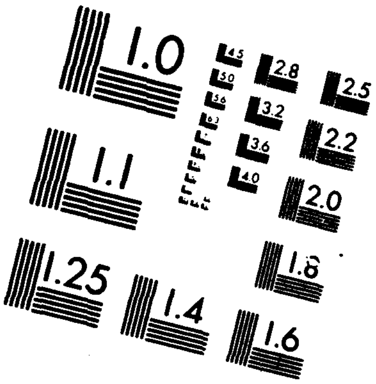
AD-A163 950 SIMULATION MODEL OF A HIGH-SPEED TOKEN-PASSING BUS FOR 1/2
AVIONICS APPLICATIONS(U) AIR FORCE INST OF TECH
WRIGHT-PATTERSON AFB OH SCHOOL OF ENGI.. J E SPIETH
UNCLASSIFIED DEC 85 AFIT/GCS/ENG/85D-15 F/G 17/2 NL


1.0
4.5
5.0
5.6
6.3
2.8
2.5
3.2
2.2
3.6
1.1
4.0
2.0
1.8
1.25
1.4
1.6

SIMULATION MODEL OF A HIGH-SPEED TOKEN-PASSING BUS FOR AVIONICS APPLICATIONS

THESIS

James E. Spieth

AFIT/GCS/ENG/85D-15

DEPARTMENT OF THE AIR FORCE
AIR UNIVERSITY
AIR FORCE INSTITUTE OF TECHNOLOGY

Wright-Patterson Air Force Base, Ohio

AFIT/GCS/ENG/85D-15

SIMULATION MODEL OF A HIGH-SPEED TOKEN-PASSING BUS FOR AVIONICS APPLICATIONS

THESIS

James E. Spieth

AFIT/GCS/ENG/85D-15

Approved for public release; distribution unlimited

AFIT/GCS/ENG/85D-15

SIMULATION MODEL OF A HIGH-SPEED TOKEN-PASSING BUS FOR AVIONICS APPLICATIONS

THESIS

Presented to the Faculty of the School of Engineering
of the Air Force Institute of Technology
Air University
In Partial Fulfillment of the
Requirements for the Degree of
Master of Science in Electrical Engineering

James E. Spieth, B.S.E.E.

December 1985

Approved for public release; distribution unlimited

Preface

The purpose of this thesis was to develop a simulation model of a token-passing bus local area network. The intended application of the simulated networks is that of the specialized aircraft avionics data bus network. The simulation model was successfully developed and validated. Initial testing was completed for one protocol and a comparison test of two protocols was accomplished.

This simulation model is one of many models and tools that will be needed to design and develop a new avionics data bus required for the next generation of aircraft and their complex avionics.

I would like to thank my advisor, Major Walter D. Seward, and my thesis committee member, Captain David A. King, for their assistance during this thesis effort. I would also like to thank my sponsor, Harold J. Alber for his technical advice and for allowing me to utilize the Systems Engineering Avionics Facility computer. Finally, I would like to thank my wife Charlotte, for her patience and support during this graduate program.

James E. Spieth

Table of Contents

List of Figures

Figure Page

List of Tables

Abstract

There are many factors of bus token-passing protocols that influence the overall performance of the protocol. Extensive analysis is needed to design a protocol with performance that can meet the requirements for a next-generation aviation electronics (avionics) data bus. The purpose of this thesis was to develop and test a token-passing bus simulation model that could be used to conduct this analysis.

This thesis developed and validated a model for simulating bus token-passing protocols for avionics applications. Two algorithms were designed that reflected the timing and operation of a distributed control token-passing protocol and a centralized control token-passing protocol. The algorithms were incorporated into an overall simulation model program which included simulation control, data collection, and data analysis functions. The simulation model program was written in the Pascal computer programming language. [illegible]

The simulation model program allows various avionics bus configurations to be defined and tested. A series of tests were conducted using the simulation model program to validate its operation and modeling capabilities. The validation tests were successful. Initial performance tests were conducted for a centralized control token-passing protocol using a bus configuration representative of a fighter-type aircraft bus network. The performance of the two types of protocols was also compared.

SIMULATION MODEL OF A HIGH-SPEED TOKEN-PASSING BUS FOR AVIONICS APPLICATIONS

I. Introduction

There is presently a great deal of interest concerning the rapidly evolving field of local area computer communication networks, commonly called local area networks (Myers, 1982:28; Stallings, 1984a:3). The main reason for this interest in local area networks (LANs) stems from the ability of the local area network to interconnect computer-based equipment so that expensive resources can be shared and data can be exchanged among the equipment (Stallings, 1984a:3). Local area networks are used in aviation electronics (avionics) applications primarily to reduce integration complexity and risk (Alber and Thomas, 1985:130).

This introductory chapter begins with a background of avionics local area networks and reasons why a new avionics local area network is needed. Next, a brief description of the problem is given followed by a statement of the objective of this thesis. A summary of the current knowledge concerning bus local area networks is then presented followed by the approach taken in this thesis. The chapter concludes with an overview of the rest of the chapters and appendices of this thesis.

Background

In the late 1960s, it became apparent that avionics integration by use of dedicated hard-wired interfaces was too costly and complex, and led to inflexible avionics systems (Gifford, 1974:85). Therefore, the concept of a standard interface which would allow different avionic equipment to exchange information via a shared serial communications link was developed (Boeing, 1980:2-2). Such a standard interface was defined and adopted by the Air Force as MIL-STD-1553. This standard defines the hardware and protocol for a serial digital data bus that is shared by the avionics connected to it. A time division multiplexing scheme is used to allow the avionic equipment to share the bus in a controlled manner. This Air Force standard was later adopted by the Department of Defense for use by all the military services.

As the next generation of military aircraft is starting to be designed, a new standard interface implemented by an avionics local area network is required. This new avionics network is needed to meet the increased information flow requirements of the more advanced and complex digital avionics that will be used in these aircraft (Ludvigson and Milton, 1985:122). Also, a broader range of avionic equipment could take advantage of reductions in weight, integration complexity and cost that avionics local area networks offer if their information flow requirements could be met by such a network.

Problem

The US Air Force's Aeronautical Systems Division engineering community has agreed on the framework for a new avionics local area network. The use of a token-passing protocol with a bus topology is the preferred method (Ludvigson and Milton, 1985:123). However, there are many factors of the token-passing protocol which could influence the performance of the avionics network, specifically the data throughput rate and message delays. These factors include bit rate, number of data words per message, maximum message size, number of overhead bits, size of the token, maximum token-passing time, centralized versus distributed token scheduling and control, physical length of the bus, and the maximum number of terminals connected to the bus. Extensive analysis is needed to determine what effect these different factors have on the data throughput rate and message delays.

Objective

The objective of this thesis is to design and test a software simulation model that will allow analysis of the effect the factors listed above have on the bus's throughput rates and message delays. This model will aid in the development of a finalized configuration for the new avionics local area network. The software model will allow simulation of the bus network on a digital computer.

Current Knowledge

Due to the popularity and interest in local area networks, there has been a great amount of work reported in the literature concerning local area networks. Studies of the token-passing bus protocol found pertinent for this thesis include the following.

> Stallings discusses the factors that determine performance for local area networks including token-passing bus protocols and develops simple performance equations (Stallings, 1984b).
>
> Ulug compares the performance of a token-passing bus protocol with different token-holding limit strategies using analytical and simulation methods (Ulug, 1984).
>
> Cherukuri *et al* evaluate the performance of a token-passing protocol for ring, baseband bus, and broadband bus topologies using analytical methods (Cherukuri *et al*, 1982).

> Stuck presents a performance comparison of the popular media access methods including the token-passing bus protocol using analytical methods (Stuck, 1983a).

> Rahimi and Jelatis validate the IEEE 802.4 token-passing protocol and evaluate its performance by using simulation methods (Rahimi and Jelatis, 1983).

Based upon the results of these analytical and simulation studies, some ideas concerning the performance of token-passing buses are generally accepted. These ideas include the following:

> Low throughput when the bus is lightly loaded because much of the bandwidth is wasted passing the token through idle stations (Rahimi and Jelatis, 1983:801; Cherukuri et al, 1982:59).

> High and stable throughput when the bus is heavily loaded because the overhead is small compared to the amount of data being transferred (Cherukuri et al, 1982:59).

> Decrease in throughput when either the header bits in the message and/or the token are increased in size because both are considered overhead. When they are increased, more time is being spent carrying the overhead bits and not data (Stallings, 1984b:30).

> Performance is sensitive to the message and token propagation delays and hence the length of the bus (Stuck, 1983a:75-76).

While there has been a great deal of work involving analytical and simulation studies of local area networks, most works are not completely applicable to the very specialized avionics application. Avionics local area networks are different from other typical local area networks for an office or distributed computing environment in two main areas. The first difference is that an avionics network has known minimum and maximum station populations (Alber, 1985). The other difference is that more is known about the message size and arrival characteristics for the avionics network compared to the other environments.

There are also disadvantages involved with the analytical method itself. The major drawback is the simplifying assumptions that must be made in order for the analysis to be workable. These assumptions limit the analysis' value for detailed performance evaluations (Rahimi and Jelatis, 1983:801).

The problem with simulation studies is the applicability of the model. The simulation might work correctly, but if the model is not an accurate representation of the system under study, the usefulness of the simulation is doubtful. For example, Stuck noted that simulation studies for bus local area networks did not model the separation of the stations on the bus (Stuck, 1983b:112). He found that the simulations assumed worst case propagation delays for signals traveling between stations. Since then, a bus local area network simulation has incorporated actual separation delays; but the study was conducted for the carrier sense, multiple access with collision detection media access method as implemented by Ethernet (Jackman and Medeiros, 1984:595).

Approach

The approach taken in this thesis in developing a simulation model of an avionics bus local area network was to develop an algorithm that reflected the timing and protocol operation of a token-passing bus network. An overall design was done for the simulation model which incorporated the algorithm and necessary simulation control, data collecting, and data analysis functions to produce a complete simulation software package. The package was written in the Pascal programming language. The simulation model allows various avionics bus configurations to be set up and tested so the effect of the different bus design factors can be analyzed. The design factors that are adjustable are listed below in general categories:

- bus environment
  - bit rate
  - number of stations
  - length of bus
  - signal propagation speed
- stations
  - separation
  - token-passing order
  - message arrival types and rates
  - message length distribution types and means
- messages
  - number of overhead bits
  - number of bits in token
  - number of bits in a data word
  - minimum and maximum number of data words
- protocol
  - centralized or distributed control
  - token holding time
  - station delay time

Besides setting up the bus environment to be tested, the simulation control function allows control of the length of the simulation run.

The data collecting function of the simulation model program collects data so that the following performance parameters can be determined by the data analysis function:

- data throughput rates
- message delays
- access delays
- message size statistics
- bus efficiency

Overview of Remaining Chapters

Chapter II presents a brief overview and analysis of local area network topology, media access methods for bus topologies, and potential protocols for the new avionics bus network. Chapter III discusses the simulation model. Chapter IV describes how the simulation model was validated and presents results from simulation runs. Chapter V is the summary of this thesis and presents some recommendations. Appendix A is the simulation model users guide and Appendix B contains the simulation model software. Appendix C contains the data files used for simulation runs.

II. Local Area Networks

This chapter presents a discussion of local area networks including topologies and bus media access methods. After each topic, an analysis is conducted that describes the strong and weak points of each topology and access method presented. Four popular token-passing protocols are then discussed and analyzed. Based upon these analyses, a topology, media access method, and protocol are selected for use in this thesis.

Definition

A local area network can be defined as "a communications network that provides interconnection of a variety of data communicating devices within a small area" (Stallings, 1984a:4).

Avionics systems consist of various equipment that depend on each other for basic information concerning the aircraft and its environment (Boeing, 1980:2-2). The avionics need to exchange this information with one another in order for each to perform their function. An example of this information exchange would be a central air data computer providing pressure altitude and air speed data to an inertial navigation unit (Boeing, 1982:6-5). Also, the avionics system is limited to the physical size of the aircraft. Thus, an avionics bus system which allows information transfer agrees with the above definition and is one example of many systems that can be described as a local area network.

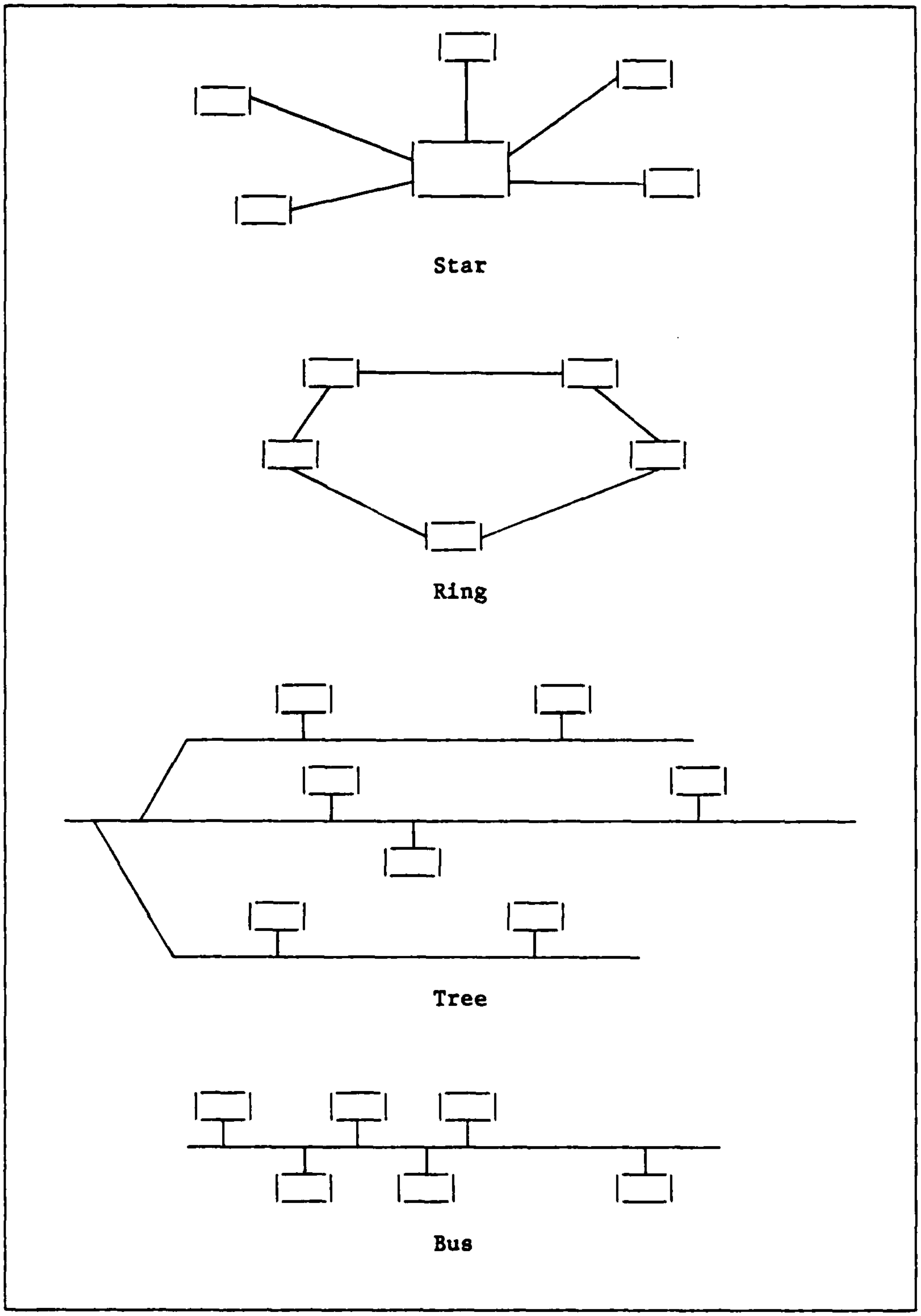
Topology

There are three main topologies that can be used to describe a local area network: the star, ring and tree topologies (Stallings, 1984a:6). These topologies are shown in Figure 1 where the square boxes represent nodes or stations. The star topology has a central node connected to all the other nodes in the network. The ring has each node connected to exactly two other nodes using point to point connections which form a physical ring. The tree topology has a trunk with multiple branches. The nodes can be located anywhere along the trunk and branches. The special case of a tree with only a trunk is called a bus topology (Stallings, 1984a:6).

Topology Analysis

The star topology leads to the central node becoming the bottleneck of the network since all messages are routed to or through it. The ring topology requires each node to actively repeat each message to the next node on the ring. Also, if one node fails, the ring is broken unless active bypass circuits are available. A break in the media can cause a single point failure of the bus topology. This failure mode can be avoided if multiple buses are used.

Chosen Topology. The bus was selected as the preferred topology for the new avionics local area network for the following reasons: (1) previous bus experience with MIL-STD-1553, (2) nodes can be connected to the bus with passive connections, (3) adding or deleting nodes is easily accomplished, and (4) multiple buses can be used for reliability. One reason for not choosing the star topology was the congestion problem at the central node. Another reason was the central node is a single point of failure which would cause the entire network to fail. The ring topology was not chosen because there is a chance that a single failure could break the ring. Active bypass circuitry can be used to pass the



Star

Ring

Tree

Bus

Figure 1. Local Area Network Topologies

signals around the break, but the bypass circuitry adds another failure mode.

Bus Media

The media used for the actual physical bus can be a radio channel, a coaxial cable, a fiber optic cable, or a twisted-shielded pair of wires. This thesis will not address the choice of media in keeping with the current trend to design media independent protocols.

Media Access Methods

Since all the nodes in the bus topology share a common communications link (the bus), the network requires a scheme for controlling the nodes' access of the bus. This is necessary because only one message can be succeısfully transmitted and received on the bus at a time (Kurose et al, 1984:44). This control is called the media, medium, channel or multiple access method and is one of the most important aspects of the bus topology. Also, due to the shared bus, all nodes can hear all messages transmitted. This characteristic is called broadcast (Stallings, 1984a:6). The media access methods can be grouped into three major categories: fixed, random or contention, and demand assignment methods (Liu et al, 1982:417).

Fixed Assignment Method. In the fixed assignment access method, all nodes on the bus are given a certain amount of time or frequency to transmit messages even if they have none to send. Examples of fixed assignment methods are time division multiplexing and frequency division multiplexing.

Random Assignment Method. Random access methods try to improve on the inefficiency of the fixed methods by only allowing nodes with messages to randomly try and transmit them. However, there is no coordination between nodes; so two or more nodes can transmit at the same time, causing their messages to collide and become useless. When a collision occurs, the whole process is repeated until all the messages are successfully transmitted (Kurose et al, 1984:46). There are various versions of the random access methods which try and minimize this loss of capacity due to collisions. Two familiar examples of random access methods are the various Aloha type methods, and the carrier sense, multiple access with collision detection (CSMA/CD) method as implemented by Ethernet (Tanenbaum, 1981:253,292).

Demand Assignment Method. The demand access methods either explicitly or implicitly exchange control information so at any time only one node is in control of the bus and allowed to transmit a message. This method avoids the problems associated with collisions (Kurose et al, 1984:45). Also, although each node is given the chance to transmit, they do not have to (IEEE 802.4, 1982:1-10). This way, the media bandwidth is not wasted by assigning it to idle nodes (Liu et al, 1982:419). Examples of demand access methods are the various token-passing and reservation schemes.

Media Access Method Analysis

The token-passing method is preferred for the new avionics bus local area network rather than other methods due to its attractive features for the aircraft avionics real-time environment application (Ludvigson and Milton, 1985:123). The fixed assignment access methods are not efficient for changing traffic loads. The message collision characteristic of the random access method is inefficient; and depending on the message transmission retry policy, can lead to non-deterministic bus access delays. This is especially true under heavy traffic loads. The attractive features of the token-passing access method include deterministic access delays; no minimum packet length requirement; good performance under heavy traffic loads; fairness to all nodes; no specialized listening while talking or collision detection circuitry; allowing implementation of multiple classes of service (priority); and easy addition or deletion of nodes (Stallings, 1984a:28; Miller and Thompson, 1982:84; IEEE 802.4, 1982:1-2).

Token-Passing Media Access Method

The token-passing method operates by having the nodes pass a special bit pattern message (the token) among themselves. The node having the token has control of the bus and can transmit any messages it might have until it has no more messages or the maximum token-holding time has expired (Miller and Thompson, 1982:80). The token is then passed to the next node. Every node is guaranteed a maximum time that it must wait between token possessions, which is called the maximum access delay time. This access delay time varies with the amount of load (nodes with messages) on the bus. A feature of this method is that it places minimal restrictions on how a node uses its bus time. This allows a node to use some other access method during its bus control time such as poll/response or requesting an acknowledgement, as long as it does not confuse the other nodes (IEEE 802.4, 1982:1-10).

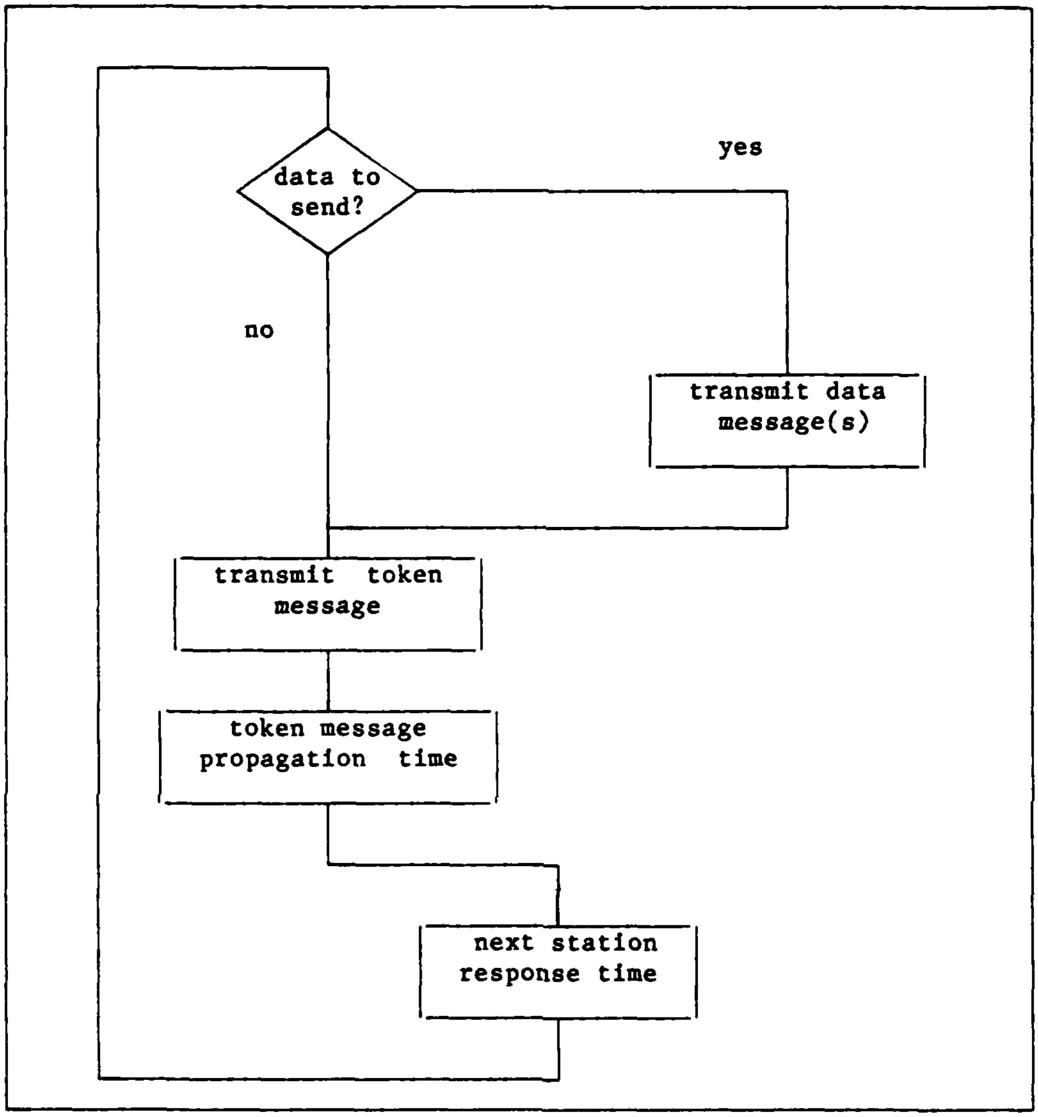


Figure 2. Simple Token-Passing Protocol Simulation

The normal steady-state operation of the token-passing protocol can be thought of as alternating data transfer and token transfer phases (Stallings, 1984b:26). Figure 2 is a simplified flow chart representation of how this steady-state operation is simulated in this thesis. The Figure starts with a station having just received the token.

Preamble	Start	Control	SA	DA	Data	CRC	End

Figure 3. Typical Message Format

Messages. In addition to carrying actual data from one station to another, a message in the token-passing media access method also carries information that is necessary for the correct operation of the token-passing method (Ludvigson and Milton, 1985:127). This information, such as synchronization, address, and error control information, is not actual data that is being transferred from one station to another and therefore is considered overhead.

Message Format. The format of a typical message is shown in Figure 3. The preamble or synchronization bits allow a receiving station to synchronize itself with the bus signal (Ludvigson and Milton, 1985:127). The start delimiter bits labeled Start in Figure 3 and the end delimiter bits labeled End in Figure 3 are used to define the beginning and end of each message (Alber and Thomas, 1985:132). The control bits are used to define the message type. There are basically two types of messages, the control message and the data message. The control messages are necessary for the operation of the token-passing media access method. Data messages carry data among the stations. Examples of control type messages would be the token message, where a station is passing the token; messages that allow new stations to join and exit the network; and messages that initialize the network (IEEE 802.4, 1982:4-3). Other information can also be included with the control bits; priority bits to indicate the priority of the message, and time tag bits to indicate when the data in the message was generated. SA indicates the source address bits and DA indicates the destination address bits in Figure 3. These bits are used to indicate the source and destination of the particular message. The data bits labeled Data in Figure 3 denote the data being transferred among stations or data associated with a control message. The data bits can be an optional field for the control type messages. For example, the control message that passes the token would not have any data bits in it. The cyclic redundancy code bits labeled CRC in Figure 3 are used for error control. This allows stations to detect errors in the messages they receive.

Token-Passing Protocols

There are currently three token-passing protocols that have been proposed for the new avionics bus network. This section will compare and analyze their features. A fourth protocol defined by the Institute of Electrical and Electronics Engineers (IEEE) in their effort to standardize local area networks will also be included in the analysis. The other three protocols have been proposed by the Society of Automotive Engineers (SAE) as part of their aerospace standards activities, by the Collins Government Avionics Division of Rockwell Corporation as part of a contract for the US Air Force's Avionics Laboratory Pave Pillar Program, and finally by the US Air Force's Systems Engineering Avionics Facility (SEAFAC). A summary of the important features of each protocol is presented in Table I. A dash (-) in a column for an item indicates that this item is undefined or not

TABLE I

COMPARISON OF TOKEN-PASSING PROTOCOLS

ITEM	IEEE 802.4	SAE	Avionics Laboratory	SEAFAC
bit rate (Mb/s)	10	50	50	50
token message size (bits)	112/176	24	-	22
token control	distributed	distributed	distributed	centralized
direction of token passing	descending	ascending	-	any order
minimum message size (bits)	112/176	72	-	70
maximum message size (bits)	65,568	65,608	-	4166
data word size (bits)	8	16	16	16
number of data words	0-8182/8174	0-4096	1-4096	0-256
maximum number of addresses	32768	128	64	128
maximum number of group addresses	32768	-	-	1
maximum bus length (meters)	-	1280	91	1000
maximum stub length (meters)	30	included	6	100
response time (microseconds)	-	.5	-	.5
levels of message priority	optional, 4	4	if needed	yes, 2
type of station servicing	limited	limited	-	limited
multiple buses	-	yes, 1-4	-	yes

applicable for that particular protocol.

The response time item in Table I refers to the maximum amount of time a station is allowed to take in responding to a token message addressed to it. That is, a station must begin transmitting a data or token message within the response time maximum limit. If no response is heard by the station that passed the token (sending station) or the central controller, depending on the type of control used in the network, an error condition is assumed to exist. The sending station or central controller then takes steps to overcome this error condition. Station servicing refers to the two different disciplines available to a station for servicing the messages it has waiting to be transmitted. The first discipline is called exhaustive servicing where all messages pending at a station are transmitted. The other discipline is called limited servicing which only allows a station to transmit pending messages until a time limit or number of messages limit is reached (Cherukuri et al, 1982:60). The next four paragraphs briefly discuss the information presented in Table I for each protocol.

IEEE Token-Passing Protocol. (IEEE 804.2, 1982) The draft IEEE Token-Passing Bus Access Method, Standard 802.4, is one of the family of standards for local area networks. The 802.4 protocol includes options for a 1 megabit per second (Mb/s) baseband network, a 5 or 10 Mb/s baseband network, and a 1 Mb/s or 5 Mb/s or 10 Mb/s broadband network. For the analysis, the 10 Mb/s baseband network was used. Referring to Table I, a maximum bus length is not directly specified in the standard because different types of cable are allowed by the standard. Instead, a minimum signal strength in dB is defined. The minimum message size item in Table I contains two values because the IEEE draft Standard allows a 16 bit or a 48 bit address field. However, the maximum size of the message is limited to 65,568 bits in both cases. Only the 16 bit address field is used in the calculation of the maximum address and maximum group address item entries.

SAE Token-Passing Protocol. (SAE, 1985) In the draft of the proposed standard, seven bits are allowed for destination and source addresses, but a total of eight bits are allowed in the address fields. The last bit is to be used for multi-cast or group addresses, but the draft standard does not define how it is to be implemented.

Avionics Laboratory Token-Passing Protocol. (Ludvigson and Milton, 1985) The contract to develop this protocol is presently in progress so many details of the protocol have not yet been defined. This is the reason why so many items are marked with a dash in Table I. The Avionics Laboratory and Rockwell Collins are allowing a token control option in the protocol. The option allows a second token control method, that of a centralized type.

SEAFAC Token-Passing Protocol. (Alber and Thomas, 1985) This protocol uses a centralized method to control the operation of the bus. One station, designated the scheduler station, is responsible for bus initiation, error detection and recovery, and setting the token passing order. This protocol has a characteristic that allows the token to be passed along with the data in one message. If a station does not have data to send, a regular token message is transmitted. Including the token in the data message decreases the bus overhead, as a separate token message does not have to be transmitted. This protocol also

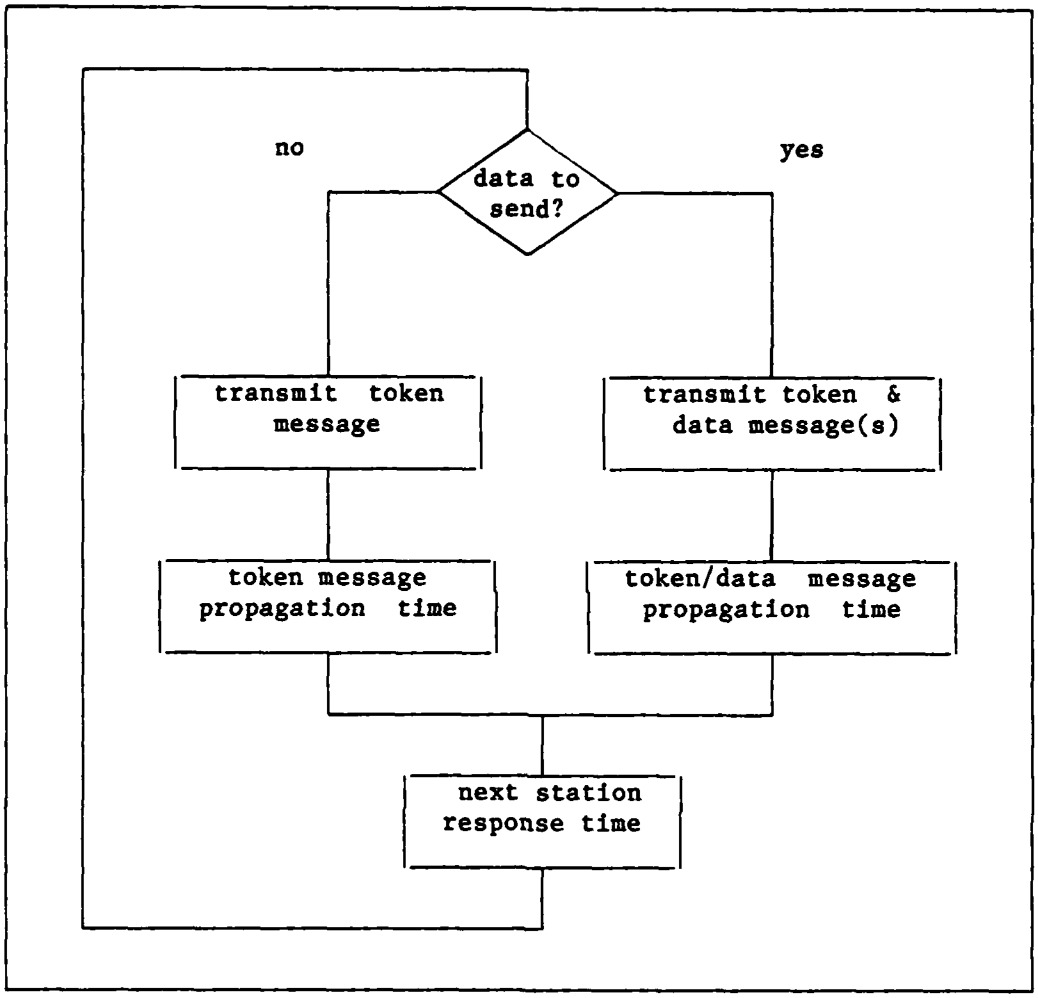

no
yes
data to send?
transmit token message
transmit token & data message(s)
token message propagation time
token/data message propagation time
next station response time

Figure 4. Simple Centralized Token-Passing Protocol Simulation

allows more than one data message to be sent by having the transmitting station pass the token back to itself until it has no more data or the token holding time is exceeded. The token is passed to the next station with the last data message transmitted. Figure 4 is a simplified flow chart representation of how this centralized protocol's steady-state operation is simulated in this thesis. The Figure starts with a station having just received the token.

Summary

Local area networks can be described by their topology and the method used to control how the stations gain access to the network. Each topology and access method has its strong and weak points which must be analyzed according to the network's application.

Since a bus topology with a token-passing access method is the preferred method for a new avionics local area network, this thesis will develop a simulation model for such a network. The simulation model will be designed with enough flexibility to model all the previously described protocols.

