A TEST E     XH$PRTY,MH$ZP#A(XH$POINTP#A-1,1),CON#A\n");
fprintf(fp,"        TEST G     XH$RR,MH$ZP#A(XH$POINTP#A-1,2),LUG#A\n");
fprintf(fp,"        SAVEVALUE  PRTY,XH$RR,H\n");
fprintf(fp,"        MSAVEVALUE ZP#A,(XH$POINTP#A-1),1,XH$PRTY,H\n");
fprintf(fp,"        SAVEVALUE  RR,1,H\n");
fprintf(fp,"        TRANSFER   ,CON#A\n");
fprintf(fp," LUG#A SAVEVALUE  PRTY,MH$ZP#A(XH$POINTP#A-1,2),H\n");
fprintf(fp,"        SAVEVALUE  POINTP#A-,1,H\n");

fprintf(fp," CON#A ADVANCE    #E\n");
fprintf(fp,"        TRANSFER   ,USE#G\n");

fprintf(fp,"* MAKE RESERVATION\n");
fprintf(fp," PHO#A TEST G     XH$PRTY,MH$ZP#A(XH$POINTP#A-1,1),ON#A\n");
fprintf(fp," SHO#A TEST L      XH$RR,P1,ON#A\n");
fprintf(fp,"       SAVEVALUE   RR,P1,H\n");
fprintf(fp,"       TRANSFER    ,ON#A\n");

fprintf(fp,"* FREE TOKEN\n");

fprintf(fp," PRE#A TEST E      XH$PRTY,MH$ZP#A(XH$POINTP#A-1,1),BON#A\n");
fprintf(fp,"       TEST L      P1,MH$ZP#A(XH$POINTP#A-1,1),BON#A\n");
fprintf(fp,"       TEST GE     XH$RR,P1,HOH#A\n");
fprintf(fp,"       SAVEVALUE   BIG,XH$RR,H\n");
fprintf(fp,"       TRANSFER    ,DOH#A\n");
fprintf(fp," HOH#A SAVEVALUE   BIG,P1,H\n");
fprintf(fp," DOH#A TEST G      XH$BIG,MH$ZP#A(XH$POINTP#A-1,2),LOG#A\n");
fprintf(fp,"       SAVEVALUE   PRTY,XH$BIG,H\n");
fprintf(fp,"       MSAVEVALUE  ZP#A,(XH$POINTP#A-1),1,XH$BIG,H\n");
fprintf(fp,"       SAVEVALUE   RR,1,H\n");
fprintf(fp,"       TRANSFER    ,BON#A\n");
fprintf(fp," LOG#A SAVEVALUE   PRTY,MH$ZP#A(XH$POINTP#A-1,2),H\n");
fprintf(fp,"       SAVEVALUE   POINTP#A-,1,H\n");
fprintf(fp,"       SAVEVALUE   RR,XH$BIG,H\n");
fprintf(fp,"\n");
fprintf(fp,"* TEST IF DEADLINE HAS BEEN MISSED !\n");
fprintf(fp,"\n");
fprintf(fp," BON#A TEST G      M1,P2,OK#A\n");
fprintf(fp,"       SAVEVALUE   FLAG,1,H\n");
fprintf(fp,"       BUFFER\n");
fprintf(fp,"       SAVEVALUE   1+,1,H\n");
fprintf(fp,"       TEST E      BV$REGEN,0,MON#A\n");
fprintf(fp,"       SPLIT       1,MON#A\n");
fprintf(fp,"       TERMINATE   1\n");
fprintf(fp," OK#A  TEST GE     P2,XH$DELTA,SAM#A\n");
fprintf(fp,"       TEST L      M1,P2-XH$DELTA,SAM#A\n");
fprintf(fp,"       ASSIGN      1,1\n");
fprintf(fp,"       TRANSFER    ,LAM#A\n");
fprintf(fp," SAM#A SAVEVALUE   DIS,M1-P2+XH$DELTA,H\n");
if (func == 1)
        fprintf(fp,"       TEST LE     XH$DIS,XH$DELTA*1/8,DA1#A\n");
else
        fprintf(fp,"       TEST LE     XH$DIS,XH$DELTA*1/2,DA1#A\n");
fprintf(fp,"       ASSIGN      1,1\n");
fprintf(fp,"       TRANSFER    ,LAM#A\n");

