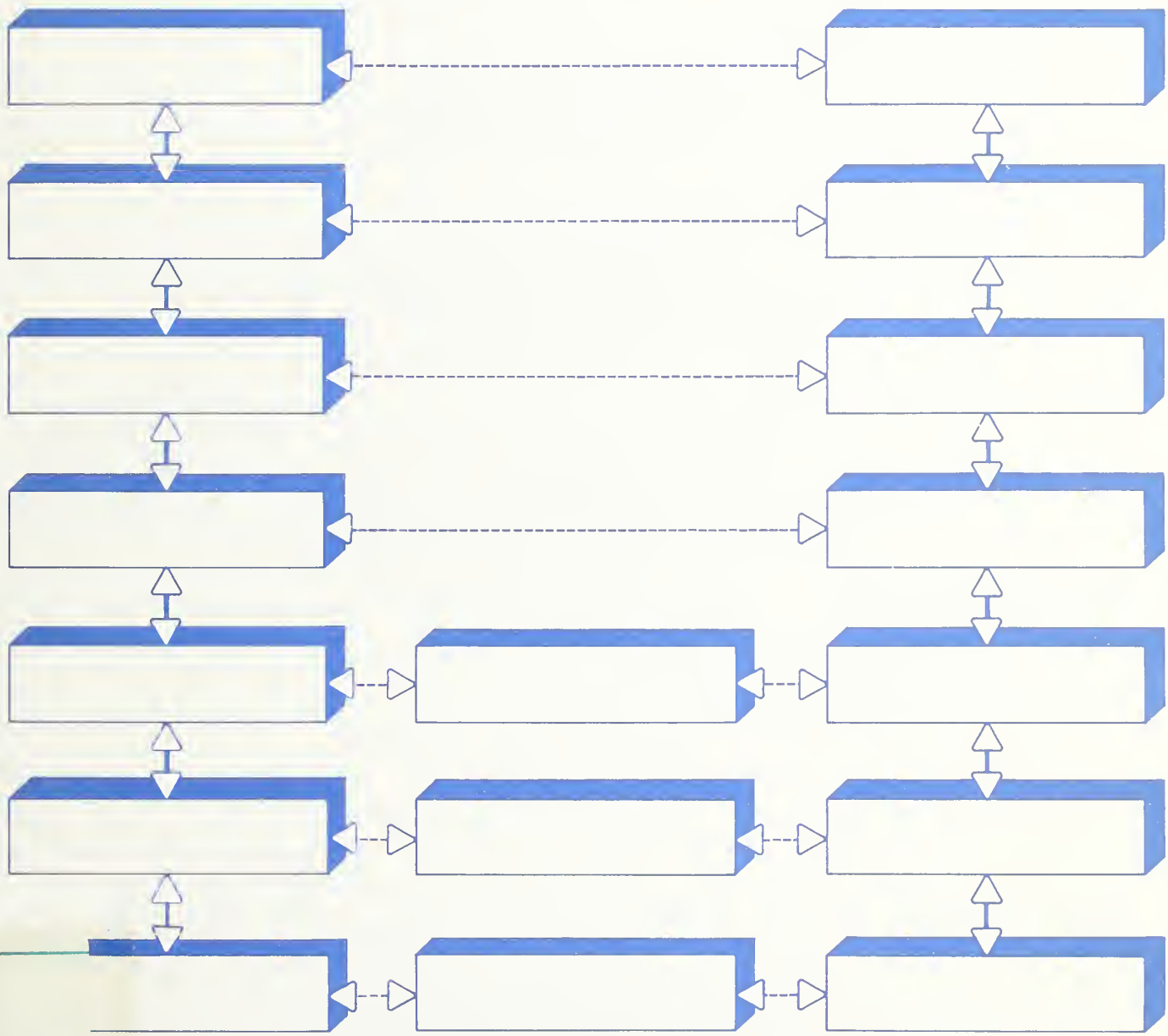
Computer Science and Technology

NBSIR 87-3516

Workshop on Factory Communications

March 17-18, 1987

Robert Rosenthal, Editor



sored by:

The National Bureau of Standards
Institute for Computer Sciences
and Technology

The Institute of Electrical and
Electronic Engineers
Industrial Electronics Society

Computer Science and Technology

NBSIR 87-3516

Workshop on Factory Communications

March 17-18, 1987

Report on

Recent Factory Communications Research in Government, Industry and Academia Including MAP Applications, Simulation and Analytic Modeling Tools

Robert Rosenthal, Editor

Systems and Network Architecture Division
Institute for Computer Sciences and Technology
National Bureau of Standards
Gaithersburg, MD 20899

Cosponsored by the
National Bureau of Standards
Institute for Computer Sciences and Technology
and the
Institute of Electrical and Electronics Engineers
Industrial Electronics Society

Issued March 1987

U.S. DEPARTMENT OF COMMERCE
Malcolm Baldrige, Secretary

National Bureau of Standards
Ernest Ambler, Director

TABLE OF CONTENTS

Acknowledgements

A workshop is only as successful as the participants make it. The opportunity for a successful workshop was made possible by a dedicated committee.

Workshop Chairman:

Dr. Alfred C. Weaver
Department of Computer Science
Thornton Hall
University of Virginia
Charlottesville, Virginia 22903
(713) 333-6792

Technical Program Chairman:

Dr. Juan Pimentel
Electrical and Computer Engineering Department
GMI Engineering and Management Institute
1700 West Third Avenue
Flint, Michigan 48502-2276
(313) 762-7992

Assistant Program Chairman:

Mr. Kevin Mills
Building 225, Room B-217
National Bureau of Standards
Gaithersburg, Maryland 20899
(301) 975-3627

NBS Liaison:

Mr. Robert Rosenthal
Building 225, Room B-217
National Bureau of Standards
Gaithersburg, Maryland 20899
(301) 975-3603

Registration Chairman and Treasurer:

Dr. Scott D. Carson
Computer Science Department
University of Maryland
College Park, Maryland 20742
(301) 454-2002

Special thanks are due to Ms. Mary Lou Fahey, Ms. Sharon Reeves, and Ms. Joan Wyrwa whose secretarial skills and local arrangements expertise made the workshop possible.

TABLE OF CONTENTS

Acknowledgements

A workshop is only as successful as the participants make it. The opportunity for a successful workshop was made possible by a dedicated committee.

Workshop Chairman:

Dr. Alfred C. Weaver
Department of Computer Science
Thornton Hall
University of Virginia
Charlottesville, Virginia 22903
(713) 333-6792

Technical Program Chairman:

Dr. Juan Pimentel
Electrical and Computer Engineering Department
GMI Engineering and Management Institute
1700 West Third Avenue
Flint, Michigan 48502-2276
(313) 762-7992

Assistant Program Chairman:

Mr. Kevin Mills
Building 225, Room B-217
National Bureau of Standards
Gaithersburg, Maryland 20899
(301) 975-3627

NBS Liaison:

Mr. Robert Rosenthal
Building 225, Room B-217
National Bureau of Standards
Gaithersburg, Maryland 20899
(301) 975-3603

Registration Chairman and Treasurer:

Dr. Scott D. Carson
Computer Science Department
University of Maryland
College Park, Maryland 20742
(301) 454-2002

Special thanks are due to Ms. Mary Lou Fahey, Ms. Sharon Reeves, and Ms. Joan Wyrwa whose secretarial skills and local arrangements expertise made the workshop possible.

LETTER FROM THE EDITOR

Factory and laboratory automation requires computer communication networks capable of providing rich connectivity among diverse computer systems, sensors, actuators, robot controllers and other devices. Protocol specifications for networks capable of meeting manufacturing and technical office requirements are evolving nationally and internationally. Networks that implement these specifications enable distributed computer applications within the factory; and, with these distributed computer applications, automated design and manufacturing is possible.

The workshop provides a technical forum for the exchange of information, research results and new applications for factory communication networks. These proceedings report recent efforts of government, industry and academic researchers in four major areas: the application of manufacturing automation protocols (MAP), the application of non-MAP protocols, the use of design and simulation tools and the use of analytic and simulation models.

The Institute for Computer Sciences and Technology at the National Bureau of Standards is pleased to co-sponsor this Workshop on Factory Communications with the Industrial Electronics Society of the Institute of Electrical and Electronics Engineers. The papers presented at the workshop and the workshop discussions represent the views of the authors and not necessarily the views of the NBS or the IEEE.

Sincerely,

Robert Rosenthal
Manager, Network Integration Program
Institute for Computer Sciences and Technology
National Bureau of Standards
Gaithersburg, Maryland, 20899

Non-MAP Factory Networks

Field Bus in the Hierarchy of Factory Communications: the Limits of a Classical Approach

J.-D. Decotignie and P.Pleinevaux

Ecole Polytechnique Fédérale de Lausanne
Laboratoire d'Informatique Technique
16, ch. de Bellerive,
CH-1007 Lausanne
SWITZERLAND

ABSTRACT

Field buses were introduced to replace the traditional wiring of sensors and actuators to programmable controllers. After a brief summary of the advantages presented by field buses , we examine the features of the interconnected devices and the requirements that these networks must fulfill. We then describe Phoebus, a prototype field bus developed for a machine tool. This implementation is compared to other proposals .
In a second part, an architecture based on the OSI reference Model is presented with elements of services and protocols which match the requirements for a machine tool field bus.
Finally, we examine a series of problems which are introduced when several automation equipments share the same field bus.

INTRODUCTION

Manufacturers have been offering industrial networks that interconnect control or management computers, CRT terminals and programmable controllers (PLC). The problem with these implementations is that, in most cases, equipment from different manufacturers cannot be directly interconnected. Specific hardware and software interfaces had to be developed to get around the compatibilty problem, leading to tremendous installation costs.
It has become obvious that costs could be drastically reduced if a standard existed for interconnection of programmable devices. General Motors has encouraged this standardization effort by introducing MAP [GM85].
The initial idea was to interconnect all intelligent devices on a single type of network. However, it quickly became apparent that, at least two types of network were needed. A backbone network that would carry information over the whole plant for management, and a real time network to be used for supervision and coordination of equipment at shop floor or flexible cell level.
Furthermore, it appeared that direct connections between sensors or actuators and the first level of automation (PLC, CNC) could be favourably replaced by a network, the *Field Bus*.
Therefore, factory communications can be represented in a three-layer structure (fig.1).

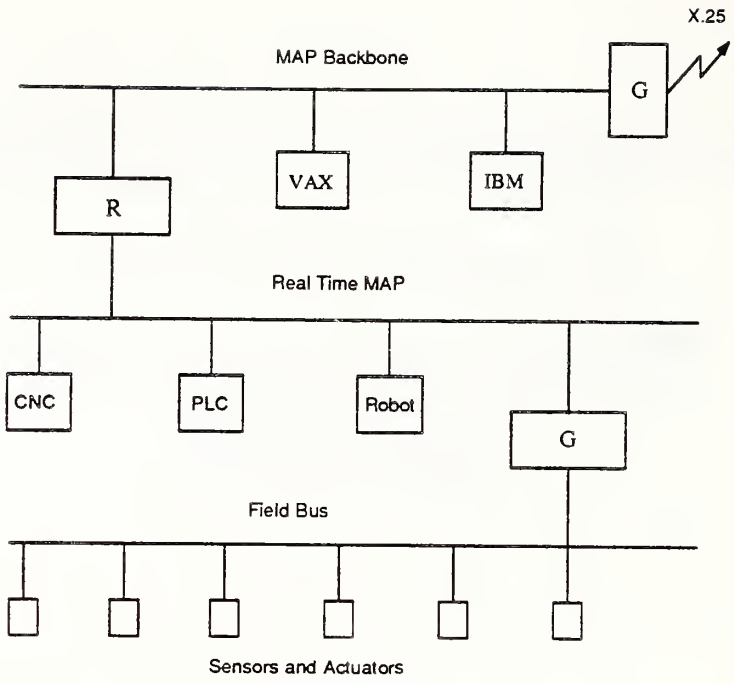


Figure 1: Hierarchy of communications in the factory

THE ADVANTAGES OF FIELD BUS

The replacement of direct connections by a network - the Field Bus - has the following advantages:

- the cabling costs can be drastically reduced.
- installation and maintenance are made easier since these operations involve the manipulation of a unique cable.
- detection and localization of cable faults are very difficult and time consuming in traditional installations; with a network these operations are easily implemented.
- expansion is made easier because of the modular nature of a network.

Furthermore, the use of a network brings new functionalities which were not previously available :

- data consistency for a large number of inputs and outputs. Sampling of inputs or locking of outputs can be synchronized so that PLCs can work on an exact picture of the process at a given time.
- improvement of analog signal quality (S/N ratio); this trend can already be seen as analog transmitters are becoming digital.
- filtering and pre-processing of inputs are possible since each node of the network has processing capabilities.

DEVICE AND TRAFFIC CHARACTERISTICS

The devices considered here are those that are traditionnally connected to the first level of automation (PLC: Programmable Logic Controller, CNC: Computerized Numerical Control). They are:
- Status indicators such as contact closures;
- On/off actuators for pumps, power supplies, lamps...;
- Process sensors, transmitters or transducers for measurement of physical parameters such as temperature, flow, pressure, level... These are usually analog;
- Counters or totalizators;
- "Intelligent" devices such as motor controllers or heat flow computers.

Sensor and actuator characteristics are summarized in Table 1. Devices are rather inexpensive. They transmit or receive small paquets of data (1 to 16 bits). The associated response times range from less than 1ms for control loops, 10ms for events (alarms, end of track...), 50ms for on/off functions, to a few seconds for physical parameters such as temperature, flow, level...

Value	nb.bits	response time	price
Rapidly varying analog values	12	≈1ms	≈$50
Slowly varying analog values	12	≈1s	≈$50
Event type logic inputs	1	≈1ms	$1-5
State logic inputs	1	20-100ms	$1-5
On/off actuators	1	20ms-1s	var.
Counters/totalisators	16	≈1ms	$200

Table 1: Sensors and actuators characteristics

All these values are sampled or updated periodically by cyclic polling. Events which require a short response time can be processed as interruptions or, more often, by short cycles within normal polling cycles.
Transmission errors should be detected but correction is insured by cyclic repetition (report by repetition). Retransmissions would alter the cycle time and should be avoided. In case of error, it is a good policy to assume that states have not changed since last cycle. On the contrary, wrong informations can be very dangerous.
Besides this synchronous traffic, parameters such as set point data or scaling can be sent to "intelligent devices". Similarly, parameters such as reasons for alarms or statistics can be read from some devices. Such messages are typically asynchronous and cannnot be verified by cyclic repetition. This means that retransmission must be allowed since it is needed to insure a correct transmission of the information.

REQUIREMENTS FOR A FIELD BUS

Machine-tool control

For a CNC project under development at EPFL [Gre87], we needed a Field bus that could replace the direct connections between the sensors and actuators usually controlled by the PLC part of the NC system and the PLC itself.
In our particular case, the requirements were:
- constant polling cycle;
- 30 meters in length;
- low attachment cost;

- small size so it could fit inside the machine-tool;
- less than 20 stations connecting less than 100 sensors and less than 100 actuators;
- response time below 20ms;
- no cross-communications;
- stations must be powered remotely;
- good noise immunity.

These requirements were drawn taking into account current characteristics of PLCs in NC systems and correspond to medium applications of field bus as drawn by IEC [Woo86].

IEC proposed requirements

As a comparison, the requirements drawn by the IEC can be summarized as follows:

- Length vs speed
  - 350m (medium distance version)
    - > 150 messages/sec. (process control)
    - < 10000 messages/sec. (manufacturing automation)
  - 40m (short distance version)
    - >5000 messages/sec. (manufacturing automation)
    - maximum possible (robotics)
- 30 stations on the line
- 60 logical addresses + subaddresses
- Response time
  - 5 ms (manufacturing automation)
  - 20 ms (process control)
- No redundancy
- Provision for backup host with mastership
- Cross communication between nodes
- Power through signaling lines (option)

PHOEBUS

As mentioned above, Pheobus is a field bus developed for a project called "Numerical Control for Machine Tools" [Gre87]. It is based on the INTEL 8344 microcontroller. It has a multidrop topology with a single master station and up to 27 secondary stations.