III. Simulation Model

This chapter begins with a brief discussion of discrete event simulation concepts. The simulation program and the token-passing algorithm are described and defined using flow charts. The various adjustable bus parameters are described next. Finally, the simulation program design is presented using structure charts.

Discrete Event Simulation

Simulation can be defined as experimentation with a model of a system (Shannon, 1983:20). What makes simulation so popular and powerful is its ability to allow a nondisruptive examination of an existing system to learn more about it or to test improvements. Simulation also can be used to explore the performance of a system that does not yet exist (Mittra, 1984:142) or which can not be readily analyzed.

There are many ways simulation models can be classified. Three examples of these classifications are according to how the model represents the system, the model's purpose, and how the model represents change within the system (Schmidt, 1984:65). When a simulation model represents change within the system as occurring only at isolated points in time, the model is classified as a discrete type. When the model represents change as continually occurring over time, the model is classified as a continuous type (Schmidt, 1984:65-66). For discrete models, the points in time are determined by the occurrence of an event. An event is when something happens to change the state of the system (Shannon, 1975:109). The time clock in discrete simulations is advanced in varying size steps of time to when the next event occurs. These steps are discrete amounts of time, hence the name discrete simulation.

In simulating an avionics bus network in this thesis, the discrete event type of simulation is used because the operation of the bus can be characterized by various events. Examples of these events are the arrival of a message at a station, the times a station begins and ends transmission of a message, and the times a station begins and ends transmission of a token message.

Entities. Entities in a system are the physical or symbolic components of the system (Schruben, 1983:101). Entities are important in a simulation model because their interactions cause events to be created (Shannon, 1975:109). For the avionics bus network, the entities of the system would be the bus media, the stations, and the messages. Entities are described by their attributes (Schruben, 1983:102). For example, the attributes of a message might be its size, its point of origin (source station), and its destination station (Fortier and Leary, 1981:221).

Simulation Language. (Schmidt, 1984:72-73) The simulation model must eventually be converted to a form understandable by the computer. This conversion is carried out by expressing the simulation model in a general purpose computer programming language or a special purpose (simulation) language. There are two main advantages in using a general purpose language. These languages provide the person conducting the simulation a maximum amount of flexibility in the design of the model. Secondly, at least one of these languages is probably known by that person. An advantage of using a simulation programming language is its built-in routines to accomplish common simulation functions. Two examples of these routines would be event manipulation subroutines and time keeping mechanisms.

Selected Language. The Pascal general purpose programming language was selected for use in this thesis for a number of reasons. The most important reason was that the flexibility of a general purpose language was needed to model the level of detail desired for the avionics bus simulation model. Even with the recent growth and popularity of simulation languages, Mittra reports that a recent survey estimated that 75% of all discrete event simulations were performed using FORTRAN or Pascal languages while most of the remaining 25% were performed using the GPSS and SIMSCRIPT simulation languages (Mittra, 1984:144).

Simulation Model Program

The simulation model program is divided into three main sections that are described below and shown in flow chart form in Figure 5. The three sections are setup, simulation, and summary.

Setup. The Setup section performs all the data input and initialization actions that are necessary before the actual simulation can take place. These actions accomplished by the Setup section of the simulation program are shown in flow chart form in Figure 6. The Setup section begins by reading in the bus configuration data. Next, the arrival time and length of the first message for each station is calculated and the message is queued at its station. Then, the variables that will store the data for all the statistics concerning the simulation are initialized. Finally, the token-passing propagation delays are calculated for every station pair and this information is

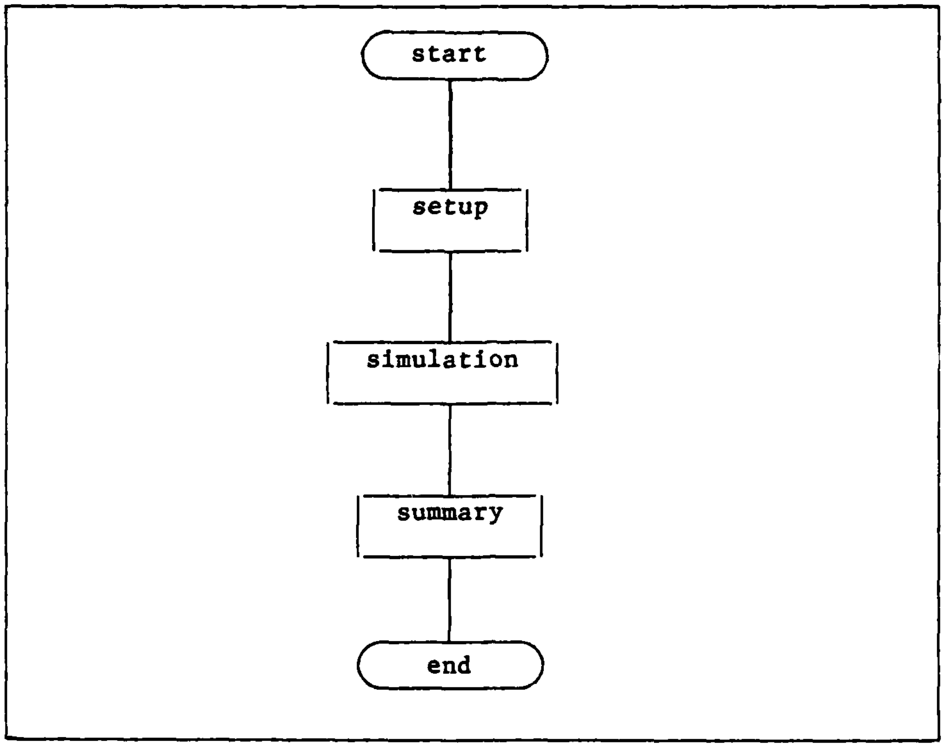


Figure 5. Simulation Model Program Flow Chart

stored. This is done so that during the simulation portion of the program, the token-passing propagation delays can simply be looked up and do not have to be calculated every time they are needed.

Simulation. This section of the simulation program performs the actual simulation of the bus. The section consists of two main routines, one for simulating distributed token-passing protocols similar to the one defined by the IEEE 802.4 specification, and the other for simulating the Systems Engineering Avionics Facility centralized token-passing protocol. This section of the simulation model program is described in detail later in this chapter under the heading Token-Passing Algorithms.

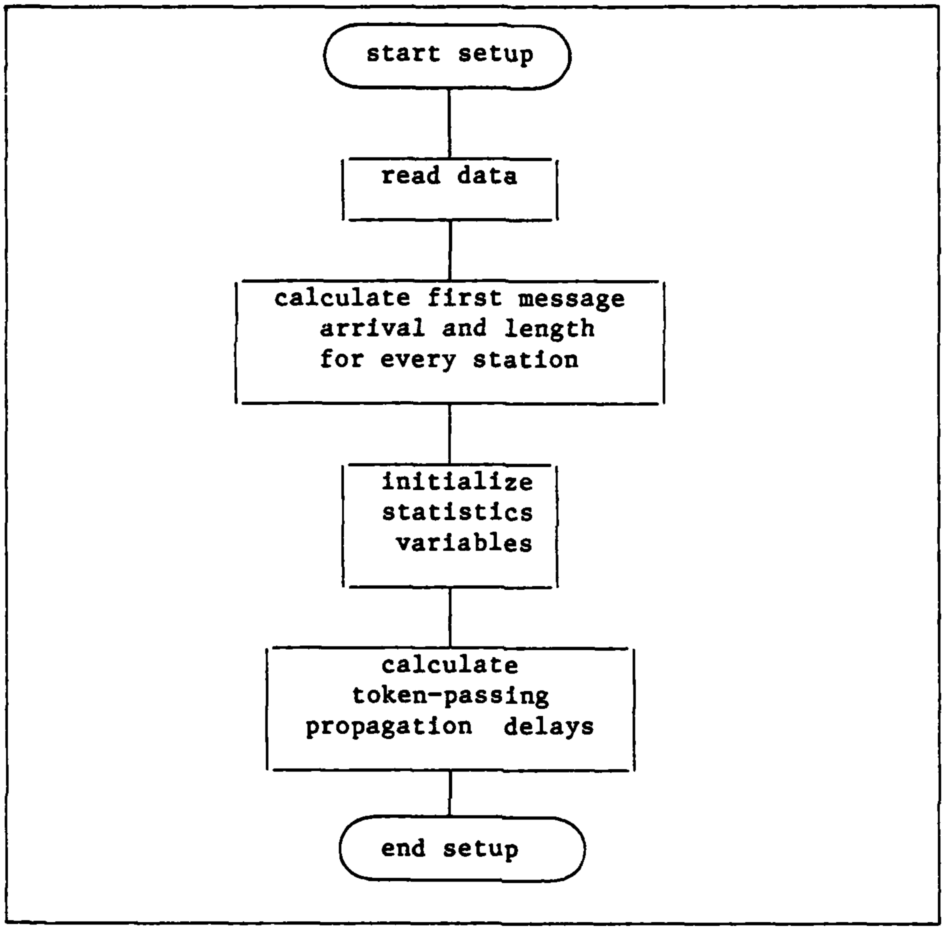

start setup
read data
calculate first message arrival and length for every station
initialize statistics variables
calculate token-passing propagation delays
end setup

Figure 6. Setup Section Flow Chart

Summary. The actions accomplished by the Summary section of the simulation model program are shown in flow chart form in Figure 7. This section performs the calculations necessary to determine the values for the various performance parameters for the bus configuration that was simulated. These parameters include: throughput, message statistics, access delays, message delays, number of token-passing cycles, and efficiency. This section then prints these parameter values in a summary format.

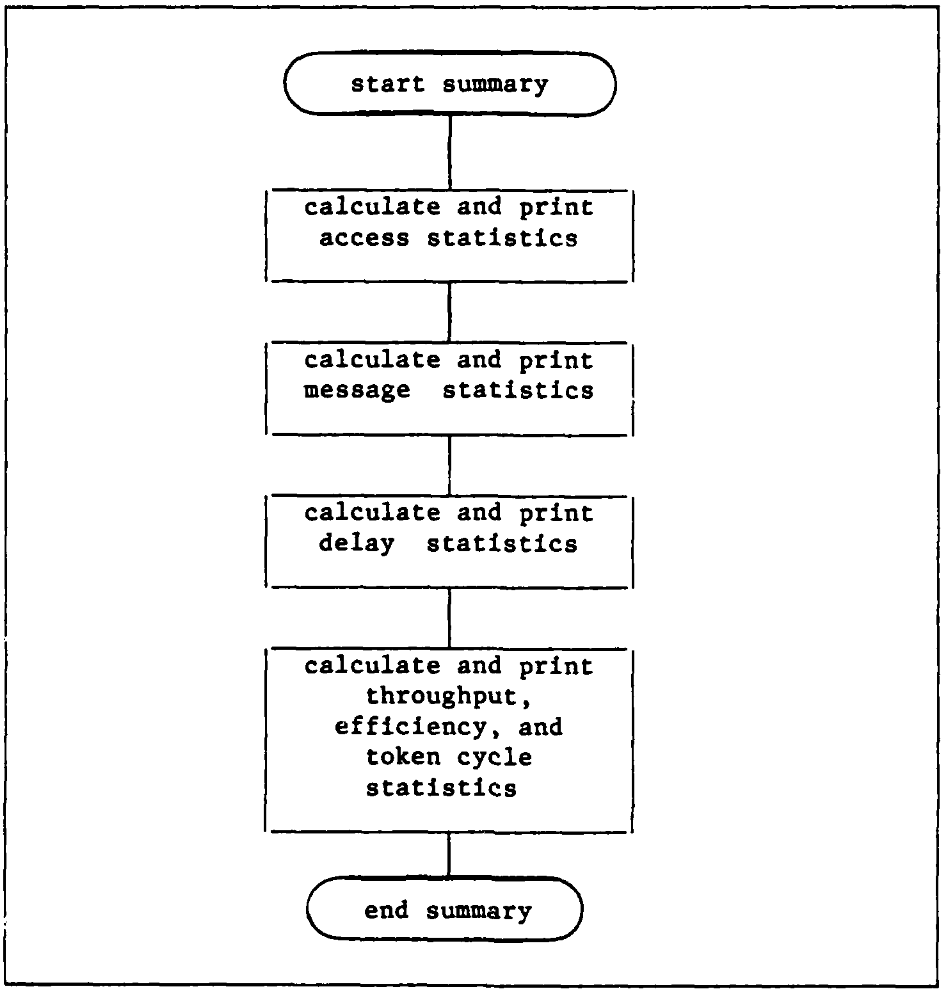

start summary
calculate and print access statistics
calculate and print message statistics
calculate and print delay statistics
calculate and print throughput, efficiency, and token cycle statistics
end summary

Figure 7. Summary Section Flow Chart

Token-Passing Algorithms

The token-passing algorithms are the main part of the simulation section of the overall program. The distributed token-passing algorithm is shown in Figure 8 and the centralized token-passing algorithm is shown in Figure 9. In the figures, the abbreviation "incr. sim." represents increase simulation. This refers to increasing the time clock of the simulation to the time of the next event. The abbreviation "msg" represents message and the abbreviation "tx" represents transmission.

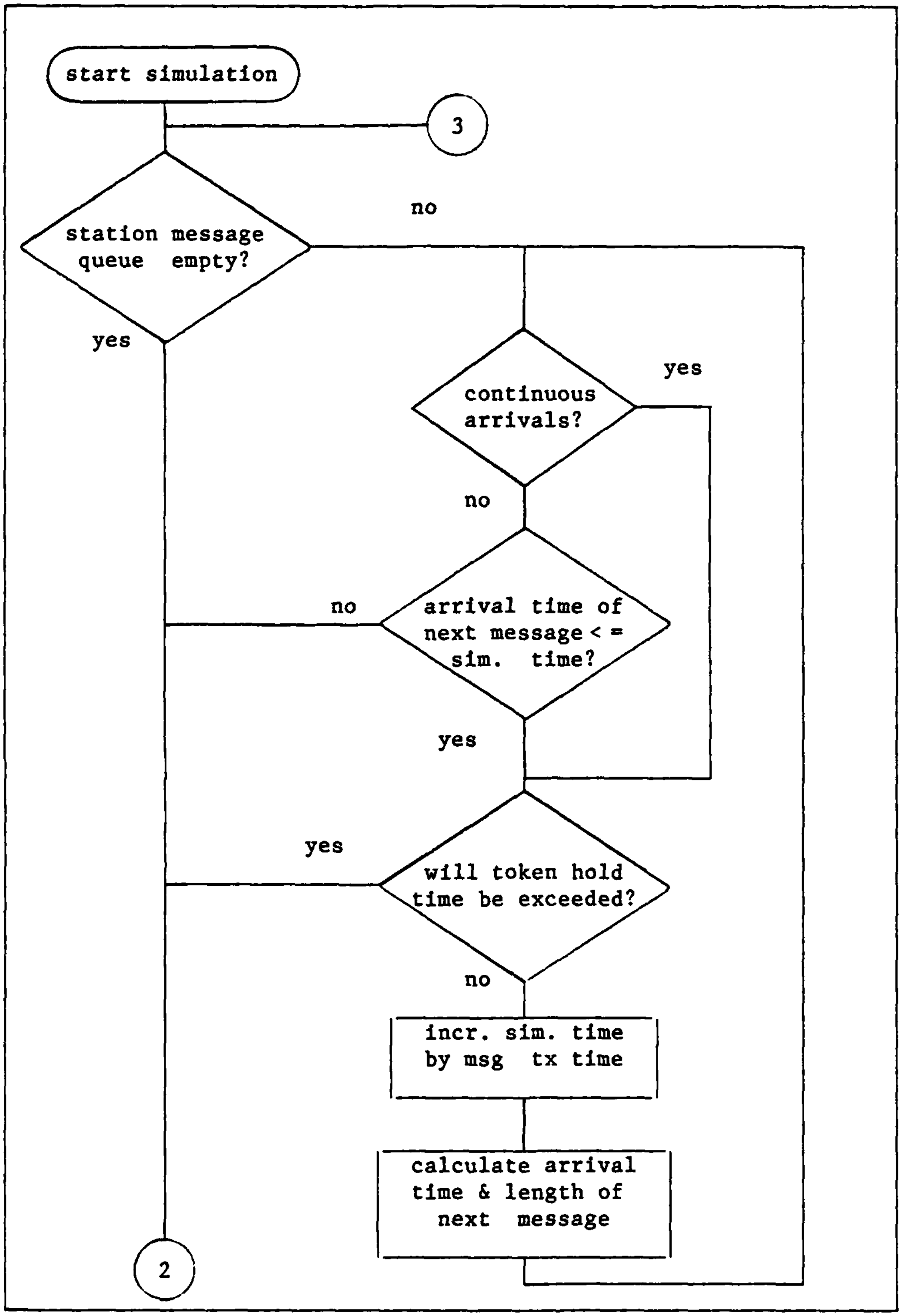

start simulation
3
station message queue empty?
no
yes
continuous arrivals?
yes
no
arrival time of next message < = sim. time?
no
yes
will token hold time be exceeded?
yes
no
incr. sim. time by msg tx time
calculate arrival time & length of next message
2

Figure 8. Distributed Token-Passing Algorithm

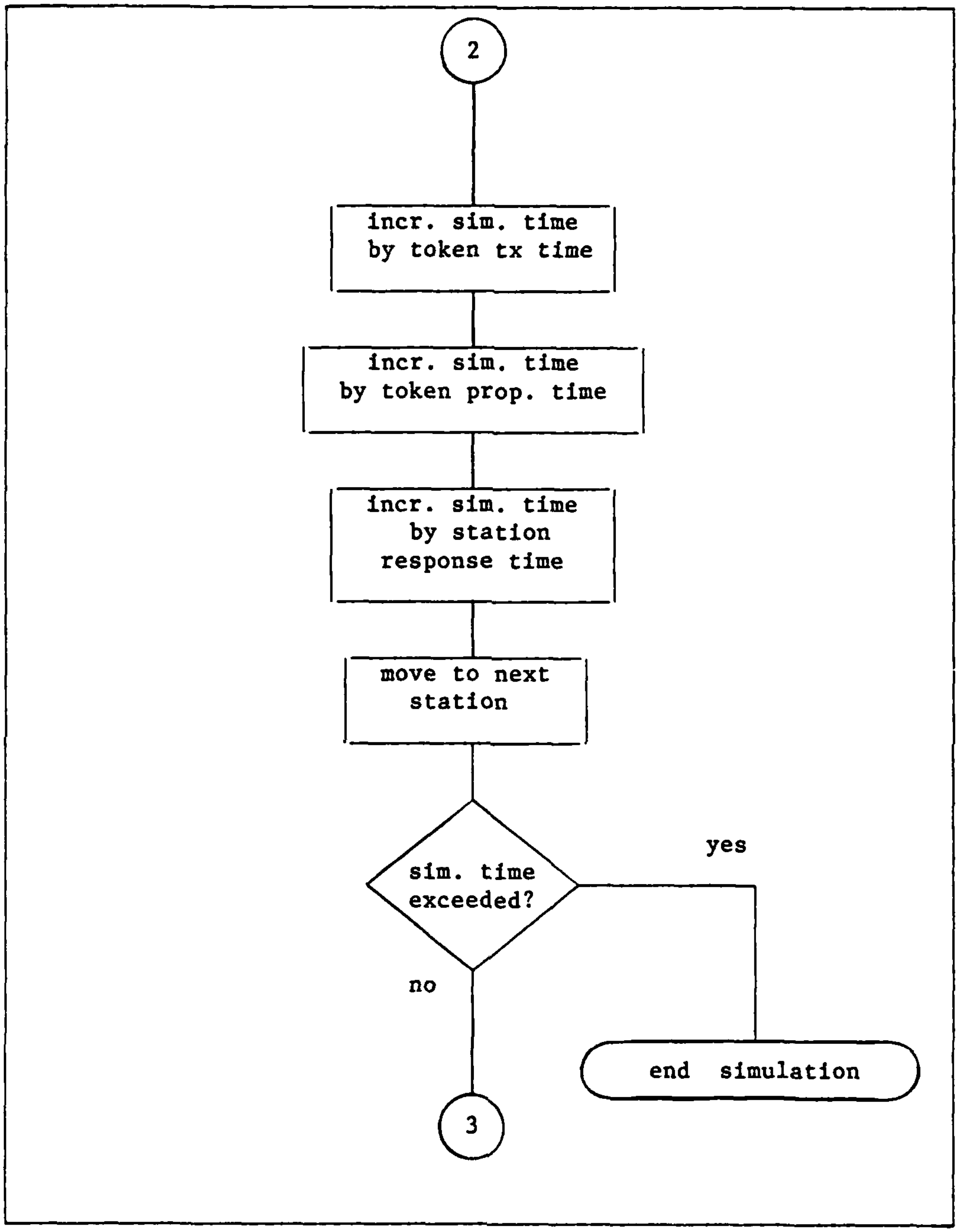

2
incr. sim. time by token tx time
incr. sim. time by token prop. time
incr. sim. time by station response time
move to next station
sim. time exceeded?
yes
no
end simulation
3

Figure 8. Distributed Token-Passing Algorithm (continued)

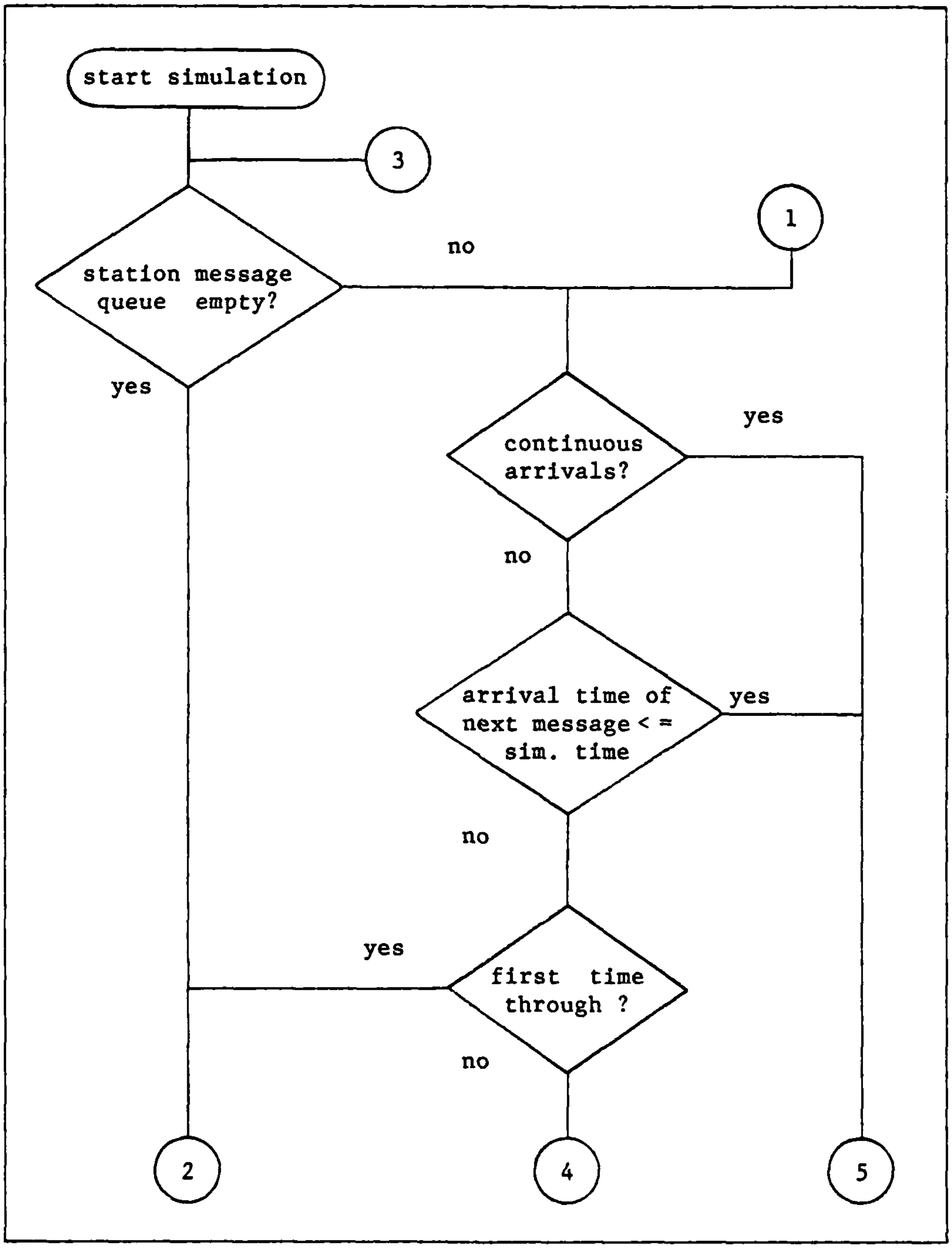


Figure 9. Centralized Token-Passing Algorithm

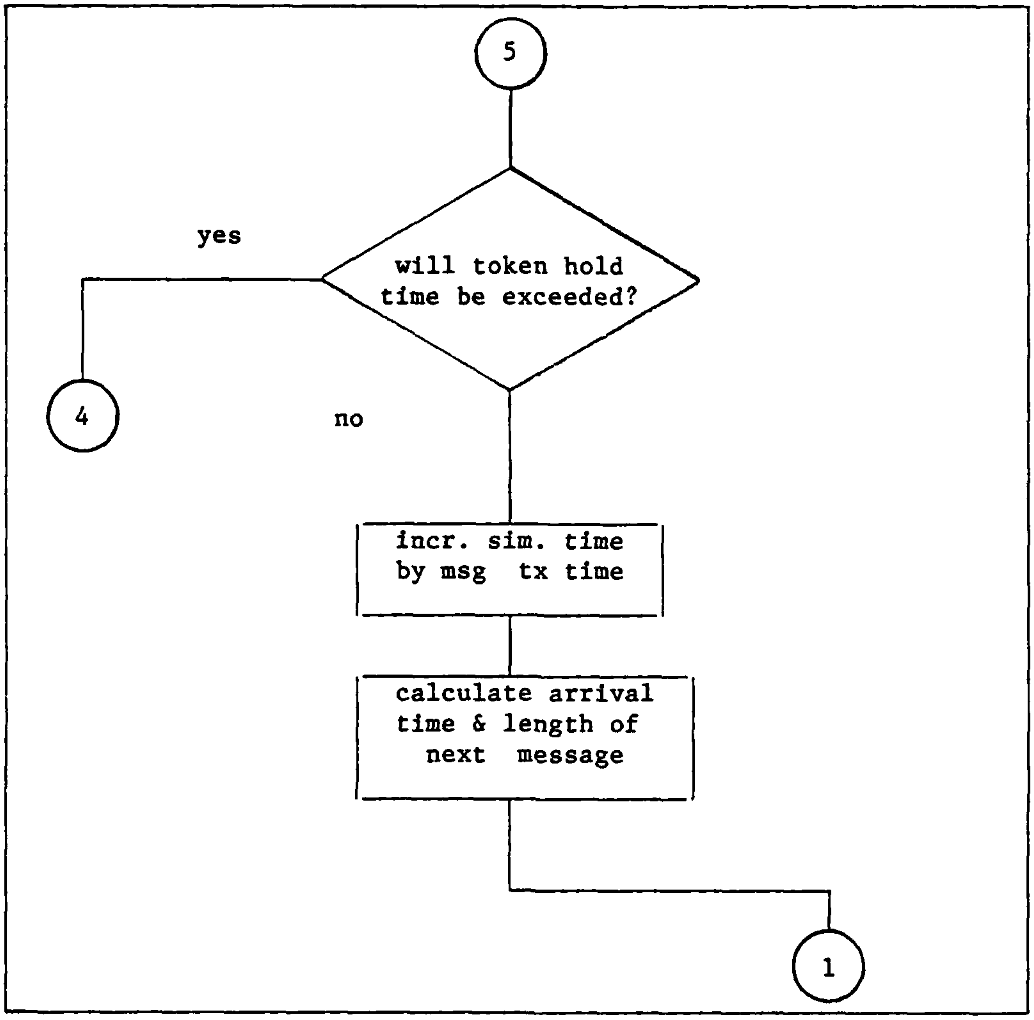

5
will token hold time be exceeded?
yes
4
no
incr. sim. time by msg tx time
calculate arrival time & length of next message
1

Figure 9. Centralized Token-Passing Algorithm (continued)

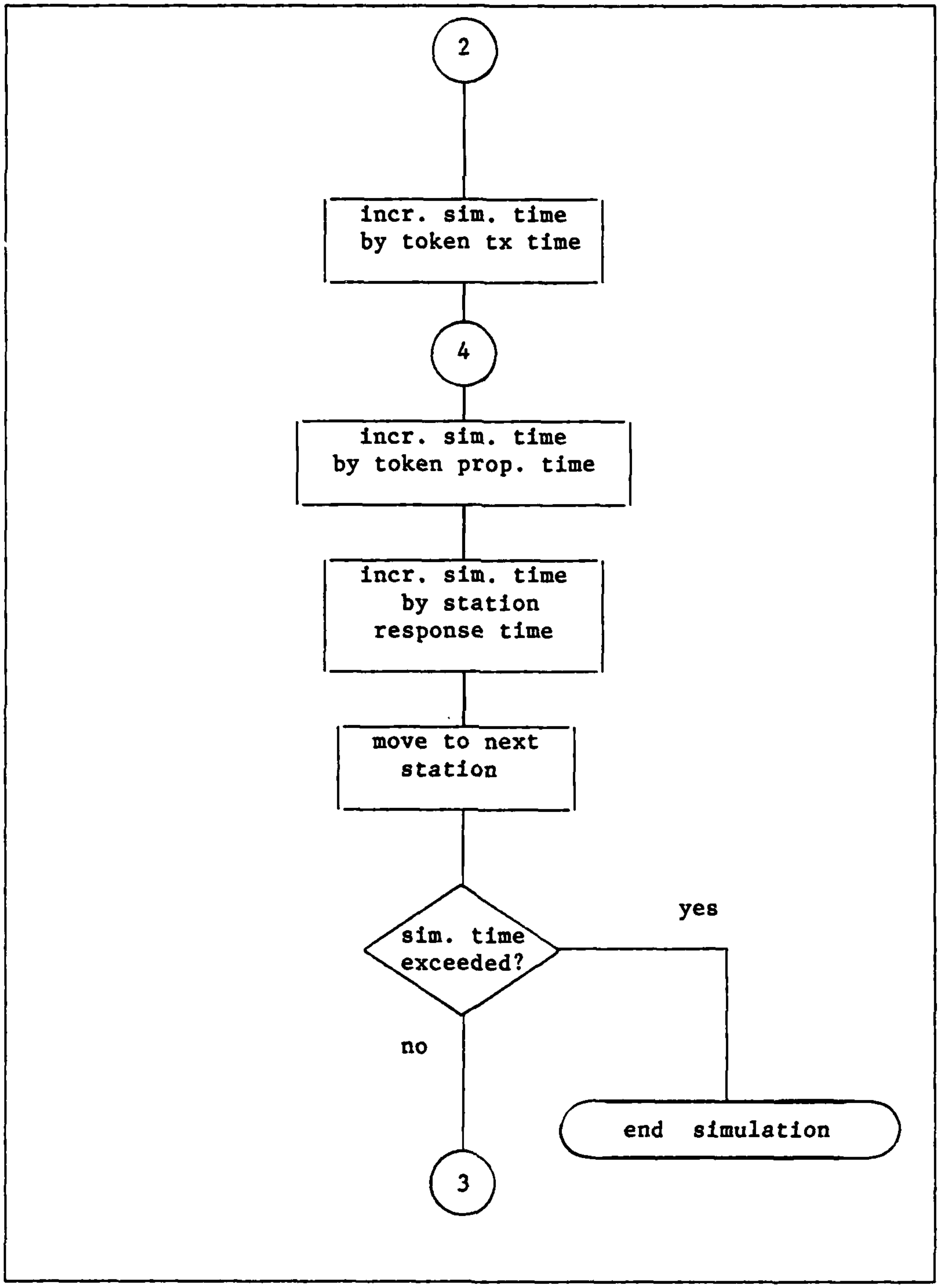


Figure 9. Centralized Token-Passing Algorithm (continued)

For both algorithms, the first decision that is made is whether the station has any messages waiting to be sent. The presence of messages waiting to be sent must be checked because a station may have a very low or even a zero rate for message arrivals. A variety of arrival rates are allowed so different traffic loading situations may be simulated. A message arrival rate of zero would correspond to a station with no messages to send. The expression "continuous arrivals" refers to the condition where a station always has messages waiting to be transmitted. Again, the reason for this condition is so different traffic loading situations can be simulated.

Token-Holding Time Limit. The token-holding time limit can be checked by a station either before a message is transmitted or after a message has been transmitted. In a token-passing protocol with a maximum message size, both types of token-hold time limit checks would result in calculable actual station token-holding times and therefore deterministic access delays. However, a before transmission check keeps the actual maximum station token-holding time lower than an after transmission check. The only way to achieve this same lower maximum token-holding time with the after transmission check would be to artificially restrict the token-holding time limit. This restriction does not allow the station to fully utilize its token-holding time limit except when it has the maximum number of maximum size messages ready to transmit. It is for this reason that both algorithms use a before transmission check of the token-holding time limit.

In the case of the centralized token-passing protocol, this before transmission check requires a station to not only check the next message in its message queue, but also to check the second message if there is one. This is because the token must be passed in the last message transmitted by the station. Thus, if the token-holding time will run out during transmission of the second message, the station only transmits the next message and includes the token passing instruction in it. However, since no actual messages are being send in the simulation model program, the centralized algorithm does not have to check the second message. It still performs a before transmission check of the token-holding time limit though. If the limit will be exceeded, we assume the token was passed in the last message transmitted.

Assumptions. Both algorithms assume that initialization of the bus has taken place; that is, a complete token-passing order has been established and each station knows to whom it is to pass the token. The centralized token-passing algorithm assumes that the token-holding time limit allows a station to transmit at least one data message when it has one or more ready to send, because that is the only means by which the token can be passed given that condition. The algorithms have been designed to model their respective token-passing protocol under the conditions of steady-state, error free operation. These assumptions are made so that the simulation program can focus on modeling the protocols to determine their performance and the effect the various bus parameters have upon that performance.

Bus Parameters

The various bus parameters that are adjustable in the simulation program are listed below. Also included are the choices available when a parameter has a limited number of values it may take on, for example the type of message arrival condition a station can have.