if (func == 1)
        fprintf(fp," DA1#A TEST LE     XH$DIS,XH$DELTA*2/8,DA2#A\n");
else
        fprintf(fp," DA1#A TEST LE     XH$DIS,XH$DELTA*3/4,DA2#A\n");
fprintf(fp,"       ASSIGN      1,2\n");
fprintf(fp,"       TRANSFER    ,LAM#A\n");
if (func == 1)
        fprintf(fp," DA2#A TEST LE     XH$DIS,XH$DELTA*3/8,DA3#A\n");
else
        fprintf(fp," DA2#A TEST LE     XH$DIS,XH$DELTA*7/8,DA3#A\n");
fprintf(fp,"       ASSIGN      1,3\n");
fprintf(fp,"       TRANSFER    ,LAM#A\n");
if (func == 1)
        fprintf(fp," DA3#A TEST LE     XH$DIS,XH$DELTA*4/8,DA4#A\n");
else
        fprintf(fp," DA3#A TEST LE     XH$DIS,XH$DELTA*15/16,DA4#A\n");
fprintf(fp,"       ASSIGN      1,4\n");
fprintf(fp,"       TRANSFER    ,LAM#A\n");
if (func == 1)
        fprintf(fp," DA4#A TEST LE     XH$DIS,XH$DELTA*5/8,DA5#A\n");
else
        fprintf(fp," DA4#A TEST LE     XH$DIS,XH$DELTA*31/32,DA5#A\n");
fprintf(fp,"       ASSIGN      1,5\n");
fprintf(fp,"       TRANSFER    ,LAM#A\n");
```

```
        if (func == 1)
                fprintf(fp," DA5#A TEST LE    XH$DIS,XH$DELTA*6/8,DA6#A\n");
        else
                fprintf(fp," DA5#A TEST LE    XH$DIS,XH$DELTA*63/64,DA6#A\n");
        fprintf(fp,"       ASSIGN     1,6\n");
        fprintf(fp,"       TRANSFER   ,LAM#A\n");
        if (func == 1)
                fprintf(fp," DA6#A TEST LE    XH$DIS,XH$DELTA*7/8,DA7#A\n");
        else
                fprintf(fp," DA6#A TEST LE    XH$DIS,XH$DELTA*127/128,DA7#A\n");
        fprintf(fp,"       ASSIGN     1,7\n");
        fprintf(fp,"       TRANSFER   ,LAM#A\n");
        if (func == 1)
                fprintf(fp," DA7#A ASSIGN     1,8\n");
        else
                fprintf(fp," DA7#A ASSIGN     1,8\n");
        fprintf(fp," LAM#A TEST E     XH$BUSY,1,DAM#A\n");
        fprintf(fp,"       TEST E     XH$TX,#A,SHO#A\n");
        fprintf(fp,"       TRANSFER   ,BAC#A\n");
        fprintf(fp," DAM#A TEST GE    P1,XH$PRTY,PHO#A\n");
        fprintf(fp,"       SAVEVALUE  FLAG,1,H\n");
        fprintf(fp,"       BUFFER\n");
        fprintf(fp,"       TRANSFER   ,CAP#A     \n");
        fprintf(fp,"       ENDMACRO\n");
        fprintf(fp,"*\n");
        fprintf(fp,"*      MACRO  ENDS\n");
        fprintf(fp,"*\n");
        }