The primary (master) station is composed of an 8344 microcontroller communicating with a PLC through a dual port RAM. This RAM contains the transmission and reception buffers along with the secondary station's descriptors.

Secondary stations have a much simpler architecture: an Intel 8344 microcontroller with program memory (EPROM + RAM) and input/output circuits.

Programming a communications protocol on the 8344 requires the selection of the SIU's response mode. Two modes are available: in the AUTO mode, the SIU implements in hardware a subset of SDLC [IBM79]. In flexible mode, only address recognition and error checking are done in hardware; frame identification, response initiation and all other functions are performed by software.

Phoebus uses the flexible mode and a subset of SDLC protocol for both the primary and the secondary stations. In this mode, the protocol may be simplified. Frame numbering for instance is not implemented in Phoebus since all frames sent by the master - command frames - are implicitly acknowledged by the corresponding response frame. The synchronization frame is the only exception to this rule. Furthermore, we cannot take advantage of the SDLC window mechanism because information is always transfered in one frame.

It can be shown that HDLC and SDLC protocols have a Hamming distance of 1 [Fun82]. To avoid this problem we have chosen to fix the Information field length to 16 bits. One may wonder if a 16 bits CRC is adequate to protect 16 bits of data.

The secondary station driver
The secondary station driver can be described by means of a finite state machine (fig.2). Basically we distinguish two states :

Disconnected state:
The machine enters this state after power up and initialization or after having received a DISC frame. The station remains in this state until it receives an SNRM command from the primary station; it enters transfer state after transmission of a UA response. The station responds to any other frame - apart from UI as will be explained later - with a DM frame.

Transfer state:
Under normal conditions, the secondary station remains in this state, waiting either for an Information frame or a Synchronization frame, UI. Upon reception of a UI, all secondary stations sample their inputs, store them in transmission buffers and lock the new value of their outputs. This value has been received in a previous data exchange in which the secondary station received the new value for its outputs and returned the sampled value of its inputs.

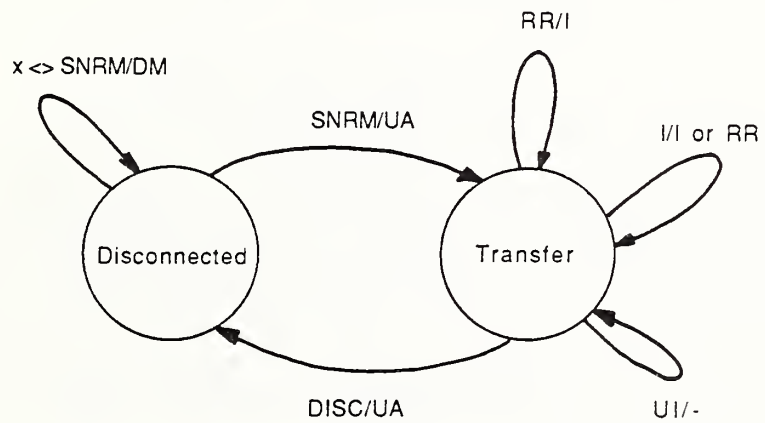


Figure 3: Secondary station automaton

The primary station driver
The primary station is responsible for collecting and distributing data to the secondary stations at regular intervals of time. It issues commands to the secondary stations and receives responses from them. These commands are entirely determined by the state of the secondary stations which is reflected in a data structure called station's descriptor. Such a descriptor contains the following information : (1) the 8 bit address of the station ; (2) the station's status which determines the next command to be sent. Status information is the current station's state and error indicators such as timeout and incorrect information field length ; (3) the number of bytes sent and received from this particular station ; (4) two pointers indicating the position of the sensor and actuator buffers in external RAM.

All this information constituting the secondary stations' descriptors is contained in a dual port RAM which can be accessed at initialization by the PLC. This processor must configure the network, i.e. indicate which stations are to be polled and in which order - reflected by the order of the descriptors in memory. Once initialized, the primary station receives a command indicating that polling can start. It issues a Synchrionization frame to sample the data and starts polling each station according to its descriptor.

Performance tests

Tests were performed to assess various parameters of Phoebus. A critical factor is the station time that we define as the time necessary to load a station's descriptor, exchange two bytes of data in each direction with the secondary station, and update its descriptor. For a transmission speed of 375 kbps, we measured a station time of 770µs. This time corresponds to 470µs of transmission and 300µs of processing time. These figures show that transmission at 2.4 Mbps - six times faster than in the test conditions - would only reduce the station time by a factor of 2. The reason is to be found in the fixed processing time of 300µs corresponding to the load and save operations performed on the station's descriptor, the frame preparation and the response processing.

If we assume now that our network contains 20 secondary stations, each receiving two bytes of information or transmitting two bytes to the primary station, the total cycle time necessary to poll the network would amount to 15.4 milliseconds. This result fulfills the requirements we have set for a machine tool.

EXISTING IMPLEMENTATIONS

The purpose of this section is not to be exhaustive but to present a few representative implementations. Some are commercial products; others are still in the research or development laboratories and may evolve but their essence will remain the same.
The first ideas for a field bus have emerged in the nuclear and military fields, giving birth to CAMAC and MIL-STD 1553 [Hav86]. The instrumentation field contributed the HPIB (IEEE 488, IEC625) bus. Among these three implementations, MIL-STD 1553 is the only one that uses a serial transmission and can meet the requirements for a field bus.
More recently, BITBUS [Int84] and several proposals such as FIP [Tho86], FB FT2 [Fun86-2], PROWAY C, and Phoebus have appeared. All these proposals have a centralized medium access control, except PROWAY C which uses a decentralized token-passing scheme.
Table 2 summarizes the main features of five implementations. The transmission efficiency is measured as the ratio of the useful bits (here 16 bits) over the number of bits required to insure the transmission. The delay between command and response is assumed to be equal to the transmission time of 25 bits [Fun86-1].
All proposals, except BITBUS and Phoebus, use a Manchester encoding technique. BITBUS and Phoebus support two alternative bit signaling methods as mentioned above (Phoebus hardware description).
Except for BITBUS, support is provided for broadcast data transmission without acknowledgement but only MIL-STD 1553 has a built-in mechanism to verify the correct reception of a broadcast frame. This is indicated by a flag in the status field transmitted as a result of a further poll.
In PROWAY C, it is assumed that each active station transmits a single broadcast data frame without acknowledge when it receives the token and then releases it.
In table 2, only the first two solutions meet the requirements of the IEC for a field bus. The other proposals are rather *low end solutions* that can handle continuous process control with a medium response time. BITBUS suffers from two drawbacks; it cannot handle

broadcast messages (required for synchronization purposes) and it does not ensure correct information protection (Hamming distance 1 [Fun82]).

	MIL 1553	FIP	PRO-WAY C	BIT-BUS	PHOE-BUS
SPEED(Mb/s)	1	3	1	0.375	0.375
LOGICAL ADDR	1024	65535	65535	65535	255
STATION NB.	32	100	100	28	32
EFFICIENCY	22%	16%	6.8%	7.4%	11.6%
HAMMING DIS.	2	3	3	1	3
ACCESS TIME [ms]	Nx.065	Nx.035	20	?	Nx.77
BROADCAST	YES	YES	YES	NO	YES
LLC Services	RDR SDA	RDR SDA,N	SDN	RDR SDA	RDR SDA
MAC	Central	Central	Distr.	Central	Central
MULTIMASTER	NO	NO	YES	NO	NO
CROSS COMM.	YES	YES	YES	YES *	NO
PROC.CAPAB.	NO	FUTUR	EXTERN	YES	YES

Table 2: Comparison of several Field Bus Proposals

All solutions, except PROWAY C, implement SDA, RDR and SDN services (see next section). In FIP, all responses can be considered as SDN services. PROWAY makes use of SDN LLC service.

The application area for field buses requires synchronization capabilities. This feature can only be insured by broadcast messages. Furthermore, correct reception of broadcast frames should be provided. Among the proposals that allow broadcast messages, only MIL-STD 1553 allows a verification of the correct reception of these messages.

Reading table 2 shows that the transmission efficiency is low even in the optimistic case presented here (16 bits of information). To keep this relative efficiency, the user is obliged to group on/off sensors and actuators in which case the field bus looses its main advantage. It is then desirable to design field buses that provide an efficiency larger than 1% for single on/off sensor or actuator.

The Open Systems Interconnection (OSI) Reference Model allows the interconnection of heterogeneous computer systems by specifying a series of protocols which must be supported in order to allow interoperation of these devices. Field bus as ordinary computer networks should have a similar architecture, but the basic features of such networks preclude at this moment the use of a full seven layer architecture:

- transmission of sensors's or actuators's states across higher level networks seems unlikely because of the long access times of such networks. Therefore, the network layer is not needed.
- the transport layer should be dropped since there is currently no protocol supporting broadcast and multicast addressing at this layer.
- the session layer, which is intimately bound to the transport layer, is not interesting in the context of field bus.

Consequently, a three layer architecture seems appropriate for a field bus (figure 3).

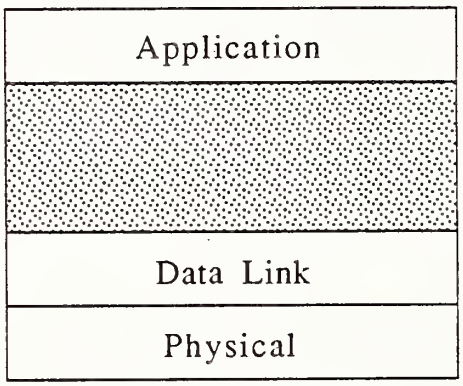


Figure 3: Field Bus Architecture

The physical layer

The following observations can be made on current implementations:

- all implementations have adopted the bus topology.
- Shielded twisted pair is the most commonly used transmission medium, although coaxial cable (FIP, ProControl_P) and fiber optics (FIP) are allowed.
- The number of supported devices is 30 or 32.
- Manchester encoding is supported by MIL STD 1553 and FIP, NRZI by BITBUS and Phoebus.

Medium access control

Classically, medium access control can be performed in a centralized or decentralized form. Decentralized medium access control seems very attractive at first glance, but leads to several drawbacks:

- interfaces are more complex;
- performance is reduced due to token management;
- cycle time cannot be constant;
- each station must know information types and cycle lengths.

On other hand, centralized access control provides important advantages:
- minimum updating time when all process variables change simultaneously;
- high protection against loss of information by using cyclic repetition;
- combination of real time acquisition and restitution of process datas with real time supervision of network topology;
- quantified constant cycle time;
- support of broadcast or multicast messages with no address or with a single address field;
- simple channel access protocol in slave station leading to simpler slave stations.

A major argument against centralized access control is reduced system availability, since the operation of the whole network depends upon a single point. This can be overcomed by powerful redundant techniques for the master station. Furthermore, "democracy" is not always an advantage in the whole system availability [Pow86].

Application layer

The integration of a field bus in a MAP network suggests that the application layer be compatible. MAP specifies MMS (Manufacturing Automation Protocol) as the application layer protocol for manufacturing applications. Interesting functionalities provided by MMS are the remote variable access and event management functions. However, a complete implementation of MMS on a field bus is not realistic because of the size of MMS. Therefore a subset must be defined.

Services and protocol

Timing and addressing constraints impose that a field bus be based on Connectionless services. Three data transfer services are generally introduced [PRO83]: Send Data with Acknowledge (SDA), Send Data with No acknowledge (SDN) and Request Data with Reply (RDR).

The SDA service allows a user to send information to a single remote station (L_DATA_ACK.request). The user receives a confirmation of receipt or non-receipt of the data (L_DATA_ACK.confirm). At the remote station, the information is passed to the remote user along with the address of the sender (L_DATA_ACK.indication). When received by the remote user, this information is known to be correct and identical to the information sent by the source station.

The SDN service allows a user to send information to a single remote station, to a group of remote stations (multicast) or to all remote stations (broadcast).The user receives a confirmation of transmission completion (L_DATA.confirm) but not of data delivery. At the remote station, the information is passed to the remote user if received correctly (L_DATA.indication).

The RDR service allows a user to request information (L_REPLY.request) previously submitted by the user at a remote station (L_REPLY_UPDATE.request). The requesting user receives either the data or an indication that the data was not available (L_REPLY.confirm). The remote user receives an indication that the submitted information has been fetched by the initiator of the service (L_REPLY.indication).

When using a field bus in conjunction with a PLC, a number of constraints must be respected:

1) The typical operation of a PLC is a cyclic one. This cycle can be broken into three phases:
    - acquisition of the sensors' state.
    - processing.
    - update of the actuators' state.

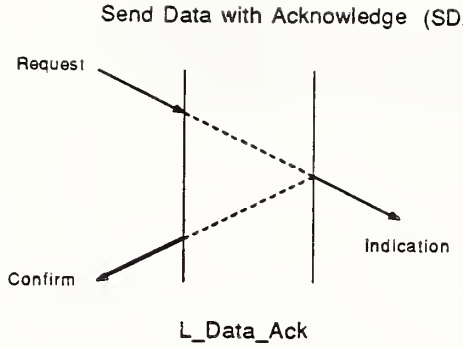


Figure 4

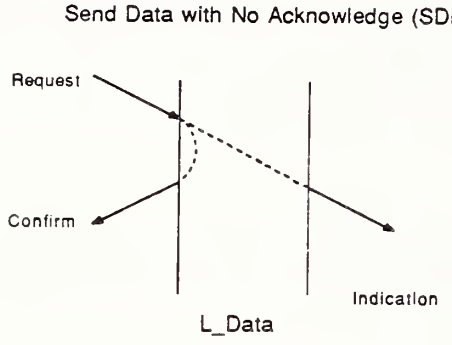


Figure 5

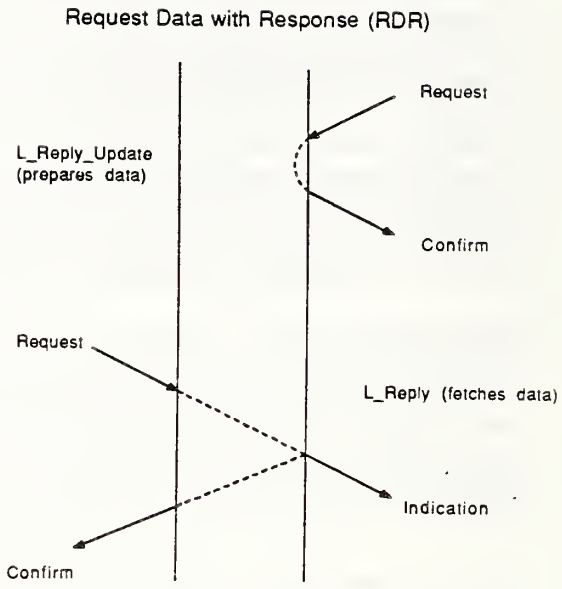


Figure 6

As a consequence, the network protocol is also cyclic, with a fixed period imposed by the PLC. This implies that all data exchanges must be executed in one and only one transaction; retransmissions must be forbidden, since their allowance doubles the maximum cycle time. This does not mean that error detection is not done. Indeed, for a PLC it may be disastrous to work with erroneous data or to update an actuator with a value different from that computed by the PLC. Information must be protected - by a CRC or simple parity, the question is open - errors must be detected and signaled, but their correction need not be immediate.

2) Data flows are from the sensors to the PLC and from the PLC to the actuators. Therefore, a master-slave protocol and a RDR service are a natural approach to this application. This consequence is due to the fact that state evaluations are centralized in the PLC and thus cross-communications between sensors and actuators are in general not desired. However, should the PLC function be distributed among secondary stations, then a peer-to-peer protocol would be a natural approach. A second case in which direct communication between a sensor and an actuator may be desirable is in the case of alarm notifications. An actuator can immediately react to a sensor state

change without waiting for a command coming from the PLC. However, both the PLC and the actuator must be notified, justifying the need for multicast or broadcast addressing.