- bus environment
    - bit rate
    - number of stations
    - length of the bus
    - station delay time
    - signal propagation speed
- protocol
    - centralized or distributed control
    - token holding time limit
    - token passing order
        - ascending
        - descending
        - fixed
- stations
    - distance from left end of the bus
    - type of message arrival condition
        - deterministic distribution
        - Poisson distribution
        - continuous case
    - message arrival rate
    - type of message length distribution
        - deterministic
        - exponential
    - message length size or mean
- messages
    - number of overhead bits
    - number of bits in token
    - number of bits in a data word
    - minimum number of data words in a message
    - maximum number of data words in a message

Message Arrival Types and Rates. All the stations on the bus may have the same type of message arrival condition. All the stations may have the same arrival rate of this condition or they may all have different rates of this condition. However, if the stations all have different types of message arrival conditions, all their rates are assumed to be different and are read in on a individual basis even though they might be the same.

Message Length Distribution Types and Means. All the stations on the bus may have the same type of message length distribution. All the stations may have the same mean for this distribution type or they may all have different means. However, if the stations all have different types of distributions, all their means are assumed to be different and are read in on a individual basis even though they might be the same.

Simulation Model Program Design

The structure chart method was used as an aid in the design of the simulation model program. The structure charts are shown in Figure 10. A goal of modularity was strived for in the design of the software so changes or added functions could easily be incorporated. The following paragraphs describe each main module of the program.

Module 0.0. This is the executive module for the program. It is decomposed into three modules that implement the functions of the three sections described earlier.

Module 1.0. This module implements the functions of the setup section. It accomplishes this by calling five other modules. The bus data configuration input is divided into two modules. The first module, bus_data_input, reads in all general bus configuration data. The second module, station_data_input, reads in the specific data about each station. An example of the modularity of the program is shown in the Node 1.0 structure chart of Figure 10. The calculate arrival and length (calc_arr_and_len) module performs the basic calculations to generate a new message and assign it an arrival time and length. This module can be called by any other module for any station when a new message is needed.

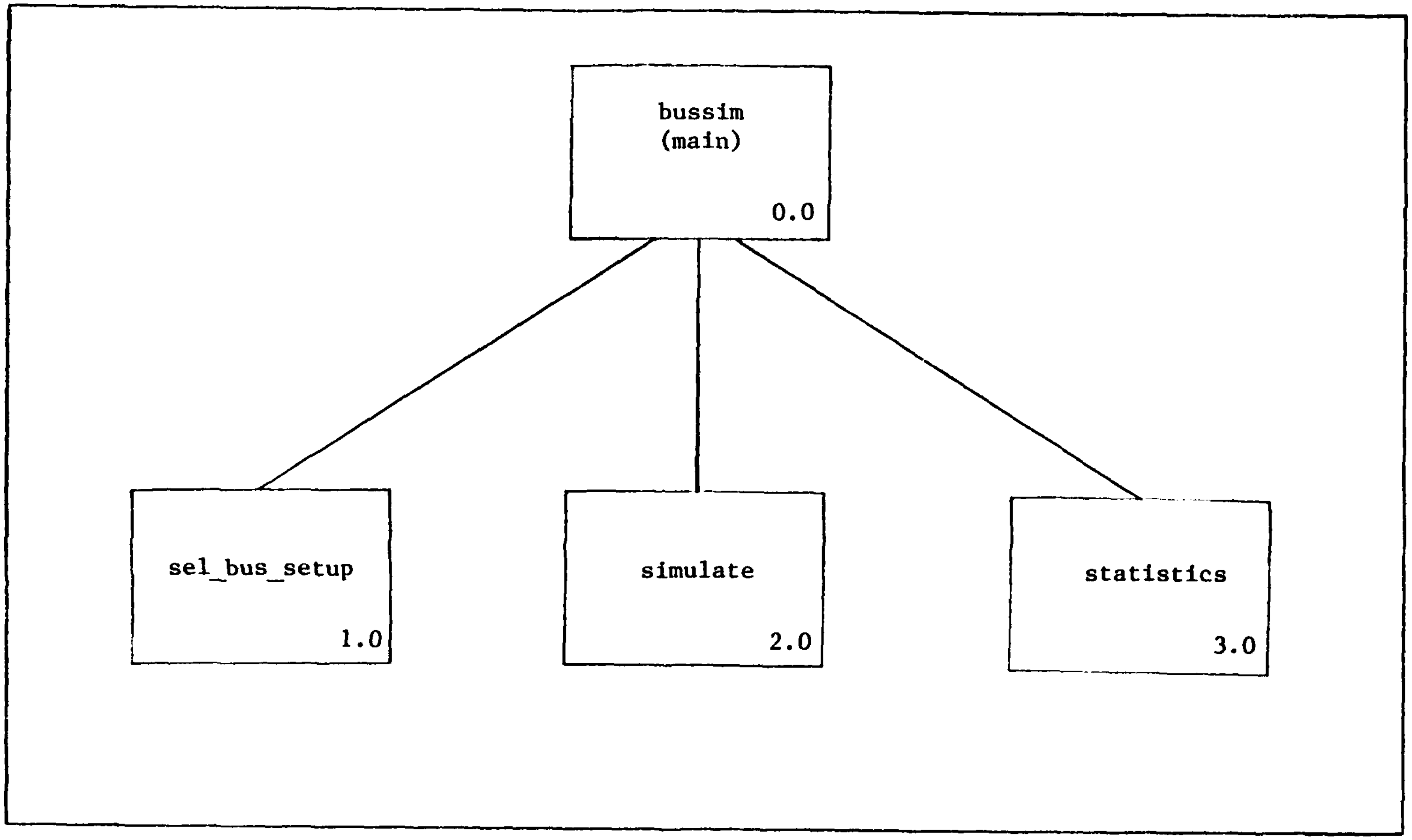


Figure 10. Structure Charts Node 0.0

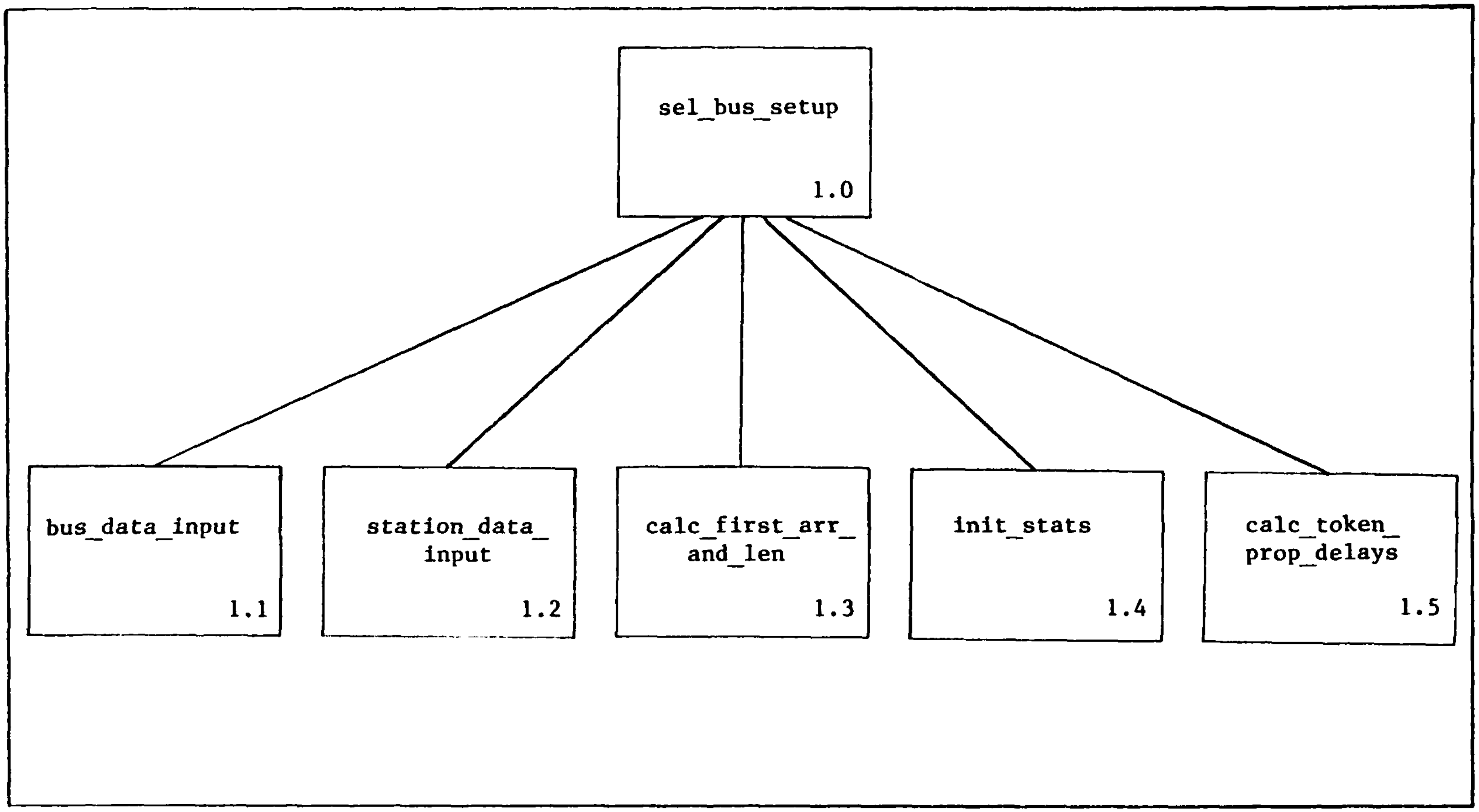


Figure 10. Structure Charts (continued) Node 1.0

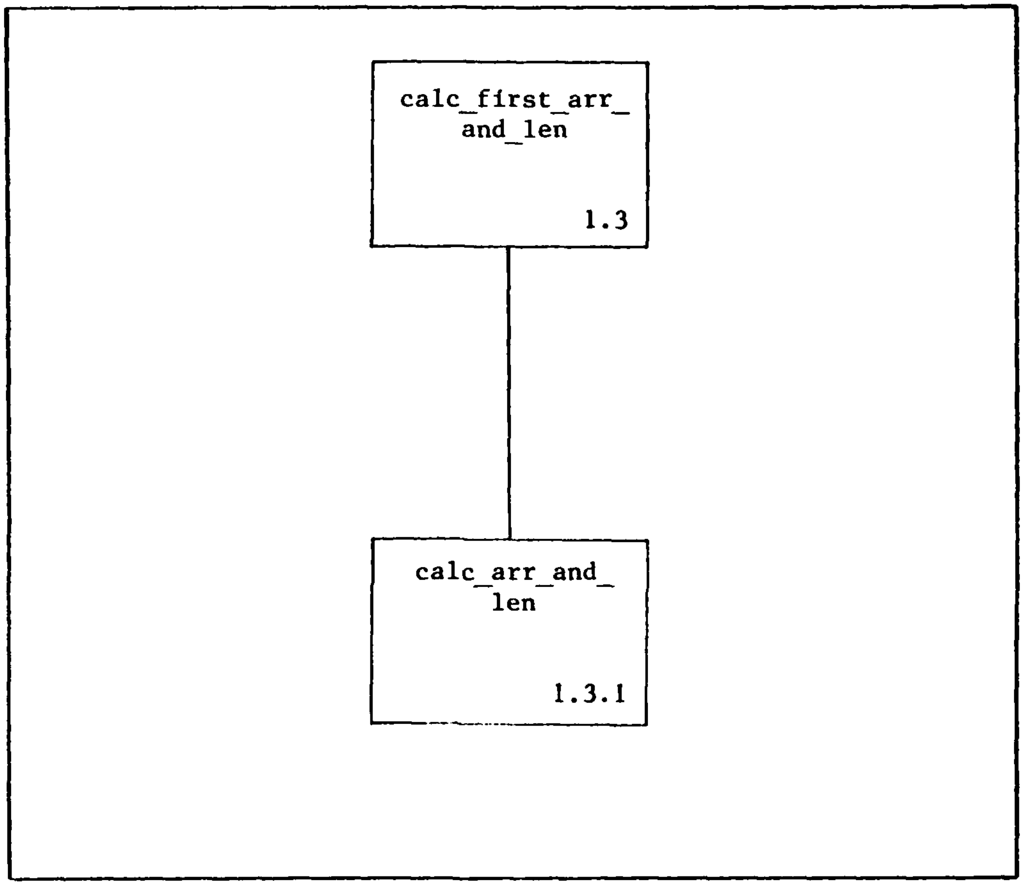


Figure 10. Structure Charts (continued) Node 1.3

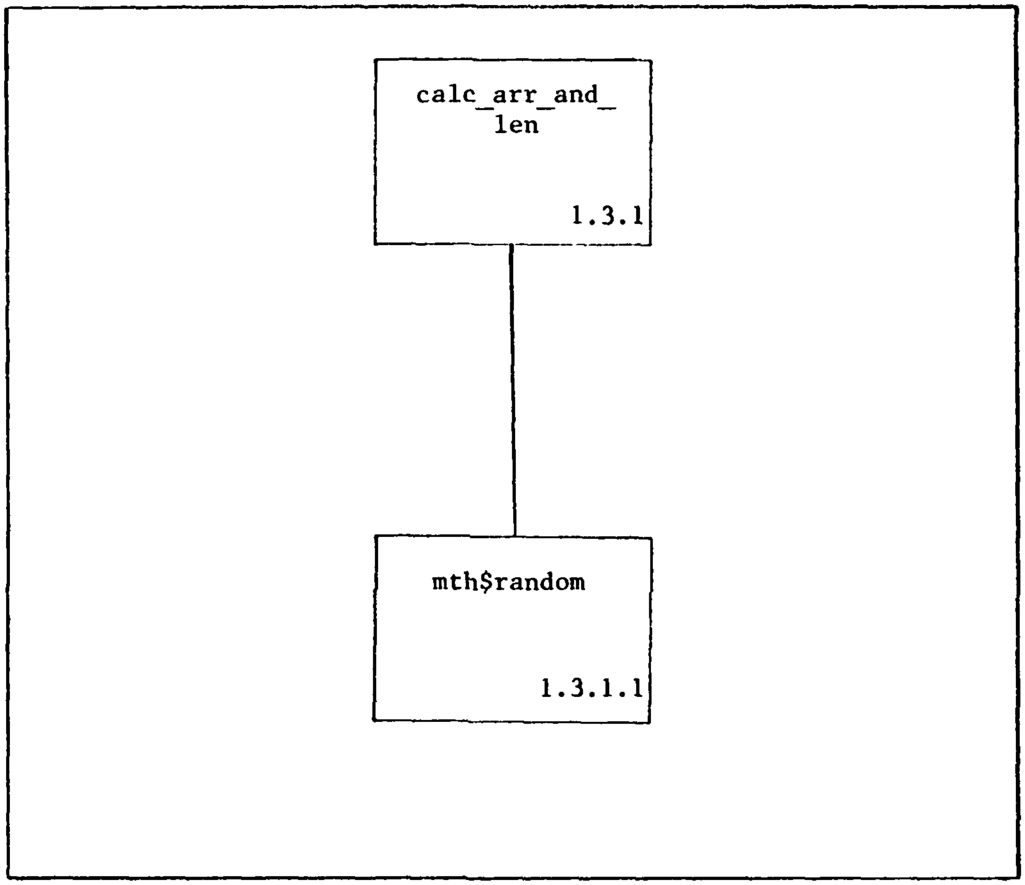


Figure 10. Structure Charts (continued) Node 1.3.1

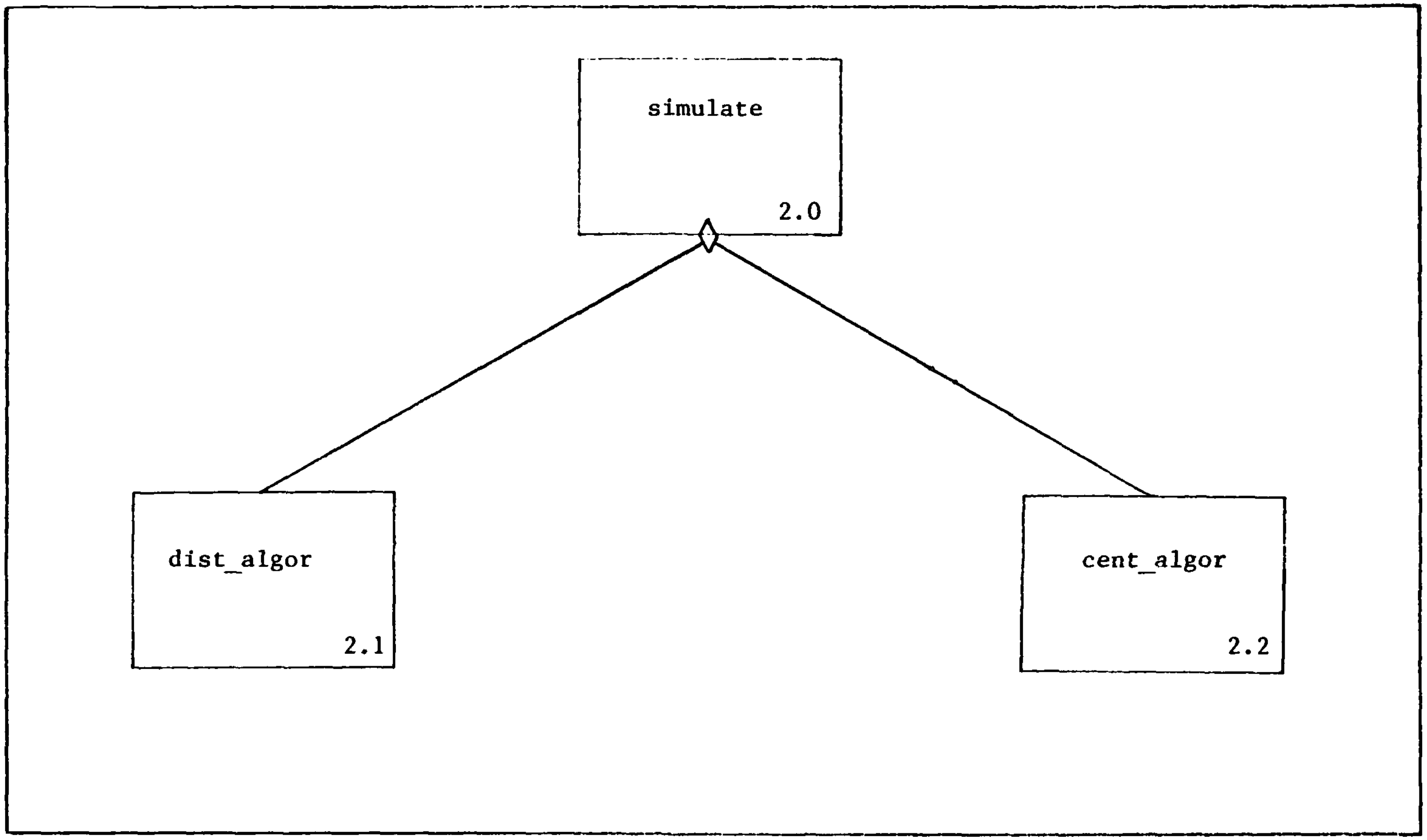


Figure 10. Structure Charts (continued) Node 2.0

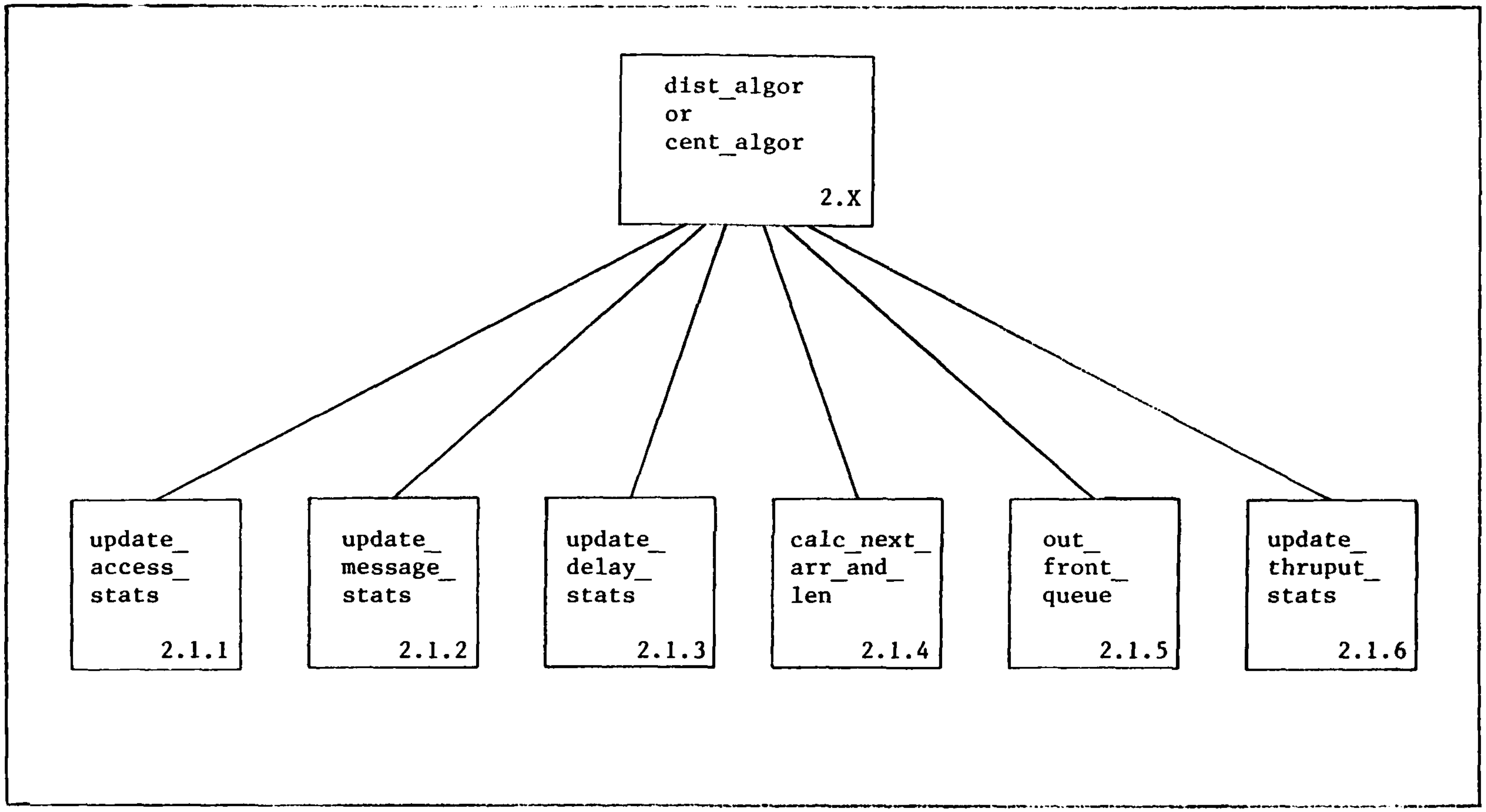


Figure 10. Structure Charts (continued) Node 2.X

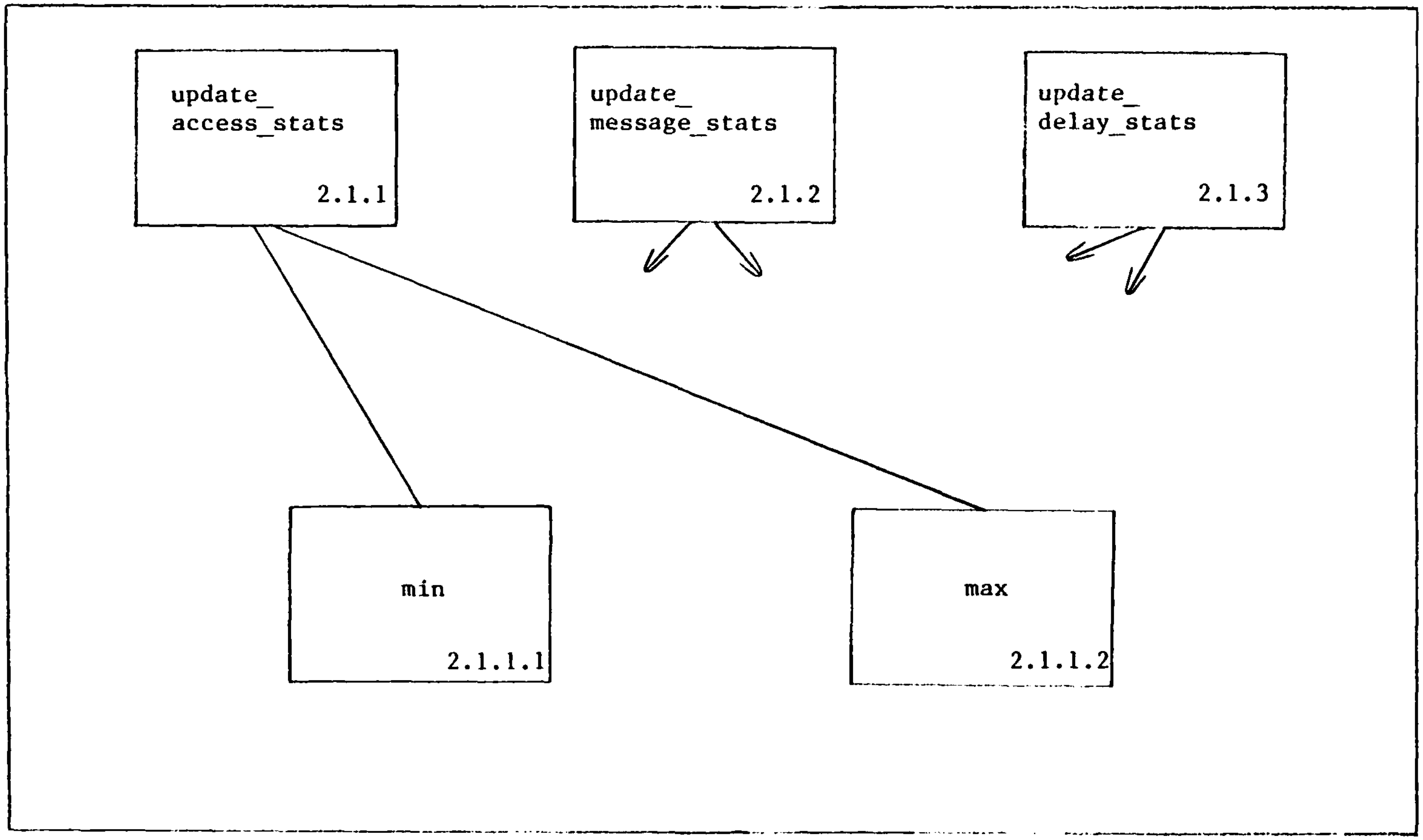


Figure 10. Structure Charts (continued) Node 2.1.X

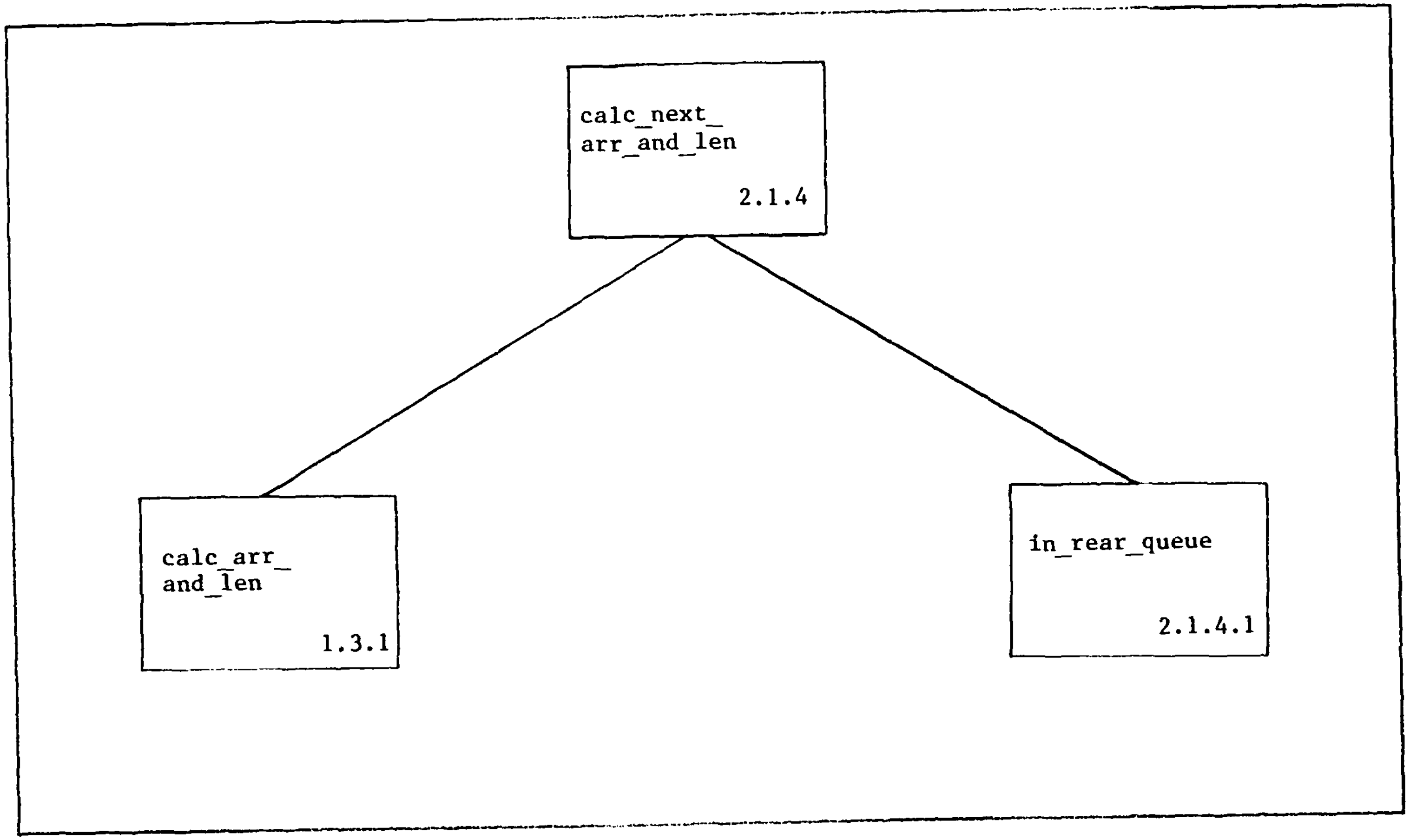


Figure 10. Structure Charts (continued) Node 2.1.4

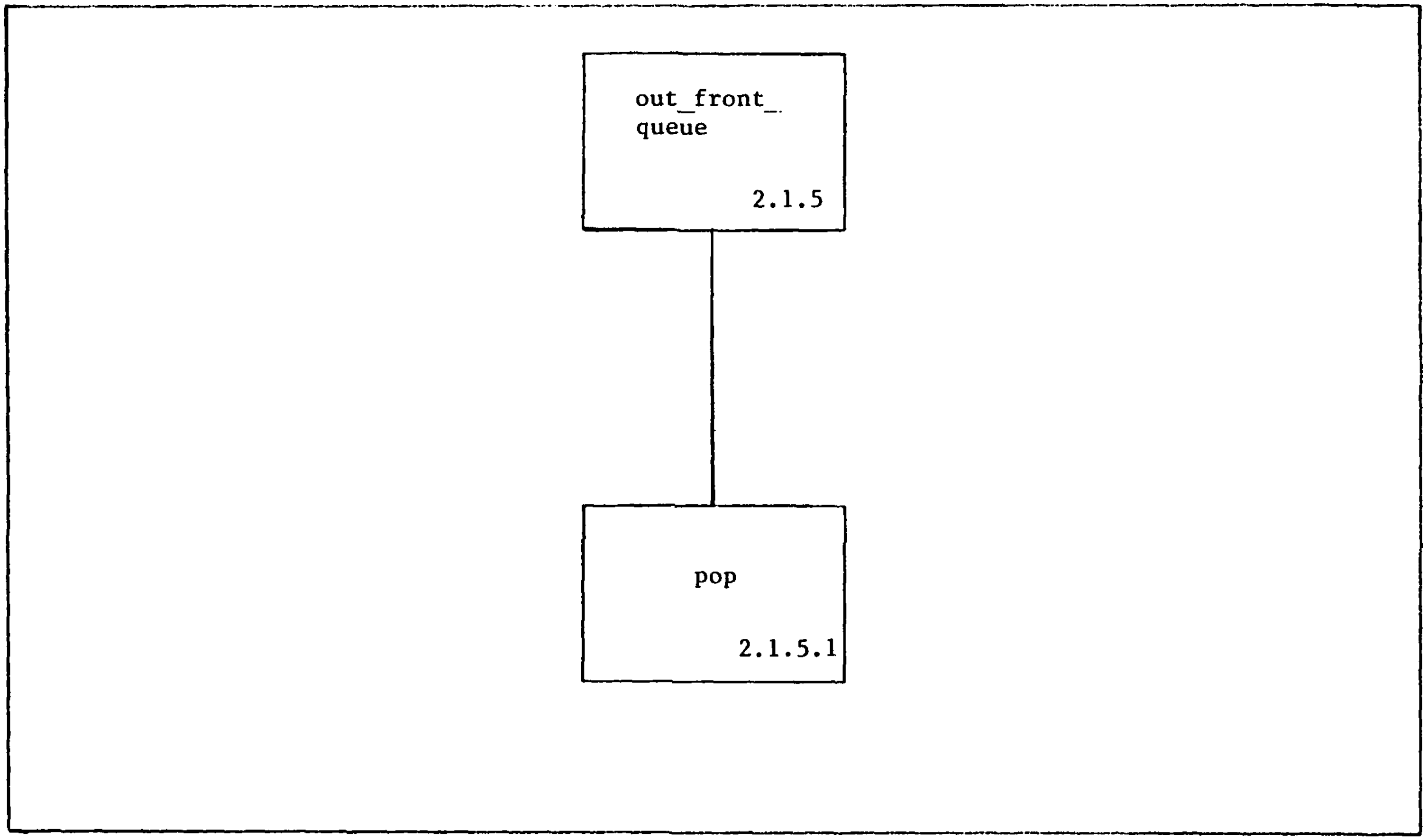


Figure 10. Structure Charts (continued) Node 2.1.5

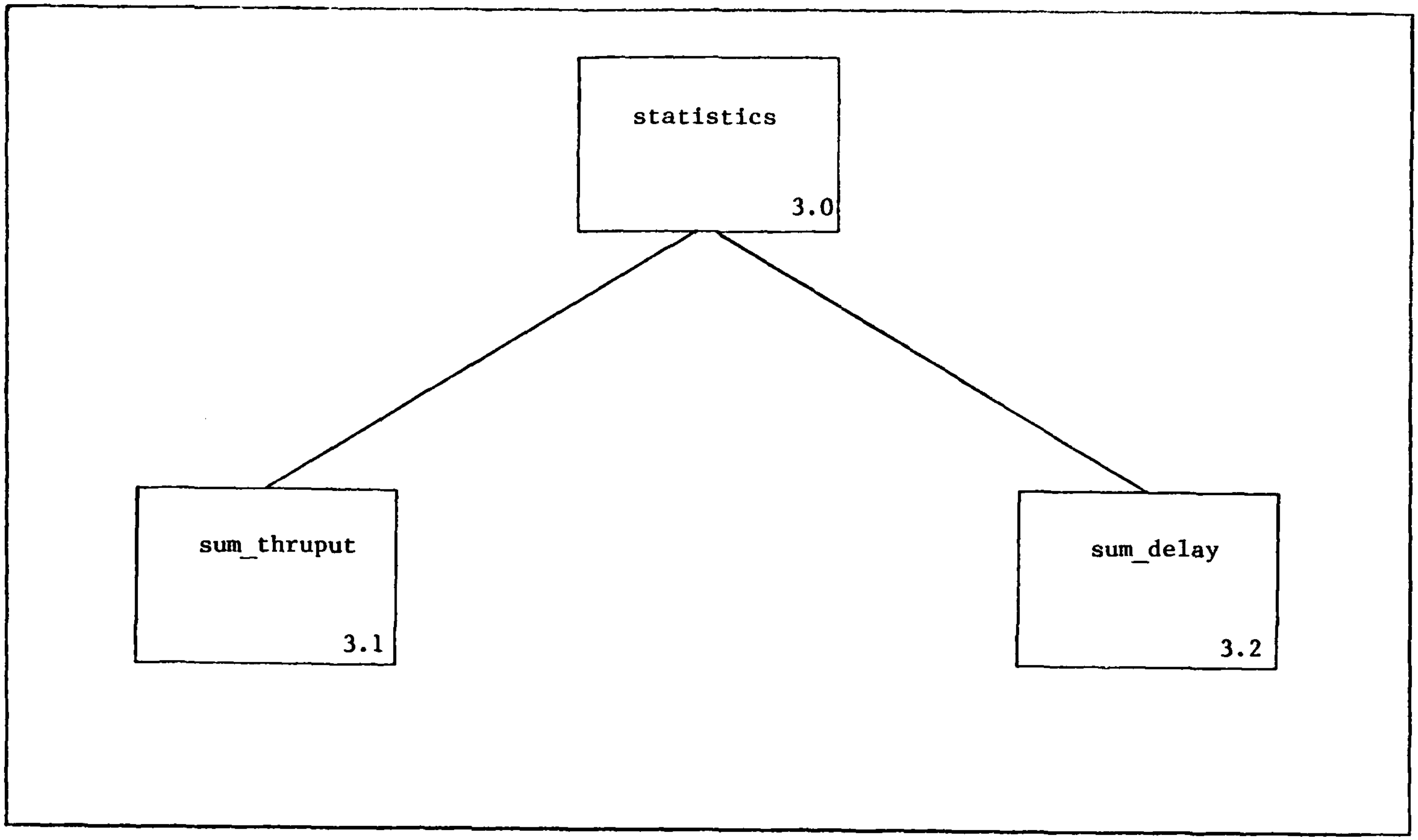


Figure 10. Structure Charts (continued) Node 3.0

Module 2.0. This module implements the functions of the simulation section. It is decomposed into two modules, one for distributed, and one for centralized token-passing protocols. During a particular execution of the simulation model program, only one of these two protocols can be simulated. Each protocol module can call on six other modules to update data statistics variables, generate a new message arrival, or remove a message that has been transmitted, as it performs the simulation. This is shown in the Node 2.X structure chart of Figure 10. Not all of the six modules might be called for a particular station. For example, if the station had continuous type message arrivals, message delays would not apply to the station. Therefore, the message delay statistics would not be calculated or updated and thus the update message delay statistics module (update_delay_stats) would not be called.