output()
/*
* This routine is the control program.
* It starts of the GPSS/H program, control is
* transferred to it after each regenerative cycle
* is over. This routine then collects statistics
* generated during the previous cycle.
* The external routine CONFID is called
* when the required number of regenerative
* cycles are over
*/
        {
        fprintf(fp,"*\n");
        fprintf(fp,"*      CONTROL CARDS\n");
        fprintf(fp,"*\n");
        fprintf(fp,"*\n");
        fprintf(fp,"       INTEGER  &I,&J\n");
        fprintf(fp,"       INTEGER  &CYCLE\n");
        fprintf(fp,"       INTEGER  &NODE \n");
        fprintf(fp,"       INTEGER  &TOTAL\n");
        fprintf(fp,"       INTEGER  &MESS  \n");
        fprintf(fp,"       INTEGER  &N(%d",nodes);
        fprintf(fp,")\n");
        fprintf(fp,"       REAL     &SUMY(%d",nodes);
        fprintf(fp,")\n");
        fprintf(fp,"       REAL     &SUMY2(%d",nodes);
        fprintf(fp,")\n");
        fprintf(fp,"       REAL     &SUMA(%d",nodes);
        fprintf(fp,")\n");
        fprintf(fp,"       REAL     &SUMA2(%d",nodes);
        fprintf(fp,")\n");
        fprintf(fp,"       REAL     &SUMYA(%d",nodes);
        fprintf(fp,")\n");
        fprintf(fp,"       REAL     &LINDELAY\n");
        fprintf(fp,"       REAL     &PERCEN\n");
        fprintf(fp,"       REAL     &SAV\n");
        fprintf(fp,"       REAL     &DEAD\n");
        fprintf(fp,"       REAL     &DELT\n");

        fprintf(fp,"       EXTERNAL &CONFID\n");

        fprintf(fp,"       LET &CYCLE=8000\n");
        fprintf(fp,"       LET &MESS=100000\n");
        fprintf(fp,"       LET &NODE=%d\n",nodes);
        fprintf(fp,"       LET &DELT=%d\n",delta);
        fprintf(fp,"       LET &DEAD=%d\n",deadline);
        fprintf(fp,"       START   1,NP\n");
        fprintf(fp,"       UNLIST  CSECHO\n");

        fprintf(fp,"       LET     &I=1\n");
        fprintf(fp," AGAIN IF  (&I'L'&CYCLE)\n");
        fprintf(fp,"            START  1,NP\n");
        fprintf(fp,"       ELSE\n");
        fprintf(fp,"            START  1\n");
        fprintf(fp,"       ENDIF\n");


        fprintf(fp,"*\n");
        fprintf(fp,"* RECORD THE IMPORTANT STATISTICS\n");
        fprintf(fp,"* \n");
        fprintf(fp,"         DO      &J=1,&NODE\n");
        fprintf(fp,"             IF  (TC&J>0)\n");
        fprintf(fp,"             LET &N(&J)=&N(&J)+1\n");
        fprintf(fp,"             LET &SUMY(&J)=&SUMY(&J)+TB&J*TC&J\n");
        fprintf(fp,"             LET &SUMY2(&J)=&SUMY2(&J)+(TB&J*TC&J)*(TB&J*TC&J)\n");
        fprintf(fp,"             LET &SUMA(&J)=&SUMA(&J)+TC&J\n");
        fprintf(fp,"             LET &SUMA2(&J)=&SUMA2(&J)+TC&J*TC&J\n");
        fprintf(fp,"             LET &SUMYA(&J)=&SUMYA(&J)+TC&J*(TB&J*TC&J)\n");
        fprintf(fp,"             LET &TOTAL=&TOTAL+TC&J\n");
        fprintf(fp,"             ENDIF\n");
        fprintf(fp,"          ENDDO\n");

        fprintf(fp,"       LET &SAV=XH1\n");
        fprintf(fp,"       LET &PERCEN=&SAV*100/(&TOTAL+&SAV)\n");

        fprintf(fp,"       RESET      F$TOKEN\n");

        fprintf(fp,"       IF  (&TOTAL>&MESS)\n");
        fprintf(fp,"          GOTO  FIN\n");
        fprintf(fp,"          ENDIF\n");
        fprintf(fp,"       IF  (&I<&CYCLE)\n");
        fprintf(fp,"          LET &I=&I+1\n");
        fprintf(fp,"          GOTO  AGAIN\n");
        fprintf(fp,"          ENDIF\n");
        fprintf(fp,"* C ROUTINE TO CALCULATE CONFIDENCE INTERVALS\n");
        fprintf(fp,"*\n");
        fprintf(fp," FIN      CALL     &CONFID(&NODE,&N(1),&SUMY(1),_\n");
        fprintf(fp,"&SUMY2(1),&SUMA(1),&SUMA2(1),&SUMYA(1),FR$TOKEN,_\n");
        fprintf(fp,"&PERCEN,&DEAD,&DELT)\n");
        fprintf(fp,"*\n");
        fprintf(fp,"       END\n");
        }

main()
/*
* This module prompts the user to enter various parameters
*/
{
int i,j;
printf("Number of stations ?\n");
nodes = range_check(2,50);
```