3) A distinction must be made between synchronous and asynchronous traffic. The synchronous traffic concerns the cyclic acquisition and update of inputs and outputs and constitutes the main part of the traffic. The asynchronous traffic involves all other information transfers: alarms, parameter setting, download of programs, network management. Efficient use of the bandwidth and the cycle period impose that the asynchronous part of the cycle be shorter than the synchronous part. Because of the finite duration of the asynchronous sub-cycle, certain messages may be delayed for one or more cycles, and a natural selection of messages must be made by means of a priority mechanism. SDA and SDN services are appropriate for asynchronous traffic, with the possibility of broadcasting and multicasting in the case of SDN.

4) In order to give the PLC a coherent picture of the process, inputs must be sampled simultaneously. This synchronization is obtained by broadcasting a message on the bus. Acknowledgement of this synchronization command may or may not be grouped. It is obvious that in the case of a ring topology, a group acknowledgement is easy to implement. But for a bus topology, a satisfactory solution does not exist. Therfore the most frequent solution is to acknowledge the command in a subsequent response.

FUTHER PROBLEMS

A simple control system can be characterized by three parameters (figure 7):
- the sample period t_e or cycle time which is the time elapsed between the beginning of a process cycle and the next one;
- the processing time t_p which includes the input state acquisition t_i, the algorithm computation time t_c and the output update time t_o;
- the response time t_r which is the time between a change in an input state and the corresponding change at ouptut as ordered by the control system ($t_r = t_e + t_p = t_e + t_c + t_i + t_o$).

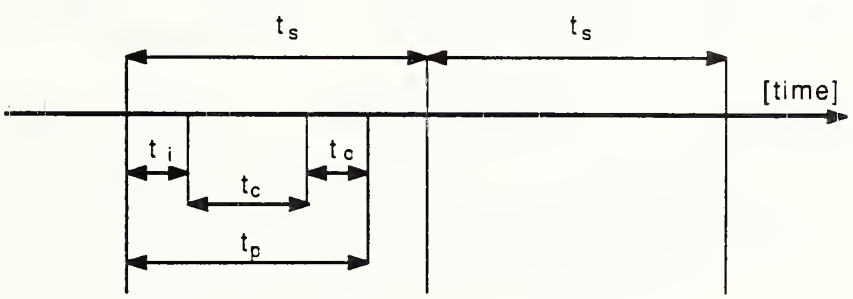


Figure 7: Simple Control System Timing

Replacing direct connections by a field bus will increase the relative importance of t_i and t_o over t_c. Let us look at some problems caused by this replacement.

A simple PLC

When a single automaton is controlling sensors and actuators through a field bus, it may or may not be the master of the network. However, its own sample period dictates the same cycle time for the field bus.
The required time to refresh the sensor state table (t_i) and to update actuators (t_o) may be far less than the sample period. If this is the case, the network is being used well below its capabilities.
On the other hand, if this time ($t_i + t_o$) is comparable with the computation time, the network introduces a large delay in the response time. Several overlapping schemes can be imagined to reduce the performance degradation [Des87]. The excess delay has generally no influence over the control algortithms.

A single motor control loop

In high performance machining applications and for high quality motor control, a sample period around 1ms or below is usually required. This means that $t_i + t_o$ should be well below 1ms and its effect should be taken into account by the control algorithms.
In order to efficiently use the field bus, other kinds of traffic should occur only during the unused period ($t_s - t_i - t_o$).

Multiaxes control

In this case, several control loops are handled via the network. To use the network efficiently, it is necessary to interleave (in time) the control loops (figure 8).
In case of non-interpolated (independant) axes, this case is equivalent to a single axis control loop. But, when axes are co-interpolated, interleaving leads to severe modifications in the curve interpolation algorithms (as well as changes in the control equations) to insure a good synchronization between axes.

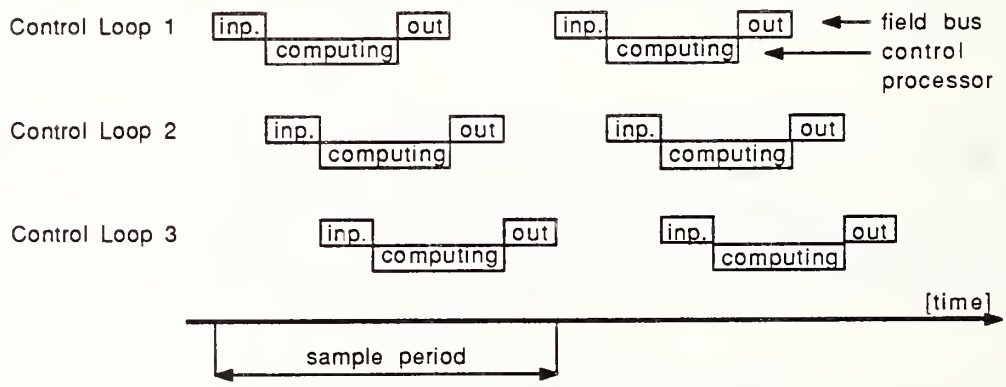


Figure 8: Several control loops on a single field bus

Multi PLCs

When several reflex automation devices run a set of sensors and actuators over a single field bus, their sample periods are generally different. If these periods are constant, the network can be used optimally by externally synchronizing the different automatons. Otherwise, the field bus will introduce another penalty due to simultaneous requests for acquisition or update. The excess delay will not be constant.

Comments

When a single automation device (PLC, control loop, CNC) is linked to its sensors and actuators via a field bus, a single master/multiple slave configuration is an obvious choice. When several reflex automatons are connected to the network, the configuration choice becomes more difficult. However, the problems mentionned above show that, due to the real time constraints, the field bus cannot be made transparent and all cycles have to be designed carefully before implementing the network. In that sense, design and management can be made much easier with a single master station.

CONCLUSION

A conventional master slave protocol like SDLC can be used to design a field bus for medium speed applications, but entails to much unnecessary overhead for high performance process control. This remark also applies to all other proposals when connecting a single on/off sensor or actuator to the field bus.
Care should be taken to minimize the delays due to higher level software that may exceed the transmission time.
It has been shown that the "old" single master/multiple slave configuration exhibits good behaviour in case of hard real time constraints. Furthermore, this configuration is much easier to handle when dealing with complex control applications. In these applications and because of the real time constraints, a field bus cannot be made transparent unless used inefficiently.
Finally, due to the complexity of handling mutliple scan cycles, it becomes necessary to provide CAD tools for building the network operations.

ACKNOWLEDGEMENT

This project was performed under grant of the EPFL inter-laboratory project "Commande Numérique de machines". The authors would like to thank N.M.Greene for her help in writing this paper.

REFERENCES

[Des87] D.Desmons, J.-D.Decotignie, "Field Bus in the Numerical Control of Machine-Tools: How to Get the Best of It", presented at the International Workshop on Industrial Automation Systems, Tokyo, Japan, February 4-6, 1987.

[Fun82] G.Funk, "Message Error Detecting Properties of HDLC Protocols", IEEE Trans.Comm., COM-30, pp.252-7, December 1982.

[Fun86-1] G.Funk, "FB-FT2", IEC SC65 (Funk) 19.

[Fun86-2] G.Funk, "Alternative Field Bus Solutions", IEC SC65 (Funk) 20, March 1986.

[GM85] General Motors Co, "MAP specifications 2.1", March 1985.

[Gre87] J.C.Gregoire, "A New Perspective on the Architecture of Numerical Control", presented at the International Workshop on Industrial Automation Systems, Tokyo, Japan, February 4-6, 1987.

[Hav86] N.Haverty, "MIL-STD 1553 - a standard for data communications", Communication & Broadcasting, Vol.10 (1), pp.29-33, January 1986.

[IBM79] IBM, "IBM Synchronous Data Link Control: General Information", document GA27-3093-2, Research Triangle Park, North Carolina, USA, 1979.

[Int84] INTEL Corp.," The Bitbus Interconnect Serial Control Bus Specifications", in Distributed Control Modules, 1984.

[Pow86] D.R.Powell, "Dependable Architectures for Real Time Local Area Networks", presented at the Advanced Seminar on Real-Time Local Area Networks, Bandol, France, April 16-18, 1986.

[PRO83] PROWAY LAN Study Document Revision B, October 1983.

[Tho86] J.P.Thomesse et al., "An Industrial Instrumentation Local Area Network", presented at IECON 86, Milwaukee, WI, September 29- October 3, 1986.

[Woo86] G.C.Wood, "Field Bus, a Developing Low Level Industrial LAN Standard", presented at EFOC/LAN, Amsterdam, Netherlands, June 23-27, 1986.

A Communication Protocol for Advanced Automation LAN (AALAN)

Lung-Sing Liang

ISDN Group, Telecommunication Laboratories,
Directorate General of Telecommunications,
Ministry of Communications, PO Box 71,
Chung-Li, Taiwan

ABSTRACT

A local area network used for advanced automation functions is described. These functions include: office automation, computer aided design/computer aided manufacturing, a robotic system, image processing, remote teaching and expert systems. The design and implementation of the layered architecture of AALAN and its relationship with the existing protocols, such as the transmission control/internet protocols (TCP/IP) and the Xerox Network System (XNS) are also described.

Keywords: computer networks, local area networks, advanced automation, transmission control/internet protocols

1. INTRODUCTION

With the advent of computer hardware technology, multifunction workstations[1] and the multimicroprocessor based computing systems[2] have evolved considerably, while the cost has correspondingly decreased. The integration of these workstations and computers to perform advanced automation, i.e. OA, CAD/CAM, robotics, image processing, remote teaching and expert systems, are becoming more urgent day by day.

The basic components of an advanced automation system will include a message management, a data management and an activity management system. These will be integrated together in a communication network[3]. A LAN differs from a conventional 'long-haul' communication network in its communication bandwidth, communication protocols and architecture. Usually, in long-haul networks the transmission bandwidth is an expensive resource and network design emphasizes communication link efficiency with complicated routing, flow control, network administration and monitoring facilities.

Unlike long-haul networks, the design philosophy of a LAN is quite different because of its high communication bandwidth, low transmission delay and low transmission error rate. In a local environment, the LAN will connect all the terminals, host computers, intelligent workstations, computer aided engineering (CAE) stations and CAD/CAM stations together. Therefore the LAN communication protocols will differ considerably from those in a long-haul network, based upon the LAN's differing technology and application environment.

The application environment will include an office information system, a robotic system, a distributed database management system (DDBMS) and a network application system. The interconnection of these stations will be difficult because of the heterogeneous hardware and software environment. A homogeneous communication system must be developed to provide the communication service necessary for advanced automation.

2. NETWORK CONFIGURATION

The proposed network architecture is shown in Figure 1. It includes PDP11/70, PDP11/34, AT&T 3B2s, Xerox 1108s, Ritge and Intel Microcomputer Developing System (MDS). Most of these devices will support the Ethernet interface without the device driver. The Xerox 1108 workstation will be dominated by the Xerox Networking System (XNS) protocol. The three VAX-11 computers are networked by DECnet running on top of an Ethernet.

The robotic network consists of five workstations, each of which consists of a Z-100 personal computer and one Hero robot. It is a loop structure with one station as the master station connected to the Ethernet. An AT&T PC 6300 is connected to one of the AT&T 3B2s via the kermit file transfer protocol[4] allowing remote teaching activity. There is an image processing system developed on the PDP11/70 and AT&T PC 6300. An office information system, IMMS[5], is developed on the PDP11/70, 3B2s and Xerox 1108s. The distributed database management system (DDBMS) is being developed on the INTEL MDS.

Due to its popularity and the existing hardware consideration, Ethernet was chosen for the physical and link layers[6]. By employing a repeater, this departmental LAN can be connected to the campus VAX DECnet/Ethernet network approximately 1 km away.

3. LAYERED ARCHITECTURE

The network architecture consists of a set of layer definitions and their protocol implementation. There are five individual layers 1-5. These are the link, network, transport, presentation and application layers. The definition and function of each layer will be described later. The interface between each layer will be known as the interprocess communication. The transmission control/internet protocols (TCP/IP)[7,8] and the XNS protocol have obtained wide acceptance from LAN and office automation vendors. The TCP/IP is a standard communication protocol running under Unix BSD 4.x. All Xerox office products are based upon PARC Universal Packet (PUP) or XNS protocols. Some other companies have also implemented the XNS protocol on top of Ethernet[9].

Figure 2 compares the XNS, TCP/IP and AALAN with the corresponding ISO Open System Interconnection Reference Model Levels 1, 2 and part of level 3 (the physical, data link and intranetwork layers), are equivalent to the XNS level 0. The XNS level 1 corresponds to layer 3b of the ISO network layer (the internetwork layer). The internet protocol/internet control message protocol (IP/ICMP) or internet datagram protocol (IDP) defines the format of an internet datagram. The IDP addresses, routes and delivers internet packets inside the network. In most

datagram protocols, the IDP does not guarantee reliability. It may lose a packet, deliver a packet which is out of sequence or even duplicate a packet. The IP and ICMP of the TCP/IP protocol is much more complicated than the corresponding IDP of the XNS protocol. In the advanced automation application, the IDP is able to interface with Ethernet. The XNS layer 2 or TCP is equivalent to ISO layer 4 — the transport layer. In the XNS layer 2[10], there are five individual subprotocols, namely, Sequence Packet Protocol (SPP), Packet Exchange Protocol (PXP), Error Handling Protocol (ERP), Echo Protocol (ECHO) and Routing Information Protocol (RIP). The XNS approach has been adopted because of its modular design scheme. In layer six, the Telnet[11], FTP[12], MAIL[13], TFIP[14] protocols of the TCP/IP are used. In layer 6 of the AALAN we have the following functions: SND, RCV, LST, DEL, SHL and LOGIN for AALAN.

These six functions will be described in more detail later. The following functions are based on the presentation layer of AALAN: (Message Management System), ESMH (Expert System for Message Handling) and some other applications, such as, DDBMS, remote teaching and videoconferencing systems in the application layer.

The software structure is shown in Figure 3. The following modules are run under Unix; the Ethernet device driver, IDP handler, PXP handler, error packet handler and time packet handler. This will briefly be described in the following sections.

Ethernet device driver

The DEC 'Deuna' Ethernet and the AT&T 3BNET Ethernet controller are used for the PDP11/70 and 3B2 respectively. The Deuna device driver (for the PDP11/70) is written in C[15]. Control of the deuna driver is via the open and close system command (to initiate and terminate the network operation), the write and read system command (to send and receive the data packet), the gtty and stty system command (to query and modify Ethernet mode and status variables). The 3B2 supermicrocomputer allows the user to develop high layer protocols, other than 3BNET, to interface with the Ethernet device driver[16]. The 3BNET supports the direct access to the Ethernet data link layer protocol. The application interface to the network is provided via a subset of the network interface (NI), the driver–user interface. The high layer protocol interface to 3BNET is also provided by the system commands such as open, close, read, write and ioctl. The ioctl command is used to get and set the parameters of the network control block.

Internet datagram protocol handler

The function of the internet datagram protocol (IDP) is to address, route and deliver internet packets in the network. The IDP consists of two parts: control and data. The control part consists of transmission control, destination address and source address. There are four fields in the transmission control: checksum, length, transport control and packet type. The checksum is a software checksum, and is in addition to the FCS error checking mechanism provided by the Ethernet link layer protocol. The checksum is calculated by add-and-left-rotate of all the 16-bit words of the internet packet except the checksum itself. It is assumed that the main function of this checksum is as an internal verification between the communication processor and the memory. Sources of the internet packets which do not want to generate the checksum and are willing to tolerate a lower communication reliability may fill the checksum field with all 'ones' (65535 in decimal notation). This value indicates that no checksum is calculated. The checksum will be recalculated when any content of the packet is altered during the routing of this packet. One byte is used for transport control. The 0–3 bit should be zeroed and the 4–7 bit is the hop count. The hop count is used by the internet router and initially should be set to zero. Each time an internet packet is rerouted, this field will be increased by one and the software checksum will be recalculated. When the hop count exceeds 16, this packet will be discarded to avoid the infinite looping of an internet packet. An error packet will also be transmitted according to the error packet protocol. The length field is the total length of the internet packet measured in bytes including the checksum field. In general, the maximum length of the internet packet is 576 byte, which includes a 30 byte header and 546 byte of data. Identification numbers are used for the network (32 bits), host (48 bits) and socket (16 bits) for source and destination station respectively. A socket identification number is the process number used for communication. Other types of socket number are well known. The packet type field is used to identify the format of the data field of the internet packet. It is also the bridge to the higher layer protocols. There are six different types used in the Xerox network system: routing information, echo, error, packet exchange, sequenced packet and experiment packet.