Module 3.0. This module implements the functions of the summary section of the program. Module 3.0, statistics, performs the calculations and printing of access delays and message statistics for each station as well as a summary. It calls upon the two other modules to calculate and print message delays (Module 3.2), and token cycles, throughput, and efficiency (Module 3.1).

Implementation

The simulation model program was designed to be run on the sponsor's Digital Equipment Corporation VAX 11/782 computer running the VAX/VMS Version 4.2 operating system. The simulation model was coded in the Pascal language. Only standardized Pascal language constructs were used so that the program could be transported to other computer systems with a minimum of changes and or problems. However, a Digital supplied uniformly distributed random number generator run time library function was used to generate random numbers for the message arrival and length distribution calculations.

The program was designed to be executed in a batch mode. This was done to avoid lengthy delays for the user when long simulations were being run. A users manual for the simulation model program is contained in Appendix A. Also included is an example of the program's output. The simulation model program software is contained in Appendix B.

Execution. A user submits the simulation model program for execution in the batch mode by invoking a command file. This command file contains the program execution instruction and also includes the data needed by the program to set up the bus configuration to be tested. The command file is created by the user, separately from the program, using a text editor. Thus, the user has the option of selecting an existing bus configuration by using an existing command file, or defining a new configuration by creating and using a new command file.

Representation

The three main parts of a bus local area network that are represented by the simulation model program are the bus, messages and stations.

Bus. The bus is simply represented as an entity that transmits bits at a certain rate with no errors. The propagation delay of signals as they move along the bus is modeled. The speed of light is multiplied by an adjustable propagation factor to determine a signal propagation rate and associated time delay.

Messages. Messages are represented as entities that move through the bus local area network. The message attributes include its source station's address, size, and arrival time. The messages are implemented as Pascal records with the message attributes as fields within that record. The form of the message record is shown below.

```
message_type  =  record
                   source_add : integer ;
                   length : real ;
                   arr_time : real ;
                 end ;
```

Stations. The bus stations are represented as entities that are part of the local area network. The station attributes include:

station address
passing address
message arrival type
message arrival rate
distance from start of bus
message length distribution type
message length distribution mean
token passing propagation time to next station
time of last bus access
message queue

The stations are implemented as Pascal records with the station attributes as fields within that record. The form of the station record is shown below. The station's message queue is represented using a single linked list.

```
station_type  =  record
                    address : integer ;
                    pass_address : integer ;
                    mess_arr_type : arrival ;
                    mess_arr_rate : real ;
                    distance : real ;
                    mess_len_type : length_distrib ;
                    mess_len_mean : real ;
                    pass_prop_time : real ;
```

```
                                        last_access : real ;
                                        front_mess_queue : message_ptr ;
                                        rear_mess_queue : message_ptr ;
                                     end ;
```

Performance Parameters

The following paragraphs describe how the various performance parameters are calculated by the simulation model program.

Access Delay. Access delay is calculated when a station has just received the token. It is the difference between the current time and the time when it last received the token.

Message Delay. Message delays are calculated if a station's message arrival distribution is poisson or constant with a non-zero arrival rate. The delay for a particular message is calculated just after it has been transmitted. The delay includes the time the message has been waiting to be transmitted (queueing time), and the time its takes to be transmitted (Bux, 1981:158).

Normalized Delay. The normalized delay is the mean message delay for all messages divided by the mean message length transmission time (Bux, 1981:169).

Token-Passing Cycle. A token passing cycle is the amount of time it takes for the token to be passed through all the stations on the bus.

Throughput. Throughput is the number of data bits transmitted during one token-passing cycle (Rahimi and Jelatis, 1983:800). It is calculated at the end of every token-passing cycle. These throughputs are then averaged and a mean throughput is printed as part of the summary statistics.

Efficiency. Efficiency is the number of data bits transmitted, divided by the total number of bits (data and overhead) transmitted during one token-passing cycle (Ludvigson and Milton, 1985:127). Efficiency is calculated at the end of every token passing cycle and a mean efficiency is printed at the end of the program as part of the summary statistics.

Summary

Simulation allows experiments to be conducted on real and non-existing systems to help answer performance and operation questions. A discrete event simulation is being used in this thesis to explore the performance of an avionics token-passing bus. An overall simulation model program incorporating token-passing simulation algorithms, simulation control, and performance calculations was designed and implemented in the Pascal language.

IV. Test Results

This chapter presents the results of testing accomplished to validate the simulation model program. The results of tests using a fighter-type aircraft bus configuration with the centralized token-passing protocol are also presented to illustrate how the different bus design factors affect this protocol's performance. Finally, a comparison test of the distributed and centralized protocols using the fighter-type bus configuration is presented.

Validation

Validation of the simulation model program was accomplished through a number of different tests. These tests were designed to selectively test a certain aspect of the program.

First Validation Test. The first validation test was designed to test the continuous type of message arrival condition using the distributed token-passing algorithm. This was carried out using a simple bus configuration which is shown in Figure 11. This first validation test is based on a distributed token-passing protocol validation and evaluation done by Rahimi and Jelatis for the IEEE 802.4 Protocol (Rahimi and Jelatis, 1983:800-801). There are six stations spaced evenly on a 500 meter long bus. All six stations have messages to send and they always have messages available. The message size is fixed at two different lengths, 864 data bits or 16224 data bits. There are 160 bits of overhead in a message and the token is also 160 bits in length. The token is passed in ascending station address order, starting with station one. A bit rate of 10 Mb/s and a station delay

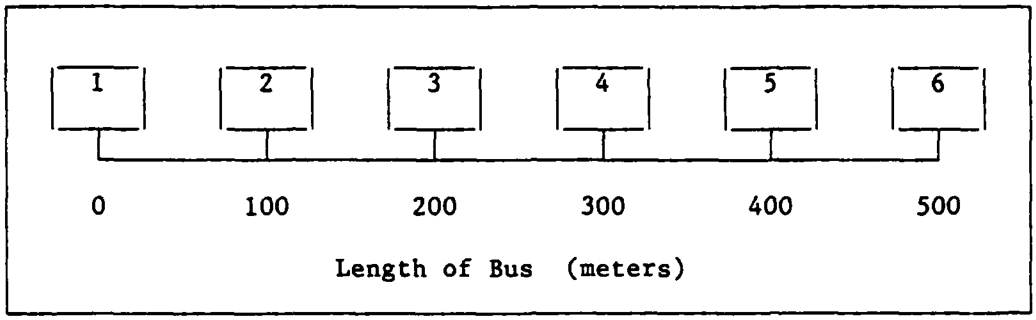


Figure 11. Simple Bus Configuration

time of 0.8 microseconds are used.

The condition of a station always having messages available to send is modeled by a continuous message arrival condition. The token-holding time was adjusted so the stations could only send 1, 2, 4, 8, or 16 of the shorter messages and 1 or 2 of the longer messages.

In order to validate their simulation model, Rahimi and Jelatis constructed simple test cases for which they could develop relatively simple formulas for the token-passing cycle time or token walk time, and throughput (Rahimi and Jelatis, 1983:800). The formulas were then used to produce analytic values for throughput which were compared to the values produced by their simulation model. They first defined the token-passing overhead time per station:

$$P = O + S + D \tag{1}$$

where

O = time to transmit the token message = 16 microseconds

S = station delay time = 0.8 microseconds

D = mean token-passing propagation delay time = 0.6667 microseconds

The formula for the token walk time is then

$$W = (N)(P) + \frac{(M)(K)(F)}{R} \qquad (2)$$

where

N = number of stations on the network = 6

M = number of stations with data to transmit

K = maximum number of messages a station can transmit

F = total message length including data and overhead in bits

R = bit rate = 10 megabits/second

The formula for throughput is

$$T = \frac{(M)(K)(L)}{W} \qquad (3)$$

where

L = number of data bits in a message

The results of this validation test using the simulation model program along with Rahimi and Jelatis' analytic and simulation results are shown in Table II. Each row of Table II represents one run of the simulation model program. Comparing the results, it is seen that this thesis simulation model program's results are in closer agreement with Rahimi and Jelatis' analytic results than their own simulation results. The probable reason for this is their more detailed simulation model. Also, they give no description of how throughput was calculated in their simulation model. Based upon the close simularity between the simulation model program's results and Rahimi and Jelatis' results, this first validation test was successful.

Table II

First Validation Test Results

Number of messages sent	Message size (bits)	Throughput (Mb/s)		
		Rahimi/Jelatis		Simulation Model Program
		analytic	simulation	
1	1024	7.20	7.10	7.21
2	1024	7.77	7.70	7.77
4	1024	8.09	7.89	8.09
8	1024	8.26	8.18	8.26
16	1024	8.35	8.26	8.35
1	16384	9.80	9.63	9.80
2	16384	9.80	9.80	9.85

Second Validation Test. The second validation test was designed to test a case in which the message load is not evenly distributed among the stations. This test utilized both the deterministic distribution and continuous case message arrival conditions. The test was carried out using the same bus configuration as the first validation test. However, in this test, only one of the six stations has messages to send and it always has messages available. This second validation test is again based on a distributed token-passing protocol validation and evaluation done by Rahimi and Jelatis for the IEEE 802.4 Protocol (Rahimi and Jelatis, 1983:800-801). Equations (1), (2), and (3) presented in the first validation test paragraph also apply to this validation test.

Table III

Second Validation Test Results

Number of messages sent	Message size (bits)	Throughput (Mb/s)		
		Rahimi/Jelatis		Simulation Model Program
		analytic	simulation	
1	1024	4.17	4.05	4.17
2	1024	5.58	5.45	5.58
4	1024	6.71	6.59	6.72
8	1024	7.48	7.34	7.48
16	1024	7.93	7.85	7.93
32	1024	8.18	8.07	8.18
1	16384	9.30	9.13	9.31
2	16384	9.60	9.13	9.60

The station that always has messages available to send is modeled by a continuous message arrival condition. The other stations with no messages are modeled by a deterministic distribution message arrival condition with an arrival rate of zero. The token-holding time is adjusted so that a station may send 1, 2, 4, 8, 16, or 32 of the shorter messages and 1 or 2 of the longer messages. The results for the test using the simulation model program along with Rahimi and Jelatis' analytic and simulation results are shown in Table III. Due to the close agreement between the simulation model program's results and Rahimi and Jelatis' results, this second validation test was successful.

Third Validation Test. The third validation test is similar to the first except it is conducted using the centralized token-passing

Table IV

Third Validation Test Results

Number of messages sent	Message size (bits)	Throughput (Mb/s)		Efficiency	
		analy.	sim.	analy.	sim.
1	1024	8.32	8.32	.8438	.8438
4	1024	8.41	8.41	.8438	.8438

protocol. The test was designed to check the continuous message arrival condition and uses the bus configuration shown in Figure 11. All the other bus parameters are the same. However, only a data message length of 864 bits and just two different token-holding time limits are used. The two token-holding time limits allow the stations to send one or four messages. The results of this validation test using the simulation model are shown with the analytical results in Table IV. This validation test was successful based upon the consistent results between the analytical calculations and the simulation program's values.

Fourth Validation Test. The fourth validation test is similar to the second except it is conducted using the centralized token-passing protocol. The test was designed to check the deterministic distribution message arrival condition with an arrival rate of zero and uses the bus configuration shown in Figure 11. All the other bus parameters are the same. However, only one data message length of 864 bits and just two different token-holding time limits are used. The two token-holding time limits allow the stations to send one or four messages. The results of this validation test using the simulation model are shown

Table V

Fourth Validation Test Results

Number of messages sent	Message size (bits)	Throughput (Mb/s)		Efficiency	
		analy.	sim.	analy.	sim.
1	1024	4.52	4.52	.4737	.4737
4	1024	6.93	6.93	.7059	.7059

along with the analytical results in Table V. Comparing the results indicates total agreement between the analytical calculations and the simulation program's results. This validation test was successful.

Fifth Validation Test. The fifth validation test was designed to test the Poisson distribution message arrival condition using the distributed token-passing algorithm. This validation test is based on evaluations of token-passing methods conducted by Cherukuri, Li, and Louis (Cherukuri et al, 1982:68). The bus configuration used for this test is shown in Figure 12. There are 50 stations spaced evenly along a 2000 meter bus. The message arrival rate is the same for all the stations. The message size is fixed at a constant length of 1000 data bits. There are 96 bits of overhead in a message and the token message is 96 bits long also. The token is passed in ascending station order starting with station number one. A bit rate of 10 Mb/s and a station delay time of 2 microseconds are used for the test. The token-holding time limit was fixed at 120 microseconds, which allowed the stations to transmit one message per access.

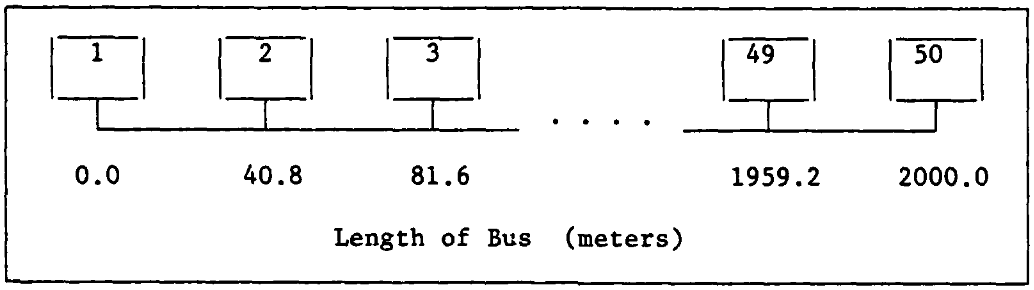


Figure 12. Large Bus Configuration

Messages arrive at the stations according to a Poisson distribution. All stations have an identical mean arrival rate. This mean arrival rate was varied between 30 and 300 messages/second and a number of runs were made using the simulation model program. The results of these runs are compared to Cherukuri's results in Figure 13. The figure shows the normalized delay (mean message delay relative to the mean message transmission time) plotted against the throughput rate relative to the bit rate. Based upon the good agreement between the simulation program's results and Cherukuri's results, this validation test was successful.

Sixth Validation Test. The sixth validation test was designed to test the exponential distribution of message lengths. This validation test is based on a distributed token-passing analysis and simulation study conducted by Ulug (Ulug, 1984). Ulug found that when stations were limited to transmitting only one message per token-holding turn, there was little variation in the mean token-passing cycle time when using fixed message lengths or exponentially distributed message lengths with the same mean (Ulug, 1984:39).

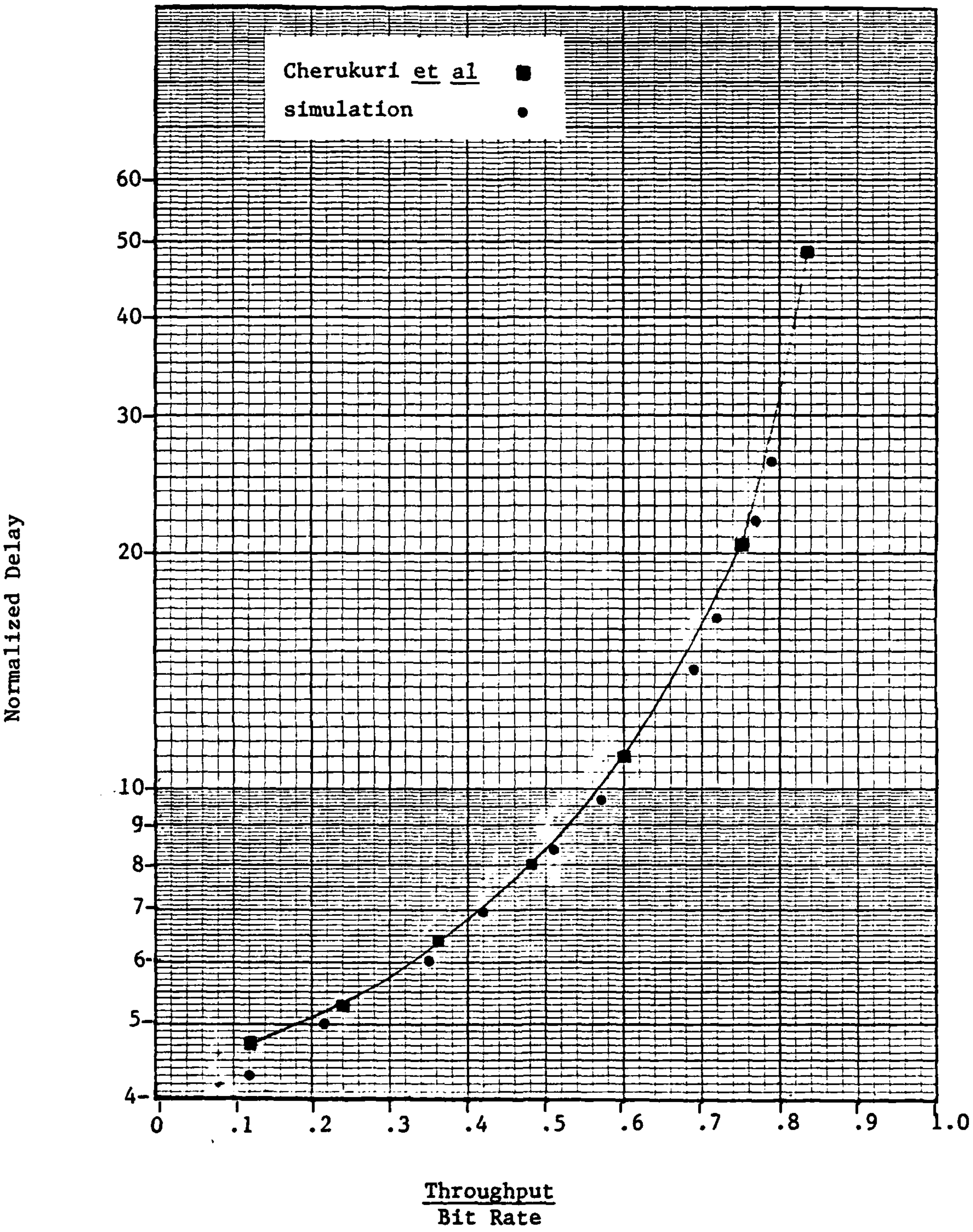


Figure 13. Fifth Validation Test Results

Table VI

Sixth Validation Test Results

total token-passing time (micro seconds)	message length distribution type	mean token-passing cycle time (milliseconds)		standard deviation	
		Ulug	Simulation Model Program	Ulug	Simulation Model Program
75.0	fixed	6.411	6.412	0.782	0.000802
125.0	fixed	10.575	10.152	0.854	0.000901
175.0	fixed	14.795	13.694	0.871	0.000798
75.0	exp.	6.252	6.308	1.219	0.001052
125.0	exp.	10.668	10.062	1.339	0.001201
175.0	exp.	14.707	13.493	1.467	0.001151

The bus configuration used for this validation test consisted of fifty stations spaced evenly on the bus. Ulug does not indicate the length of the bus but rather specifies a total token-passing time which includes token transmission time, token propagation time, and the station delay time. Three different values are used for this total token-passing time; 75 microseconds, 125 microseconds, and 175 microseconds. A message is 864 bits in length. The message overhead is 160 bits and the token is also 160 bits in length. A 5 megabit/second bit rate is used.

The results of this test using the simulation model program and Ulug's results are shown in Table VI. The difference between the simulation model program's mean token-passing cycle times and Ulug's times exists because he did not state at what message arrival rate (load) his tests were run. Also, a decimal point error must have been made in his calculation of the standard deviations. However, when comparing the values in Table VI, a general consistency for the mean token-passing cycle times and standard deviations between Ulug's results and the simulation model program's results can be seen. This validation test was considered successful.

Validation Tests Summary. All the validation tests were successful. Since no other simulation or analytical results exist for the cases of Poisson type arrivals and exponentially distributed message lengths using the centralized token-passing protocol, a comparison check between the distributed and centralized software was done. These checks, along with the knowledge that the same support procedures (calc_arr_and_len for example) are called by both protocol algorithms, added to the confidence that the centralized protocol algorithm was also working correctly. These validation tests and checks show that the simulation program successfully models a bus local area network using a distributed or centralized token-passing protocol.

Aircraft Test Case

Initial performance tests of the centralized token-passing bus protocol while varying some of the bus design factors were conducted using the simulation model program. The bus configuration tested was representative of an avionics local area network for a fighter-type aircraft. The aircraft size information is based on the dimensions of the F-15 aircraft and the basic bus configuration was suggested by Alber

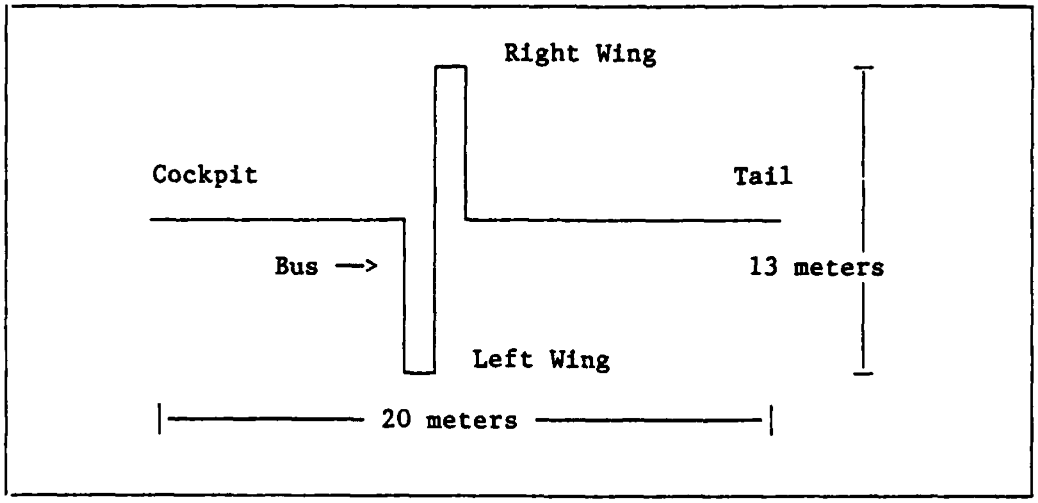


Figure 14. Fighter-Type Aircraft Bus Configuration

(Alber, 1985). The bus was 60 meters in length and had 30 stations connected to it. The stations were positioned on the bus corresponding to where avionics would normally be located. These locations included the cockpit area, an equipment bay aft of the cockpit, both wing areas and the tail section. The stations were divided evenly among these five areas, six stations to an area. The bus configuration is shown in Figure 14. The bus starts at the cockpit (distance 0.0 meters), goes to the left wing, then to the right wing and ends at the tail (distance 60.0 meters). The bus length is greater than the physical dimensions of the aircraft because the actual routing distances through the aircraft for the bus cable are taken into account.

Details of the bus configuration are as follows. There are 70 bits of overhead in a message and the token message is 22 bits in length. A data word consists of 16 bits. There can be a minimum of zero data words and a maximum of 256 data words in a message. A bit rate of 50 megabits/second is used and the station delay time is 0.5 microseconds.

The stations are allowed to hold the token for a maximum of 83.32 microseconds, which is the time it would take to transmit a message with the maximum number of data words in it (256). The token-passing cycle starts with station number one and the token is passed in ascending order based upon the station's address. The Poisson distribution type of message arrival condition is used and messages have exponentially distributed lengths. In order to keep the message size within the minimum and maximum limits, the initial length generated is checked by the simulation program to make sure it is greater than or equal to the minimum length and less than or equal to the maximum length. If the message length is outside these limits, its value is changed to whichever limit it exceeded.

First Test Case. The first test case for this fighter-type aircraft bus configuration was a comparison of equal and unequal station message arrival rates. For unequal message arrival rates, the stations were divided into three classes: low, medium, and high. These classes had a message arrival rate ratio of 1/5/50. This condition of unequal message arrival rates was used to simulate the different data output or data update rates that avionics have or require. For example, mission or fire control computers produce and/or require high rates of data messages per second in order to accomplish their functions with the level of accuracy needed for modern military aircraft. The stations' addresses, location on the bus, distance from the left end of the bus, and message arrival class are shown in Table VII.

For the equal message arrival rate simulation, all the stations had the same mean message arrival rate. This mean message arrival rate was

Table VII

Aircraft Test Case Station Data

Station	Location	Distance (meters)	Message Arrival Class
1	Cockpit	2.0	Low
2	Cockpit	2.5	Medium
3	Cockpit	3.0	High
4	Cockpit	3.5	High
5	Cockpit	4.0	Medium
6	Cockpit	4.5	Low
7	Equip. Bay	5.0	Low
8	Equip. Bay	6.0	Medium
9	Equip. Bay	7.0	High
10	Equip. Bay	8.0	High
11	Equip. Bay	9.0	Medium
12	Equip. Bay	10.0	Low
13	Left Wing	13.0	Low
14	Left Wing	14.0	Medium
15	Left Wing	15.0	High
16	Left Wing	16.0	High
17	Left Wing	17.0	Medium
18	Left Wing	18.0	Low
19	Right Wing	28.0	Low
20	Right Wing	29.0	Medium
21	Right Wing	30.0	High
22	Right Wing	31.0	High
23	Right Wing	32.0	Medium
24	Right Wing	33.0	Low
25	Tail	55.0	Low
26	Tail	56.0	Medium
27	Tail	57.0	High
28	Tail	58.0	High
29	Tail	59.0	Medium
30	Tail	60.0	Low

varied from 300 messages per second to 3500 messages per second. For the unequal message arrival rate simulation, all stations in a class had the same mean message arrival rate. However, the mean rates were different among the three classes. The mean message arrival rates were varied from 10 to 2000 messages per second for the Low class, 50 to 10,000 for the Medium class, and 500 to 100,000 for the High class.

Both message arrival rate simulations used exponentially distributed messages lengths with a mean of 64 data words. The results of these simulations are shown in Figure 15 as normalized delay-throughput curves.

From the results, it can be seen that the unequal arrival rate case has smaller delays for medium load (throughput/bit rate) values. At low load values, both the equal and unequal arrival rate conditions delays are very similar; while at very high load values, the unequal rate condition has higher delays. This difference in delay values indicates the assumption of equal mean message arrival rates for stations can lead to pessimistic delay values.

The command files used to provide the bus configuration data to the simulation model program for this test case and the other following test cases are contained in Appendix C.

Second Test Case. The second test case for the fighter-type aircraft bus configuration was a comparison of different mean message lengths. Simulations were done with mean message lengths of 32 data words, 64 data words and 128 data words. All three simulations used the unequal station message arrival rate condition from the first test case with the same mean arrival rates. The results of this second test case are shown in Figure 16 as normalized delay-throughput curves.

From the test case results, it can be seen that a larger mean message size results in higher throughputs for the same amount of delay. This improvement in performance can be explained by comparing two networks with different mean message lengths. The network with a larger mean message size will be transmitting a larger number of data bits for

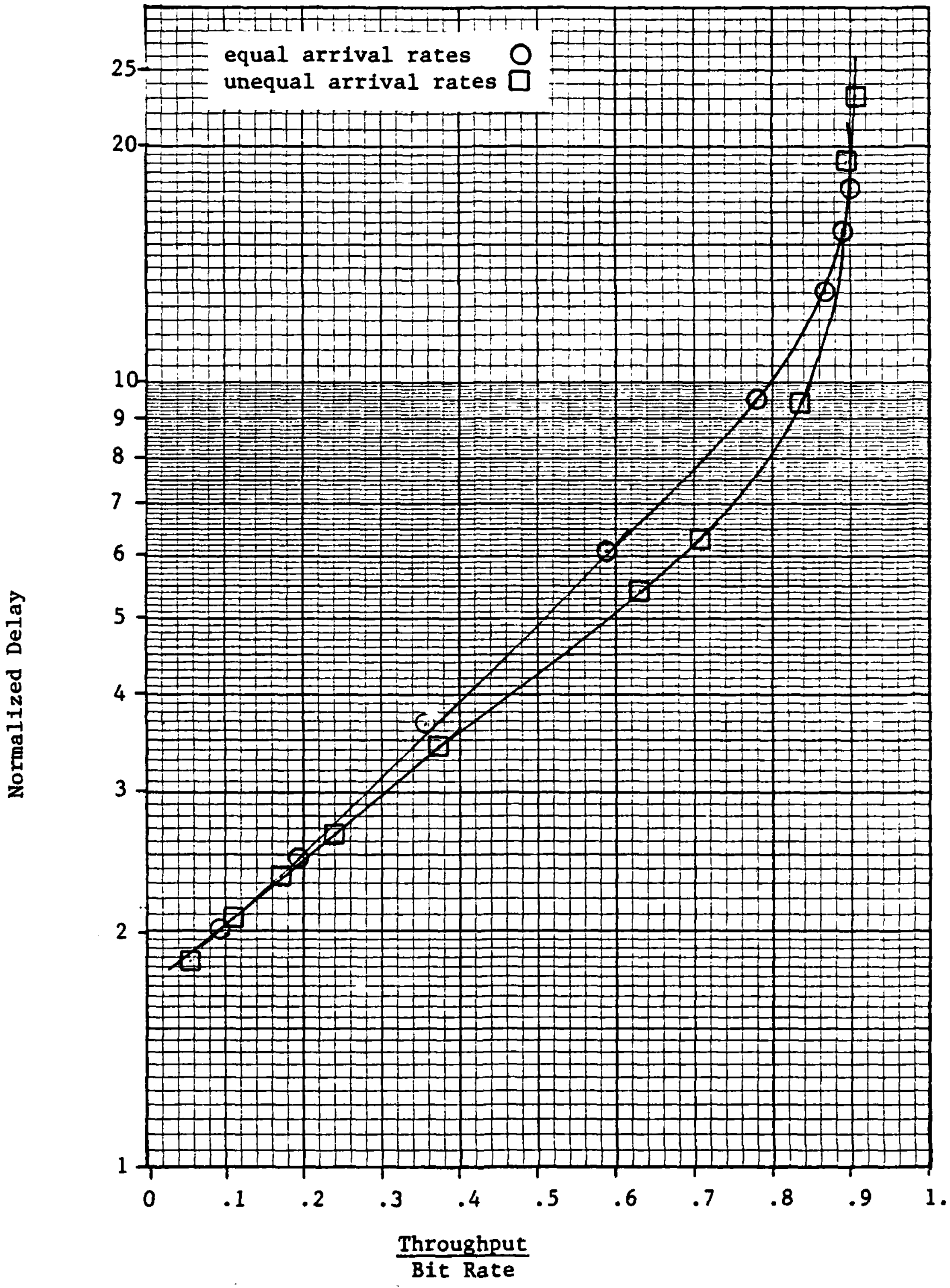


Figure 15. First Test Case Results

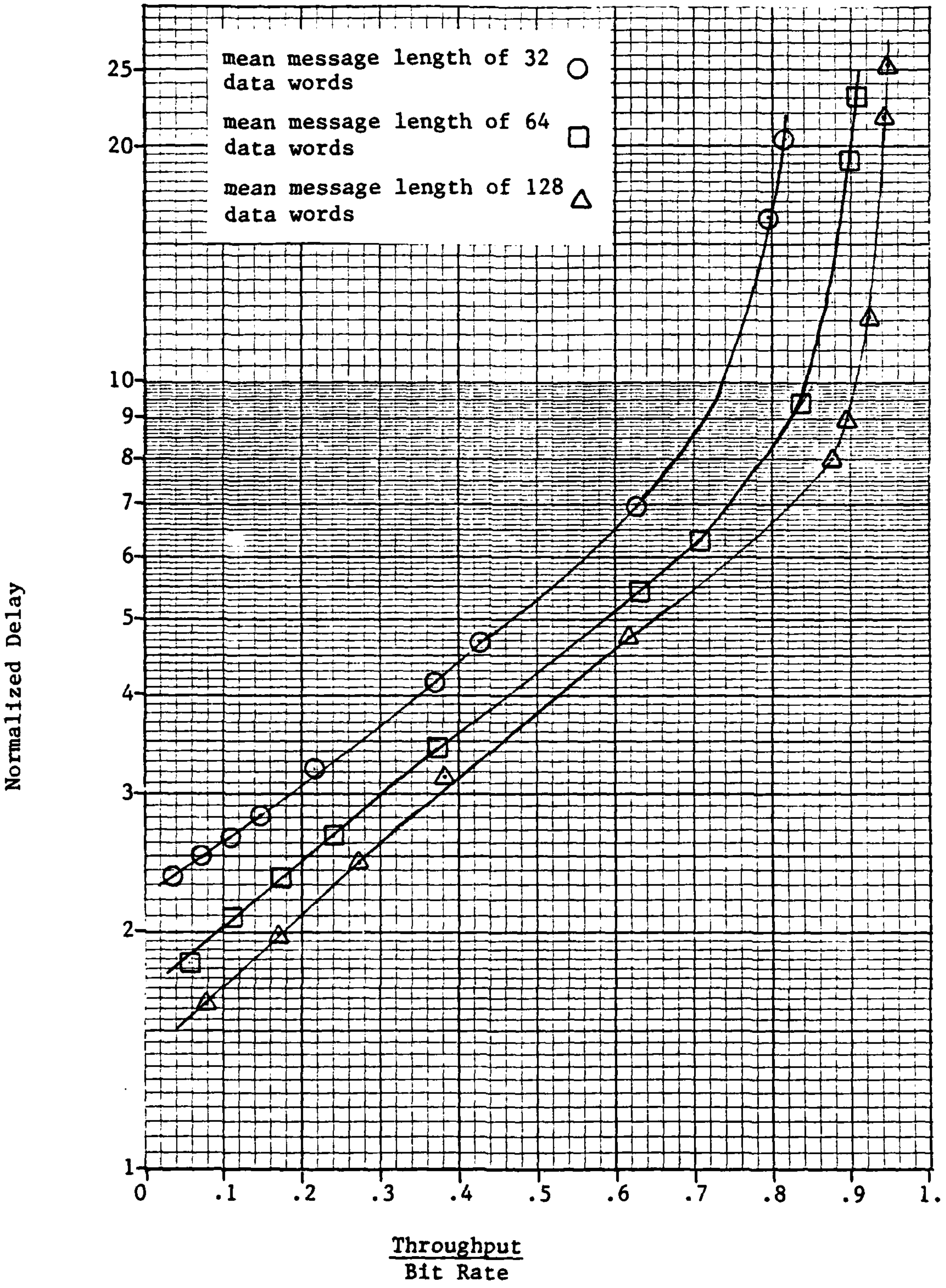


Figure 16. Second Test Case Results

1-2-3-4-5-6-7-8-9-10-11-12-13-14- 28-29-30-1

optimum token-passing sequence

1-30-2-29-3-28-4-27-5-26-6-25-7- 14-17-15-16-1

worst-case token-passing sequence

Figure 17. Token-Passing Sequences

the same number of bus accesses. Rahimi and Jelatis also noted higher throughput and efficiency values for longer message lengths (Rahimi and Jelatis, 1983:801).