```
printf("Line delay ?\n");
stn_lat = range_check(1,20);
printf("Service Time: 1.Exponential 2.Constant\n");
ser_ind = range_check(1,2);
if (ser_ind == 1)
        printf("Exponential service time ?\n");
else
        printf("Constant service time ?\n");
ser_time = range_check(1, 500);

printf("1.Symmetric ring 2.Assymetric ring\n");
sym = range_check(1,2);
if (sym ==1)
        {
        printf("Inter-arrival time distribution: 1.Exponential 2.Constant\n");
        arr_ind = range_check(1,2);
        if (arr_ind == 1)
                {
                printf("Exponential interarrival time ?\n");
                arr_time = range_check(1,500000);
                }
        else
                {
                printf("Constant interarrival time ?\n");
                arr_time = range_check(1,500000);
                }
        }
else
        {
        ring_ind = 1;
        while (ring_ind < nodes+1)
        {
        printf("1.Individual entry 2.Group entry ?\n");
        grp = range_check(1,2);
        if (grp ==1)
                {
                printf("Node %d",ring_ind);
                printf(": Inter-arrival time distribution: 1.Exponential 2.Constant ?\n");
                arr_ind = range_check(1,2);
                ring[ring_ind][1] = arr_ind;
                if (arr_ind == 1)
                        {
                        printf("Node %d",ring_ind);
                        printf(": Exponential interarrival time ?\n");
                        arr_time = range_check(1,500000);
                        ring[ring_ind][2] = arr_time;
                        }
                else
                        {
                        printf("Node %d",ring_ind);
                        printf(": Constant interarrival time ?\n");
                        arr_time = range_check(1,500000);
                        ring[ring_ind][2] = arr_time;
                        }
                ring_ind = ring_ind + 1;
                }
        else
                {
                printf("group size ?\n");
                grp = range_check(1,nodes-ring_ind+1);
                printf("Nodes %d",ring_ind);
                printf("-%d",ring_ind+grp-1);
                printf(": Inter-arrival time distribution: 1.Exponential 2.Constant ?\n");
                arr_ind = range_check(1,2);
                for (j=ring_ind; j<ring_ind+grp; j++)
                        ring[j][1] = arr_ind;
                if (arr_ind == 1)
                        {
                        printf("Node %d",ring_ind);
                        printf("-%d",ring_ind+grp-1);
                        printf(": Exponential interarrival time ?\n");
                        arr_time = range_check(1,500000);
                        for (j=ring_ind; j<ring_ind+grp; j++)
                                ring[j][2] = arr_time;
                        }
                else
                        {
                        printf("Node %d",ring_ind);
                        printf("-%d",ring_ind+grp-1);
                        printf(": Constant interarrival time ?\n");
                        arr_time = range_check(1,500000);
                        for (j=ring_ind; j<ring_ind+grp; j++)
                                ring[j][2] = arr_time;
                        }
                ring_ind = ring_ind + grp;
                }
                }
        }
printf("Deadlines: 1.Exponentially distributed 2.Constant\n");
dead_ind = range_check(1,2);
if (dead_ind == 1)
        printf("mean for exponentially distributed deadlines ?\n");
else
        printf("constant deadline ?\n");
deadline = range_check(1,100000);
printf("delta ?\n");
delta = range_check(0,100000);
printf("Priority Function: 1.Linear 2.Non-linear ?\n");
func = range_check(1,2);

/* calculate value of rho */
if (sym == 1)
        {
        rho = (float)nodes*ser_time/arr_time;
        }
else
        {
        rho = 0.0;
        for (i=1; i<nodes+1; i++)
                {
                rho = rho + (float)ser_time/ring[i][2];
                }
        }
printf("rho is  %f\n",rho);
if (rho>1.00)
        {
        printf("WARNING:queues will build up!\n");
        }

/* print gpss code */

if ((fp = fopen("gpss.gps","w"))<0)
        {
        perror("fopen");
        exit(1);
        }
header();
setup();
macro();

fprintf(fp,"*\n");
fprintf(fp,"*      CALL  MACRO\n");
fprintf(fp,"*\n");