Packet exchange protocol handler

The packet exchange protocol (PXP) is used to transmit a request packet and receive a response with a reliability higher than that obtained by directly transmitting a datagram, and lower than that by using sequence packet protocol (SPP). The PXP is a simple one packet protocol. The PXP will retransmit a request packet without duplicate detection when the request station is unable to acquire the response packet. The ID field in PXP is a 32-bit field used to cross check between the client and the server. If the ID is the same for both the requestor and replier, then the request–response packet pair completes the transaction. The number for the ID is a monotonically increasing number. The client type field contains the registered value which identifies the client of the protocol.

Error protocol

The error packet handler (ERP) enables any station in the network to recognize the occurrence of an error condition. The ERP can also be used as a diagnostic tool and as a mechanism for improving the network performance. An error protocol packet is returned from the destination socket of the host detecting the error. This packet is sent to the source socket of the packet when an error occurs. The contents of the error packet are two 16-bit words, indicating the error number and the error parameter followed by at least 42 byte of the offending packet. The sending socket can use the received error packet to carry out a check on itself. The error number identifies the error type, e.g. checksum incorrect, specified socket non-existent, packet too large to be forwarded etc. The error parameter is only specified for certain types of error. An error protocol will not respond to a broadcast or multicast packet to prevent the generation of a large number of unnecessary error packets.

Time packet handler

The time packet handler is used to form a time request packet according to the format shown in Figure 4. It also disassembles the time acknowledgement packet to obtain the time information.

4. AALAN HARDWARE

The hardware for AALAN is a microprocessor based

controller implementing the Ethernet protocol, for example: Deuna (Unibus), Excelan (Unibus, Multibus), Ungermann-Bass (Ridge), 3BNet controller (3B2 supermicrocomputer) and Xerox Ethernet controller (Xerox 1108 workstation). Some of the stations allow the user to develop his own high layer protocol other than their standard protocol such as 3BNet, XNS, PUP etc. The device driver is developed with Deuna for PDP11/70 running PWB/Unix.

5 .AALAN PRESENTATION LAYER

The presentation and application layer protocols are implemented on a PDP11/70, a 3B2 and a Xerox 1108 (Lisp machine). There are six presentation layer functions. From the design view point of AALAN, each station has a 'general server' and a 'client' as shown in Figure 5. The general server consists of a file server, an echo server, a command execution server, time server etc. The 'client' part includes the function 'SND', 'RCV', 'DEL', 'LST', 'SHL' and 'LOGIN'. The server is always in the active state ready to receive the service request from the 'client' part of another station. Communication only occurs between the client and the remote station; there is no direct communication link between the 'client' and the 'server' in the same local station. The 'server' and 'client' employ the services of the lower layer communication protocols. Six different patterns are used to identify the corresponding protocols. We can assume that each 'client' is an application process or is controlled by another application process. Once the 'client' issues a specific call request, the general server will classify the request according to the request type, socket number and the presentation protocol pattern. Depending on the request class, one of five different actions may be taken. These are time server, echo server, login server, shell server and file server. The file server will include the 'SND', 'RCV', 'DEL' and 'LST' functions.

Any incorrect format encountered during the request and acknowledgement phases between the 'client' and 'server' will invoke the error protocol to issue an error packet. The time server and echo server will be described in later papers. This paper will focus on the file server, shell server and the login server, which are written in C language and the interlisp D programming language.

The corresponding functions will be described in the following sections.

SND

syntax

SND [station name]

The 'SND' function can be used by an interactive user directly or interface with an application program. The following is an example of how to send a file to a remote host (PDP11/70) station from a local 3B2 supermicrocomputer.

```
%SND PDP11-70
LOGIN: LIANG
PASSWORD: ++++++++
PLEASE INPUT SOURCE FILENAME?
: LANMANUAL
PLEASE INPUT DESTINATION FILENAME?
: LANMANUAL.1
FILE LANMANUAL.1 COPIED
MORE FILE TO SEND (Y OR N)?N
%
```

The whole directory can also be transferred using the pdpsnd(3bsnd) shell language procedure. The login security check is also provided to ensure that the user has the access right to the file.

RCV

syntax

RCV [station name]

The 'RCV' function is used to receive a file from a remote station. The 'LST' function may be used to examine the directory at the remote station to make sure that the file you expect to receive actually exists.

The operation of 'RCV' is very similar to the 'SND' function. The login security checking is also provided. The 'PDPRCV' (3brcv) can also be used to carry out a remote directory copy.

DEL

syntax

DEL [station name]

A file in the remote station can be removed by the 'DEL' function. If file does not exist, then a 'no such file' error-message will be returned to the user.

In order to make the correct deletion, the 'LST' function may be used (see earlier).

SHL

syntax

SHL [station name] 'command string'

A local procedure may be submitted from the local station. This function may easily be interfaced to some application program. The 'command string' can be any command text or a series of commands up to 512 byte. Regardless of whether the command is executed or not, the results will not be passed back to the station which issued the command. The following is an example of how to issue a 'mail' request and 'print out' a hardcopy of a file, from the 3B2 station to the PDP11/70 station.

```
SHL PDP11-70 'MAIL CHANG < LET1;
PR LANMANUAL|OPR'.
```

The first part is to mail a file named 'LET1' to the login user 'CHANG' and the 2nd part is to copy the file 'LANMANUAL' to the line printer. The login security check is also implemented.

LST

syntax

LST [station name]

This function is designed to display the contents of a directory at a remote station. Before a file transfer or a file deletion, the 'LST' command should be used to check the remote directory.

This command can also be used to replace the 'SHL' function. The execution results will be returned to the station which issued the command originally e.g. LST 3B2 'CAT LANMANUAL'.

We can also check the users currently logged in on the 3B2 from the PDP11/70 by issuing the command LST 3B2 'WHO'.

In short, the 'LST' function is the most general function and can be easily adapted to interface with many application programs. As with the other functions security checking is also implemented.

Other software modules

There are some other software modules, such as a time server, an echo server, an SPP (Sequence Packet Protocol) and other transport layer protocols. (The time server and echo server will be described in other papers.) The SPP provides the reliable transmission of successive internet packets with sequence control, window size flow control,

ID of process and a scheme to avoid duplicated packets. The command patterns for 'SND', 'RCV', 'LST', 'DEL', 'SHL' and 'LOGIN' are shown as follows:

	REQUEST	ACK	NAK
SND	sssssssss	sssssssss	nnnnnnnnn
RCV	rrrrrrrr	rrrrrrrr	nnnnnnnnn
LST	llllllllll	llllllllll	nnnnnnnnn
DEL	ddddddddd	ddddddddd	nnnnnnnnn
SHL	hhhhhhhhhh	hhhhhhhhhh	nnnnnnnnn
LOGIN	gggggggggg	gggggggggg	nnnnnnnnn

The AALAN will support a user-friendly interface. The user can interface with AALAN using either an interactive mode or through program handshaking. For example, in MMS the query message is broadcast to multiple stations. The query message can be flexible enough to be in the form of a file or a remote executable command. The respond message will be obtained via the LST or SHL functions. Similarly, in a distributed ruled-based ESMH system the user condition message will be processed by the SND and RCV functions. From the design view point of user application systems, benefits can be obtained from the flexible presentation layer interface.

6. CONCLUSION

The high-level protocols such as XNS, TCP/IP and AALAN will be widely used on different kinds of LANs for various applications. The design considerations will be concentrated on modularity, simplicity, portability, ease of interface etc. All these high layer protocols in AALAN are written in C. In the near future, these protocols will be built as utility software under a different operating system.

Further enhancement of existing functions and the introduction of new functions into the AALAN may be easily accomplished. This is due to its modular design.

It is possible to predict that portable, compact and modularly designed high level protocols for LANs will become more popular.

REFERENCES

1 *AT & T Personal Computer 7300 user's manual* AT & T Technologies, USA (1984)
2 **Liu, A C** *A testbed for distributed processing (TED)* ECE Department, IIT, Chicago, USA (1985)
3 **Chang, S K** *Office information system design* ECE Department IIT, Chicago, USA (1985)
4 **Cruz, F D** *Kermit protocol manual* Columbia University Center for Computing Activities, New York, USA (April 1984)
5 **Chang, S K** *et al. An intelligent* message management *system* Global Telecom, New Orleans USA (December 1985)
6 **Krueger, V** *Focus conference: opportunity in the US telecommunication market* Nielsen dataquest (October 1985)
7 **Postel, J** *DARPA internet protocol specification* ISC, USC RFC-791 (September 1981) p 47-100
8 **Postel, J** 'DARPA transmission control protocol specification' ISC, USC RFC-793 (September 1981) p 131-221
9 **Lien, M** *Experience in implementing XNS protocols* Bridge Communications, USA
10 *Internet transport protocol* Xerox system standard, Xerox Corp., USA (December 1981)
11 **Postel, J** *Telnet protocol specification* ISC, USC RFC-764 (June 1980) p 225-239
12 **Postel, J** *File transfer protocol* ISC, USC RFC-765 (June 1980) p 243-312
13 **Sluizer, K and Postel, J** *Mail transfer protocol* ISC, USC RFC-785 (May 1981)
14 **Sollins, K R** *Trivial file transfer protocol* MIT, USA (June 1981) p 379-396
15 **Draizen, H** *The user's manual of DEUNA* ISL ECE Department IIT, Chicago, USA (1985)
16 *AT & T 3B2 Computer Unix System 5.2,3BNET application interface manual* AT & T Technologies, USA (1984)

ISO reference level	XNS level	XNS protocol	TCP/IP protocol	AALAN level	AALAN
7	4	PRINT, MAIL FILE CLEARING HOUSE	Telnet FIP, MAIL TFIP	5	DDBMS, MMS ESMH
6	3	COURIER TIME SERVER		4	SND, RCV, LST DEL, SHL, LOGIN
5					
4	2	RIP, ECHO ERROR, PXP SPP	TCP/UDP	3	RIP, ECHO ERROR, PXP SPP
3b	1	IDP	IP & ICMP	2	IDP
3a	0	.Ethernet .X.25 .DDX .TI .LEASED		1	Ethernet
2					
1		LINE			

Figure 2. Protocol relationships comparing XNS, TCP/IP and AALAN with the corresponding ISO Reference Models

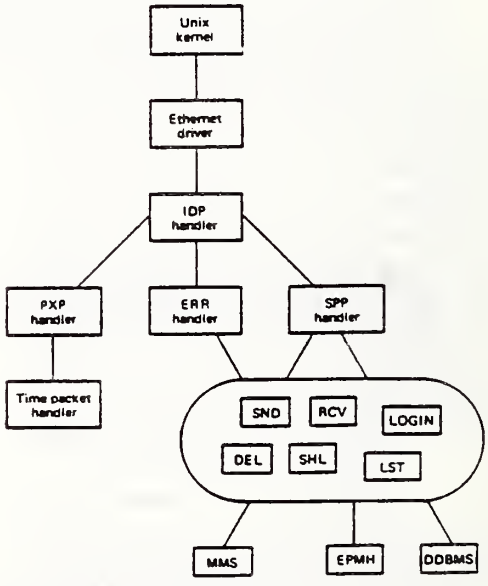


Figure 3. AALAN software structure

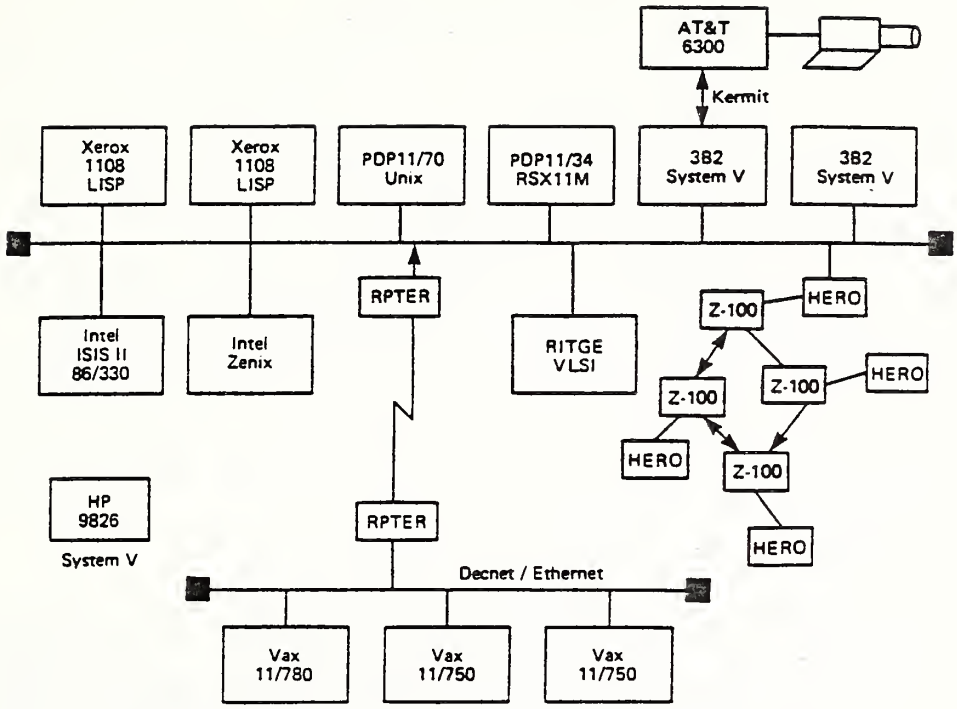


Figure 1. Network architecture of AALAN

Destination address	
Source address	
Type	
Checksum	
Transport control	PXP
Destination network	
Destination host	
Destination socket	
Source network	
Source host	
Source socket	
Identification field	
Client type	

Figure 4. Format of time request packet

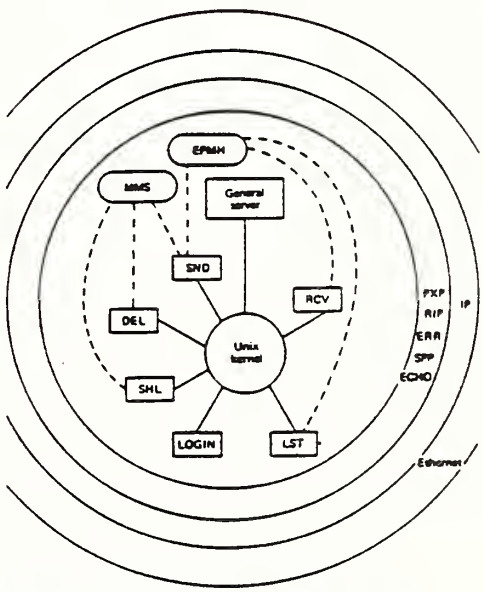


Figure 5. Relationship between client and server

LOCAL AREA NETWORKS FOR

AN INDUSTRIAL AUTOMATION SYSTEM

Leo Sintonen

Tampere University of Technology
Computer Systems Laboratory
P.O.B. 527, 33101 Tampere
Finland

Abstract

A modular robotics system is defined. The communication structure is described and discussed. It is found that by the use of modern high-speed local area network technology together with a suitable system architecture a modular system can be built and reasonable real-time processing requirements can be met.

Contents

1. INTRODUCTION

The purpose of the project is to study the design of a flexible and modular industrial automation system that can be used for assembly, manipulation and production automation. The communication architecture is defined and described.

The system consists of modules and a central controlling computer, which are connected together by a high-speed bus. Modules which are programmable components for production, manipulation or measurement, can have internal communication subsystems. A part of this project is to study the communication requirements and communication architecture.

This paper describes the basic system architecture in chapters 2 and 3. Communication is described in chaptes 4 and 5. A module subsystem communication-protocol is suggested and described in chapter 6.

2. SYSTEM ARCHITECTURE

An industrial automation system in this context is a system used for automated manufacturing, assembly, transportation etc. in a factory scale.