Third Test Case. The third test case for this fighter-type aircraft bus configuration was a comparison of optimum and worst-case token-passing sequences. An optimum token-passing sequence, where the token is passed to adjacent stations, was compared to a worst-case sequence. In the worst-case sequence, the token is passed to the farthest unvisited station (Cherukuri et al, 1982:59). These sequences are shown in Figure 17. For these simulations, an exponential message length distribution with a mean of 64 data words was used. The unequal station message arrival rate condition was used with the same mean arrival rates as in the previous test cases. The results of these simulations are shown in Figure 18 as normalized delay-throughput curves.

From the test case results, it can be seen that there is not a significant performance difference between the two token-passing sequences. It is known that propagation delays due to station separation and bus length effect the performance of bus networks (Stuck,

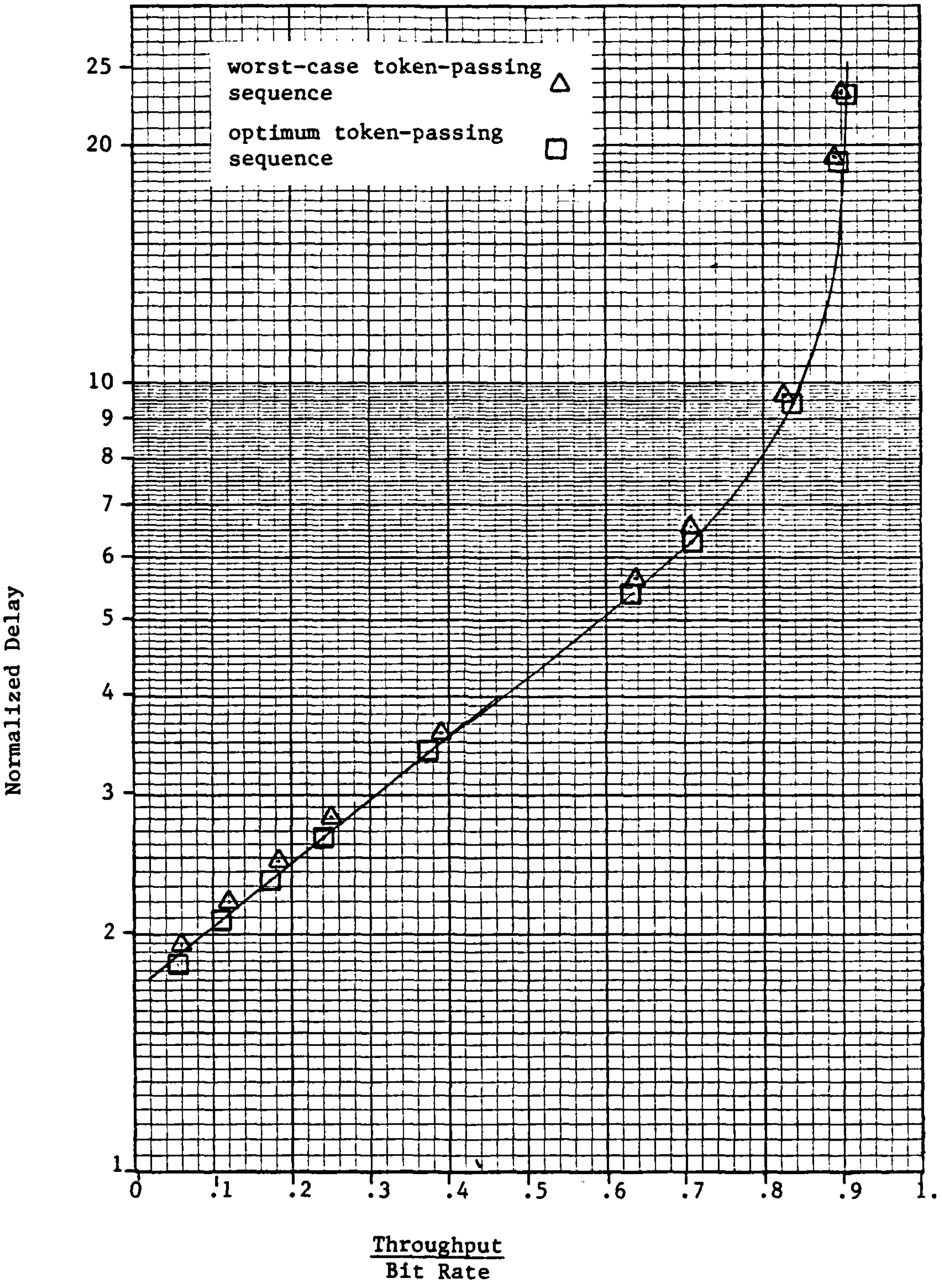


Figure 18. Third Test Case Results

1983a:75-76). However, in the case of the avionics bus configuration simulated in this test case, the token-passing sequence does not effect the performance of the bus because of the small distances involved and the high bit rate. Since most avionics buses will be short in length, the performance degradation due to propagation delays will not be a significant factor.

Fourth Test Case. The fourth test case using the fighter-type aircraft bus configuration was a comparison of different bit rates. Simulations were performed with bit rates of 25 megabits/second, 40 megabits/second and 50 megabits/second. All three simulations used the unequal station message arrival rate condition from the first test case with the same mean arrival rates. The results of this test are shown in Figure 19 as normalized delay-throughput curves.

The test case results indicate no large differences in the normalized delay-throughput/bit rate ratio curves for the three different bit rates. However, when the non-normalized mean message delays shown in Table VIII are compared, the delays at 25 Mbits/second are approximately twice as long as the delays at 50 Mbits/second and the delays at a bit rate of 40 Mbits/second are approximately a third longer than at 50 Mbits/second.

Fifth Test Case. The fifth test case using the centralized control protocol in a fighter-type aircraft bus configuration involved varying the maximum message size. Operation of the centralized control protocol was simulated with a maximum message size of 256 data words, 1024 data words, and 4096 data words. A maximum message length of 256 data words was chosen for one of the test cases because this is the size limit used

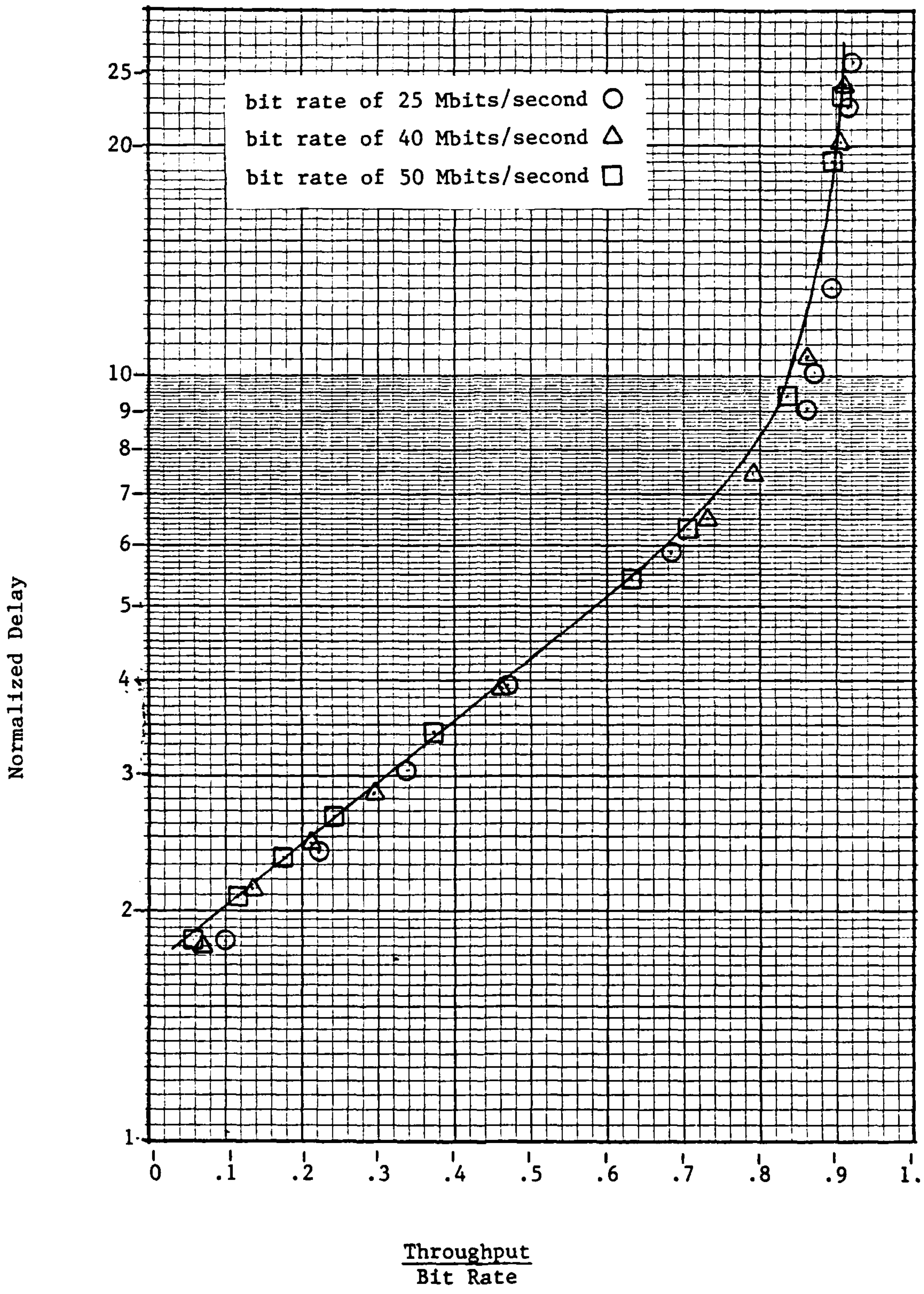


Figure 19. Fourth Test Case Results

Table VIII

Fourth Test Case Mean Message Delays

mean arrival rate (messages/second)	Mean Message Delay (milliseconds)		
	25 Mbits/sec	40 Mbits/sec	50 Mbits/sec
186.6	0.080	0.049	0.040
373.3	0.103	0.057	0.045
560.0	0.132	0.066	0.051
746.6	0.167	0.076	0.056
1120.0	0.249	0.104	0.074
1866.0	0.393	0.177	0.117
2240.0	0.428	0.200	0.135
3733.3	0.565	0.286	0.203
18666.0	0.967	0.540	0.408
37333.3	1.114	0.644	0.497

in the Systems Engineering Avionics Facility centralized protocol. A maximum message length of 4096 data words was chosen for this test because this is the size limit used in both the SAE and Avionics Laboratory distributed protocols. A mean message length of 64 data words was used for this test case. This test used the unequal station message arrival rate condition from the first test case with the same mean arrival rates. The results of this test are shown in Figure 20 as normalized delay-throughput curves.

From the test case results, it can be seen that the normalized delay-throughput curves are very similar for the different maximum message lengths. Also, there are no major differences in the mean

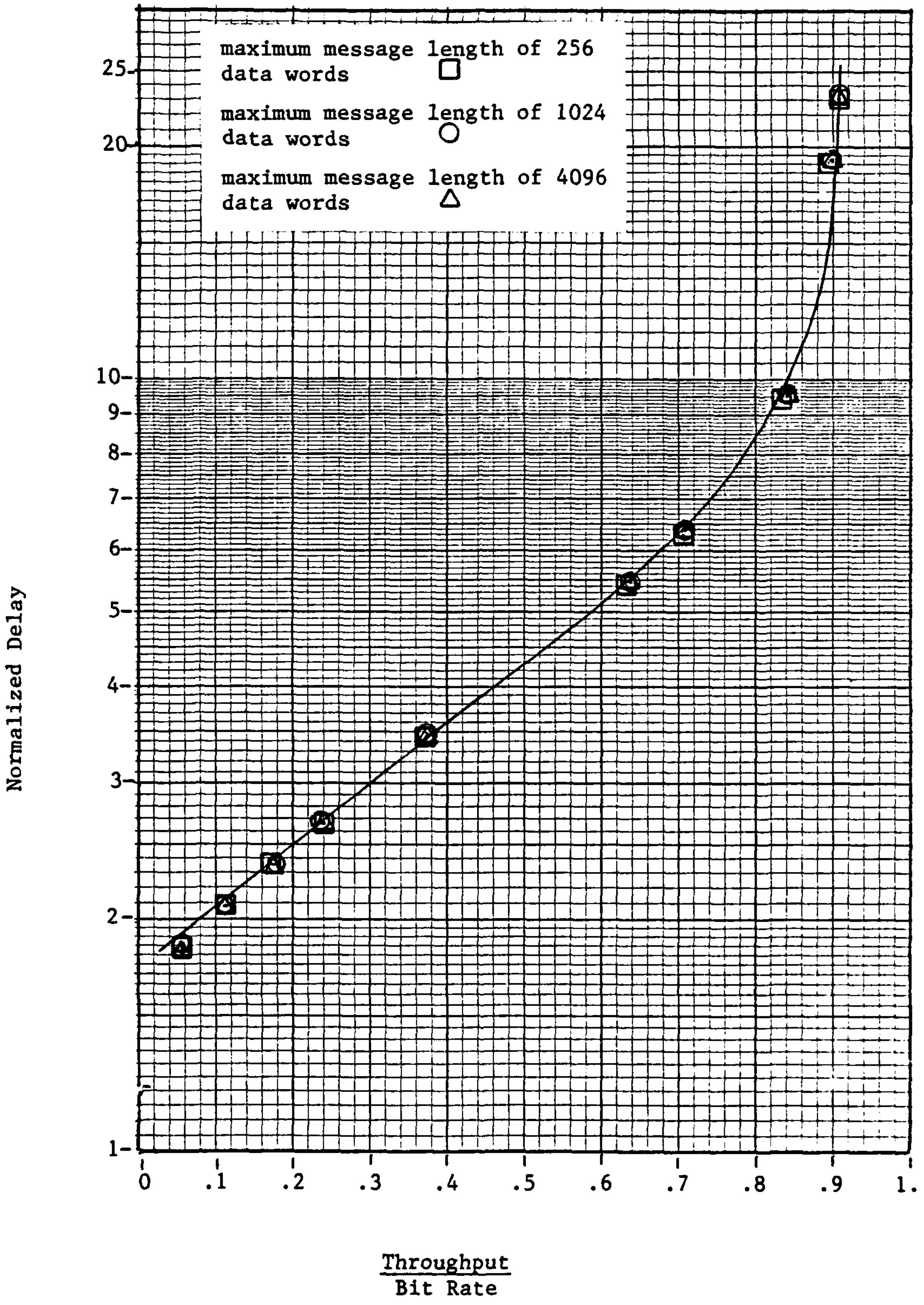


Figure 20. Fifth Test Case Results

message delays. Two conditions are likely to have contributed to this lack of variation in performance parameters for this test case. The first is that the same value for the mean message length (64 data words) was used for all three maximum message length simulations. The second condition is that the same message arrival rates were also used for all three simulations. The simulations with the smaller maximum message lengths (256 and 1024 data words) actually should have had slightly higher message arrival rates. This would be necessary because multiple messages would be required using those protocols to transfer the same amount of data as could be transferred with one message of 4096 data words using the third protocol.

Sixth Test Case. The sixth test case examined the effect of a different type of message arrival distribution on the centralized protocol in the fighter-type aircraft bus configuration. A deterministic distribution message arrival condition was compared to a Poisson distribution condition. The unequal station message arrival condition from the first test case with the same mean arrival rates was used in this test. The deterministic distribution used the same mean message arrival rates used for the Poisson distribution. The results of this test are shown in Figure 21 as normalized delay-throughput curves.

Having messages arrive according to a Poisson distribution is the normally assumed condition in queueing systems analyses, as this type of arrival condition successfully models the random arrival of messages in real systems (Tanenbaum, 1981:58). In current MIL-STD-1553 avionics bus systems, information is usually updated at a bus station at a deterministic rate. This is done because the bus controller follows a

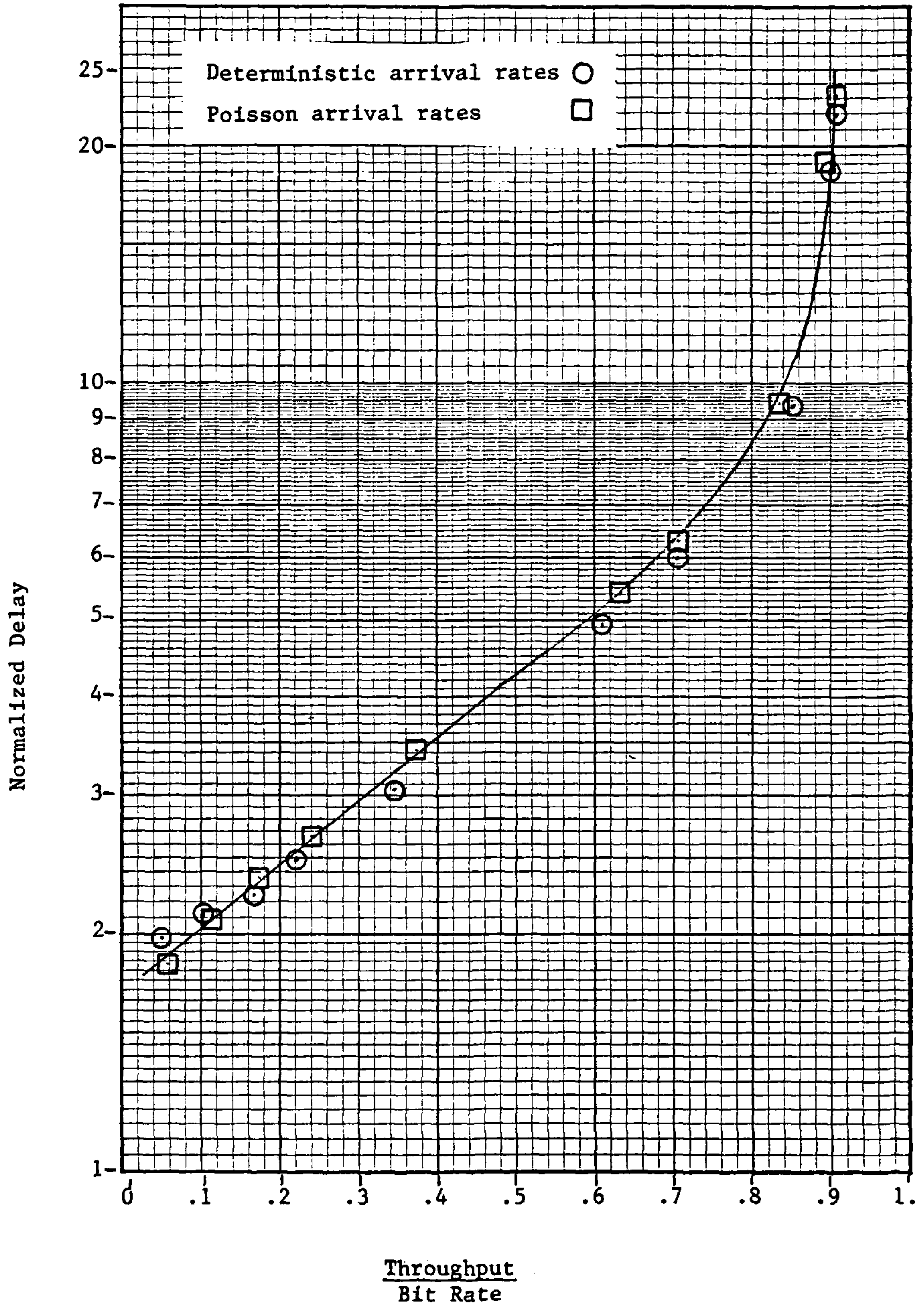


Figure 21. Sixth Test Case Results

defined timing cycle in requesting and providing data to the stations due to the constraints of its hardware and software components (Boeing, 1980:5-3). However, with a token-passing protocol for a new avionics network, this practice could still be used but would no longer be necessary. From the test case results shown in Figure 21, it can be seen that no significant differences exist between the deterministic and Poisson arrival distribution delay-throughput curves.

Seventh Test Case. The seventh test case compares the distributed control and the centralized control protocols using the fighter-type bus configuration. The centralized control protocol used for this test is the same one used for all the previous tests. It is based on the protocol proposed by the Systems Engineering Avionics Facility. The distributed control protocol tested is based upon the IEEE 802.4 protocol. Although the IEEE protocol has a maximum bit rate of 10 megabits/second, a bit rate of 50 megabits/second is used in the distributed control protocol for this test in order to obtain a more realistic comparison. A 50 megabits/second bit rate might be obtained with the IEEE protocol if fiber optics were used as the media. The results of this comparison test are shown in Figure 22 as normalized delay-throughput curves.

The results show that the distributed control protocol's normalized delay-throughput/bit rate ratio curve is shifted up and to the left of the centralized protocol's curve. This means that for the same throughput value, the distributed protocol has a higher message delay time than the centralized protocol. From the other aspect, for the same delay value, the distributed protocol has a lower throughput rate than

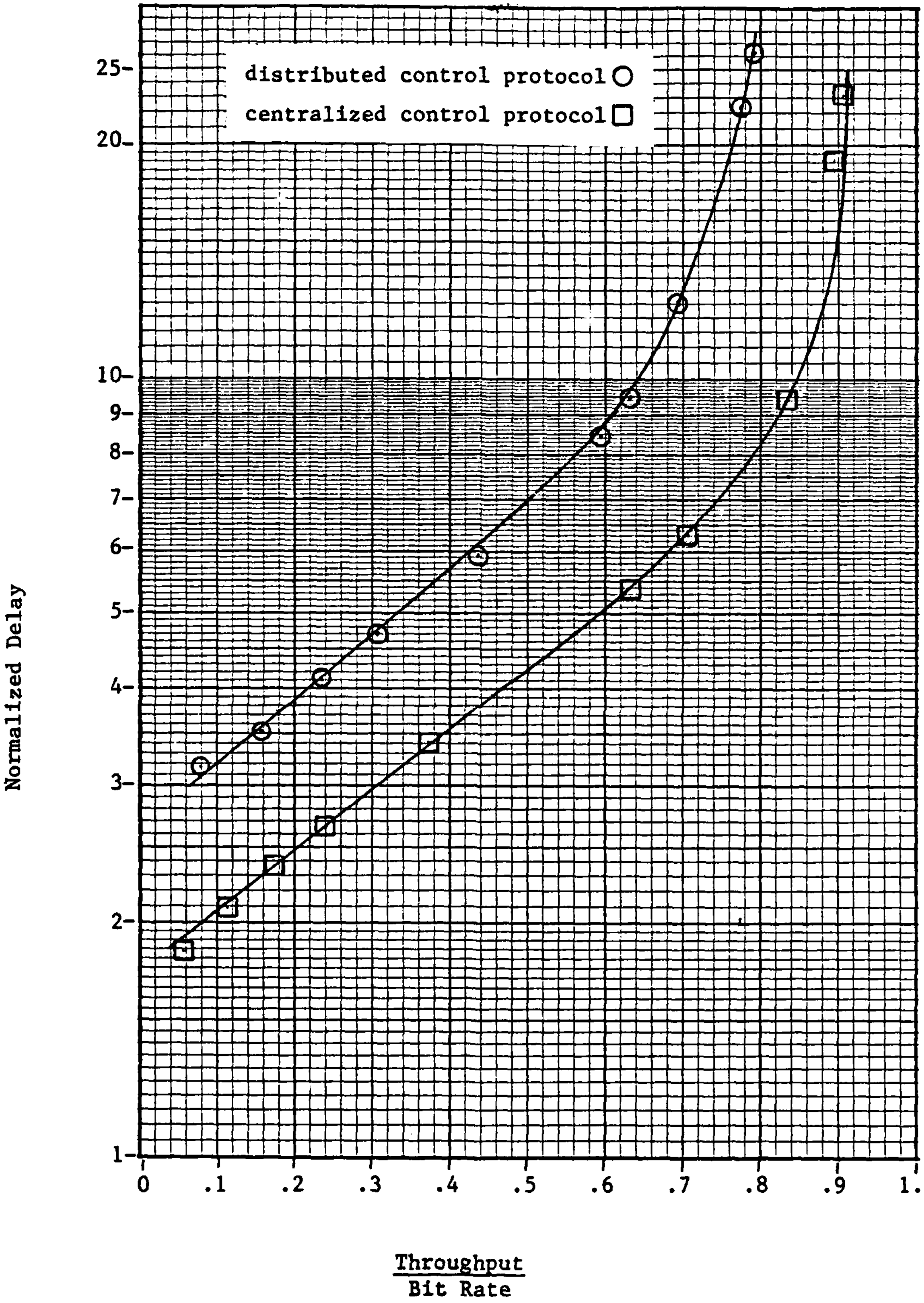


Figure 22. Seventh Test Case Results

the centralized protocol. This poorer performance of the distributed protocol results from its separate token message and larger number of overhead bits in a data message as compared to the centralized protocol. However, as discussed in the next chapter, the distributed protocol may still be preferred in practice for other reasons in spite of its poorer delay-throughput performance.

Summary

The simulation model program developed by this thesis was validated through a series of tests. These tests allowed the operation and performance of the simulation model program to be compared to published or analytical operation and performance results. Initial tests were conducted using the centralized token-passing protocol in a fighter-type aircraft avionics bus configuration to determine the different bus design factors' influence on the protocol's performance. A comparison test between the distributed control and centralized control protocols was also conducted.

V. Summary and Recommendations

This chapter summarizes the testing conducted in Chapter IV and includes a discussion of how the testing results relate to an avionics network environment. A summary of this thesis is then presented along with recommendations for future studies.

Summary of Test Results

The validation and initial performance/parameter variation tests from Chapter IV are summarized and discussed in the following paragraphs.

Validation Tests. Validation of the simulation model was conducted in stages with each stage adding a protocol characteristic to the model. As each characteristic was added, the complexity of the model increased. The bus and protocol parameters for many of the tests, especially for the distributed control protocol, were designed to duplicate testing conducted by other investigators and reported in the literature. Thus, the simulation model program's results could be and were compared to published results. In the case of the centralized control protocol where no previous work existed, the test results were compared to analytical results for the simple test cases. The only check that could be accomplished for the more complicated test cases, was a software cross-check of the centralized control protocol algorithm with the distributed control protocol algorithm. The successful outcome of all the validation tests indicates the ability of the simulation program to model a token-passing protocol bus network.

Parameter Variation/Performance Tests. The parameter variation/performance tests were conducted to determine the effect of the different bus and protocol parameters on the performance of the centralized control protocol. A comparison test between the centralized and distributed protocols was also conducted.

Message Arrival Conditions. The various avionic equipment on a network all have different information transfer requirements based upon their function. These information transfer requirements can be translated into messages sizes and transmission rates which are usually different for each station. This results in a condition of numerous unequal station message arrival rates. This message arrival condition could be modeled by the simulation model program since it allows all stations to have different message arrival rates.

Bit Rate Variation. When the bit rate was varied in the Fourth Test Case, no large differences were noted in the normalized delay-throughput curves for the three rates. However, when the non-normalized mean message delays were compared, there were differences. The non-normalized mean message delays at 25 megabits/second were more than twice as long as the delays at 50 megabits/second. This comparison raises the question of whether throughput or message delay is the most important performance parameter in an avionics network. In most cases, message delay is considered the most important parameter because of the real-time nature of an avionics network. A station, as part of the aircraft's avionics system, requires information to perform its function. Since the aircraft and the environment around it are changing at a rapid rate, information can become "old" and thus worthless very quickly because the information represents an condition of the aircraft or its environment that no longer exists. Thus, limits on mean and worst case message delays are required to make sure the information is received by a station before the information becomes old and useless.

For some avionics, small message delays are not as critical due to the function the avionics are performing or the type of information being processed. In this case, message delays are not as important and network tradeoffs can be made. For example, a lower bit rate could be used reducing the complexity and the cost of the bus and station hardware components.

Although the 25 megabits/second bit rate is exactly half of the 50 megabit/second bit rate, the delays for the 25 megabit/second bit rate were more than twice the 50 megabit/second delays. The reason for this difference is a slightly different total number of messages transmitted/total number of station accesses ratio and a slightly different actual mean message length for each bit rate simulation.

Protocol Comparison. In the Seventh Test Case, the centralized control protocol had a better normalized delay-throughput curve than the distributed control protocol. However, from an overall avionics network system viewpoint, additional factors besides delay-throughput performance must be considered. While the distributed protocol does have more overhead bits in a message, these bits allow more bus operation options. These options include more addresses which allow additional stations on the bus, and more subaddresses for groups of stations. Also with the distributed control protocol, control of the network is shared equally by all stations. This avoids problems of single points of failure that the centralized control protocol has with its one scheduler station, or one scheduler station and one backup scheduler station. Finally, since the IEEE 802.4 protocol is a widely-accepted standardized protocol, a network using it could benefit in a number of possible ways. These benefits might include having multiple sources of proven low-cost station hardware components available for use and a large base of user experience to draw upon.

The IEEE 802.4 protocol is not, however, completely problem free. The biggest difficulty associated with the protocol is its complexity. This complexity is derived from all the bus control functions that must be implemented by all the stations that make up a network. These functions include initiating a token-passing sequence, recovering from fault conditions caused by lost or multiple tokens, and allowing stations to enter or exit the token-passing sequence (Myers, 1982:36).

Thesis Summary

This thesis developed and validated a model for simulating bus token-passing protocols for avionics applications. The purpose of the simulation model was to explore the effect of the different bus and protocol parameters on the performance of the bus. Two separate protocols were modeled, that of a protocol with distributed control and a protocol with centralized control. The model was developed as part of an overall simulation program which included simulation control, data collection, and data analysis functions. The simulation model program was written in the Pascal computer programming language. A series of tests were conducted using the simulation model program to validate its operation and modeling capabilities. The validation tests were successful. Initial performance tests under varying bus and protocol design parameter conditions were conducted for the centralized token-passing protocol using the simulation model program. Finally, the performance of the centralized control and distributed control protocols were compared.

The simulation model program developed by this thesis provides an analysis capability for the centralized control protocol where none existed before. Due to the large number of bus and protocol parameters that can be varied, the simulation model program provides a capability for detailed analysis of the performance of a protocol in a variety of network configurations and environments. Also, since the simulation model program is modular in design and construction, additional protocol characteristics can be added if a more detailed modeling capability is needed or as more characteristics become known as the definitions for the next-generation avionics network protocol are refined.

Recommendations

Four recommendations concerning the simulation model program are presented and discussed in the following paragraphs.

First Recommendation. The testing done in Chapter IV using the centralized token-passing protocol algorithm was not meant to be a comprehensive test of the performance of this protocol. It was meant to demonstrate the capabilities of the simulation model program and explore basic performance questions concerning the centralized token-passing protocol. Thus, the first recommendation would be to continue the performance testing of the centralized token-passing protocol. An example of additional tests that could be accomplished would be testing the protocol with a standard network message scenario being developed under the Avionics Laboratory contract (Klass, 1985:169). The scenario is meant to realistically define the number of stations and their message loads for an advanced fighter avionics network.