/* Generate macro calls. One for each station
* on the ring. The following are the parameters
* for the macro calls.
* #A - Station number
* #B - Deadline
* #C - Message service time
* #D - not used
* #E - Station latency
* #F - Message inter-arrival time
* #G - Next station
*/

for (i=1;i<nodes+1;i++)
        {
        fprintf(fp," MAIN  MACRO      %d",i);
        fprintf(fp,",%d",deadline);
        fprintf(fp,",V$SERS");
        fprintf(fp,", ,%d",stn_lat);
        if (sym == 1)
                fprintf(fp,",V$ARRS");
        else
                fprintf(fp,",V$ARR%d",i);
        if (i==nodes)
                fprintf(fp,",1");
        else
                fprintf(fp,",%d",i+1);
        fprintf(fp,"     STATION %d\n",i);
        if (i==1)
                fprintf(fp,"       UNLIST     MACX\n");
        }
fprintf(fp,"       PAGE\n");

output();

}
```

```
/* This module is meant for the dynamic case*/

#include <stdio.h>

void CONFID(NODE,N,SUMY,SUMY2,SUMA,SUMA2,SUMYA,UTIL,PERCEN,DL,DELT)

/*
*CCCCCCCCCCCCCCCCCCCCCCCCC
*                        C
*      Declarations      C
*                        C
*CCCCCCCCCCCCCCCCCCCCCCCCC
*
*      I,J,F         DUMMY VARIABLES
*      NODE          # OF NODES
*      NN1           N()*(N()-1)
*      TOTAL         TOTAL # OF MESSAGES TO THE SYSTEM
*      UTIL          UTILIZATION OF SERVER
*      Z             A FACTOR NEEDED TO CALCULATE CONFIDENCE INTERVALS
*
*      When  variables below start with a T, they become the
*      variables for all nodes comdined.
*
*      N(NODE)       NUMBER OF CYCLES RUN FOR EACH NODE
*      R()           MEAN WAITING TIME
*      RVAL()        +-INTERVAL FOR CONFIDENCE INTERVALS
*      S2()          VARIANCE OF WAITING TIME
*      S11()         VARIANCE OF Y
*      S22()         VARIANCE OF ALPHA
*      S12()         COVARIANCE OF ALPHA AND Y
*      SUMY(NODE)    SUM OF Y
*      SUMY2(NODE)   SUM OF SQUARE OF Y
*      SUMA(NODE)    SUM OF ALPHA
*      SUMA2(NODE)   SUM OF SQUARE OF ALPHA
*      SUMYA(NODE)   SUM OF ALPHA*Y
*      MEANY()       MEAN OF Y
*      VARIY()       VARIANCE OF Y
*      MEANA()       MEAN OF ALPHA
*      VARIA()       VARIANCE OF ALPHA
*      PERCEN        PERCENTAGE OF MESSAGES LOST
*      DL            DEADLINE
*      DELT          DELTA
*
*
*      NOTE: 1. The factor Z needed to calculate the 90% confidence
*            intervals is  1.645. But if different percentage is
*            used, you could just change the value for Z in the
*            initialization section.
*
*            2. To compile this program perform the following steps
*               i)   cc -r -c CONFID.c
*               ii)  ld -r CONFID.o -lm -o CONFID.o
*
*/

int *NODE, *N;
double *SUMY,*SUMY2,*SUMA,*SUMA2,*SUMYA,*UTIL,*PERCEN,*DL;
double *DELT;

        {
        int i, j;

        double tn, tsumy, tsumy2, tsuma, tsuma2;
        double tsumya,percen,dl,delta;

        double z,total;
        int node, n[100];
        double sumy[100],sumy2[100],suma[100],suma2[100],sumya[100],util;

        double tr, trval, ts2, ts11, ts22, ts12;
        double tmeany, tvariy, tmeana, tvaria;

        double nn1;

        double r[100], rval[100], s2[100], s11[100], s22[100], s12[100];
        double meany[100], variy[100], meana[100], varia[100];

        FILE *fp;