The system is modular, allowing change of manufactoring programs. The system is controlled by a central computing facility. System modules are different kinds of machines needed to perform the process, fig.1. Typically there are numerically controlled machines (NC:s), robots, servo systems, transducers, manipulators e.t.c. These machines are connected to the central computing system by the communication facility, which is a bus-type local area network system.

The system of fig.1. performs a system process which consists of the processes of the individual machines, controlled by the central computer.

The computing structure is distributed. The system task given to the system (from the outside world, from CAD/CAM -system) is divided into subtasks distributed to the computing submodules of the system. Computing submodules are robot controllers, servo controllers or in general controller modules existing in different machines, or measurement on alarm data collection units.

In order to perform the distributed system task modules communicate with other modules and with the central computer via the system bus.

3. SOFTWARE STRUCTURE

The computing submodule can be of type controller or of type interface, depending on the complexity of the task it should handle.

Functions of the type controller are:

- to implement the control algorithm needed to perform the module subtask
- to maintain communication to the system bus
- to handle the communication inside the communication subsystem of the module (internal communication)

All this is performed under the control of the submodule executive.

Computing submodules of type interface are used to collect simple numerical data, e.g. measurement values (position, velocity, force, e.t.c.) or more sophisticated data, like picture data, and send it to a computing module via the system bus. Also, simple commands and valves coming from the system bus are transmitted through this interface to the actuators. The function of the computing module of the type interface are:

- to implement the communication interface to the system bus
- to handle the data and command representation
- to transmit data and/or commands from and to the actual device

4. COMMUNICATION ARCHITECTURE

During the configuration phase, the central computer downloads program to the memory of computing subsystems. A part of this configuration phase may also be the learning phase of robots. Information is transferred mainly via the system bus from/to the central computer, but also during the learning phase in the robot's communication subsystem.

During the operational phase the basic communication is within a module. This reduces the communication load on the system bus.

Module - to - module communication is basicly synchronization information during the operational phase caused by different event conditions.

Status, alarm and command messages are exchanged between the control computer and modules.

Numerical data, video, digitized picture and voice transmission may be needed in some cases between the central computer and modules.

Communication levels

The levels of the communication architecture with reference to the OSI- model are depicted in fig. 2.

At the representation level there is an agreement of the representation of data values, e.g. floating point format and other, in order to make possible to use diverse equipment in the system.

There is also a harmonization of the basic module command set. This set includes commands to operate the system from the central computer. Examples from this set are:

- Run — Initiate the predetermined sequence (machine dependent).
- Stop — Stop the operation. Can be restarted only by Run.
- Break — Abort the sequence. Can be restarted by Continue. For abnormal conditions.
- Continue — Continues the sequence from the previous Break-point.
- Home — Go to predefined settings.

Communication between communication software modules in different computing subsystems is based on logical channels between these modules. Logical channels are established during the configuration phase and remain fixed through the operational phase.

At the bus level, there is intention to use to some standard industrial protocol, like IEEE 802.2 at the LLC-level and IEEE 802.4 at the MAC-and physical level /1/. The LLC-level is connectionless type in order to avoid overlapping connections between same program entities.

This arrangement means that in many cases the communicating entity can be identified by simple address, which is the same as the bus station address, or by a bus address + a generic name. This arrangement is different from the OSI- model, and the purpose is to reduce the communication overhead.

At the physical level the choice of the protocol is dictated by the factors of the deviced response time and the environmental electromagnetic disturbance conditions.

Broadband communications should be, at least theoretically, less sensitive to noise than levelband communications. In addition broadband system makes possible to use multiple channels in the same medium. This feature can be used to reduce transfer delay under heavy load conditions. There is no protocol standard yet suggesting this type of operation.

5. MESSAGES AND TRANSFER DELAYS

Typical messages during the operational phase are quite short. Long messages occur during the program downloading phase, but the response time is not critical in that operation.

The header and other overhead belonging to a link-level frame is typically 10 to 20 bytes. It can be estimated that the overhead of embedded higher-level protocols is of the same magnitude. Thus, the total overhead in a message is 20 to 40 bytes.

Numerical data value e.g. in the floating-point format occupies 4 bytes. A message containing e.g. values for ten parameters would also take 40 bytes. It is obvious that the average message length will be in the order of one hundred bytes. Visual (picture) information messages are considerably longer. A typical bit map memory has size of 2 M bytes.

The transfer delay of the token-passing type of operation has been analyzed widely, see e.g. references in /2/. The transfer delay including queueing delays, is given by the expression.

$$T = \tau + \frac{n \cdot w}{2} + \frac{n \cdot w \cdot (1-\rho)}{2(1-\rho)} + \frac{\rho \cdot \tau}{1-\rho} \quad ;$$

where τ = average message transmission time, n = number of stations, w = 'walking' time of token (per station), $\rho = n \cdot \lambda \cdot \tau$, λ = messages/s per station.

Sample values for 1 Mbps, 5 Mbps and 10 Mbps bit rates are given in fig. 3. Parameters selected are for a typical task profile derived form the application of the system. Results show, that e.g. for a 5 Mbps system with 20 messages/s per station the average delay is approximately 0,6 ms.

It is rather difficult to approximate the delay of higher levels. It depends of the protocol level and of the scheduling of the program and the processing time of the computer. Practical experience and related simulation results /3/ have shown that the response time of higher layers is considerable longer than that of the bus itself.

The time delay caused by the higher levels can be reduced by designing the protocols such that the implementation is rather simple. This can be done by rejecting some services defined by the OSI-model but not relevant in this type of application. This should result in simpler and faster buffer management and reduced intertask communication, thus reducing the total running time of the program.

If the central computing unit is used to sample the values of some modules of the systems, it is obvious, that the maximum workload will be in this processor. The maximum usable sampling rate is thus dependent of the processing capacity of the central processor, and not of the communication bus. With the approximation above the sampling rate will be in the order of 10 to 100 samples/s.

Transmission of picture information takes considerably longer time. E.g. transmission of contents of 1 Mbyte picture memory takes 1,6 seconds at the 5 Mbps bit rate. By the use of compression techniques this time can be shortened. It is obvious, that in order not to lengthen the bus response time considerably the picture information should be transmitted in a separate channel whether in digital or in analog form.

6. COMMUNICATION SUBSYSTEM

The task of the communication subsystem is to connect different devices of the cell, transducers, intelligent sensors and actuators e.t.c. to the controlling processor.

Communication is typically between the control processor and a device (= substation). Typically there is no need for communication between substations. The control processor inputs are measurement values, positional data e.t.c. and the processor sends basically commands, and/or numerical data.

The number of different devices in a cell is in the order of ten, and the distance between a device and the control processor is in the range of 0,1 to 10 meters.

The communication protocol was designed to be as simple as possible but still effective and reliable. The implementation is based on existing microprocessor serial communication VLSI which makes the solution also cost-effective.

The architecture of the communication subsystem is a bus, consisting of a single pair, double pair (Rx/Tx), or a coaxial cable, fig 4. The length of the cable can be up to 400 meters, and the bit rate can be several hundreds of kilobits per second (< 0,5 Mbps).

The communication method is polling. CPU gives permission to send to a substation in order. This substation sends it's data or an acknowledgement to the CPU. There is only one message on the bus at a time. Communication is asynchronous and the length of the message is 62 byte (one line) maximum but can be variable otherwise.

Control processor (CPU) sends polling messages and data messages. The format of these messages is the following:

CPU sends:

SA	Type = Poll
8	8

Polling message

A	B	C	D
SA	Type	Length	Data
8	8	8	n x 8

message to a substation + poll

Fields B and C can be combined to a single 8-bit field.

SA = substation address

Substations send always a response to the message from the CPU. The message types are the following:

Substation sends:

0000	ACK/NACK	SA
8	8	8

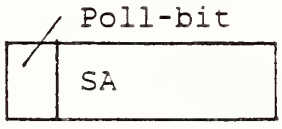
Polling message received, nothing to send.

A	B	C	D	E
0000	Type a data	byte count	data	SA
8	8	8	n x 8	8

Data message from a substation

Poll to the next station is the acknowledgement, that CPU has received the message correctly.

If the number of stations is less than 2 = 128, CPU messages (polling messages) and substation responses can be separated by one bit:



0	SA

CPU sends

1	SA

Substation sends

Address and type fields A and B are combined to one byte in the CPU message as well as the fields A and E in the substation message.

Error control is based on the error detecting by parity checking, and repeated transmissions. This should give a rather good performance in normal conditions. More powerful error checking method may be necessary in difficult environmental conditions and can be easily added to this protocol. In very severe electrical disturbance cases shielded cables or optical fiber cables have been used with good results.

The time of the polling period on the response time of this system can be approximated with constant message lenghts. Let the number of substations be 10, the average message length be 5 x 2 + 1 = 11 bytes with 10 bits/byte, and let the bit rate be 0,5 Mbps. Then

CPU sending time 0,02 ms
substation sending time 0,22 ms
CPU latency time 0,02 ms.

Thus the time of the total polling period is 2,5 ms.

7. CONCLUSIONS

A communication system architecture for a distributed industrial robotics automation system is outlined. The system is represented in a layered harmonised form. Protocols do not exist at the moment to implement all the levels.

Broadband multichannel token-passing LAN is shown to be a good communication solution in regard to reliability, efficiency and expandability. The response time of the LAN-level communication system is in the order of milliseconds. The total response time will depend largely on the capacity of the processors. A protocol for subsystem communication is represented.

8. REFERENCES

/1/ IEEE 802.x series of draft standards for LAN:s

/2/ J.C. Hammond - P.J.P. O'Reilly: 'Performance Analysis of Local Computer Networks', Addison-Wesley 1986.

/3/ W. Hoffman - T. Kersting: 'Simulation of Ethernet under Real-Time Conditions', Process Automation 1984.

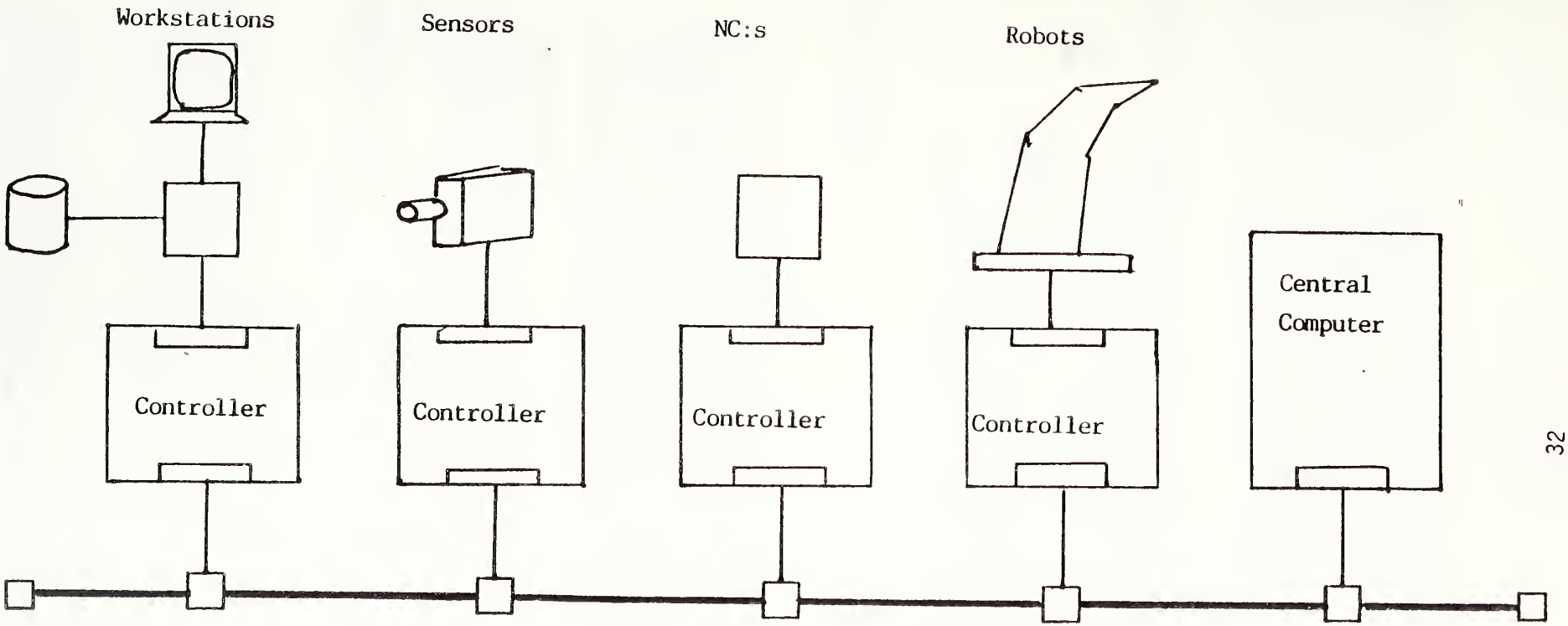
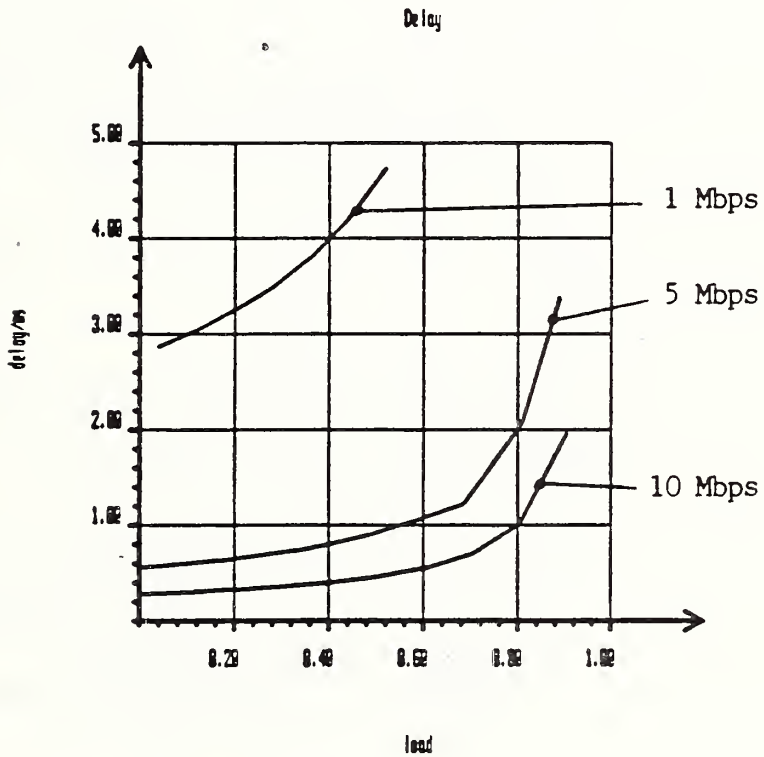


Fig. 1. Modular industrial automation system.

7 Application		Application	
6 Presentation		Data and Command presentation	
5 Session			
4 Transport		Process-to-process links	
3 Network			
2 Data link		IEEE 802.2, connectionless	
1 physical		Broadband multichannel (IEEE802.4)	

Fig. 2. System communication layers and their relation to OSI-layers.



Number of stations n = 50
Average message lenght = 100 bytes
Walking time w = 40*bit time

Fig. 3. Delay curves of a token-passing system.