Second Recommendation. The second recommendation is to use the simulation model program to explore the performance of other token-passing protocols. This would include the protocols being considered for use in the next-generation avionics local area network and the IEEE 802.4 protocol in an avionics configuration. The IEEE protocol is included in this recommendation because of the large-scale interest, previous studies, and current work involving this standardized protocol.

Third Recommendation. The third recommendation involves an improvement to the simulation model program. This recommendation would involve changing the bus and station data input method to a more interactive type of user interface. This would involve changing the building of the command file, which executes the program, from an off-line to an interactive "question and answer" type interface. This would relieve the user from many of the details concerning the formatting of the bus configuration and station data.

Fourth Recommendation. The fourth recommendation concerns an addition to the protocol algorithms. It is recommended that the capability for message and/or station priorities be added to the protocol algorithms. The capability for message and station priorities is important because it allows time-critical messages to be transmitted with low delays without undue performance degradation for other messages or stations. The priority option is especially important in real-time environments such as avionics networks. For example, a message priority capability could be obtained in the simulation model program by adding additional message queues (linked lists) in the record representation of a station. Each additional message queue would represent lower priority messages. Also, an additional field would have to be added to the record representation of a message to indicate the message's priority.

Summary

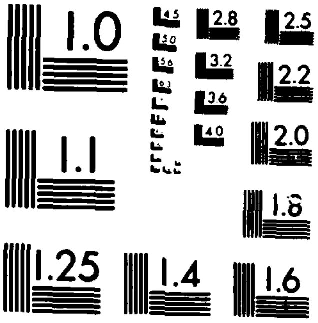
The validation and initial performance tests conducted using the simulation model program were summarized and discussed in relation to the avionics network environment. The thesis was summarized and recommendations made for the improvement of the simulation program and additional study efforts. The validated simulatio.ı model program developed by this thesis can be a tool for further token-passing protocol performance testing and evaluation.

AD-A163 950 SIMULATION MODEL OF A HIGH-SPEED TOKEN-PASSING BUS FOR AVIONICS APPLICATIONS(U) AIR FORCE INST OF TECH WRIGHT-PATTERSON AFB OH SCHOOL OF ENGI.. J E SPIETH 2/2

UNCLASSIFIED DEC 85 AFIT/GCS/ENG/85D-15 F/G 17/2 NL

END

FILMED



MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS 1963-A

Appendix A. User's Guide

This appendix is the simulation model program user's guide. How the program is executed is discussed first. This is followed by an explanation of the format for the input data and definitions of the input data variables.

Program Execution

The simulation model program is executed in a batch mode. It is submitted for execution by using the "submit" command. The format of the submit command is:

```
submit/log=[]      file name
```

There are various other options of the submit command and the user should consult the various Digital Equipment Corporation reference manuals, such as the VAX/VMS Command Language User's Guide and the Programming in VAX-11 Pascal Manual for more detailed information. The "log=[]" qualifier names the program's output file the same name as "file name" but with a ".log" extension and places it in the current default directory. The file named by "file name" is a command file that contains the program execution statement and the input data needed by the simulation model program. The file named by "file name" should have a ".com" extension. This command file is created using a text editor such as Digital Equipment Corporation's EDT.

Command File

The command file has three parts consisting of the program execution statement, the bus data input lines, and the station data input lines. These three parts will be described in the following paragraphs.

Program Execution Statement. The first line in the command file is always the program execution statement. This statement is shown below.

```
$ run   disk$user:[spieth.bus]bussim
```

This statement instructs the VAX/VMS operating system to run the file "bussim" in the directory "spieth.bus" on the "user" disk. The file "bussim" contains the main Pascal module of the simulation model program. The initial dollar sign must be included in the statement.

Bus Data Input Lines. The data for the bus configuration that is to be simulated is passed to the simulation model program by including the data in the command file. The data follows the program execution statement in the command file. The format of the bus data input lines are shown below. The description includes the name of the variable, the actual name used in the program, what type the variable is, and an explanation of the meaning of the variable. The line numbers listed are for reference only and are not ṗart of the command file. The lower case letters after the line number refer to the order of the variables on the line. On line 3, for example, the variable description labeled 3a would be first on the line followed by 3b and 3c, etc. Blank spaces between the variables on the same line are ignored by the program.

For real type variables, a digit must be listed both before and after the decimal point (0.5 instead of .5). When the explanation for a variable states that it is not currently implemented in the program, or it is not used based upon a certain condition, a value still has to be included in the data for this variable in order to avoid execution errors.

Line Number Variable Name Type

2a. random number generator seed seed integer

This is the seed for the Digital Equipment Corporation random number generator which is used in generating arrival times for Poisson arrivals and message lengths for exponential distributions.

3a. number of stations num_stations integer

This is the total number of stations on the bus.

3b. first station first_station integer

This is the address of the station that the simulation program should begin the token-passing cycles with. It is currently not implemented in the program. The program begins the token-passing cycle with the station whose data is listed first in the station data part of the input data.

3c. bit rate bit_rate real

The rate at which the bits are transmitted in bits/second.

4a. propagation factor prop_factor real

A value less than one that is multiplied times the speed of light to give a bus signal propagation speed. Typically this factor is 0.6667.

4b. length of the bus bus_length real

The length of the bus in meters.

4c. station delay time stat_delay real

The amount of time (in seconds) it takes a station to start transmitting either a data or token message once it has received the token.

5a. type of bus control bus_control enumerated

The type of control, either distributed (distrib) or centralized (central), used in the token-passing protocol. Enumerated type variables can not be read in or printed out. Therefore, for input, a 0 is meant to be distributed type control and a 1 to be centralized type control.

Line Number Variable Name Type

5b. direction of token passing token_pass enumerated

Defines in what order or direction the token is passed based upon the stations' addresses. There are three types; ascending (ascen), descending (descen), and fixed (fixed). For input; 0 equals ascen, 1 equals descen, and 2 equals fixed. This variable is not implemented in the program at this time.

5c. type of token-holding limit token_hold_type enumerated

Defines if the token-holding limit is a time value (time) or a limit in terms of number of messages (num) that can be transmitted.

5d. token holding limit token_hold_limit real

The length of time a station may transmit messages (in seconds) or the number of messages a station can transmit during one token-holding turn.

6a. same/different arrival type station_arr enumerated

Defines if all the stations have the same type of arrival condition. There are two choices, same (same) or different (diff). For input; 0 equals same, and 1 equals different.

6b. arrival type station_arr_type enumerated

If the stations all have the same type of arrival condition, this variable defines which type it is. There are three choices, arrivals according to: a deterministic distribution (constant_arr), a Poisson distribution (Poisson), or a continuous case (contin). For input; 0 equals deterministic, 1 equals Poisson, and 2 equals continuous. This variable is not used when the stations have different types of arrival conditions; that is, when the station_arr variable equals diff.

6c. same/different arrival rate station_rate enumerated

Defines if all the stations have the same arrival rate. There are two choices: same (same) or different (diff). For input, 0 equals same, and 1 equals different. This variable is not used when the stations have different types of arrival conditions; that is, when the station_arr variable equals diff.

6d. arrival rate station_arr_rate real

This variable represents the arrival rate of messages to a station (messages/second). This variable is only used if the stations all have the same type of arrival condition with the same rate; that is, both the station_arr and station_rate variables must be equal to same.

7a. same/different length distribution station_len enumerated

Defines if all the stations have the same type of message length distribution. There are two choices, same (same) or different (diff). For input; 0 equals same and 1 equals different.

7b. length distribution type station_len_type enumerated

If the stations all have the same type of message length distribution, this variable defines which type it is. There are two choices, deterministic (constant_len) or exponential (exp). For input; 0 equals deterministic and 1 equals exponential. This variable is not used when the stations have different types of message length distributions; that is, when the station_len variable equals different (diff).

7c. same/different length mean station_mean enumerated

Defines if all the stations have the same mean message length. There are two choices: same (same) or different (diff). For input, 0 equals same and 1 equals different. This variable is not used when the stations have different types of message length distributions; that is, when the station_len variable equals different (diff).

7d. length mean station_len_mean real

The mean message length in data words. This variable is only used if the stations all have the same type of message length distribution with the same mean message length; that is, both the station_len and station_mean variables must be equal to same. Note: when the message length distribution is of the deterministic type, the mean represents the size of the message.

8a. number of bits in token token_bits real

The number of bits in the token message.

8b. number of overhead bits in a message overhead_bits real

The number of header and trailer bits (overhead) in a message. Includes everything but the data bits.

8c. number of bits in one data word bits_per_data_word real

The number of bits in one data word.

9a. minimum number of data words min_data_words real

Defines the minimum number of data words allowed to be sent in a message. Usually would be zero or one.

9b. maximum number of data words max_data_words real

Defines the maximum number of data words allowed to be sent in a message.

10. simulation stop time sim_stop_time real

The value of the simulation clock when the simulation should be stopped. The simulation clock starts at value 0.0.

11. calculate token passing propagation delay times flag
calc_pass_prop_time boolean

Determines if the token-passing propagation time delays should be calculated or read in.

Station Data Input Lines. Depending on a station's type of arrival condition and type of message length distribution, from one to three lines of data may need to be read in to completely describe a station. Lines XX, YY, and ZZ define the data that is on these lines. At the minimum, there would be one line of data for each station. At the maximum, three lines of data could be used for each station. The first station listed is the station the simulation model program will begin the token-passing cycle with. The rest of the stations are then listed in the order they are passed the token.

Line Number Variable Name Type

XXa. station address (pointer)^.attrib.address integer

The station's address.

XXb. passing address (pointer)^.attrib.pass_address integer

The address of the station that the current station passes the token to.

XXc. distance (pointer)^.attrib.distance real

The distance from the left starting point of the bus to the current station in meters. Only used if the calculate token-passing propagation delay times flag is true (1). A value needs to be listed in either case.

Line Number Variable Name Type

XXd. token-passing propagation delay time
(pointer) ^.attrib.pass_prop_time real

The time (in seconds) it takes the token to propagate from the current station to the next station. Only used if the calculate token-passing propagation delay times flag is false (0). Values only need to be listed when the flag is false.

YYa. station arrival rate (pointer)^.attrib.mess_arr_rate real

The particular station's message arrival rate. This line of data is needed only if the stations all have the same type of message arrival condition but different arrival rates. That is, the station_arr variable would be equal to same (same) and the station_rate variable would be equal to different (diff).

OR

YYa. station arrival type (pointer)^.attrib.mess_arr_type enumerated

YYb. station arrival rate (pointer) ^.attrib.mess_arr_rate real

The particular station's type of message arrival condition and the arrival rate. This line of data is needed only if the stations all have different types of message arrival conditions. That is, the station_arr variable would be equal to different (diff).

ZZa. station message length mean
(pointer)^.attrib.mess_len_mean real

The particular station's mean message length. This line of data is needed only if the stations all have the same type of message length distribution but different means. That is, the station_len variable is equal to same (same) and the station_mean variable is equal to different (diff).

OR

ZZa. station message length distribution
(pointer)^.attrib.mess_len_type enumerated

ZZb. station message length mean
(pointer)^.attrib.mess_len_mean real

The particular station's type of message length distribution and its mean. This line of data is needed only if the stations all have different types of message length distributions. That is, the station_len variable is equal to different (diff).

Sample Output.

```
! Second Test Case  Mean Message Length = 32 Data Words
$ run  disk$user:[spieth.bus]bussim
select bus configuration module
868
30              1       50.0e6
0.666666666     60.0    0.5e-6
1               2       0       83.32e-6
0               1       1       400.0
0               1       0       32.0
22.0          70.0    16.0
0.0          256.0
0.6
station data input module
    1
1       2       2.0
40.0
2       3       2.5
200.0
3       4       3.0
2000.0
4       5       3.5
2000.0
5       6       4.0
200.0
6       7       4.5
40.0
7       8       5.0
40.0
8       9       6.0
200.0
9       10      7.0
2000.0
10      11      8.0
2000.0
11      12      9.0
200.0
12      13      10.0
40.0
13      14      13.0
40.0
14      15      14.0
200.0
15      16      15.0
2000.0
16      17      16.0
2000.0
17      18      17.0
200.0
18      19      18.0
40.0
```

```
19      20      28.0
40.0
20      21      29.0
200.0
21      22      30.0
2000.0
22      23      31.0
2000.0
23      24      32.0
200.0
24      25      33.0
40.0
25      26      55.0
40.0
26      27      56.0
200.0
27      28      57.0
2000.0
28      29      58.0
2000.0
29      30      59.0
200.0
30      1       60.0
40.0
calc first arr and len module
init stats module
         1     2.50000E-09
         2     2.50000E-09
         3     2.50000E-09
         4     2.50000E-09
         5     2.50000E-09
         6     2.50000E-09
         7     5.00000E-09
         8     5.00000E-09
         9     5.00000E-09
        10     5.00000E-09
        11     5.00000E-09
        12     1.50000E-08
        13     5.00000E-09
        14     5.00000E-09
        15     5.00000E-09
        16     5.00000E-09
        17     5.00000E-09
        18     5.00000E-08
        19     5.00000E-09
        20     5.00000E-09
        21     5.00000E-09
        22     5.00000E-09
        23     5.00000E-09
        24     1.10000E-07
        25     5.00000E-09
        26     5.00000E-09
        27      5.00000E-09
        28      5.00000E-09
        29      5.00000E-09
        30      2.90000E-07
simulation control module entered
centralized algorithm procedure called
```

Station Address	Number of access	Average access delay (seconds)	Minimum access delay (seconds)	Maximum access delay (seconds)
1	16009.0	3.74811E-05	2.72989E-05	1.93834E-04
2	16009.0	3.74811E-05	2.72989E-05	1.93834E-04
3	16009.0	3.74811E-05	2.72989E-05	1.93834E-04
4	16008.0	3.74817E-05	2.72989E-05	1.93834E-04
5	16008.0	3.74818E-05	2.72989E-05	1.98632E-04
6	16008.0	3.74818E-05	2.72989E-05	1.98632E-04
7	16008.0	3.74818E-05	2.72989E-05	1.98632E-04
8	16008.0	3.74818E-05	2.72989E-05	1.98632E-04
9	16008.0	3.74818E-05	2.72989E-05	1.98632E-04
10	16008.0	3.74817E-05	2.72989E-05	1.79127E-04
11	16008.0	3.74817E-05	2.72989E-05	1.77890E-04
12	16008.0	3.74817E-05	2.72989E-05	1.77890E-04
13	16008.0	3.74817E-05	2.72989E-05	1.77890E-04
14	16008.0	3.74817E-05	2.72989E-05	1.77890E-04
15	16008.0	3.74817E-05	2.72989E-05	1.77890E-04
16	16008.0	3.74812E-05	2.72989E-05	1.82033E-04
17	16008.0	3.74812E-05	2.72989E-05	1.87766E-04
18	16008.0	3.74812E-05	2.72989E-05	1.87766E-04
19	16008.0	3.74812E-05	2.72989E-05	1.87766E-04
20	16008.0	3.74812E-05	2.72989E-05	1.87766E-04
21	16008.0	3.74812E-05	2.72989E-05	1.91659E-04
22	16008.0	3.74812E-05	2.72989E-05	1.82986E-04
23	16008.0	3.74812E-05	2.72989E-05	1.78799E-04
24	16008.0	3.74812E-05	2.72989E-05	1.78799E-04
25	16008.0	3.74812E-05	2.72989E-05	1.78799E-04
26	16008.0	3.74812E-05	2.72989E-05	1.78799E-04
27	16008.0	3.74812E-05	2.72989E-05	1.78799E-04
28	16008.0	3.74812E-05	2.72989E-05	1.93834E-04
29	16008.0	3.74812E-05	2.72989E-05	1.93834E-04
30	16008.0	3.74812E-05	2.72989E-05	1.93834E-04

Station Address	Number of data mess. sent	Average message length	Minimum message length	Maximum message length
		(in bits and including	overhead)	
1	23.0	583.39	70.00	2422.00
2	129.0	537.10	70.00	3366.00
3	1145.0	607.89	70.00	3622.00
4	1161.0	580.83	70.00	3766.00
5	117.0	559.16	70.00	2646.00
6	21.0	596.48	86.00	1350.00
7	31.0	520.06	70.00	1782.00
8	117.0	590.48	70.00	1894.00
9	1098.0	580.35	70.00	4134.00
10	1120.0	568.33	70.00	3302.00
11	119.0	657.03	70.00	2262.00
12	20.0	624.40	86.00	1990.00
13	28.0	631.71	86.00	1718.00
14	128.0	624.25	70.00	2294.00
15	1155.0	565.63	70.00	3382.00
16	1164.0	569.78	70.00	3238.00
17	129.0	617.97	70.00	3382.00
18	18.0	614.00	150.00	1798.00
19	25.0	498.80	86.00	1926.00
20	116.0	666.14	86.00	2694.00
21	1128.0	559.97	70.00	4166.00
22	1098.0	577.32	70.00	3174.00
23	129.0	596.39	70.00	3798.00
24	28.0	490.00	86.00	1382.00
25	27.0	592.07	70.00	2374.00
26	131.0	548.17	86.00	2102.00
27	1072.0	579.55	70.00	3222.00
28	1051.0	551.61	70.00	3990.00
29	125.0	534.51	70.00	2230.00
30	22.0	479.45	102.00	1462.00

Total number of accesses	Total average access delay (seconds)	Total number of messages	Total average message length (bits)
480243.0	3.74814E-05	12675.0	575.75

```
            Average      Minimum      Maximum
Station     message      message      message
Address      delay        delay        delay
           (seconds)    (seconds)    (seconds)

   1       3.861E-05    4.381E-06    1.300E-04
   2       3.539E-05    4.739E-06    1.363E-04
   3       3.250E-05    2.176E-06    1.520E-04
   4       3.276E-05    1.580E-06    1.665E-04
   5       3.020E-05    2.384E-06    1.123E-04
   6       3.161E-05    1.189E-05    7.206E-05
   7       3.589E-05    7.298E-06    7.908E-05
   8       3.349E-05    3.368E-06    9.684E-05
   9       3.271E-05    2.369E-06    1.357E-04
  10       3.256E-05    2.086E-06    1.381E-04
  11       3.390E-05    3.397E-06    9.298E-05
  12       3.622E-05    1.433E-05    7.147E-05
  13       3.171E-05    3.383E-06    7.702E-05
  14       3.167E-05    2.831E-06    8.844E-05
  15       3.203E-05    2.213E-06    1.444E-04
  16       3.261E-05    2.325E-06    1.379E-04
  17       3.411E-05    2.369E-06    1.098E-04
  18       3.377E-05    8.017E-06    7.111E-05
  19       3.586E-05    5.729E-06    9.835E-05
  20       3.669E-05    4.709E-06    1.895E-04
  21       3.196E-05    2.116E-06    1.073E-04
  22       3.253E-05    2.444E-06    1.782E-04
  23       3.439E-05    3.636E-06    1.066E-04
  24       2.603E-05    5.305E-06    4.649E-05
  25       3.159E-05    5.648E-06    9.596E-05
  26       2.870E-05    3.368E-06    7.364E-05
  27       3.290E-05    2.176E-06    1.148E-04
  28       3.175E-05    2.265E-06    1.274E-04
  29       3.083E-05    3.487E-06    9.008E-05
  30       3.705E-05    1.283E-05    8.708E-05

Total average message delay was:  3.25021E-05   seconds

The normalized delay is:          3.21323   (without overhead)

The normalized delay is:          2.82257   (with overhead)
```

```
Total number of complete
      token-passing cycles was      16008.00

The mean token-passing cycle time is   3.74811E-05   seconds

The mean throughput was    7.199736  megabits/second

With a bit rate of   50.0000  megabits/second
   the ratio of throughput to the bit rate is:    0.143995

The mean efficiency was    0.211945


  SPIETH       job terminated at 10-OCT-1985 17:53:11.35

  Accounting information:
  Buffered I/O count:            53     Peak working set size:   367
  Direct I/O count:              70     Peak page file size:     469
  Page faults:                  482     Mounted volumes:           0
  Charged CPU time:     0 00:00:58.62   Elapsed time:     0 00:01:01.78
```

Appendix B. Simulation Model Program Software

This appendix contains the Pascal software for the simulation model program. The modules that make up the software are grouped according to function and placed into seven files for ease of editing and configuration control. Six of the files are then "included" in the seventh file, using the VAX/VMS operating system %include directive, to make the complete program. The %include directives are in the declaration section of the main Pascal module in the file bussim.pas. The modules and what file they are contained in are listed below. Except for the bussim.pas file which is listed first, the other files are listed in their "included" order. The modules within a file are listed in their compilation order. The software follows this listing.

File Name	Module Name	Module Number
- bussim.pas		
	main	0.0
- declar.pas		
	constant, type, and variable declarations	
- queue.pas		
	pop	2.1.5.1
	out_front_queue	2.1.5
	in_rear_queue	2.1.4.1
- stats.pas		
	min	2.1.1.1
	max	2.1.1.2
	sum_delay	3.2
	sum_thruput	3.1
	statistics	3.0