/*
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C                                 C
C  INITIALIZATION  OF  VARIABLES  C
C                                 C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
*/

        node = *NODE;
        percen    = *PERCEN;
        dl        = *DL;
        delta     = *DELT;
        util = *UTIL;
        for (i=1; i<node+1; i++)
                {
                n[i]     = *(N+i-1);
                sumy[i]  = *(SUMY+i-1);
                sumy2[i] = *(SUMY2+i-1);
                suma[i]  = *(SUMA+i-1);
                suma2[i] = *(SUMA2+i-1);
                sumya[i] = *(SUMYA+i-1);
                }
        if ((fp = fopen("stat","w"))<0)
                {
                perror("fopen");
                exit(1);
                }
        z = 1.645;
        util = util/1000;
        total = 0.0;
        for (i=1; i<node+1; i++)
                {
                r[i] = 0.0;
                rval[i] = 0.0;
                s2[i]=0.0;
                s12[i]=0.0;
                s11[i]=0.0;
                s22[i]=0.0;
                meany[i] = 0.0;
                variy[i] = 0.0;
                meana[i] = 0.0;
                varia[i] = 0.0;
                }


/*
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C                                      C
C  CALCULATE THE IMPORTANT PARAMETERS  C
C                                      C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
*/
```

```
        for (i=1; i<node+1; i++)
                {
                total = total + suma[i];
                if (n[i] >= 2)
                        {
                        meany[i] = sumy[i]/n[i];
                        variy[i] = sumy2[i]/n[i] - meany[i]*meany[i];
                        meana[i] = suma[i]/n[i];
                        varia[i] = suma2[i]/n[i] - meana[i]*meana[i];
                        nn1 = n[i] * (n[i]-1);
                        s11[i] = sumy2[i]/(n[i]-1) - sumy[i]*sumy[i]/nn1;
                        s22[i] = suma2[i]/(n[i]-1) - suma[i]*suma[i]/nn1;
                        s12[i] = sumya[i]/(n[i]-1) - sumy[i]*suma[i]/nn1;
                        r[i] = meany[i]/meana[i];
                        s2[i] = s11[i] - 2*r[i]*s12[i] + r[i]*r[i]*s22[i];
                        if ((s2[i]>=0) && (n[i]>=0))
                                rval[i] = z*sqrt(s2[i])/(meana[i]*sqrt(n[i]));
                        }
                }
/*
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C                                           C
C      OUTPUT STATISTICS IN A NEAT FORMAT    C
C                                           C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
*/
        fprintf(fp,"****************************\n");
        fprintf(fp,"*                          *\n");
        fprintf(fp,"*                          *\n");
        fprintf(fp,"*   SUMMARY OF STATISTICS   *\n");
        fprintf(fp,"*   UTILIZATION = %f",util);
        fprintf(fp,"  *\n");
        fprintf(fp,"*                          *\n");
        fprintf(fp,"*                          *\n");
        fprintf(fp,"****************************\n");
        fprintf(fp,"TOTAL # OF MESSAGES %f\n",total);
        fprintf(fp,"Z USED %f\n",z);
        fprintf(fp,"_____________________________________________\n");

        for (i=1; i<node+1; i++)
                {
                fprintf(fp,"STATION # %d\n",i);
                fprintf(fp,"TOTAL # of CYCLES %d\n",n[i]);
                fprintf(fp,"S2 = %f\n",s2[i]);
                fprintf(fp,"S11 = %f\n",s11[i]);
                fprintf(fp,"S22 = %f\n",s22[i]);
                fprintf(fp,"S12 = %f\n",s12[i]);
                fprintf(fp,"SUM OF CYCLE WAITING TIME =  %f\n", sumy[i]);
                fprintf(fp,"SUM OF SQUARES = %f\n",sumy2[i]);
                fprintf(fp,"SUM OF # OF MESSAGES =   %f\n", suma[i]);
                fprintf(fp,"SUM OF SQUARES = %f\n",suma2[i]);
                fprintf(fp,"SUM OF PROD. OF # MESS. AND WAITING TIME =  %f\n", sumya[i]);
                fprintf(fp,"MEAN WAITING TIME / CYCLE = %f\n", meany[i]);
                fprintf(fp,"VARIANCE = %f\n", variy[i]);
                fprintf(fp,"MEAN # OF MESSAGES / CYCLE = %f\n", meana[i]);
                fprintf(fp,"VARIANCE = %f\n", varia[i]);
                fprintf(fp,"MEAN WAITING TIME = %f",r[i]);
                fprintf(fp,"+-%f\n",rval[i]);
                fprintf(fp,"_____________________________________________\n");
                }
/*
C
C      CALCULATE  THE  STATISTICS FOR  IDENTICAL STATIONS
C
*/
        tn      = 0.0;
```