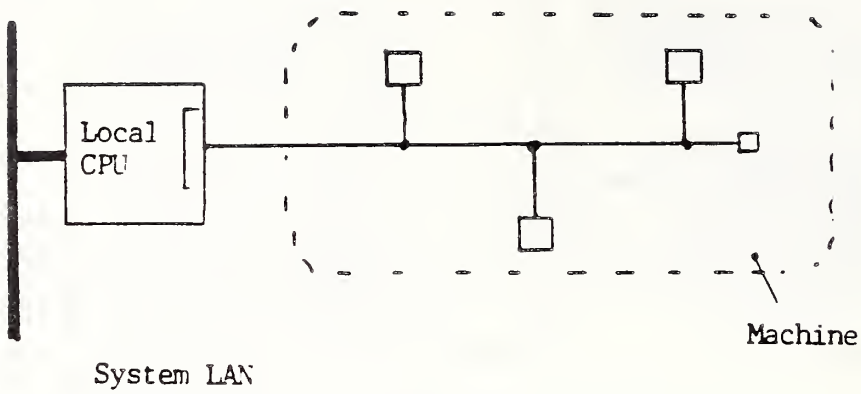


Fig. 4a. The local LAN

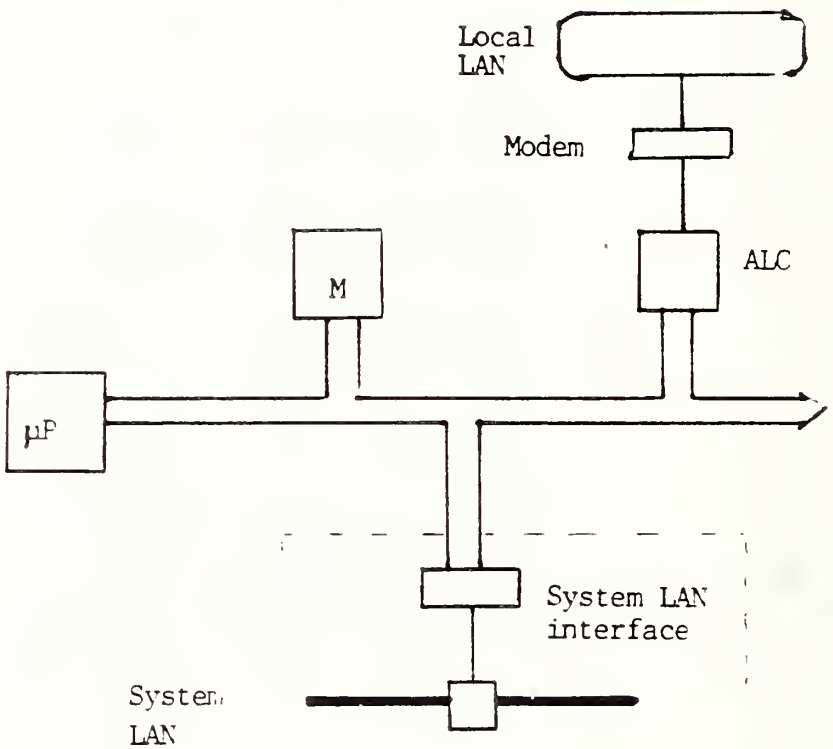


Fig. 4b. Block diagram of the local LAN interface.

REAL-TIME MAP:
A NETWORK ARCHITECTURE STANDARD FOR MANUFACTURING CELLS

By

Vijay Chauhan

Honeywell Corporate Systems Development Division
MN63-C060
1000 Boone Avenue North
Golden Valley, Minnesota 55427

Workshop on Factory Communications

Co-sponsored by
IEEE Industrial Electronics Society and
National Bureau of Standards

to be held March 17-18, 1987 at
National Bureau of Standards
Gaithersburg, Maryland

ABSTRACT

In the wake of the Manufacturing Automation Protocol (MAP) development, efforts to establish network architecture standards in the area of manufacturing automation have reached a frenetic pace. Market opportunity is the basic driver for this feverish activity by all vendors of industrial automation products. In 1984, the U.S. market for process and programmable controllers was $1.1 billion and is expected to increase to $2.6 billion by 1990—a growth rate of 15.9 percent. This paper addresses the architecture and communication needs/standards of a segment of this market that is relevant to Honeywell--real-time control at the manufacturing cell level.

SECTION 1
INTRODUCTION

The manufacturing industry is going through a rapid standardization phase in the area of factory-floor device communications. The lack of standards has driven up device interconnection costs because each interconnect is custom-made for a given pair of devices. This trend limited the gains realized by increased automation and caused General Motors to look into ways devices could interoperate at the "plug-campatible" level, similar to that achieved by the telephone industry. The result was the MAP specification. This specification is a suite of communication protocols derived from IEEE, ANSI, ISO, etc., standards and is based on the seven-layer reference model for Open Systems Interconnection (OSI). MAP is intended to provide standard interconnect solutions on a plant-wide basis; however, for real-time communications, the MAP architecture is considered too cumbersome to meet timing requirements for manufacturing-cell-level communications. This is due largely to the seven layers of communications overhead present in the MAP architecture. To solve this, a collapsed architecture that bypasses four layers of protocol and has increased network responsiveness has been designed to provide standardized interconnect solutions at the cell level for real-time control. This architecture is termed Real-Time (RT) MAP. The collapsed architecture, however, is reduced in functionality when compared to the seven-layer MAP architecture.

Section 2 describes the RT MAP architecture, the protocols associated with the architecture, and the status of the standards. Since the architecture is based on draft proposals that are evolving toward international standard status, the architecture is a moving target.

Section 3 presents some of the research work done within Honeywell with regard to the performance study of the Real-Time MAP architecture. A prototype network has been built for performance modeling; the status of the network is presented.

SECTION 2
REAL-TIME MAP ARCHITECTURE AND STANDARDS

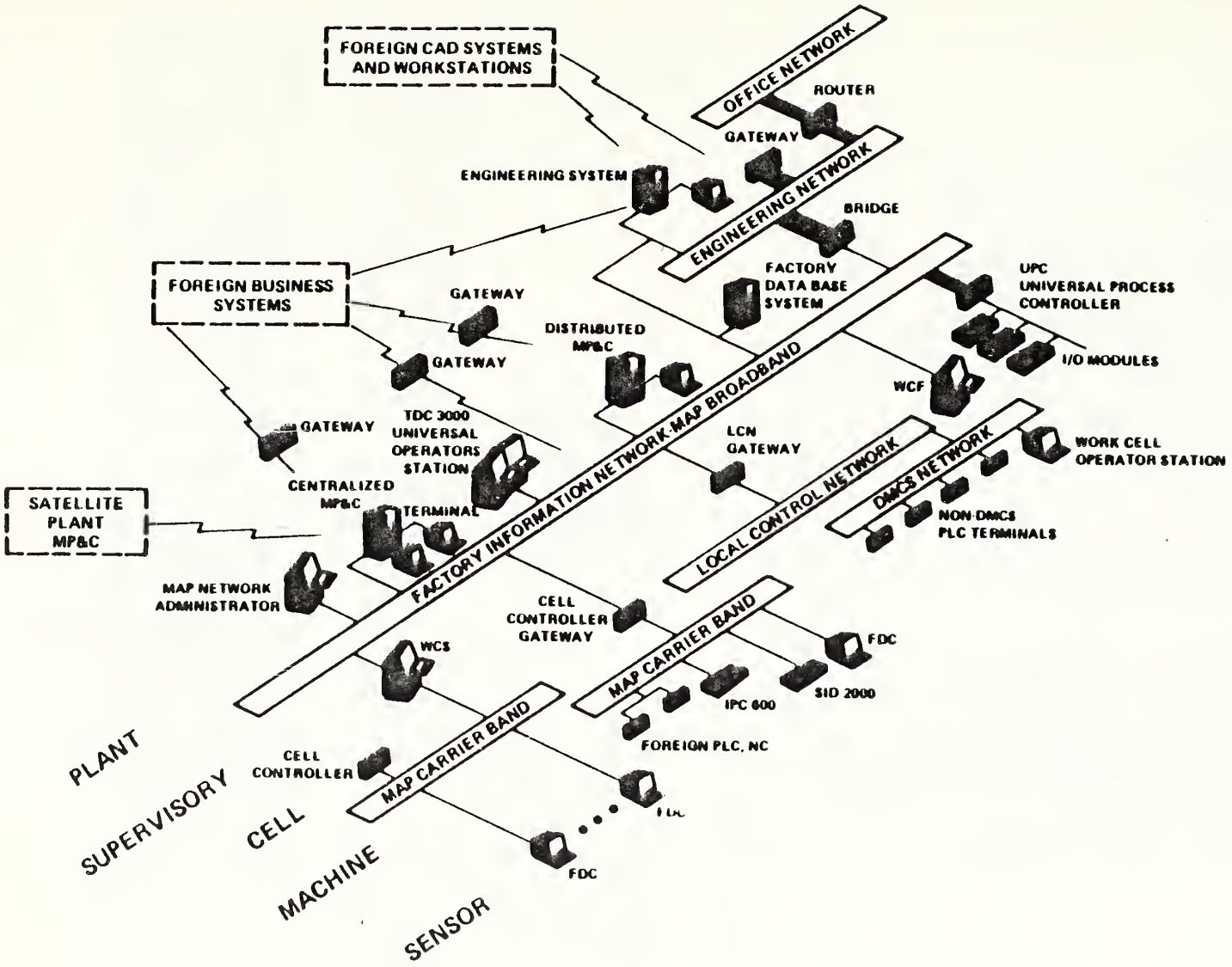
ARCHITECTURE

The discrete manufacturing architecture can be hierarchically divided into six levels, starting from the enterprise level and going down to the sensor level (Figure 2-1). At the enterprise level, whole plants distributed geographically over large distances are connected via wide-area networks (WANs). At the sensor level, a varied collection of sensors are linked together via potentially simple proprietary networks. Located within this spectrum of network architectures are the interconnect solutions offered by MAP and RT MAP. The MAP architecture is ideal for a backbone network that is dispersed throughout the entire plant. This architecture is designed to carry high volumes of data of a heterogeneous nature (data, voice, facsimile, etc.). In an attempt to be "everything to everybody," the MAP architecture resulted in a hefty set of network protocols that were loaded with functionality. However, the one functionality that had to be sacrificed (to provide for the rest) was network responsiveness (i.e., the ability to get things across to the other host quickly). At the level of the manufacturing cell, the maximum tolerable end-to-end response time is on the order of 20 to 100 milliseconds. At this level of the manufacturing architecture hierarchy, network responsiveness takes priority over any other functionality. A standard network architecture that provided the required network responsiveness was needed; the RT MAP architecture was designed to fulfill this need. Layers of protocols present in the MAP architecture whose functionality was not a priority were eliminated. RT MAP is a collapsed version of the MAP architecture with a smaller set of protocols that results in a smaller set of code/hardware and, hence, lower networking overhead.

Figure 2-2 contrasts the full MAP and RT MAP architectures. The MAP architecture, which is based on the reference model for Open Systems Interconnection (OSI) provided by the International Standards Organization (ISO), specifies a network protocol for each layer of the reference model (currently the presentation layer protocol is not specified). The RT MAP architecture was derived from the MAP architecture by removing protocols that provided functions thought to be dispensible.

Real-Time MAP goals include:

- Minimal machine instruction cycles to deliver protocol data units to the application interface;
- Immediate message acknowledgement;
- Broadcast capability;
- PROWAY and 802 compatibility.



File No. 6-0237

Figure 2-1. Discrete Manufacturing Architecture

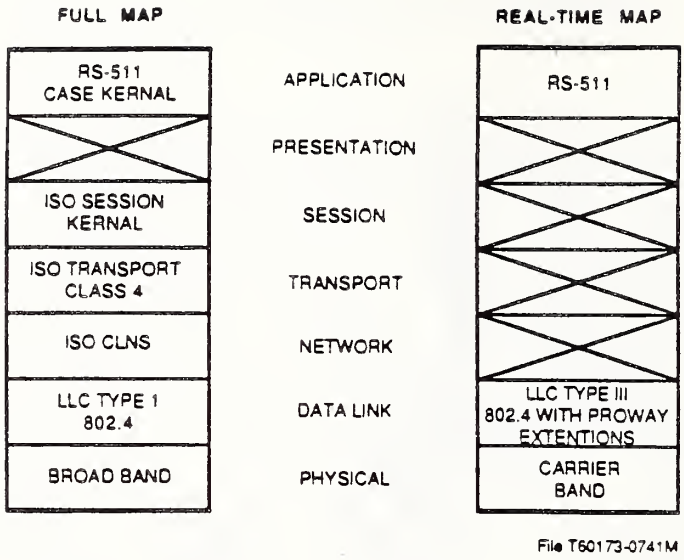


Figure 2-2. Layer Specification

The following are Real-Time MAP design criteria:

Access time	25 ms
Signalling rate	5 Mbps
Signalling method	Phase coherent
No. of stations	32
Distance (perimeter)	1 km (3300 ft)
Typical message length	16 to 20 octets
Maximum message length	<1K octets

To provide real-time communications capability, the following two requirements must be met: (1) a facility to provide immediate acknowledgement for every message transmitted by a sending station, and (2) a decrease from the MAP standard in the time for a message to travel from the physical layer to the application layer.

The first of these requirements is provided for by including PROWAY's extension to the MAC layer specified by the IEEE 802.4 standard. The second requirement is met by bypassing some of the intermediate layers of the OSI reference model. This bypassing of layers can only be achieved by sacrificing some of the functionality normally provided by the bypassed layers. The layers that have been bypassed to provide a real-time architecture are network, transport, session and presentation. Figure 2-3 represents the RT MAP architecture.

The notable differences between the RT MAP protocols and their counterparts in the MAP architecture are:

- Physical Layer--In the MAP architecture, a broadband coaxial cable based on the specifications laid down by the IEEE 802.4 standards committee is identified as the physical layer. This specification is based on the standard CATV cable used by the cable TV industry. The bandwidth is divided into channels, each approximately 6 MHz wide. Thus it is possible to pass information of various kinds (voice/data/facsimile, etc.), each on a different channel. This requires modems that are fairly intelligent and frequency-agile; it also requires active components such as head-end remodulators, etc. RT MAP specifies a single-channel, frequency-shift-keying (FSK), carrierband, phase-coherent signalling method. Here the modems are relatively simple and therefore are expected to be cheaper (a requirement for the RT MAP architecture).

- Data Link Layer--At the MAC sublayer of the data link layer, the MAP architecture specifies the IEEE 802.4 standard, and the RT MAP architecture specifies the IEEE 802.4 standard with some extensions for implementing the immediate acknowledgment functions. At the LLC sublayer, the MAP architecture specifies a simple datagram TYPE 1 service. The RT MAP architecture, on the other hand, specifies a Class

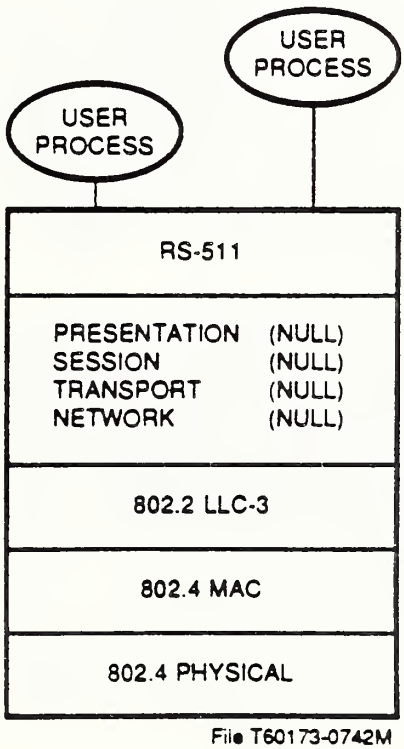


Figure 2-3. RT MAP Architecture

3 service (a union of TYPE 1 and TYPE 3). The TYPE 3 service provides an acknowledged connectionless service that results in reduced transport layer functionality.

- Network, Transport, Session, Presentation Layers--The MAP architecture specifies protocols as shown in Figure 2-2, while the RT MAP architecture has null protocols for these layers.

- Application Layer--At the application layer, the MAP architecture specifies a CASE protocol, and the RT MAP architecture does not. In addition, the MAP architecture specifies a host of application-specific service elements such as RS-511, Vertical Terminal Protocol (VTP), FTAM, etc., while the RT MAP architecture specifies only RS-511 as the application layer.

In meeting the RT MAP goals, the following functionality has been compromised:

- High-quality guaranteed delivery;
- Session services, e.g., data synchronization;
- Real-time global message delivery (off-segment);
- Indefinite message length;
- Reduced flow control;
- Concept of connection.