File Name	Module Name	Module Number
- update.pas		
	update_access_stats	2.1.1
	update_message_stats	2.1.2
	update_delay_stats	2.1.3
	update_thruput_stats	2.1.6
- setup.pas		
	mth$random	1.3.1.1
	bus_data_input	1.1
	station_data_input	1.2
	calc_arr_and_len	1.3.1
	calc_first_arr_and_len	1.3
	init_stats	1.4
	calc_token_prop_delays	1.5
	sel_bus_setup	1.0
- simulate.pas		
	calc_next_arr_and_len	2.1.4
	dist_algor	2.1
	cent_algor	2.2
	simulate	2.0

```
(****************************************************************
*                                                               *
*    DATE: 23 Sep 1985                                          *
*    VERSION: 1.3                                               *
*                                                               *
*    TITLE: bus simulation                                      *
*    FILENAME: bussim.pas                                       *
*    COORDINATOR: Jim Spieth                                    *
*    PROJECT: Avionics Bus Simulation Model                     *
*    OPERATING SYSTEM: VAX/VMS, Version 4.2 on VAX-11/782       *
*    LANGUAGE: Pascal                                           *
*    USE: main pascal program                                   *
*    CONTENTS: bussim                                           *
*    FUNCTION: simulate token-passing bus                       *
*                                                               *
****************************************************************)
```

```
(****************************************************************
*                                                               *
*    DATE: 23 Sep 1985                                          *
*    VERSION: 1.3                                               *
*                                                               *
*    NAME: bussim  (main)                                       *
*    MODULE NUMBER: 0.0                                         *
*    DESCRIPTION: main, executive module for bussim program     *
*    PASSED VARIABLES: none                                     *
*    RETURNS: none                                              *
*    GLOBAL VARIABLES USED: none                                *
*    GLOBAL VARIABLES CHANGED: sim_clock   num_cycle            *
*              total_cyc_time    total_thruput   total_eff      *
*    FILES READ: none                                           *
*    FILES WRITTEN: none                                        *
*    MODULES CALLED: sel_bus_setup                              *
*                    simulate                                   *
*                    statistics                                 *
*    CALLING MODULES: none                                      *
*                                                               *
*    AUTHOR: Jim Spieth                                         *
*    HISTORY: 1.3  23 Sep 85  added total_cyc_time              *
*             1.2  18 Sep 85  added update.pas include file     *
*             1.1  24 Aug 85  added global thruput variables    *
*             1.0  19 Aug 85  original                          *
****************************************************************)
program bussim(input, output) ;

   %include  'declar.pas'
   %include  'queue.pas'
   %include  'stats.pas'
   %include  'update.pas'
   %include  'setup.pas'
   %include  'simulate.pas'

begin
   sim_clock := 0.0 ;
   num_cycle := 0.0 ;
   total_thruput := 0.0 ;
   total_eff := 0.0 ;
   total_cyc_time := 0.0 ;

   sel_bus_setup ;
   simulate ;
   statistics ;
end.
```

```
(****************************************************************
*                                                               *
*    DATE: 24 Sep 1985                                          *
*    VERSION: 1.6                                               *
*                                                               *
*    TITLE: declarations for program bussim                     *
*    FILENAME: declar.pas                                       *
*    COORDINATOR: Jim Spieth                                    *
*    PROJECT: Avionics Bus Simulation Model                     *
*    OPERATING SYSTEM: VAX/VMS, Version 4.2 on VAX-11/782       *
*    LANGUAGE: Pascal                                           *
*    USE: %include file in program bussim                       *
*    CONTENTS: constant, type and var                           *
*    FUNCTION: contain all declarations for program bussim      *
*                                                               *
****************************************************************)
const   speed_light = 3.0e8 ;
type
        choice = (same, diff) ;
        arrival = (constant_arr, poisson, contin) ;
        length_distrib = (constant_len, exp) ;
        control = (distrib, central) ;
        token_pass_type = (ascen, descen, fixed) ;
        token_hold_limit_type = (time, num) ;

   stats_type = record
                   address : integer ;
                   num_access : real ;
                   sum_access : real ;
                   min_access : real ;
                   max_access : real ;
                   num_messages : real ;
                   sum_mess_len : real ;
                   min_mess_len : real ;
                   max_mess_len : real ;
                   sum_mess_delay : real ;
                   min_mess_delay : real ;
                   max_mess_delay : real ;
               end ;    (* of record *)
   stats_ptr = ^stats ;
   stats = record
              data : stats_type ;
              next : stats_ptr ;
           end ;   (* of record *)

   message_type = record
                     source_add : integer ;
                     length : real ;
                     arr_time : real ;
                  end ;  (* of record *)
   message_ptr = ^message ;
   message = record
                info : message_type ;
                next_message : message_ptr
             end ;  (* of record *)

   station_ptr = ^station ;
   station_type = record
                     address : integer ;
                     pass_address : integer ;
                     mess_arr_type : arrival ;
                     mess_arr_rate : real ;
                     distance : real ;
                     mess_len_type : length_distrib ;
                     mess_len_mean : real ;
                     pass_prop_time : real ;
                     last_access : real ;
                     front_mess_queue : message_ptr ;
                     rear_mess_queue : message_ptr ;
                  end ;   (* of record *)

   station = record
                attrib : station_type ;
                next_station : station_ptr
             end ;   (* of record *)

var
   front_station,  current_station     : station_ptr ;

   front_stats,   current_stats        : stats_ptr ;

   bit_rate,           prop_factor,
   sig_prop,           sig_delay,
   bus_length,         stat_delay,
   overhead_bits,      token_bits,
   bits_per_data_word, token_hold_limit,
   min_data_words,     max_data_words,
   sim_clock,          sim_stop_time,
   total_thruput,      total_eff,
   total_cyc_time,     num_cycle,
   station_arr_rate,   station_len_mean     : real ;

   first_station, num_stations, seed        : integer ;

   station_arr,  station_rate,
   station_len,  station_mean     : choice ;

   station_arr_type     : arrival ;
   station_len_type     : length_distrib ;
   bus_control          : control ;
   token_pass           : token_pass_type ;
   token_hold_type      : token_hold_limit_type ;
   calc_pass_prop_time  : boolean ;
```

```
(****************************************************************
*                                                               *
*    DATE: 18 Aug 1985                                          *
*    VERSION: 1.0                                               *
*                                                               *
*    TITLE: queue (linked list) related procedures              *
*    FILENAME:  queue.pas                                       *
*    COORDINATOR: Jim Spieth                                    *
*    PROJECT: Avionics Bus Simulation Model                     *
*    OPERATING SYSTEM: VAX/VMS, Version 4 on VAX-11/782         *
*    LANGUAGE: Pascal                                           *
*    USE: %include file for bussim program                      *
*    CONTENTS: pop                                              *
*               out_front_queue                                 *
*               in_rear_queue                                   *
*    FUNCTION: procedures for queue (linked list) operations    *
*                                                               *
****************************************************************)


(****************************************************************
*                                                               *
*    DATE: 18 Aug 1985                                          *
*    VERSION: 1.0                                               *
*                                                               *
*    NAME: pop                                                  *
*    MODULE NUMBER: 2.1.5.1                                     *
*    DESCRIPTION: removes first message in queue (linked list)  *
*    PASSED VARIABLES: list - pointer to front of list          *
*    RETURNS: list                                              *
*    GLOBAL VARIABLES USED: none                                *
*    GLOBAL VARIABLES CHANGED: none                             *
*    FILES READ: none                                           *
*    FILES WRITTEN: none                                        *
*    MODULES CALLED: none                                       *
*    CALLING MODULES: out_front_queue                           *
*                                                               *
*    AUTHOR: Jim Spieth                                         *
*    HISTORY: Adapted from Dale and Orshalick, 1983:443         *
*                                                               *
****************************************************************)
procedure pop(var list : message_ptr ) ;

var ptr : message_ptr ;

begin
   ptr := list ;
   list := list^.next_message ;
   dispose(ptr)
end ;
```

```
(****************************************************************
*                                                               *
*    DATE: 18 Aug 1985                                          *
*    VERSION: 1.0                                               *
*                                                               *
*    NAME: out_front_queue                                      *
*    MODULE NUMBER: 2.1.5                                       *
*    DESCRIPTION: removes message from front of queue (linked   *
*                 list) and checks for empty queue              *
*    PASSED VARIABLES: front - pointer to front of list         *
*                      rear - pointer to rear of list           *
*    RETURNS: front, rear                                       *
*    GLOBAL VARIABLES USED: none                                *
*    GLOBAL VARIABLES CHANGED: none                             *
*    FILES READ: none                                           *
*    FILES WRITTEN: none                                        *
*    MODULES CALLED: pop                                        *
*    CALLING MODULES: dist_algor                                *
*                                                               *
*    AUTHOR: Jim Spieth                                         *
*    HISTORY: Adapted from Dale and Orshalick, 1983:454         *
*                                                               *
****************************************************************)
procedure out_front_queue(var front, rear : message_ptr ) ;

begin
   if (front = nil)
      then writeln('queue is empty')
      else
         begin
         pop( front ) ;
         if (front = nil)
            then rear := nil
         end
end ;
```

```
(*****************************************************************
*                                                                *
*    DATE: 18 Aug 1985                                           *
*    VERSION: 1.0                                                *
*                                                                *
*    NAME: in_rear_queue                                         *
*    MODULE NUMBER: 2.1.4.1                                      *
*    DESCRIPTION: inserts element at rear of queue               *
*    PASSED VARIABLES: front, rear, element                      *
*    RETURNS: front, rear                                        *
*    GLOBAL VARIABLES USED: none                                 *
*    GLOBAL VARIABLES CHANGED: none                              *
*    FILES READ: none                                            *
*    FILES WRITTEN: none                                         *
*    MODULES CALLED: none                                        *
*    CALLING MODULES: calc_next_arr_and_len                      *
*                                                                *
*    AUTHOR: Jim Spieth                                          *
*    HISTORY: Adapted from Dale and Orshalick, 1983:454          *
*                                                                *
*****************************************************************)
procedure in_rear_queue( var front, rear : message_ptr ;
                        var one_mess : message_type ) ;

var
   ptr : message_ptr ;

begin
   new(ptr) ;
   ptr^.info := one_mess ;
   ptr^.next_message := nil ;
   if rear = nil
      then
         begin
         rear := ptr ;
         front := ptr ;
         end
      else
         begin
         rear^.next_message := ptr ;
         rear := ptr ;
         end
end ;
```

```
(****************************************************************
*                                                               *
*    DATE: 24 Sep 1985                                          *
*    VERSION: 1.7                                               *
*                                                               *
*    TITLE: statistics                                          *
*    FILENAME: stats.pas                                        *
*    COORDINATOR: Jim Spieth                                    *
*    PROJECT: Avionics Bus Simulation Model                     *
*    OPERATING SYSTEM: VAX/VMS, Version 4.2 on VAX-11/782       *
*    LANGUAGE: Pascal                                           *
*    USE: %include file for bussim program                      *
*    CONTENTS: min                                              *
*              max                                              *
*              sum_delay                                        *
*              sum_thruput                                      *
*              statistics                                       *
*    FUNCTION: perform statistics operations                    *
*                                                               *
****************************************************************)


(****************************************************************
*                                                               *
*    DATE: 18 Aug 1985                                          *
*    VERSION: 1.0                                               *
*                                                               *
*    NAME: min      (function)                                  *
*    MODULE NUMBER: 2.1.1.1                                     *
*    DESCRIPTION: picks minimum of two real numbers             *
*    PASSED VARIABLES: real1 and real2                          *
*    RETURNS: mimimum of the two                                *
*    GLOBAL VARIABLES USED: none                                *
*    GLOBAL VARIABLES CHANGED: none                             *
*    FILES READ: none                                           *
*    FILES WRITTEN: none                                        *
*    MODULES CALLED: none                                       *
*    CALLING MODULES: update_access_stats                       *
*                     update_message_stats                      *
*                     update_delay_stats                        *
*    AUTHOR: Jim Spieth                                         *
*    HISTORY:                                                   *
*                                                               *
****************************************************************)
function min (real1, real2 : real ) : real ;

begin
   if real1 > real2
      then min := real2
      else min := real1
end ;
```

```
(****************************************************************
*                                                               *
*    DATE: 18 Aug 1985                                          *
*    VERSION: 1.0                                               *
*                                                               *
*    NAME: max         (function)                               *
*    MODULE NUMBER: 2.1.1.2                                     *
*    DESCRIPTION: picks maximum of two reals                    *
*    PASSED VARIABLES: reall, real2                             *
*    RETURNS: maximum of the two                                *
*    GLOBAL VARIABLES USED: none                                *
*    GLOBAL VARIABLES CHANGED: none                             *
*    FILES READ: none                                           *
*    FILES WRITTEN: none                                        *
*    MODULES CALLED: none                                       *
*    CALLING MODULES: update_access_stats                       *
*                     update_message_stats                      *
*                     update_delay_stats                        *
*    AUTHOR: Jim Spieth                                         *
*    HISTORY:                                                   *
*                                                               *
****************************************************************)
function max (reall, real2 : real ) : real ;

begin
   if reall > real2
      then max := reall
      else max := real2
end ;
```

```
(*****************************************************************
*                                                                *
*    DATE: 18 Sep 1985                                           *
*    VERSION: 1.1                                                *
*                                                                *
*    NAME: sum_delay                                             *
*    MODULE NUMBER: 3.2                                          *
*    DESCRIPTION: calculates and prints message delay values     *
*    PASSED VARIABLES: none                                      *
*    RETURNS: nothing                                            *
*    GLOBAL VARIABLES USED: num_stations, front_stats            *
*                           front_station, bit_rate              *
*    GLOBAL VARIABLES CHANGED: current_stats  current_station    *
*    FILES READ: none                                            *
*    FILES WRITTEN: none                                         *
*    MODULES CALLED: none                                        *
*    CALLING MODULES: statistics                                 *
*                                                                *
*    AUTHOR: Jim Spieth                                          *
*    HISTORY:                                                    *
*        1.1  18 Sep 85 added 2nd form of normalized delay       *
*        1.0  30 Aug 85 original                                 *
*                                                                *
*****************************************************************)
procedure sum_delay(total_num_mess, total_ave_mess_len : real) ;

var
    h, one, yes : integer ;

    total_sum_delay, ave_delay,
    total_ave, full_delay_ratio, delay_ratio         : real ;

begin
one := 0 ;
total_sum_delay := 0.0 ;
current_stats := front_stats ;
current_station := front_station ;
case station_arr of
   same : case station_arr_type of
             constant_arr : case station_rate of
                               diff : yes := 1 ;
                               same : if station_arr_rate = 0.0
                                        then yes := 0
                                        else yes := 1
                            end ;  (* of case *)
             poisson : yes := 1 ;
             contin : yes := 0 ;
          end ; (* of case *)
   diff : yes := 1 ;
end ;  (* of case  *)
```

```
if yes = 1 then
begin
for h := 1 to num_stations do
 begin
 if current_station^.attrib.mess_arr_type <> contin
  then
   if ((current_station^.attrib.mess_arr_type <> constant_arr) or
         (current_station^.attrib.mess_arr_rate <> 0.0 ))
    then
     begin
     if one = 0 then
      begin
      page ;
      writeln('              Average       Minimum       Maximum') ;
      writeln('Station       message       message       message') ;
      writeln('Address        delay         delay         delay' ) ;
      writeln('             (seconds)     (seconds)     (seconds)');
      writeln ;
      end ;
     total_sum_delay := total_sum_delay +
                      current_stats^.data.sum_mess_delay ;
     if current_stats^.data.num_messages = 0.0
      then ave_delay := 0.0
      else ave_delay := current_stats^.data.sum_mess_delay /
                         current_stats^.data.num_messages ;
     writeln(current_stats^.data.address:4, '       ',
                  ave_delay:10, '    ',
                  current_stats^.data.min_mess_delay:10, '    ',
                  current_stats^.data.max_mess_delay:10 ) ;
     one := one + 1 ;
     current_station := current_station^.next_station ;
     current_stats := current_stats^.next ;
     end ;
   end ;   (*  for  *)
if one > 0 then
  begin
  total_ave := total_sum_delay / total_num_mess ;
  delay_ratio := total_ave / ((total_ave_mess_len - overhead_bits) /
                                                        bit_rate) ;
  full_delay_ratio := total_ave / (total_ave_mess_len / bit_rate) ;
  writeln ;
  writeln('Total average message delay was: ', total_ave, '   seconds');
  writeln ;
  writeln('The normalized delay is:  ', delay_ratio:15:5,
          '   (without overhead)') ;
  writeln ;
  writeln('The normalized delay is:  ', full_delay_ratio:15:5,
          '   (with overhead)') ;
  writeln ;
  end ;
end ;   (* if yes *)
end ;
```

```
(*****************************************************************
*                                                                *
*    DATE: 23 Sep 1985                                           *
*    VERSION: 1.1                                                *
*                                                                *
*    NAME: sum_thruput                                           *
*    MODULE NUMBER: 3.1                                          *
*    DESCRIPTION: calculates and prints throughput values        *
*    PASSED VARIABLES: none                                      *
*    RETURNS: nothing                                            *
*    GLOBAL VARIABLES USED: num_cycle, total_thruput             *
*                           total_eff, total_cyc_time            *
*    GLOBAL VARIABLES CHANGED: none                              *
*    FILES READ: none                                            *
*    FILES WRITTEN: none                                         *
*    MODULES CALLED: none                                        *
*    CALLING MODULES: statistics                                 *
*                                                                *
*    AUTHOR: Jim Spieth                                          *
*    HISTORY: 1.1 23 Sep 85 added total_cyc_time & ave_cyc_time *
*             1.0 23 Aug 85 original                             *
*                                                                *
*****************************************************************)
procedure sum_thruput ;

var
     ave_thruput, ave_eff, ave_cyc_time,
     mod_bit_rate, thruput_ratio           : real ;

begin
   mod_bit_rate := bit_rate / 1.0e6 ;
   ave_thruput := (total_thruput / num_cycle) / 1.0e6 ;
   ave_eff  := total_eff / num_cycle ;
   ave_cyc_time := total_cyc_time / num_cycle ;
   thruput_ratio := ave_thruput / mod_bit_rate ;
   page ;
   writeln('Total number of complete ') ;
   writeln('      token-passing cycles was', num_cycle:14:2) ;
   writeln ;
   writeln('The mean token-passing cycle time is  ', ave_cyc_time,
           '   seconds') ;
   writeln ;
   writeln('The mean throughput was', ave_thruput:12:6,
           '  megabits/second') ;
   writeln ;
   writeln('With a bit rate of', mod_bit_rate:10:4,'  megabits/second');
   write  ('   the ratio of throughput to the bit rate') ;
   writeln(' is:', thruput_ratio:12:6) ;
   writeln ;
   writeln('The mean efficiency was', ave_eff:12:6) ;
   writeln ;
end ;
```

```
(****************************************************************
*                                                               *
*    DATE: 27 Sep 1985                                          *
*    VERSION: 1.3                                               *
*                                                               *
*    NAME: statistics                                           *
*    MODULE NUMBER: 3.0                                         *
*    DESCRIPTION: calculates and prints station and summary     *
*                 statistics                                    *
*    PASSED VARIABLES: none                                     *
*    RETURNS: nothing                                           *
*    GLOBAL VARIABLES USED: front_stats                         *
*    GLOBAL VARIABLES CHANGED: current_stats                    *
*    FILES READ: none                                           *
*    FILES WRITTEN: none                                        *
*    MODULES CALLED: sum_thruput      sum_delay                 *
*    CALLING MODULES: bussim (main)                             *
*                                                               *
*    AUTHOR: Jim Spieth                                         *
*    HISTORY:                                                   *
*        1.3  27 Sep 85  improved headings                      *
*        1.2  30 Aug 85  added sum_delay call                   *
*        1.1  23 Aug 85  added sum_thruput call                 *
*        1.0  19 Aug 85  original                               *
*                                                               *
****************************************************************)
procedure statistics ;

var
  i       : integer ;
  total_num_access, total_access,
  ave_access,
  total_num_mess, total_mess_len,
  ave_mess_len,
  total_ave_access, total_ave_mess_len     : real ;

begin
  total_num_access := 0.0 ;
  total_access := 0.0 ;
  ave_access := 0.0 ;
  total_num_mess := 0.0 ;
  total_mess_len := 0.0 ;
  ave_mess_len := 0.0 ;
  total_ave_access := 0.0 ;
  total_ave_mess_len :=  0.0 ;
  page ;
writeln('            Number      Average       Minimum       Maximum');
writeln('Station       of        access        access        access');
writeln('Address     access       delay         delay         delay');
writeln('                      (seconds)     (seconds)     (seconds)');
  writeln ;
  current_stats := front_stats ;
```

```
for i := 1 to num_stations do
   begin
   total_num_access := total_num_access +
                       current_stats^.data.num_access ;
   total_access := total_access + current_stats^.data.sum_access ;
   ave_access := current_stats^.data.sum_access /
                   (current_stats^.data.num_access - 1.0 ) ;
   writeln(current_stats^.data.address:4,
           current_stats^.data.num_access:13:1,'  ', ave_access,
              '  ',current_stats^.data.min_access,
              '  ', current_stats^.data.max_access ) ;
   current_stats := current_stats^.next ;
   end ;
page ;
writeln('            Number ') ;
writeln('              of        Average       Minimum       Maximum ');
writeln('Station      data       message       message       message ');
writeln('Address     mess.       length        length        length  ');
writeln('             sent        ( in bits and including overhead ) ');
writeln ;
current_stats := front_stats ;
for i := 1 to num_stations do
   begin
   total_num_mess := total_num_mess + current_stats^.data.num_messages ;
   total_mess_len := total_mess_len + current_stats^.data.sum_mess_len ;
   if current_stats^.data.num_messages = 0.0
      then ave_mess_len := 0.0
      else ave_mess_len := current_stats^.data.sum_mess_len /
                           current_stats^.data.num_messages ;
   writeln(current_stats^.data.address:4,
           current_stats^.data.num_messages:13:1, ave_mess_len:14:2,
           current_stats^.data.min_mess_len:14:2,
           current_stats^.data.max_mess_len:14:2 ) ;
   current_stats := current_stats^.next ;
   end ;
total_ave_access := total_access / (total_num_access - num_stations) ;
total_ave_mess_len := total_mess_len / total_num_mess ;
writeln ;
writeln ;
writeln ;
writeln('   Total         Total         Total          Total  ') ;
writeln('   number        average       number        average ') ;
writeln('     of          access          of          message ') ;
writeln('  accesses        delay        messages       length  ') ;
writeln('                (seconds)                      (bits) ') ;
writeln ;
writeln(total_num_access:10:1, '    ', total_ave_access,
         total_num_mess:12:1,   total_ave_mess_len:14:2 ) ;

sum_delay(total_num_mess, total_ave_mess_len) ;
sum_thruput ;
end ;    (**** statistics ****)
```

```
(*****************************************************************
*                                                                *
*    DATE: 23 Sep 1985                                           *
*    VERSION: 1.1                                                *
*                                                                *
*    TITLE: update statistics                                    *
*    FILENAME: update.pas                                        *
*    COORDINATOR: Jim Spieth                                     *
*    PROJECT: Avionics Bus Simulation Model                      *
*    OPERATING SYSTEM: VAX/VMS, Version 4.2 on VAX-11/782        *
*    LANGUAGE: Pascal                                            *
*    USE: %include file for bussim program                       *
*    CONTENTS: update_access_stats                               *
*              update_message_stats                              *
*              update_delay_stats                                *
*              update_thruput_stats                              *
*    FUNCTION: perform data collection and updates               *
*                                                                *
*****************************************************************)
```

```
(****************************************************************
*                                                               *
*    DATE: 31 Aug 1985                                          *
*    VERSION: 1.0                                               *
*                                                               *
*    NAME: update_access_stats                                  *
*    MODULE NUMBER: 2.1.1                                       *
*    DESCRIPTION: updates access delay times for each station   *
*    PASSED VARIABLES: time of station's last access            *
*    RETURNS: time of this access = sim_clock                   *
*    GLOBAL VARIABLES USED: sim_clock     current_stats         *
*    GLOBAL VARIABLES CHANGED: none                             *
*    FILES READ: none                                           *
*    FILES WRITTEN: none                                        *
*    MODULES CALLED: min    max                                 *
*    CALLING MODULES: dist_algor   cent_algor                   *
*                                                               *
*    AUTHOR: Jim Spieth                                         *
*    HISTORY:                                                   *
*                                                               *
****************************************************************)
procedure update_access_stats( var last_access : real ;
                                      pass_cyc : integer ) ;

var
  current_delay : real ;

begin
  current_stats^.data.num_access := current_stats^.data.num_access
                                    + 1.0 ;
  if pass_cyc > 1
    then
      begin
      current_delay := sim_clock - last_access ;
      current_stats^.data.sum_access := current_stats^.data.sum_access
                                          + current_delay ;
      current_stats^.data.min_access := min(
                                current_stats^.data.min_access,
                                        current_delay ) ;
      current_stats^.data.max_access := max(
                                current_stats^.data.max_access,
                                        current_delay ) ;
      end ;
  last_access := sim_clock ;
end ;
```

```
(****************************************************************
*                                                               *
*    DATE: 21 Aug 1985                                          *
*    VERSION: 1.0                                               *
*                                                               *
*    NAME: update_message_stats                                 *
*    MODULE NUMBER: 2.1.2                                       *
*    DESCRIPTION: updates station's message statistics          *
*    PASSED VARIABLES: message length                           *
*    RETURNS: nothing                                           *
*    GLOBAL VARIABLES USED: current_stats                       *
*    GLOBAL VARIABLES CHANGED: none                             *
*    FILES READ: none                                           *
*    FILES WRITTEN: none                                        *
*    MODULES CALLED: min   max                                  *
*    CALLING MODULES: dist_algor   cent_algor                   *
*                                                               *
*    AUTHOR: Jim Spieth                                         *
*    HISTORY:                                                   *
*                                                               *
****************************************************************)
procedure update_message_stats ( mess_len : real ) ;

begin
   current_stats^.data.num_messages := current_stats^.data.num_messages
                                       + 1.0 ;
   current_stats^.data.sum_mess_len := current_stats^.data.sum_mess_len
                                       + mess_len ;
   current_stats^.data.min_mess_len := min(
                                current_stats^.data.min_mess_len,
                                        mess_len ) ;
   current_stats^.data.max_mess_len := max(
                                current_stats^.data.max_mess_len,
                                        mess_len ) ;
end ;
```

```
(****************************************************************
*                                                               *
*    DATE: 30 Aug 1985                                          *
*    VERSION: 1.0                                               *
*                                                               *
*    NAME: update_delay_stats                                   *
*    MODULE NUMBER: 2.1.3                                       *
*    DESCRIPTION: updates station's message delay statistics    *
*    PASSED VARIABLES: message arrival time                     *
*    RETURNS: nothing                                           *
*    GLOBAL VARIABLES USED: current_stats    sim_clock          *
*    GLOBAL VARIABLES CHANGED: none                             *
*    FILES READ: none                                           *
*    FILES WRITTEN: none                                        *
*    MODULES CALLED: min   max                                  *
*    CALLING MODULES: dist_algor   cent_algor                   *
*                                                               *
*    AUTHOR: Jim Spieth                                         *
*    HISTORY:                                                   *
*                                                               *
****************************************************************)
procedure update_delay_stats ( arr_time : real ) ;

var
   delay : real ;

begin
   delay := sim_clock - arr_time ;
   current_stats^.data.sum_mess_delay :=
              current_stats^.data.sum_mess_delay + delay ;
   current_stats^.data.min_mess_delay :=
              min(current_stats^.data.min_mess_delay, delay ) ;
   current_stats^.data.max_mess_delay :=
              max(current_stats^.data.max_mess_delay, delay ) ;
end ;
```

```
(****************************************************************
*                                                               *
*    DATE: 23 Sep 1985                                          *
*    VERSION: 1.1                                               *
*                                                               *
*    NAME: update_thruput_stats                                 *
*    MODULE NUMBER: 2.1.6                                       *
*    DESCRIPTION: calculates and updates data for throughput    *
*                 calculations                                  *
*    PASSED VARIABLES: token-passing cycle time - cyc_time      *
*                      sum of data bits - sum_data              *
*                      sum of overhead bits - sum_over          *
*    RETURNS: sum_data and sum_over set to zero                 *
*    GLOBAL VARIABLES USED: none                                *
*    GLOBAL VARIABLES CHANGED: total_thruput   total_eff        *
*                              total_cyc_time  num_cycle        *
*    FILES READ: none                                           *
*    FILES WRITTEN: none                                        *
*    MODULES CALLED: none                                       *
*    CALLING MODULES: dist_algor    cent_algor                  *
*                                                               *
*    AUTHOR: Jim Spieth                                         *
*    HISTORY:                                                   *
*        1.1  23 Sep 85  added total_cyc_time                   *
*        1.0  23 Aug 85  original                               *
*                                                               *
****************************************************************)
procedure update_thruput_stats(cyc_time : real ;  var sum_data,
                  sum_over : real ) ;

var
  eff,
  thruput     : real ;

begin
  num_cycle := num_cycle + 1.0 ;
  thruput := sum_data / cyc_time ;
  eff := sum_data / (sum_data + sum_over) ;
  total_thruput := total_thruput + thruput ;
  total_eff := total_eff + eff ;
  total_cyc_time := total_cyc_time + cyc_time ;
  sum_data := 0.0 ;
  sum_over := 0.0 ;
end ;
```

```
(****************************************************************
*                                                               *
*    DATE: 24 Sep 1985                                          *
*    VERSION: 3.1                                               *
*                                                               *
*    TITLE: Setup                                               *
*    FILENAME: setup.pas                                        *
*    COORDINATOR: Jim Spieth                                    *
*    PROJECT: Avionics Bus Simulation Model                     *
*    OPERATING SYSTEM: VAX/VMS, Version 4.2 on VAX-11/782       *
*    LANGUAGE: Pascal                                           *
*    USE: %include file for program bussim                      *
*    CONTENTS: mth$random                                       *
*              bus_data_input                                   *
*              station_data_input                               *
*              calc_arr_and_len                                 *
*              calc_first_arr_and_len                           *
*              init_stats                                       *
*              calc_token_prop_delays                           *
*              sel_bus_setup                                    *
*    FUNCTION: setup modules for bus configuration              *
*                                                               *
****************************************************************)



(****************************************************************
*                                                               *
*    DATE: 7 Sep 1985                                           *
*    VERSION: 1.0                                               *
*                                                               *
*    NAME: mth$random   (DEC run-time library function)         *
*    MODULE NUMBER: 1.3.1.1                                     *
*    DESCRIPTION: uniform random number generator               *
*    PASSED VARIABLES: seed                                     *
*    RETURNS: random number                                     *
*    GLOBAL VARIABLES USED: none                                *
*    GLOBAL VARIABLES CHANGED: none                             *
*    FILES READ: none                                           *
*    FILES WRITTEN: none                                        *
*    MODULES CALLED: none                                       *
*    CALLING MODULES: calc_arr_and_len                          *
*                                                               *
*    AUTHOR: Digital Equipment Corp.                            *
*    HISTORY:                                                   *
*                                                               *
****************************************************************)
[external,asynchronous] function mth$random (var seed : integer )
               : real ; extern ;
```

```
(*****************************************************************
*                                                                *
*    DATE: 24 Sep 1985                                           *
*    VERSION: 2.1                                                *
*                                                                *
*    NAME: bus_data_input                                        *
*    MODULE NUMBER: 1.1                                          *
*    DESCRIPTION: reads in bus attributes (data)                 *
*    PASSED VARIABLES: none                                      *
*    RETURNS: nothing                                            *
*    GLOBAL VARIABLES USED: none                                 *
*    GLOBAL VARIABLES CHANGED: seed  num_stations  first_station*
*       bit_rate   prop_factor  bus_length      stat_delay       *
*     bus_control  token_pass  token_hold_type  token_hold_limit*
*  station_arr  station_arr_type  station_rate  station_arr_rate*
*  station_len  station_len_type  station_mean  station_len_mean*
*      token_bits  overhead_bits  bits_per_data_word             *
*      min_data_words  max_data_words  sim_stop_time             *
*    FILES READ: input                                           *
*    FILES WRITTEN: none                                         *
*    MODULES CALLED: none                                        *
*    CALLING MODULES: sel_bus_setup                              *
*                                                                *
*    AUTHOR: Jim Spieth                                          *
*    HISTORY:                                                    *
*        2.1  24 Sep 85  added token_hold_type                   *
*        2.0  26 Aug 85  added enumerated reads and cases        *
*        1.0  20 Aug 85  original                                *
*                                                                *
*****************************************************************)
procedure bus_data_input ;

var     k, l, m : integer ;

begin
   readln(seed) ;
   readln(num_stations, first_station, bit_rate ) ;
   readln(prop_factor, bus_length, stat_delay ) ;
   readln(k, l, m, token_hold_limit ) ;
   case k of
      0 : bus_control := distrib ;
      1 : bus_control := central ;
   end ;   (* case *)
   case l of
      0 : token_pass := ascen ;
      1 : token_pass := descen ;
      2 : token_pass := fixed ;
   end ;  (* of case *)
   case m of
      0 : token_hold_type := time ;
      1 : token_hold_type := num ;
   end ;  (* of case *)
   readln(k, l, m, station_arr_rate ) ;
   case k of
      0 : station_arr := same ;
      1 : station_arr := diff ;
   end ;  (* of case *)
   case l of
      0 : station_arr_type := constant_arr ;
      1 : station_arr_type := poisson ;
      2 : station_arr_type := contin ;
   end ;  (* of case *)
   case m of
      0 : station_rate := same ;
      1 : station_rate := diff ;
   end ;  (* of case *)
   readln(k, l, m, station_len_mean ) ;
   case k of
      0 : station_len := same ;
      1 : station_len := diff ;
   end ;  (* of case *)
   case l of
      0 : station_len_type := constant_len ;
      1 : station_len_type := exp ;
   end ;  (* of case *)
   case m of
      0 : station_mean := same ;
      1 : station_mean := diff ;
   end ;  (* of case *)
   readln(token_bits, overhead_bits, bits_per_data_word ) ;
   readln(min_data_words, max_data_words ) ;
   readln(sim_stop_time ) ;
end ;     (**** bus_data_input ****)
```

```
(****************************************************************
*                                                               *
*    DATE: 13 Sep 1985                                          *
*    VERSION: 1.2                                               *
*                                                               *
*    NAME: station_data_input                                   *
*    MODULE NUMBER: 1.2                                         *
*    DESCRIPTION: reads in station attributes (data)            *
*    PASSED VARIABLES: none                                     *
*    RETURNS: nothing                                           *
*    GLOBAL VARIABLES USED: front_station    current_station    *
*                   station_arr          station_len            *
*                   station_arr_type     station_len_type       *
*                   station_rate         station_mean           *
*                   station_arr_rate     station_len_mean       *
*             calc_pass_prop_time        bits_per_data_word     *
*    GLOBAL VARIABLES CHANGED: front_station current_station    *
*                              calc_pass_prop_time              *
*    FILES READ: input                                          *
*    FILES WRITTEN: none                                        *
*    MODULES CALLED: none                                       *
*    CALLING MODULES: sel_bus_setup                             *
*                                                               *
*    AUTHOR: Jim Spieth                                         *
*    HISTORY:   1.2  13 Sep 85  added calc_pass_prop_time       *
*               1.1  27 Aug 85  added station case statements   *
*               1.0  18 Aug 85  original                        *
*                                                               *
****************************************************************)
procedure station_data_input ;

var     temp_station : station_type ;
        ptr : station_ptr ;
        h, i, j, k : integer ;
begin
   writeln('station data input module') ;
   readln( h ) ;
   if h = 0
      then calc_pass_prop_time := false
      else calc_pass_prop_time := true ;
   for i := 1 to num_stations do
      begin
      if calc_pass_prop_time
         then
            readln(temp_station.address,
                   temp_station.pass_address,
                   temp_station.distance )
         else
            readln(temp_station.address,
                   temp_station.pass_address,
                   temp_station.distance,
                   temp_station.pass_prop_time ) ;
      case station_arr of
         same : begin
                temp_station.mess_arr_type := station_arr_type ;
                case station_rate of
                  same : temp_station.mess_arr_rate := station_arr_rate;
                  diff : readln(temp_station.mess_arr_rate) ;
                end ; (* of case *)
                end ;
         diff : begin
                readln( j, temp_station.mess_arr_rate ) ;
                case j of
                   0 : temp_station.mess_arr_type := constant_arr ;
                   1 : temp_station.mess_arr_type := poisson ;
                   2 : temp_station.mess_arr_type := contin ;
                end ;  (* of case *)
                end ;
      end ;  (* of case *)
      case station_len of
         same : begin
                temp_station.mess_len_type := station_len_type ;
                case station_mean of
                  same : temp_station.mess_len_mean := station_len_mean;
                  diff : readln(temp_station.mess_len_mean) ;
                end ;  (* of case *)
                end ;
         diff : begin
                readln(k, temp_station.mess_len_mean ) ;
                case k of
                   0 : temp_station.mess_len_type := constant_len ;
                   1 : temp_station.mess_len_type := exp ;
                end ;  (* of case *)
                end ;
      end ;  (* of case *)
      temp_station.mess_len_mean := temp_station.mess_len_mean *
                                    bits_per_data_word ;
      temp_station.last_access := 0.0 ;
      temp_station.front_mess_queue := nil ;
      temp_station.rear_mess_queue := nil ;
      new(ptr) ;
      if i = 1
         then front_station := ptr
         else current_station^.next_station := ptr ;
      ptr^.attrib := temp_station ;
      if i = num_stations
         then ptr^.next_station := front_station
         else ptr^.next_station := nil ;
      current_station := ptr ;
      end ;
end ;          (****  station_data_input  ****)
```

```
(****************************************************************
*                                                               *
*    DATE: 7 Sep 1985                                           *
*    VERSION: 2.0                                               *
*                                                               *
*    NAME: calc_arr_and_len                                     *
*    MODULE NUMBER: 1.3.1                                       *
*    DESCRIPTION: calculates arrival time and length of a new   *
*                 message for a station                         *
*    PASSED VARIABLES: station arrival type and rate,           *
*                      station length distribution type and mean*
*    RETURNS: message arrival time and length                   *
*    GLOBAL VARIABLES USED: sim_clock     seed                  *
*       bits_per_data_word    min_data_words    max_data_words  *
*    GLOBAL VARIABLES CHANGED: none                             *
*    FILES READ: none                                           *
*    FILES WRITTEN: none                                        *
*    MODULES CALLED: none                                       *
*    CALLING MODULES: calc_first_arr_and_len                    *
*                                                               *
*    AUTHOR: Jim Spieth                                         *
*    HISTORY:                                                   *
*        2.0  updated dummy calculations to real ones           *
*        1.0  20 Aug 85 dummy constant, Poisson & exp calcs     *
*                                                               *
****************************************************************)
procedure calc_arr_and_len(arr_type : arrival ;
                           arr_rate : real ;
                           len_type : length_distrib ;
                           len_mean : real ;
                           var arrival_time : real ;
                           var message_len : real ) ;

var   length : integer ;

begin
   case arr_type of
     constant_arr : begin
                    if arr_rate =  0.0
                      then arrival_time := 0.0
                      else arrival_time := sim_clock + (1.0 / arr_rate);
                    end ;
     poisson : arrival_time := sim_clock -
                          ((1.0 / arr_rate) * ln(mth$random(seed))) ;
     contin : arrival_time := 0.0 ;
   end ;    (* of case *)
   case len_type of
      constant_len : message_len := len_mean ;
      exp : begin
            message_len := abs(len_mean * ln(mth$random(seed))) ;
            message_len := message_len / bits_per_data_word ;
            length := round(message_len) ;
            message_len := length ;
            if message_len > max_data_words
              then message_len := max_data_words
              else if message_len < min_data_words
                     then message_len := min_data_words ;
            message_len := message_len * bits_per_data_word ;
            end ;
   end      (* of case *)
end ;       (****  calc_arr_and_len  ****)
```

```
(****************************************************************
*                                                               *
*    DATE: 27 Aug 1985                                          *
*    VERSION: 1.1                                               *
*                                                               *
*    NAME: calc_first_arr_and_len                               *
*    MODULE NUMBER: 1.3                                         *
*    DESCRIPTION: calculates arrival time and length of first   *
*                 messages for all stations                     *
*    PASSED VARIABLES: none                                     *
*    RETURNS: none                                              *
*    GLOBAL VARIABLES USED: current_station   front_station     *
*    GLOBAL VARIABLES CHANGED: current_station                  *
*    FILES READ: none                                           *
*    FILES WRITTEN: none                                        *
*    MODULES CALLED: calc_arr_and_len                           *
*    CALLING MODULES: sel_bus_setup                             *
*                                                               *
*    AUTHOR: Jim Spieth                                         *
*    HISTORY:                                                   *
*        1.1  27 Aug 85  added test for constant arr rate of 0.0*
*        1.0  20 Aug 85  original                               *
*                                                               *
****************************************************************)
procedure calc_first_arr_and_len ;

var
   i : integer ;
   arrival_time, message_len : real ;
   temp_mess : message_type ;
   ptr : message_ptr ;

begin
   writeln('calc first arr and len module') ;
   current_station := front_station ;
```

```
     for i := 1 to num_stations do
        begin
        if ((current_station^.attrib.mess_arr_type <> constant_arr) or
            (current_station^.attrib.mess_arr_rate <> 0.0 ))
          then
            begin
            calc_arr_and_len(
                         current_station^.attrib.mess_arr_type,
                         current_station^.attrib.mess_arr_rate,
                         current_station^.attrib.mess_len_type,
                         current_station^.attrib.mess_len_mean,
                                     arrival_time, message_len) ;
            temp_mess.source_add := current_station^.attrib.address ;
            temp_mess.length := message_len ;
            temp_mess.arr_time := arrival_time ;
            new(ptr) ;
            ptr^.info := temp_mess ;
            ptr^.next_message := nil ;
            current_station^.attrib.front_mess_queue := ptr ;
            current_station^.attrib.rear_mess_queue := ptr ;
            end ;
        current_station := current_station^.next_station
        end
  end ;        (****  calc_first_arr_and_len  ****)
```

```
(****************************************************************
*                                                               *
*    DATE: 22 Aug 1985                                          *
*    VERSION: 1.0                                               *
*                                                               *
*    NAME: init_stats                                           *
*    MODULE NUMBER: 1.4                                         *
*    DESCRIPTION: initializes the station statistics linked list*
*    PASSED VARIABLES: none                                     *
*    RETURNS: nothing                                           *
*    GLOBAL VARIABLES USED: front_station                       *
*    GLOBAL VARIABLES CHANGED: front_stats   current_stats      *
*    FILES READ: none                                           *
*    FILES WRITTEN: none                                        *
*    MODULES CALLED: none                                       *
*    CALLING MODULES: sel_bus_setup                             *
*                                                               *
*    AUTHOR: Jim Spieth                                         *
*    HISTORY:                                                   *
*                                                               *
****************************************************************)
procedure init_stats ;

var
   temp_stats : stats_type ;
   ptr : stats_ptr ;
     j : integer ;

begin
   writeln('init stats module') ;
   current_station := front_station ;
   for j := 1 to num_stations do
     begin
     temp_stats.address := current_station^.attrib.address ;
     temp_stats.num_access := 0.0 ;
     temp_stats.sum_access := 0.0 ;
     temp_stats.min_access := 1.0e4 ;
     temp_stats.max_access := 1.0e-9 ;
     temp_stats.num_messages := 0.0 ;
     temp_stats.sum_mess_len := 0.0 ;
     temp_stats.min_mess_len := 1.0e6 ;
     temp_stats.max_mess_len := 1.0 ;
     temp_stats.sum_mess_delay := 0.0 ;
     temp_stats.min_mess_delay := 1.0e4 ;
     temp_stats.max_mess_delay := 1.0e-9 ;
     new(ptr) ;
     if j = 1
       then
         begin
         front_stats := ptr ;
         current_stats := ptr ;
         end
       else
         current_stats^.next := ptr ;
     ptr^.data := temp_stats ;
     if j = num_stations
        then ptr^.next := front_stats
        else ptr^.next := nil ;
     current_station := current_station^.next_station ;
     if j > 1 then current_stats := ptr ;
     end ;
end ;         (****  init_stats  ****)
```

```
(****************************************************************
*                                                               *
*    DATE: 13 Sep 1985                                          *
*    VERSION: 1.1                                               *
*                                                               *
*    NAME: calc_token_prop_delays                               *
*    MODULE NUMBER: 1.5                                         *
*    DESCRIPTION: calculates token-passing propagation times    *
*    PASSED VARIABLES: none                                     *
*    RETURNS: nothing                                           *
*    GLOBAL VARIABLES USED: front_station  num_stations         *
*                           prop_factor    speed_light          *
*                           calc_pass_prop_time                 *
*    GLOBAL VARIABLES CHANGED: current_station  sig_prop        *
*                              sig_delay                        *
*    FILES READ: none                                           *
*    FILES WRITTEN: none                                        *
*    MODULES CALLED: none                                       *
*    CALLING MODULES: sel_bus_setup                             *
*                                                               *
*    AUTHOR: Jim Spieth                                         *
*    HISTORY:  1.1  13 Sep 85  added calc_pass_prop_time test   *
*              1.0  31 Aug 85  original                         *
*                                                               *
****************************************************************)
procedure calc_token_prop_delays ;

var      distance : real ;
         p, q     : integer ;
begin
   sig_prop := prop_factor * speed_light ;
   sig_delay := 1.0 / sig_prop ;
   current_station := front_station ;
   if calc_pass_prop_time
      then
      for p := 1 to num_stations do
        begin
        distance := abs(current_station^.attrib.distance -
                     current_station^.next_station^.attrib.distance ) ;
        current_station^.attrib.pass_prop_time := distance * sig_delay ;
        writeln(current_station^.attrib.address, '    ',
                current_station^.attrib.pass_prop_time ) ;
        current_station := current_station^.next_station ;
        end
      else
      for q := 1 to num_stations do
         begin
         writeln(current_station^.attrib.address, '    ',
                 current_station^.attrib.pass_prop_time ) ;
         current_station := current_station^.next_station ;
         end ;
end ;
```

```
(****************************************************************
*                                                               *
*    DATE: 31 Aug 1985                                          *
*    VERSION: 1.0                                               *
*                                                               *
*    NAME: sel_bus_setup                                        *
*    MODULE NUMBER: 1.0                                         *
*    DESCRIPTION: executive for modules to setup bus config.    *
*    PASSED VARIABLES: none                                     *
*    RETURNS: nothing                                           *
*    GLOBAL VARIABLES USED: none                                *
*    GLOBAL VARIABLES CHANGED: none                             *
*    FILES READ: none                                           *
*    FILES WRITTEN: none                                        *
*    MODULES CALLED: bus_data_input                             *
*                    station_data_input                         *
*                    calc_first_arr_and_len                     *
*                    init_stats                                 *
*                    clac_token_prop_delays                     *
*    CALLING MODULES: bussim (main)                             *
*                                                               *
*    AUTHOR: Jim Spieth                                         *
*    HISTORY:                                                   *
*                                                               *
****************************************************************)
procedure sel_bus_setup ;

begin
   writeln('select bus configuration setup module') ;
   bus_data_input ;
   station_data_input ;
   calc_first_arr_and_len ;
   init_stats ;
   calc_token_prop_delays ;
end ;
```

```
(****************************************************************
*                                                               *
*    DATE: 24 Sep 1985                                          *
*    VERSION: 1.4                                               *
*                                                               *
*    TITLE: simulate                                            *
*    FILENAME: simulate.pas                                     *
*    COORDINATOR: Jim Spieth                                    *
*    PROJECT: Avionics Bus Simulation Model                     *
*    OPERATING SYSTEM: VAX/VMS, Version 4.2 on VAX-11/782       *
*    LANGUAGE: Pascal                                           *
*    USE: %include file for program bussim                      *
*    CONTENTS: simulate                                         *
*              calc_next_arr_and_len                            *
*              dist_algor                                       *
*              cent_algor                                       *
*    FUNCTION: performs the simulation of the bus               *
*                                                               *
****************************************************************)
```

```
(*****************************************************************
*                                                                *
*    DATE: 21 Aug 1985                                           *
*    VERSION: 1.0                                                *
*                                                                *
*    NAME: calc_next_arr_and_len                                 *
*    MODULE NUMBER: 2.1.4                                        *
*    DESCRIPTION: calculates arrival time and length of next     *
*                 message for current station                    *
*    PASSED VARIABLES: none                                      *
*    RETURNS: none                                               *
*    GLOBAL VARIABLES USED: current_station                      *
*    GLOBAL VARIABLES CHANGED: none                              *
*    FILES READ: none                                            *
*    FILES WRITTEN: none                                         *
*    MODULES CALLED: calc_arr_and_len                            *
*    CALLING MODULES: dist_algor                                 *
*                                                                *
*    AUTHOR: Jim Spieth                                          *
*    HISTORY:                                                    *
*                                                                *
*****************************************************************)
procedure calc_next_arr_and_len ;

var
   arrival_time,
   message_len    : real ;
   temp_mess      : message_type ;

begin
   calc_arr_and_len(
                    current_station^.attrib.mess_arr_type,
                    current_station^.attrib.mess_arr_rate,
                    current_station^.attrib.mess_len_type,
                    current_station^.attrib.mess_len_mean,
                                  arrival_time, message_len) ;
   temp_mess.source_add := current_station^.attrib.address ;
   temp_mess.length := message_len ;
   temp_mess.arr_time := arrival_time ;
   in_rear_queue(current_station^.attrib.front_mess_queue,
                 current_station^.attrib.rear_mess_queue, temp_mess) ;
end ;
```

```
(*****************************************************************
*                                                                *
*    DATE: 24 Sep 1985                                           *
*    VERSION: 1.3                                                *
*                                                                *
*    NAME: dist_algor                                            *
*    MODULE NUMBER: 2.1                                          *
*    DESCRIPTION: simulates distributed token-passing algorithm *
*    PASSED VARIABLES: none                                      *
*    RETURNS: none                                               *
*    GLOBAL VARIABLES USED: sim_clock   pass_cycle  bit_rate    *
*                           current_station    current_stats    *
*                           token_bits  stat_delay  num_stations*
*                           token_hold_limit  token_hold_type   *
*    GLOBAL VARIABLES CHANGED: current_station  current_stats   *
*    FILES READ: none                                            *
*    FILES WRITTEN: none                                         *
*    MODULES CALLED: calc_next_arr_and_len                       *
*                    out_front_queue                             *
*                    update_access_stats                         *
*                    update_message_stats                        *
*                    update_delay_stats                          *
*                    update_thruput_stats                        *
*    CALLING MODULES: simulate                                   *
*                                                                *
*    AUTHOR: Jim Spieth                                          *
*    HISTORY:                                                    *
*        1.3  24 Sep 85  added token_hold_type                   *
*        1.2  31 Aug 85  added update_delay_stats call           *
*        1.1  29 Aug 85  added update_access_stats call          *
*        1.0  21 Aug 85  original                                *
*                                                                *
*****************************************************************)
procedure dist_algor ;

var   send, station_count,
      pass_cycle             : integer ;

      data_len, mess_tx, mess_len, cycle_time, sum_data_bits,
      sum_over_bits, hold_limit     : real ;

begin
writeln('dist algor module') ;
send := 1 ;                                (** send=1=yes, send=0=no **)
station_count := 1 ;
pass_cycle := 1 ;
cycle_time := sim_clock ;
sum_data_bits := 0.0 ;
sum_over_bits := 0.0 ;
current_station := front_station ;
current_stats := front_stats ;
hold_limit := token_hold_limit ;
```

```
while (sim_clock < sim_stop_time) do
  begin
  update_access_stats(current_station^.attrib.last_access, pass_cycle) ;
  if (current_station^.attrib.front_mess_queue <> nil)
    then
      begin
      while (send = 1)  do
        begin
        if current_station^.attrib.mess_arr_type <> contin then
          if current_station^.attrib.front_mess_queue^.info.arr_time
                > sim_clock then send := 0 ;
        if send = 1 then
          begin
          data_len :=
              current_station^.attrib.front_mess_queue^.info.length ;
          mess_len := data_len + overhead_bits ;
          mess_tx := mess_len / bit_rate ;
          case token_hold_type of
             time : if mess_tx > hold_limit
                      then send := 0 ;
             num : if hold_limit = 0.0
                      then send := 0 ;
          end ;  (* of case *)
          if send = 1 then
              begin
              update_message_stats(mess_len) ;
              sim_clock := sim_clock + mess_tx ;
              if current_station^.attrib.mess_arr_type <> contin then
                update_delay_stats(
   current_station^.attrib.front_mess_queue .info.arr_time) ;
              calc_next_arr_and_len ;
              out_front_queue(current_station^.attrib.front_mess_queue,
                          current_station^.attrib.rear_mess_queue) ;
              sum_data_bits := sum_data_bits + data_len ;
              sum_over_bits := sum_over_bits + overhead_bits ;
              case token_hold_type of
                 time : hold_limit := hold_limit - mess_tx ;
                 num  : hold_limit := hold_limit - 1.0 ;
              end ;  (* of case *)
              end ;
          end ;
        end ;  (* send while *)
      end ;  (* nil if *)
  sim_clock := sim_clock + (token_bits / bit_rate) ;
  sum_over_bits := sum_over_bits + token_bits ;
  sim_clock := sim_clock + current_station^.attrib.pass_prop_time ;
  sim_clock := sim_clock + stat_delay ;
  current_station := current_station^.next_station ;
  current_stats := current_stats^.next ;
  station_count := station_count + 1 ;
  send := 1 ;
  hold_limit := token_hold_limit ;
  if station_count > num_stations then
    begin
    pass_cycle := pass_cycle + 1 ;
    station_count := 1 ;
    cycle_time := sim_clock - cycle_time ;
    update_thruput_stats(cycle_time, sum_data_bits, sum_over_bits) ;
    cycle_time := sim_clock ;
    end ;
end        (**** while ****)
end ;      (**** dist_algor ****)
```

```
(****************************************************************
*                                                               *
*    DATE: 24 Sep 1985                                          *
*    VERSION: 2.1                                               *
*                                                               *
*    NAME: cent_algor                                           *
*    MODULE NUMBER: 2.2                                         *
*    DESCRIPTION: simulates centralized token-passing algorithm *
*    PASSED VARIABLES: none                                     *
*    RETURNS: none                                              *
*    GLOBAL VARIABLES USED: sim_clock   pass_cycle  bit_rate    *
*                           current_station   current_stats     *
*                           token_bits  stat_delay  num_stations*
*                           token_hold_limit  token_hold_type   *
*    GLOBAL VARIABLES CHANGED: current_station  current_stats   *
*    FILES READ: none                                           *
*    FILES WRITTEN: none                                        *
*    MODULES CALLED: calc_next_arr_and_len                      *
*                    out_front_queue                            *
*                    update_access_stats                        *
*                    update_message_stats                       *
*                    update_delay_stats                         *
*                    update_thruput_stats                       *
*    CALLING MODULES: simulate                                  *
*                                                               *
*    AUTHOR: Jim Spieth                                         *
*    HISTORY:                                                   *
*        2.1  24 Sep 85 added token_hold_type                   *
*        2.0  17 Sep 85 simplified (deleted next message lines) *
*        1.0  31 Aug 85 original                                *
*                                                               *
****************************************************************)
procedure cent_algor ;
var   send, send_token, pass,
      station_count, pass_cycle      : integer ;

      data_len, mess_tx, mess_len,
      cycle_time, sum_data_bits,
      sum_over_bits, hold_limit      : real ;
begin
writeln('centralized algorithm procedure called') ;
send := 1 ;                             (** send=1=yes, send=0=no **)
send_token := 0 ;
pass := 0 ;
station_count := 1 ;
pass_cycle := 1 ;
cycle_time := sim_clock ;
sum_data_bits := 0.0 ;
sum_over_bits := 0.0 ;
current_station := front_station ;
current_stats := front_stats ;
hold_limit := token_hold_limit ;
```

```
while (sim_clock < sim_stop_time) do
begin
update_access_stats(current_station^.attrib.last_access, pass_cycle ) ;
if current_station^.attrib.front_mess_queue = nil
  then send_token := 1
  else
    while (send = 1) do
      begin
      if current_station^.attrib.mess_arr_type <> contin
        then if current_station^.attrib.front_mess_queue^.info.arr_time
                         > sim_clock
               then
                 begin
                 send := 0 ;
                 if pass > 0
                   then send_token := 0
                   else send_token := 1
                 end ;
      if send = 1
        then
          begin
          data_len :=
                current_station^.attrib.front_mess_queue^.info.length ;
          mess_len := data_len + overhead_bits ;
          mess_tx := mess_len / bit_rate ;
          case token_hold_type of
             time : if mess_tx > hold_limit
                      then
                        begin
                        send := 0 ;
                        if pass > 0
                          then send_token := 0
                          else send_token := 1
                        end ;
             num : if hold_limit = 0.0
                     then
                       begin
                       send := 0 ;
                       if pass > 0
                         then send_token := 0
                         else send_token := 1
                       end ;
          end ;  (* of case *)
          if send = 1   then
              begin                                    (* send message *)
              update_message_stats(mess_len) ;
              sim_clock := sim_clock + mess_tx ;
              if current_station^.attrib.mess_arr_type <> contin then
                update_delay_stats(
        current_station^.attrib.front_mess_queue^.info.arr_time) ;
              calc_next_arr_and_len ;
              out_front_queue(current_station^.attrib.front_mess_queue,
                        current_station^.attrib.rear_mess_queue) ;
              sum_data_bits := sum_data_bits + data_len ;
              sum_over_bits := sum_over_bits + overhead_bits ;
              pass := pass + 1 ;
              case token_hold_type of
                 time : hold_limit := hold_limit - mess_tx ;
                 num  : hold_limit := hold_limit - 1.0 ;
              end ;  (* of case *)
              end ;
          end ;
      end ;   (* send while  *)
if send_token = 1
  then
    begin
    sim_clock := sim_clock + (token_bits / bit_rate) ;
    sum_over_bits := sum_over_bits + token_bits ;
    end ;
sim_clock := sim_clock + current_station^.attrib.pass_prop_time ;
sim_clock := sim_clock + stat_delay ;
current_station := current_station^.next_station ;
current_stats := current_stats^.next ;
station_count := station_count + 1 ;
send := 1 ;
send_token := 0 ;
pass := 0 ;
hold_limit := token_hold_limit ;
if station_count > num_stations
  then
    begin
    pass_cycle := pass_cycle + 1 ;
    station_count := 1 ;
    cycle_time := sim_clock - cycle_time ;
    update_thruput_stats(cycle_time, sum_data_bits, sum_over_bits) ;
    cycle_time := sim_clock ;
    end ;
  end ;      (**** while ****)
end ;        (**** cent_algor ****)
```

```
(*****************************************************************
*                                                                *
*    DATE: 24 Aug 1985                                           *
*    VERSION: 1.0                                                *
*                                                                *
*    NAME: simulate                                              *
*    MODULE NUMBER: 2.0                                          *
*    DESCRIPTION: executive for bus simulation                   *
*    PASSED VARIABLES: none                                      *
*    RETURNS: none                                               *
*    GLOBAL VARIABLES USED: bus_control                          *
*    GLOBAL VARIABLES CHANGED: none                              *
*    FILES READ: none                                            *
*    FILES WRITTEN: none                                         *
*    MODULES CALLED: dist_algor                                  *
*                    cent_algor                                  *
*    CALLING MODULES: bussim (main)                              *
*                                                                *
*    AUTHOR: Jim Spieth                                          *
*    HISTORY:                                                    *
*                                                                *
*****************************************************************)
procedure simulate ;

begin
   writeln('simulation control module entered') ;
   case  bus_control  of
      distrib : dist_algor ;
      central : cent_algor
      end  (*  of case  *)
end ;
```