```
tsumy  = 0.0;
tsumy2 = 0.0;
tsuma  = 0.0;
tsuma2 = 0.0;
tsumya = 0.0;

tr     = 0.0;
trval  = 0.0;
ts2    = 0.0;
ts11   = 0.0;
ts22   = 0.0;
ts12   = 0.0;
tmeany = 0.0;
tvariy = 0.0;
tmeana = 0.0;
tvaria = 0.0;

for (i=1; i<node+1; i++)
        {
        tn = tn + n[i];
        tsumy = tsumy + sumy[i];
        tsumy2 = tsumy2 + sumy2[i];
        tsuma = tsuma + suma[i];
        tsuma2 = tsuma2 + suma2[i];
        tsumya = tsumya + sumya[i];
        }

if (tn >= 2)
        {
        tmeany = tsumy/tn;
        tvariy = tsumy2/tn - tmeany*tmeany;
        tmeana = tsuma/tn;
        tvaria = tsuma2/tn - tmeana*tmeana;
        nn1 = tn*(tn-1);
        ts11 = tsumy2/(tn-1) - tsumy*tsumy/nn1;
        ts22 = tsuma2/(tn-1) - tsuma*tsuma/nn1;
        ts12 = tsumya/(tn-1) - tsumy*tsuma/nn1;
        tr = tmeany/tmeana;
        ts2 = ts11 - 2*tr*ts12 + tr*tr*ts22;
        if ((ts2>=0) && (tn>=0))
                trval = z*sqrt(ts2)/(tmeana*sqrt(tn));
        }
fprintf(fp,"FOR IDENTICAL STATIONS \n");
fprintf(fp,"TOTAL # of CYCLES %f\n",tn);
fprintf(fp,"S2 = %f\n",ts2);
fprintf(fp,"S11 = %f\n",ts11);
fprintf(fp,"S22 = %f\n",ts22);
fprintf(fp,"S12 = %f\n",ts12);
fprintf(fp,"SUM OF CYCLE WAITING TIME =  %f\n", tsumy);
fprintf(fp,"SUM OF SQUARES = %f\n",tsumy2);
fprintf(fp,"SUM OF # OF MESSAGES =  %f\n",tsuma);
fprintf(fp,"SUM OF SQUARES = %f\n",tsuma2);
fprintf(fp,"SUM OF PROD. OF # MESS. AND WAITING TIME =  %f\n", tsumya);
fprintf(fp,"MEAN WAITING TIME / CYCLE = %f\n", tmeany);
fprintf(fp,"VARIANCE = %f\n", tvariy);
fprintf(fp,"MEAN # OF MESSAGES / CYCLE = %f\n", tmeana);
fprintf(fp,"VARIANCE = %f\n", tvaria);
fprintf(fp,"MEAN WAITING TIME = %f\n",tr);
fprintf(fp," +-%f\n",trval);
fprintf(fp,"DEADLINE = %f\n",dl);
fprintf(fp,"DELTA    = %f\n",delta);
fprintf(fp,"PERCENTAGE OF MESSAGES LOST = %f\n",percen);
fprintf(fp,"______________________________________________\n");
fclose(fp);
}
```

Bibliography

[Bou87] Boudenant, J., Feydel, B. and Rolin, P. LYNX: an IEEE 802.3 compatible deterministic protocol. In *Proceedings INFOCOM 87*, pages 573–579, IEEE Computer Society, March 1987.

[Box86] Boxma, O.J. and Meister, B. Waiting-time approximations for cyclic-service systems with switch-over times. *Performance Evaluation Review*, 14(1):254–259, May 1986.

[Cra77] Crane, M.A. and Lemoine, A.J. An introduction to the regenerative method for simulation analysis. In *Lecture Notes in Control and Information Sciences*, pages 1–109, Spring-Verlag, 1977.

[Hen83] Henriksen, J.O. and Crain, R.C. *GPSS/H User's Manual.* Wolverine Software Corporation, Annandale, VA 22003-2653, 1983.

[IEE85] IEEE Standards Board. *Token Ring Access Method and Physical Layer Specifications.* The Institute of Electrical and Electronics Engineers, 345 East 47th Street, NY 10017, 1985.

[Kim83] Kim, B.G. An adaptive token ring serving real-time traffic. In *COMPSAC 83*, pages 105–107, IEEE Computer Society, November 1983.