The RT MAP cell level architecture connects to the plantwide network (which is potentially based on the MAP architecture) via a gateway, as shown in Figure 2-4. However, a gateway is not the only type of device that can be used to interconnect MAP and RT MAP networks. Routers and bridges are alternative interconnect devices (Figures 2-5 and 2-6, respectively). A router provides connection at the network layer, while the bridge provides connection at the data link layer. The trade-offs in selecting a bridge, router, or gateway involve issues related to cost, performance, dissimilarity of networks, security, etc.

Two types of node architectures have been identified for an RT MAP network (Figure 2-4). One type of node is expected to have collapsed RT MAP architecture, as specified in Figure 2-3 and shown in Figure 2-4. The second type of node is expected to have a dual architecture. One part of the dual architecture exhibits the RT MAP architecture and the other part exhibits the MAP architecture. This dual-architecture node is also known as a MAP/RT MAP node. The purpose of the MAP/RT MAP node is to provide a way for MAP nodes to obtain faster response times on control and time-critical networks.

NOTE: The RT MAP architecture node is also known as a Mini-MAP node. The dual-architecture (MAP/RT MAP) node is also known as an Enhanced Performance Architecture (EPA) node. These two are the only accurate names to date. All other names should be ignored to reduce confusion.

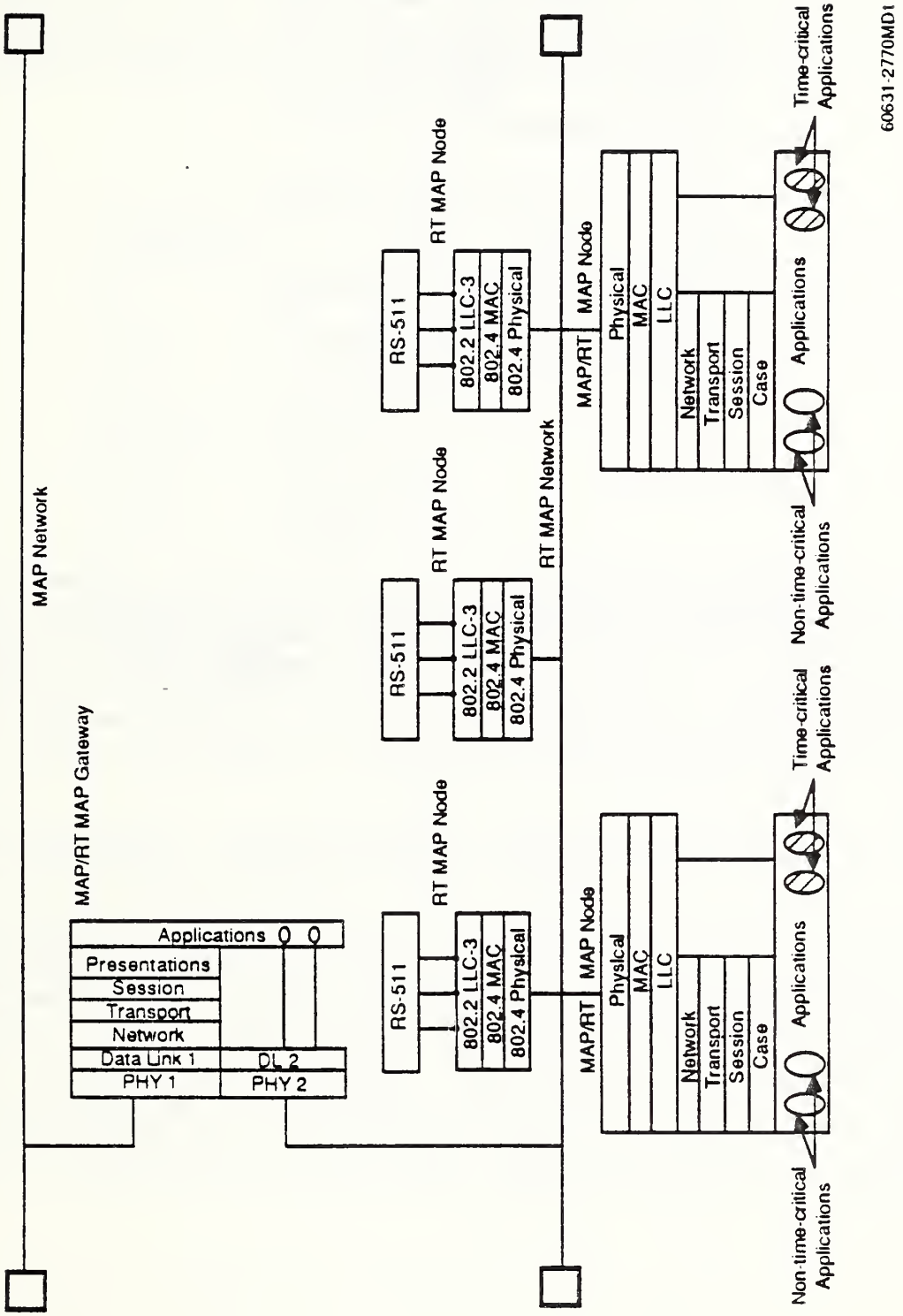


Figure 2-4. RT MAP/MAP Network Interconnect

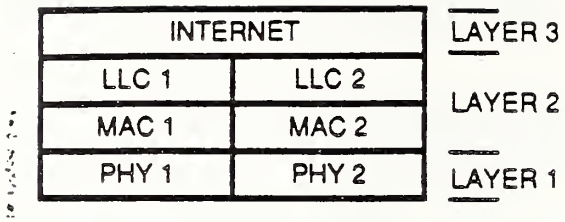


Figure 2-5. Router Architecture

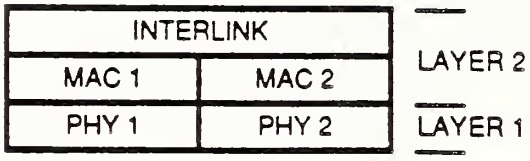


Figure 2-6. Bridge Architecture

Standards

The real force behind the development of RT MAP is its conformance to standards that are defined by recognized standards bodies and committees. Not until these standards are officially established, whether they are interim standards or final, will the full capability for interconnection of devices from different vendors be realized. Therefore, tracking and driving the standards must parallel their implementation. This is accomplished by active participation at appropriate standards meetings and through establishment of a corporate strategy on the level of participation.

All protocols associated with the RT MAP architecture are at various stages of becoming either international standards or IEEE standards. Table 2-1 shows the currently identified work items that relate to the RT MAP architecture.

The standardization process within MAP is as follows. Architectures/Items for Standardization (IFS) are defined and designed by various MAP subcommittees. The subcommittee passes its work on to the MAP Technical Review Committee (TRC). When the TRC approves the work item, it must decide in which revision of the MAP specification the approved work item should be introduced. It also must decide what changes to the entire MAP specification are required to keep it consistent with the recent addition. Typically, the MAP subcommittees develop architectures based on protocols that are being standardized by bodies such as IEEE, EIA, ANSI and ISO. Thus, the proposed architectures are based on either existing or evolving standards.

All protocols associated with RT MAP are emerging standards. No implementation of the RT MAP architecture exists in its entirety; thus, questions related to performance, throughput, size of code, protocol overhead, reliability, etc., have previously had no implementation-based answers. Section 3 describes efforts within Honeywell to answer these questions.

Table 2-1. Status of Standards

Standard	1986	1987	1988
LLC	Draft Proposal1Q86	IEEE Standard 1Q87	IS*
RS-511	EIA Draft Proposal	EIA Standard 1Q87	IS 1Q88
Network Management	Draft Proposal in IEEE	IEEE Standard 2Q87?	IS
Directory Services	Draft Proposal in MAP Subcommittee and ANSI	ANSI Standard 2Q87?	IS?
User Interface		MAP Standard 2Q87	

* International Standard

60551-3437-1

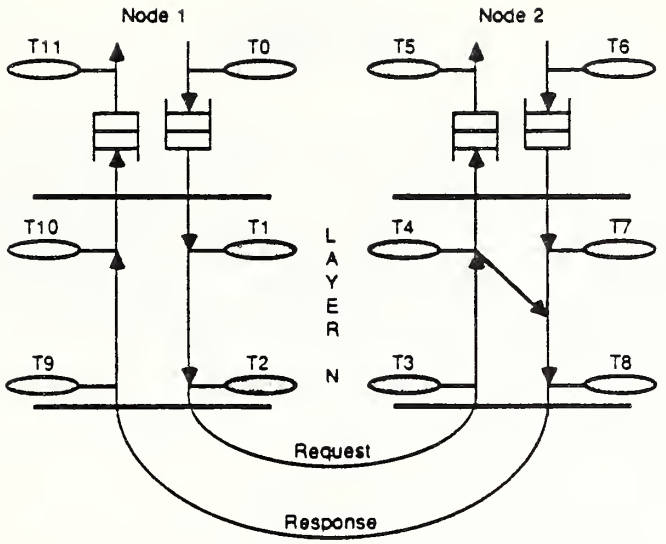
SECTION 3
REAL-TIME MAP PROTOTYPE NETWORK

In early 1986, a research program was put in place within Honeywell with a specific goal of understanding the performance, protocol and service offered by the Real-Time MAP architecture. The goals of the program are:

- Build a real-time MAP network by the end of 1986;
- Integrate typical factory floor, cell level devices (WCC 1250, IPC 620);
- Obtain performance measures for the prototype network such as end-to-end delay, throughput, etc. (See Figure 3-1);
- Integrate performance measurement hooks for layer-by-layer analysis of the architecture;
- Build a generalized multi-node performance model for Real-Time MAP;
- Validate performance model against prototype network.

The prototype three-node network (Figure 3-2) consists of two WCC 1250 microcomputers and an IPC 620 programmable controller. The WCC 1250 is a 68000-based, multiuser, multitasking supermicro manufactured by Digital Datacom, Inc., a Honeywell subsidiary. The IPC 620 is a programmable controller manufactured by IPCD. Each of the hosts will interface to the RT MAP network via a Network Interface Unit (NIU). The major task of the program is to build an NIU that plugs into the backplane of the host. The NIU will be 68000 based, with a Token Bus Controller (TBC) (supplied by Motorola) that will implement the MAC sublayer and a carrierband modem (supplied by Control Data Systsems/Computrol). The NIU will implement the entire RT MAP architecture, leaving the host to perform the application-oriented tasks.

Currently the three-node network with RS-511 and LLC Class 3 protocol implemented in software is up and running. The network is going through an oplimization phase. Simultaneously an evaluation effort for petri-net/queueing network performance modeling tools is underway.

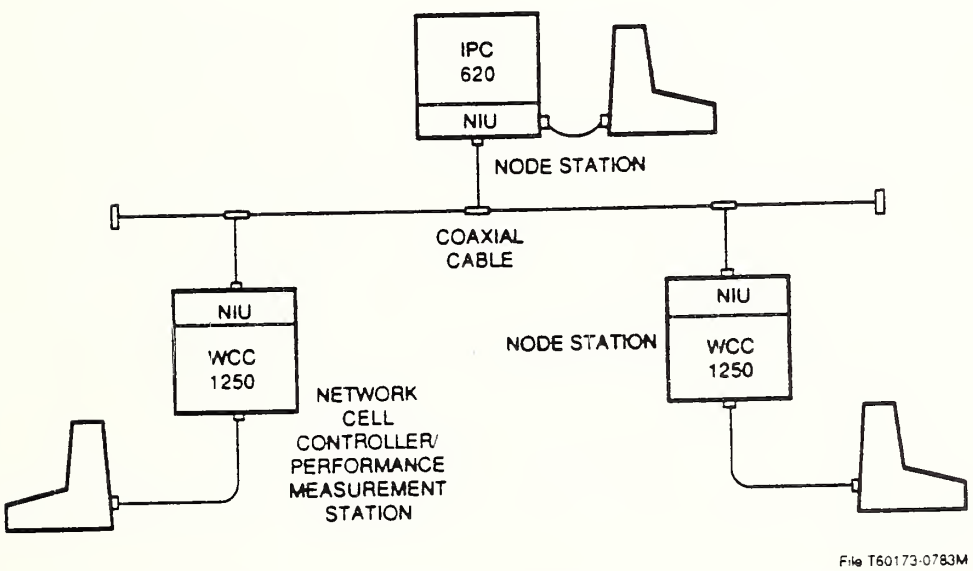


Performance Measures for Layer N:

- Response Time: T11 - T0
- End-to-End Delay: T5 - T0
- Queueing Delay: T1 - T0 (for a request)
- Service Time. (T2 - T1) + (T10 - T11) for Node 1

60551-3417MD1

Figure 3-1. Performance Measurements



File T60173-0783M

Figure 3-2. Prototype Network for Real-Time MAP

SECTION 4
CONCLUSION

Increased automation implies an increase in the amount of control and functionality distributed to devices that are closer to the shop floor. Together with the need for peer-to-peer, high-performance communications, Real-Time MAP with its lower protocol overhead and its connectionless mode of service has been designed to provide for these needs. Studies along the lines described in this paper will help ultimately in designing better communication solutions for the manufacturing automation industry.

SECTION 5
REFERENCES

1. ANSI/IEEE Std 802.2-1985

2. ANSI/IEEE Std 802.4-1985

3. ISA SP72.01 - PROWAY LAN

4. General Motors MANUFACTURING AUTOMATION PROTOCOL ver 2.1

5. General Motors MAP/Enhanced Performance Architecture Overview, November 1985

Applying MAP Networks

Open System Interconnection for Real-Time Factory Communications: Performance Results

Cornelis Franx
Philips
Eindhoven, The Netherlands

Kevin Mills
National Bureau of Standards
Gaithersburg, MD USA

This paper considers the applicability of Open System Interconnection (OSI) protocols, as now defined and implemented, for use in real-time factory communications. Factory communications requirements are described by outlining the hierarchial nature of the factory network architecture and by defining the nature of real-time at the lowest level of hierarchy. Two possible solutions for real-time factory communications are described: 1) the full MAP seven layer architecture and 2) the MAP enhanced performance architecture (EPA). Measured performance of a five layer OSI protocol implementation is described with special emphasis on one-way delays. Measurement results are also given for throughput. The ability of present OSI standards to guarantee real-time performance is evaluated. A flow control problem is identified concerning use of an OSI transport protocol over a type 1 class 1 logical link control protocol.

I. Introduction

Data communications within a factory must meet a hierarchy of performance requirements including real-time at the lowest levels and time-critical within a workcell. The Manufacturing Automation Protocol (MAP) standard, requiring all seven OSI protocol layers, was thought inadequate for real-time and time critical applications. Thus, a subset of protocols defining an Enhanced Performance Architecture (EPA) has been added to the MAP standard. The purpose of this paper is threefold: 1) to describe the hierarchical performance requirements within a typical automated factory, 2) to explain the differences between the full seven layer MAP standard and the three layer EPA, and 3) to provide measured performance results for applications using the lower four layers of the MAP standard.* Each of these topics is covered in a separate section below. Some conclusions are drawn with respect to the performance possibilities of EPA and the full MAP protocols and areas of further research are indicated.

II. Factory Communications Requirements

Communication requirements in a factory depend on the type of production process carried on in the factory. So the production process and the related production control architecture have to be described before the communication requirements can be stated.

In Philips there are many factories with discrete assembly lines, for instance to produce radio sets. The control structure of these lines will be based on the National Bureau of Standards hierarchical model for production control that was adopted by Philips (see Figure II-1) [ALB81]. The controller processes at different levels in the hierarchy will be implemented on separate computer systems. In the automation module and device control layers individual systems control robots, positioners and local workpiece transport within an assembly station. The workstation controller sequences the activities of the automation module controllers to execute assembly tasks with elapsed time in the two to five second range. The workcell controller routes incoming parts to available stations, coordinating the area transport and the assembly workstations. Data communication between the computer systems is required when their controller processes interact.

---

*Certain commercial equipment is identified in this paper in order to adequately specify the experimental procedure. Such identification does not imply recommendation or endorsement by the National Bureau of Standards, nor does it imply that the equipment identified is necessarily the best available for the purpose.