Appendix C. Test Case Command Files

This appendix contains the command files used to execute the seven test case simulations in Chapter IV. The first line of each file is a comment indicating which test case and condition the file was used for. For most of the test cases, ten simulations were made with different message arrival rates to generate data which was presented as delay-throughput curves. However, only one command file representing one message arrival rate set is included in this appendix for each condition of each test case.

```
! First Test Case    Equal Message Arrival Rates
$ run  disk$user:[spieth.bus]bussim
707
30             1        50.0e6
0.666666666    60.0     0.5e-6
1              2        0       83.32e-6
0              1        0       1500.0
0              1        0       64.0
22.0          70.0    16.0
0.0          256.0
0.6
    1
1       2       2.0
2       3       2.5
3       4       3.0
4       5       3.5
5       6       4.0
6       7       4.5
7       8       5.0
8       9       6.0
9       10      7.0
10      11      8.0
11      12      9.0
12      13      10.0
13      14      13.0
14      15      14.0
15      16      15.0
16      17      16.0
17      18      17.0
18      19      18.0
19      20      28.0
20      21      29.0
21      22      30.0
22      23      31.0
23      24      32.0
24      25      33.0
25      26      55.0
26      27      56.0
27      28      57.0
28      29      58.0
29      30      59.0
30      1       60.0
```

```
! First Test Case   Unequal Message Arrival Rates
$ run  disk$user:[spieth.bus]bussim
480
30              1       50.0e6
0.666666666     60.0    0.5e-6
1               2       0       83.32e-6
0               1       1       400.0
0               1       0       64.0
22.0          70.0     16.0
0.0             256.0
0.6
    1
1       2       2.0
10.0
2       3       2.5
50.0
3       4       3.0
500.0
4       5       3.5
500.0
5       6       4.0
50.0
6       7       4.5
10.0
7       8       5.0
10.0
8       9       6.0
50.0
9       10      7.0
500.0
10      11      8.0
500.0
11      12      9.0
50.0
12      13      10.0
10.0
13      14      13.0
10.0
14      15      14.0
50.0
15      16      15.0
500.0
16      17      16.0
500.0
17      18      17.0
50.0
18      19      18.0
10.0
19      20      28.0
10.0
20      21      29.0
50.0
```

```
21      22      30.0
500.0
22      23      31.0
500.0
23      24      32.0
50.0
24      25      33.0
10.0
25      26      55.0
10.0
26      27      56.0
50.0
27      28      57.0
500.0
28      29      58.0
500.0
29      30      59.0
50.0
30      1       60.0
10.0
```

```
! Second Test Case   Mean Message Length = 128 Data Words
$ run  disk$user:[spieth.bus]bussim
480
30              1       50.0e6
0.666666666     60.0    0.5e-6
1               2       0       83.32e-6
0               1       1       400.0
0               1       0       128.0
22.0          70.0     16.0
0.0             256.0
0.6
    1
1       2       2.0
10.0
2       3       2.5
50.0
3       4       3.0
500.0
4       5       3.5
500.0
5       6       4.0
50.0
6       7       4.5
10.0
7       8       5.0
10.0
8       9       6.0
50.0
9       10      7.0
500.0
10      11      8.0
500.0
11      12      9.0
50.0
12      13      10.0
10.0
13      14      13.0
10.0
14      15      14.0
50.0
15      16      15.0
500.0
16      17      16.0
500.0
17      18      17.0
50.0
18      19      18.0
10.0
19      20      28.0
10.0
20      21      29.0
50.0
```

```
21      22      30.0
500.0
22      23      31.0
500.0
23      24      32.0
50.0
24      25      33.0
10.0
25      26      55.0
10.0
26      27      56.0
50.0
27      28      57.0
500.0
28      29      58.0
500.0
29      30      59.0
50.0
30      1       60.0
10.0
```

```
! Second Test Case   Mean Message Length = 32 Data Words
$ run  disk$user:[spieth.bus]bussim
480
30              1       50.0e6
0.666666666     60.0    0.5e-6
1               2       0       83.32e-6
0               1       1       400.0
0               1       0       32.0
22.0          70.0     16.0
0.0             256.0
0.6
    1
1       2       2.0
10.0
2       3       2.5
50.0
3       4       3.0
500.0
4       5       3.5
500.0
5       6       4.0
50.0
6       7       4.5
10.0
7       8       5.0
10.0
8       9       6.0
50.0
9       10      7.0
500.0
10      11      8.0
500.0
11      12      9.0
50.0
12      13      10.0
10.0
13      14      13.0
10.0
14      15      14.0
50.0
15      16      15.0
500.0
16      17      16.0
500.0
17      18      17.0
50.0
18      19      18.0
10.0
19      20      28.0
10.0
20      21      29.0
50.0
```

```
21       22       30.0
500.0
22       23       31.0
500.0
23       24       32.0
50.0
24       25       33.0
10.0
25       26       55.0
10.0
26       27       56.0
50.0
27       28       57.0
500.0
28       29       58.0
500.0
29       30       59.0
50.0
30       1        60.0
10.0
```

```
! Third Test Case  Worst Case Token-Passing Sequence
$ run  disk$user:[spieth.bus]bussim
480
30              1       50.0e6
0.666666666     60.0    0.5e-6
1               2       0       83.32e-6
0               1       1       400.0
0               1       0       64.0
22.0          70.0     16.0
0.0          256.0
0.6
    1
1       30       2.0
10.0
30       2      60.0
10.0
2       29       2.5
50.0
29       3      59.0
50.0
3       28       3.0
500.0
28       4      58.0
500.0
4       27       3.5
500.0
27       5      57.0
500.0
5       26       4.0
50.0
26       6      56.0
50.0
6       25       4.5
10.0
25       7      55.0
10.0
7       24       5.0
10.0
24       8      33.0
10.0
8       23       6.0
50.0
23       9      32.0
50.0
9       22       7.0
500.0
22      10      31.0
500.0
10      21       8.0
500.0
21      11      30.0
500.0
```

```
11        20        9.0
50.0
20        12       29.0
50.0
12        19       10.0
10.0
19        13       28.0
10.0
13        18       13.0
10.0
18        14       18.0
10.0
14        17       14.0
50.0
17        15       17.0
50.0
15        16       15.0
500.0
16         1       16.0
500.0
```

```
! Fourth Test Case  Bit Rate = 25 megabits/second
$ run  disk$user:[spieth.bus]bussim
480
30              1       25.0e6
0.666666666     60.0    0.5e-6
1               2       0       166.64e-6
0               1       1       400.0
0               1       0       64.0
22.0          70.0     16.0
0.0             256.0
0.6
    1
1       2       2.0
10.0
2       3       2.5
50.0
3       4       3.0
500.0
4       5       3.5
500.0
5       6       4.0
50.0
6       7       4.5
10.0
7       8       5.0
10.0
8       9       6.0
50.0
9       10      7.0
500.0
10      11      8.0
500.0
11      12      9.0
50.0
12      13      10.0
10.0
13      14      13.0
10.0
14      15      14.0
50.0
15      16      15.0
500.0
16      17      16.0
500.0
17      18      17.0
50.0
18      19      18.0
10.0
19      20      28.0
10.0
20      21      29.0
50.0
```

```
21      22      30.0
500.0
22      23      31.0
500.0
23      24      32.0
50.0
24      25      33.0
10.0
25      26      55.0
10.0
26      27      56.0
50.0
27      28      57.0
500.0
28      29      58.0
500.0
29      30      59.0
50.0
30      1       60.0
10.0
```

```
! Fourth Test Case  Bit Rate = 40 megabits/second
$ run  disk$user:[spieth.bus]bussim
480
30              1       40.0e6
0.666666666     60.0    0.5e-6
1               2       0       104.15e-6
0               1       1       400.0
0               1       0       64.0
22.0          70.0     16.0
0.0             256.0
0.6
    1
1       2       2.0
10.0
2       3       2.5
50.0
3       4       3.0
500.0
4       5       3.5
500.0
5       6       4.0
50.0
6       7       4.5
10.0
7       8       5.0
10.0
8       9       6.0
50.0
9       10      7.0
500.0
10      11      8.0
500.0
11      12      9.0
50.0
12      13      10.0
10.0
13      14      13.0
10.0
14      15      14.0
50.0
15      16      15.0
500.0
16      17      16.0
500.0
17      18      17.0
50.0
18      19      18.0
10.0
19      20      28.0
10.0
20      21      29.0
50.0
```

```
21          22          30.0
500.0
22          23          31.0
500.0
23          24          32.0
50.0
24          25          33.0
10.0
25          26          55.0
10.0
26          27          56.0
50.0
27          28          57.0
500.0
28          29          58.0
500.0
29          30          59.0
50.0
30          1           60.0
10.0
```

```
! Fifth Test Case  Maximum Message Length = 1024 data words
$ run  disk$user:[spieth.bus]bussim
480
30              1       50.0e6
0.666666666     60.0    0.5e-6
1               2       0       329.08e-6
0               1       1       400.0
0               1       0       64.0
22.0          70.0     16.0
0.0           1024.0
0.7
    1
1       2       2.0
10.0
2       3       2.5
50.0
3       4       3.0
500.0
4       5       3.5
500.0
5       6       4.0
50.0
6       7       4.5
10.0
7       8       5.0
10.0
8       9       6.0
50.0
9       10      7.0
500.0
10      11      8.0
500.0
11      12      9.0
50.0
12      13      10.0
10.0
13      14      13.0
10.0
14      15      14.0
50.0
15      16      15.0
500.0
16      17      16.0
500.0
17      18      17.0
50.0
18      19      18.0
10.0
19      20      28.0
10.0
20      21      29.0
50.0
```

```
21       22        30.0
500.0
22       23        31.0
500.0
23       24        32.0
50.0
24       25        33.0
10.0
25       26        55.0
10.0
26       27        56.0
50.0
27       28        57.0
500.0
28       29        58.0
500.0
29       30        59.0
50.0
30       1         60.0
10.0
```

```
! Fifth Test Case  Maximum Message Length = 4096 data words
$ run  disk$user:[spieth.bus]bussim
480
30              1       50.0e6
0.666666666     60.0    0.5e-6
1               2       0       1.312120e-3
0               1       1       400.0
0               1       0       64.0
22.0          70.0     16.0
0.0           4096.0
0.8
    1
1       2       2.0
10.0
2       3       2.5
50.0
3       4       3.0
500.0
4       5       3.5
500.0
5       6       4.0
50.0
6       7       4.5
10.0
7       8       5.0
10.0
8       9       6.0
50.0
9       10      7.0
500.0
10      11      8.0
500.0
11      12      9.0
50.0
12      13      10.0
10.0
13      14      13.0
10.0
14      15      14.0
50.0
15      16      15.0
500.0
16      17      16.0
500.0
17      18      17.0
50.0
18      19      18.0
10.0
19      20      28.0
10.0
20      21      29.0
50.0
```

```
21       22       30.0
500.0
22       23       31.0
500.0
23       24       32.0
50.0
24       25       33.0
10.0
25       26       55.0
10.0
26       27       56.0
50.0
27       28       57.0
500.0
28       29       58.0
500.0
29       30       59.0
50.0
30       1        60.0
10.0
```

```
! Sixth Test Case  Deterministic Message Arrivals
$ run  disk$user:[spieth.bus]bussim
480
30              1       50.0e6
0.666666666     60.0    0.5e-6
1               2       0       83.32e-6
0               0       1       400.0
0               1       0       64.0
22.0          70.0     16.0
0.0          256.0
0.6
    1
1       2       2.0
10.0
2       3       2.5
50.0
3       4       3.0
500.0
4       5       3.5
500.0
5       6       4.0
50.0
6       7       4.5
10.0
7       8       5.0
10.0
8       9       6.0
50.0
9       10      7.0
500.0
10      11      8.0
500.0
11      12      9.0
50.0
12      13      10.0
10.0
13      14      13.0
10.0
14      15      14.0
50.0
15      16      15.0
500.0
16      17      16.0
500.0
17      18      17.0
50.0
18      19      18.0
10.0
19      20      28.0
10.0
20      21      29.0
50.0
```

```
21       22       30.0
500.0
22       23       31.0
500.0
23       24       32.0
50.0
24       25       33.0
10.0
25       26       55.0
10.0
26       27       56.0
50.0
27       28       57.0
500.0
28       29       58.0
500.0
29       30       59.0
50.0
30       1        60.0
10.0
```

```
! Seventh Test Case  Distributed Control Protocol
$ run  disk$user:[spieth.bus]bussim
480
30              1       50.0e6
0.666666666     60.0    0.5e-6
0               1       0        1.31136e-3
0               1       1        400.0
0               1       0        128.0
112.0        112.0     8.0
0.0         8182.0
1.8
    1
1       2       2.0
10.0
2       3       2.5
50.0
3       4       3.0
500.0
4       5       3.5
500.0
5       6       4.0
50.0
6       7       4.5
10.0
7       8       5.0
10.0
8       9       6.0
50.0
9       10      7.0
500.0
10      11      8.0
500.0
11      12      9.0
50.0
12      13      10.0
10.0
13      14      13.0
10.0
14      15      14.0
50.0
15      16      15.0
500.0
16      17      16.0
500.0
17      18      17.0
50.0
18      19      18.0
10.0
19      20      28.0
10.0
20      21      29.0
50.0
```

```
21      22      30.0
500.0
22      23      31.0
500.0
23      24      32.0
50.0
24      25      33.0
10.0
25      26      55.0
10.0
26      27      56.0
50.0
27      28      57.0
500.0
28      29      58.0
500.0
29      30      59.0
50.0
30      1       60.0
10.0
```

Bibliography

Alber, Harold J., Group Leader, Multiplex Standards Group. Personal interview. US Air Force, ASD/ENASF, Wright-Patterson AFB, Ohio, 29 March 1985.

Alber, Harold J. and Wayne A. Thomas. "A Dual Channel High Speed Fiber Optics Multiplex Data Bus System," Proceedings of The IEEE 1985 National Aerospace and Electronics Conference. 130-135. IEEE, New York, 1985.

Boeing Military Airplane Company. MIL-STD-1553 Multiplex Applications Handbook. Final Technical Report, Contract F33165-78-C-0112. Boeing Military Airplane Company, Seattle, Washington, 1980.

Bux, W. "Local-Area Subnetworks: A Performance Comparison," Proceedings of the IFIP Working Group 6.4 International Workshop on Local Networks (Local Networks For Computer Communications). 157-180. North-Holland Publishing Company, Amsterdam, Netherlands, 1981.

Cherukuri, Rao et al. "Evaluation of Token Passing Schemes in Local Area Networks," Proceedings of the Computer Networking Symposium. 57-68. IEEE Computer Society Press, Silver Spring, Maryland, 1982.

Dale, Nell and David Orshalick. Introduction to PASCAL and Structured Design. Lexington, Massachusetts: D. C. Heath and Company, 1983.

Fortier, Paul J. and Richard G. Leary. "A General Simulation Model For The Evaluation Of Distributed Processing Systems," Proceedings of the 14th Annual Simulation Symposium. 215-226. IEEE Computer Society Press, Silver Spring, Maryland, 1981.

Gifford, Charles A. "A Military Standard For Multiplex Data Bus," Proceedings of The IEEE 1974 National Aerospace and Electronics Conference. 85-88. IEEE, New York, 1974.

Institute Of Electrical and Electronics Engineers (IEEE) Draft Standard 802.4, Token-Passing Bus Access Method and Physical Layer Specification. IEEE, New York, 1982.

Jackman, John J. and D. J. Medeiros. "Modeling And Analysis Of Ethernet Networks," Proceedings of the 1984 Winter Simulation Conference. 595-600. IEEE, New York, 1984.

Klass, Philip J. "Collins Developing High-Speed Data Bus for Military Aircraft," Aviation Week & Space Technology, 123: 165-169+ (October 21, 1985).

Kurose, James F. et al. "Multiple-Access Protocols and Time-Constrained Communication," Computing Surveys, 16: 43-70 (March 1984).

Liu, Ming T. et al. "Performance Evaluation of Channel Access Protocols for Local Computer Networks," Proceedings of the 25th IEEE Computer Society International Conference. 417-426. IEEE Computer Society Press, Silver Spring, Maryland, 1982.

Ludvigson, M. T. and K. L. Milton. "High Speed Bus For Pave Pillar Applications," Proceedings of The IEEE 1985 National Aerospace and Electronics Conference. 122-129. IEEE, New York, 1985.

Miller, C. Kenneth and David M. Thompson. "Making a case for token passing in local networks," Data Communications, 11: 79-88+ (March 1982).

Mittra, Sitansu S. "Discrete system simulation concepts," Simulation, 43: 142-144 (September 1984).

Myers, Ware. "Toward a Local Network Standard," IEEE Micro Magazine, 2: 28-45 (August 1982).

Rahimi, Said K. and George D. Jelatis. "LAN Protocol Validation and Evaluation," IEEE Journal On Selected Areas In Communications, SAC-1: 790-802 (November 1983).

Schmidt, J. W. "Introduction To Simulation," Proceedings of the 1984 Winter Simulation Conference. 65-73. IEEE, New York, 1984.

Schruben, Lee. "Modeling Systems Using Discrete Event Simulation," Proceedings of the 1983 Winter Simulation Conference. 101-107. IEEE, New York, 1983.

Shannon, Robert E. "Simulation: An Overview," Proceedings of the 1983 Winter Simulation Conference. 19-22. IEEE, New York, 1983.

Shannon, Robert E. Systems Simulation: The Art and Science. Englewood Cliffs, New Jersey: Prentice-Hall, Inc., 1975.

Society of Automotive Engineers. Proposed Linear Token Passing Bus Media Access Method. Draft SAE Standard. SAE, St. Paul, Minnesota, February, 1985.

Stallings, William. "Local Networks," Computing Surveys, 16: 3-41 (March 1984).

Stallings, William. "Local Network Performance," IEEE Communications Magazine, 22: 27-36 (February 1984).

Stuck, Bart W. "Calculating the Maximum Mean Data Rate in Local Area Networks," IEEE Computer Magazine, 16: 72-76 (May 1983).

Stuck, Bart W. "Which local net bus access is most sensitive to traffic congestion?," Data Communications, 12: 107-120+ (January 1983).

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

Ulug, M. E. "Comparison Of Token Holding Time Strategies For A Static Token Passing Bus," Proceedings of the Computer Networking Symposium. 37-44. IEEE Computer Society Press, Silver Spring, Maryland, 1984.

VITA

James E. Spieth was born on 11 March 1952 in Cleveland, Ohio. He graduated from high school in Strongsville, Ohio in 1970 and attended Ohio Northern University from which he received the degree of Bachelor of Science in Electrical Engineering in May 1975. After graduation, he accepted a position with the USAF's Aeronautical Systems Division (ASD) as an aircraft electrical power system engineer. In 1981, he changed positions within ASD and moved to the Systems Engineering Avionics Facility. He remained there as an avionics systems engineer until entering the School of Engineering, Air Force Institute of Technology, in October 1984.

Permanent Address: 1552 Stormy Court
Xenia, Ohio 45385

UNCLASSIFIED
SECURITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE

1a. REPORT SECURITY CLASSIFICATION
UNCLASSIFIED

1b. RESTRICTIVE MARKINGS

2a. SECURITY CLASSIFICATION AUTHORITY

2b. DECLASSIFICATION/DOWNGRADING SCHEDULE

3. DISTRIBUTION/AVAILABILITY OF REPORT
Approved for public release;
distribution unlimited

4. PERFORMING ORGANIZATION REPORT NUMBER(S)
AFIT/GCS/ENG/85D-15

5. MONITORING ORGANIZATION REPORT NUMBER(S)

6a. NAME OF PERFORMING ORGANIZATION
School of Engineering

6b. OFFICE SYMBOL *(If applicable)*
AFIT/ENG

7a. NAME OF MONITORING ORGANIZATION

6c. ADDRESS *(City, State and ZIP Code)*
Air Force Institute of Technology
Wright-Patterson AFB, Ohio 45433-6583

7b. ADDRESS *(City, State and ZIP Code)*

8a. NAME OF FUNDING/SPONSORING ORGANIZATION
Aeronautical Systems Division

8b. OFFICE SYMBOL *(If applicable)*
ENASF

9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER

8c. ADDRESS *(City, State and ZIP Code)*
ASD/ENASF
Wright-Patterson AFB, Ohio 45433-6503

10. SOURCE OF FUNDING NOS.

PROGRAM ELEMENT NO.	PROJECT NO.	TASK NO.	WORK UNIT NO.

11. TITLE *(Include Security Classification)*
See Box 19

12. PERSONAL AUTHOR(S)
James E. Spieth, B.S.E.E.

13a. TYPE OF REPORT
MS Thesis

13b. TIME COVERED
FROM ________ TO ________

14. DATE OF REPORT *(Yr., Mo., Day)*
1985 December

15. PAGE COUNT
170

16. SUPPLEMENTARY NOTATION

17. COSATI CODES

FIELD	GROUP	SUB. GR.
01	03	

18. SUBJECT TERMS *(Continue on reverse if necessary and identify by block number)*
Avionics, Computer Communications, Digital Simulation, Local Area Networks

19. ABSTRACT *(Continue on reverse if necessary and identify by block number)*

Title: SIMULATION MODEL OF A HIGH-SPEED TOKEN-PASSING BUS FOR AVIONICS APPLICATIONS

Thesis Chairman: Walter D. Seward, Major, USAF
Assistant Professor of Electrical and Computer Engineering

Approved for public release: IAW AFR 190-1/.
Lynn W...
LYNN E. WOLAVER 16 JAN 86
Dean for Research and Professional Development
Air Force Institute of Technology (ATC)
Wright-Patterson AFB OH 45433

20. DISTRIBUTION/AVAILABILITY OF ABSTRACT
UNCLASSIFIED/UNLIMITED ☒ SAME AS RPT. □ DTIC USERS □

21. ABSTRACT SECURITY CLASSIFICATION
UNCLASSIFIED

22a. NAME OF RESPONSIBLE INDIVIDUAL
Walter D. Seward, Major, USAF

22b. TELEPHONE NUMBER *(Include Area Code)*
513 255-3576

22c. OFFICE SYMBOL
AFIT/ENG

EDITION OF 1 JAN 73 IS OBSOLETE.

UNCLASSIFIED
SECURITY CLASSIFICATION OF THIS PAGE

There are many factors of bus token-passing protocols that influence the overall performance of the protocol. Extensive analysis is needed to design a protocol with performance that can meet the requirements for a next-generation aviation electronics (avionics) data bus. The purpose of this thesis was to develop and test a token-passing bus simulation model that could be used to conduct this analysis.

This thesis developed and validated a model for simulating bus token-passing protocols for avionics applications. Two algorithms were designed that reflected the timing and operation of a distributed control token-passing protocol and a centralized control token-passing protocol. The algorithms were incorporated into an overall simulation model program which included simulation control, data collection, and data analysis functions. The simulation model program was written in the Pascal computer programming language.

The simulation model program allows various avionics bus configurations to be defined and tested. A series of tests were conducted using the simulation model program to validate its operation and modeling capabilities. The validation tests were successful. Initial performance tests were conducted for a centralized control token-passing protocol using a bus configuration representative of a fighter-type aircraft bus network. The performance of the two types of protocols was also compared.