[Kle76] Kleinrock, L. *Queueing Systems, Vol 1: Theory; Vol 2: Applications.* John Wiley and Sons, New York, 1975-76.

[Knu81] Knuth, D.E. *The Art of Computer Programming, Vol. 2.* Addison-Wesley Publishing Company Inc., 1981.

[Kur84] Kurose, J.F., Schwartz, M. and Yemini, Y. Access protocols and time-constrained communication. *ACM Computing Surveys*, 16(1):43–70, March 1984.

[Leh86] Lehoczky, J.P. and Sha, L. Performance of real-time bus scheduling algorithms. *Performance 86*, 44–53, 1986.

[Liu84] Liu, M.T. and Rouse, D.M. A study of ring networks. In *Ring Technology Local Area Networks*, pages 1–39, Elsevier Science I olishers B.V., 1984.

[Lo88] Lo, Edward *Performance Evaluation and Comparison of a Token Ring Network with Full Latency Stations and Dual Latency Stations.* Master's thesis, Dept. of Elec. Engineering, University of British Columbia, May 1988.

[Mac85] Macnair, C.A. and Sauer, C.H. *Elements of Practical Performance Modeling.* Prentice-Hall, Inc., Englewood Cliffs, N.J. 07632, 1985.

[Ped87a] Peden, J.H. and Weaver, A.C. Performance of priorities on an 802.5 token ring. *Computer Communications Review*, 17(5):58–66, August 1987.

[Ped87b] Peden, J.H. *Performance Analysis of the IEEE 802.5 Token Ring.* Master's thesis, Dept. of Comp. Science, University of Virginia, Jan 1987.

[Ped88] Peden, J.H. and Weaver, A.C. Are priorities useful on an 802.5 token ring? *IEEE Transactions on Industrial Electronics*, 35(3):361–365, August 1988.

[Rom81] Rom, R. and Tobagi, F.A. Message based priority functions in local multiaccess communication systems. *Computer Networks*, 5(4):273–286, July 1981.

[Ros86a] Ross, F.E. FDDI - a tutorial. *IEEE Communications Magazine*, 24(5):10–17, May 1986.

[Ros86b] Ross, F.E. Fiber, farther, faster. In *Proceedings INFOCOM*, pages 323–330, IEEE Computer Society, April 1986.

[Sch74] Schriber, T.J. *Simulation using GPSS.* John Wiley and Sons, New York, 1974.

[Sch87] Schwartz, M. *Tellecommunications Networks: Protocols, Modeling and Analysis.* Addison-Wesley Publishing Company Inc., 1987.

[Sev87] Sevcik, K.C. and Johnson, M.J. Cycle time properties of the FDDI token ring protocol. *IEEE Transactions on software Engg.*, SE-3(3):376–385, March 1987.

[She85] Shen, Z., Masuyama, S., Muro, S. and Masegawa, T. Performance evaluation of prioritized token ring protocols. *Teletraffic Issues in an Advanced Information Society*, 5:648–654, 1985.

[Sta84] Stallings, W. Local networks. *ACM Computing Surveys*, 16(1):3–41, March 1984.

[Sta88] Stallings, W. *Data and Computer Communications.* McMillan Publishing Company, 866 Third Ave, N.Y.,10022, 1988.

[Str88] Strosnider, J.K., Marchok, T. and Lehoczky, J. Advanced real-time scheduling using the IEEE 802.5 token ring. In *Real-Time Systems Symposium*, pages 42–52, IEEE Computer Society, December 1988.

[Tak86] Takagi, Hideaki. *Analysis of P 'ling Systems.* The MIT Press, 1986.