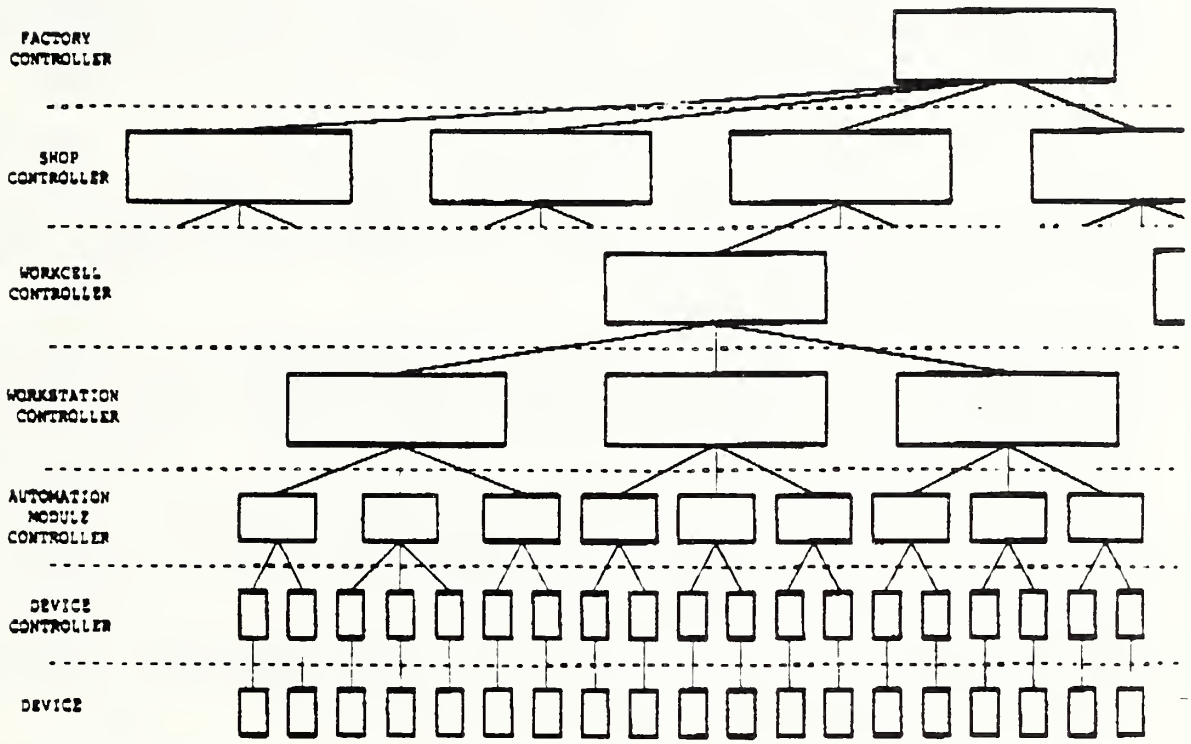


FIGURE II-1. LOGICAL MODEL OF A PRODUCTION CONTROL HIERARCHY

The time requirements of a controller process become less stringent as the process is positioned higher in the model. Controllers at the top of the model are concerned with long term planning and control, at the bottom of the model they do direct real-time control. At the upper layers large batches of data (> 1Mb) have to be transported at one time, for instance a daily production plan. This batch data must be transported within several minutes. Going down in the model time requirement become shorter and more stringent, but the amount of data to be transported at one time decreases as well.

In the Philips environment, from the workcell controller down, communication times on the order of one second can reduce production throughput by 10-20%. Consider, for example, the communication between workcell controller and workstation controller. At this level work orders and status reports are exchanged with message lengths of 100 - 500 bytes. A message must be sent and delivered within 100 to 200 ms.

Communication at the next lower level between automation module controllers and workstation controller is more critical, with 100 ms being a maximum time for communications. Communication within the workstation becomes highly time critical, especially between device controllers and automation modules. Control loops at this level require small amounts of data (< 20 bytes) to be transmitted very fast (within 10 ms) with a repetition frequency of up to once every 10 ms. All figures presented here are meant to give a global indication of the communication requirements and they only concern the delivery of production control information.

When there are tight time limits like those at the bottom level, the traffic is often called "real-time" generally without defining the meaning of "real-time". In this paper it has the following meaning:

> Communication is real-time when a message must be passed from one process to another within a previously specified time limit in order for the process to correctly perform its function.

The data communication network has to guarantee that except in the case of system or component failure the message will be transferred within that time limit. To prevent messages with a more relaxed time limit from interfering with urgent messages and delaying them, priorities have to be allocated to messages. Higher priority messages are handled before messages with a lower priority.

Another important aspect of real-time communication is its error behavior. In case of a link failure only limited time should be spent on error recovery procedures, then the application processes must be warned to enable them to take appropriate action.

As mentioned eariler, real-time communication can be found at the bottom of the model, however, it is important to realize that not all communication is for the purpose of production control. Communication for software downloading

and reporting of production statistics has different requirements. Generally the time limits are not very tight in these cases but reliability is more important, requiring more extensive error recovery efforts.

The different communication requirements at the bottom two layers of the factory model demonstrate the need for a network that offers a real-time communication service as well as a more reliable, higher level communication service. Without a standardized network offering both types of service, proprietary networking solutions are inevitable.

III. OSI and MAP Enhanced Performance Architecture

Two possible architectures for real-time factory communications will be discussed, the "full" MAP architecture based on the seven layer OSI reference model and the three layer Enhanced Performance Architecture (EPA). In MAP version 2.1 a selection of protocols and options for six layers of the OSI model (Fig. II-2) has been made. The missing protocol for the presentation layer will be supplied later. Based on this selection of protocols, vendors have started to make interoperable data communication products for the factory.

Almost from the beginning of MAP there were doubts from the process control industry that seven layer MAP (called "full" MAP) could meet the performance requirements for real-time applications. This led to the introduction of a new, three layer architecture called the enhanced performance architecture (EPA) (Figure II-3). Many of the ideas behind EPA were adopted from Proway which is a local area network for the industrial environment defined by ISA SP 72 [ISA72].

Both full MAP and EPA use at their bottom layers the IEEE 802.4 token bus [IEE85]. The first important difference between the two is at the data link layer where EPA uses logical link control (LLC) type 3, acknowledged connectionless service, instead of LLC type 1, unacknowledged connectionless service [IEE84]. LLC type 3 confirms the arrival of data at the destination LLC layer and retransmits data in case of errors. More precisely when a station has the token it sends an LLC type 3 frame and then waits for a returning LLC type 3 frame that is in fact an acknowledgement. Without this response, a timeout will occur and the LLC frame will be retransmitted. Only after the acknowledgement is received or the maximum number of retransmissions is exhausted can the token be passed to the next station. It is notable that this LLC protocol interacts with the operation of the token passing protocol at the MAC layer.

The time the station can hold the token (token-hold time) is limited; therefore, the LLC type 3 acknowledgement and retransmission scheme has to work very fast, and only three retransmissions are allowed. Without a tight limit on the token-hold time the total performance of the token bus could degrade severely when there are repeated LLC type 3 failures.

The LLC type 3 timing characteristics make it impossible to transit a bridge because a bridge introduces unacceptably large message delays when it passes LLC frames from one network segment to another. As a result the use of LLC type 3 is limited to a single segment.

LAYERS	MAP V2.1 PROTOCOLS	MAP/EPA PROTOCOLS
Layer 7 Application	ISO FTAM (DP) 8571 File Transfer Protocol, Manufacturing Message Format Standard (MMFS) and Common Application Service Elements (CASE)	EIA MMS RS 511 Manufacturing Message Service
Layer 6 Presentation	NULL (ASCII and Binary Encoding)	NONE
Layer 5 Session	ISO Session (IS) 8327 Basic Combined Subset and Session Kernel, Full Duplex	NONE
Layer 4 Transport	ISO Transport (IS) 8073 Class 4	NONE
Layer 3 Network	ISO Internet (DIS) 8473 Connectionless Network Service (CLNS)	NONE
Layer 2 Data Link	ISO Logical Link Control (DIS) 8802/2 (IEEE 802.2) Class 1, Type 1	Class 3, Type 3
Layer 1 Physical	ISO Token Passing Bus (DIS) 8802/4 (IEEE 802.4) Broadband	Carrierband

FIGURE II-2. MAP V2.1 AND FUTURE MAP/EPA PROTOCOL STACK

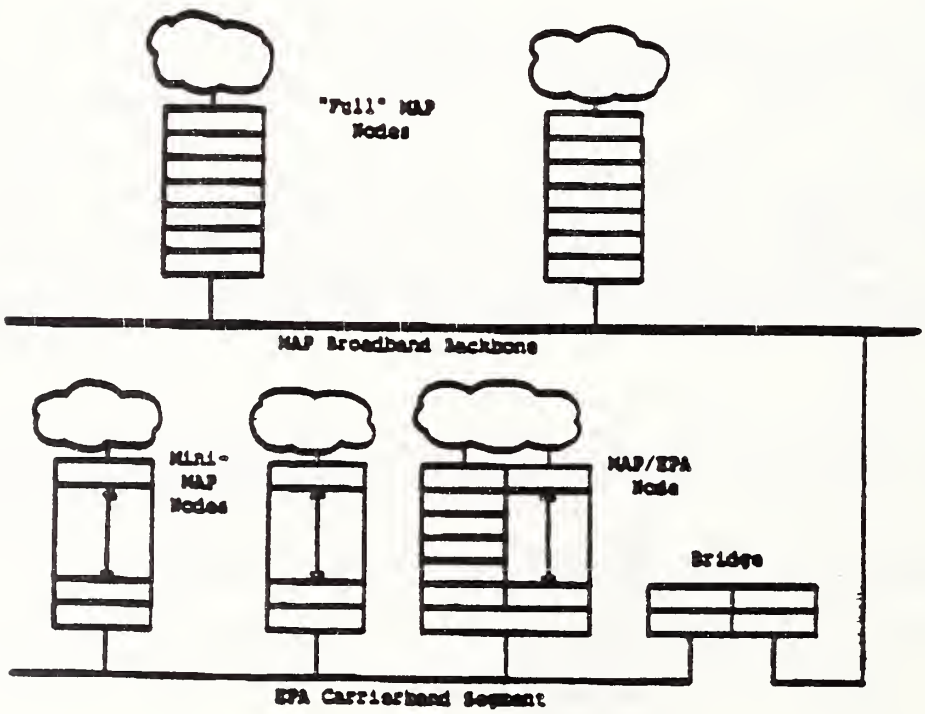


FIGURE II-3. NODES ON AN EPA SEGMENT

The use of LLC 3 to provide real-time message delivery is not without potential problems. When a token-holding station is awaiting an immediate acknowledgement several events can occur. For example, the token-holding station may retransmit the message or the token-holding station may give up and pass the token. Either of these messages may collide with the acknowledgement. Collision with the retransmitted message will cause another retransmission. Collision with the token will invoke token recovery procedures.

It is very likely that EPA, in addition to the bottom two layers with IEEE 802.4 token bus and LLC type 3, will ultimately have a third protocol layer containing Manufacturing Message Service (MMS) which runs directly on top of the LLC. The mapping between MMS and LLC has already been described in a document prepared by MAP's Programmable Devices Committee.

In the EPA specification layers 3 - 6 of the OSI model are absent. Because of the missing layers EPA offers reduced functionality when compared to "full" MAP. The consequences of skipping four layers will be discussed briefly, starting at the network layer.

Because there is no network layer it is impossible to traverse intermediate systems (routers). But, since intermediate systems introduce relatively large and unpredictable message delays, this limitation is necessary to achieve real-time performance. Without a network layer it is possible to leave out the integrity assurance protocol of the transport layer as well because LLC type 3 already offers end-to-end acknowledgment. Some of the missing transport functions such as, resequencing, flow control, and multiple associations, are covered by the mapping between LLC type 3 and MMS.

The next higher layer, the session layer, offers dialogue control and resynchronization which are not suited for real-time applications. Therefore, the session layer can be skipped as well. Finally EPA has a presentation syntax (X.409) [CCI84] and an application layer protocol when the layer seven protocol, MMS [EIA85], is added to EPA.

The use of EPA and "full" MAP can be combined on a single EPA segment (Figure III-3). On the segment there can be MAP/EPA nodes with the three and seven layer stack and Mini-MAP nodes with the three layer stack only. The choice between "full" MAP or EPA should be based on the communication requirements of the applications. Applications in a MAP/EPA node select the seven layer stack to get the functionality of "full" MAP and then are able to communicate with all other MAP/EPA nodes on the segment and with "full" MAP nodes on the broadband backbone. Real-time applications select the EPA stack to get faster responses and then are able to communicate with other MAP/EPA nodes and Mini-MAP nodes on the same segment. Real-time applications must be aware of the reduced functionality of EPA.

The main differences between "full" MAP and EPA have been pointed out. Now the suitability of both architectures to handle real-time communication will be discussed. The functionality of "full" MAP is not always required, and the associated protocol activities are sometimes undesirable for real-time communication (i.e., message routing, extensive retransmissions and resynchronization). When the seven layer OSI stack is used, message delays

are unpredictable and can not be controlled. One of the main reasons for this is the lack of control over the allocation of system resources making it impossible to give real-time communication priority over other communication and activities like network management (see Section IV). Without priorities all communication is treated in the same way regardless of its urgency and it is possible that large file transfers will interfere with short real-time messages and delay them.

EPA is in a better position to provide real-time communication than "full" MAP. Because it has only three protocol layers, processing delays can be much smaller. Moreover, it has a priority mechanism at the LLC layer that, when MMS honors priorities as well, permits full application-controlled use of priorities. Message delays are predictable when the number of nodes in the segment is known and the segment is in a stable operating condition. It should be realized that this is only true for messges with the highest priority -- lower priority messages are delayed by higher priority messages making it more difficult to predict their delay. LLC type 3 has a favorable error recovery mechanism because it performs retransmissions in a short time and then warns the application. Other failures delaying the transmission of a message should be reported as well (i.e., the collapse of the token ring).

For real-time communications the EPA stack seems to have attractive properties, but Philips is concerned about the cost to produce error-free application software for EPA and about the portability and flexibility of this software. Also, conformance and interoperability testing will be difficult for EPA because there are many options, such as connectionless or connection-oriented use of MMS, MMS subsets, and non-token stations.

IV. OSI Measured Performance

In order to assess the suitability of OSI protocol standards for factory applications, the National Bureau of Standards (NBS) and Philips conducted a cooperative project to measure the communication performance in a small testbed network using OSI protocol implementations. Intel Corporation donated hardware and software for the project. A brief description of the testbed network is given, followed by a discussion of the performance results.

A. The Testbed Network

The local area network testbed implemented at the NBS is illustrated in Figure IV-1. Four Intel 310 nodes and a passive, real-time monitor are connected to a CSMA/CD local network.* A global clock circuit is connected to each Intel 310 node to provide a synchronized measurement clock. The internal architecture of each node is shown in Figure IV-2.

OSI communication services are provided by Intel's iNA-960 software [INT84A] running on a 186/51 COMMputer ™ board [INT84B]. The 186/51 contains two processing elements: an 80186 (transport, network, and logical link control) and an 82586 (media access control). Traffic generation and measurement software runs on a separate host board based on an 80286 processor [INT83]. Communication between the host and COMMputer board is via message passing using the Multibus Interprocessor Protocol (MIP) [INT84A, Appendix E].

Figure IV-3 shows how the global clock board provides a synchronized 100 us pulse to each Intel 80286 CPU board. The clock pulse is connected to a 16-bit programmable interval timer (PIT). The PIT overflows every 6.5 seconds causing a 16-bit software clock to be updated. The entire 32-bit clock is available to user software.

Figure IV-4 illustrates the time delay measurements made within the measurement software. A user task requests communications services by issuing a request block (RB) to iNA-960 via a system call. The time required to return from the system call is measured as T1. Once the iNA-960 has provided a requested service, the RB is returned to the user program. T3 measures the time elapsed between issuance of the RB and its return. An RB normally contains a user message within it. T2 measures one-way delay for user messages. T4 measures the duration of an experiment.

The variables controlled by the traffic generation software are listed below (Table IV-1). Another set of variables, such as retransmission timer values and transport message sizes, are controlled on a connection-by-connection basis using iNA-960 network management services. The network management services are also used to monitor lower level measures such as collision counts, count of packets sent and received, and number of packets dropped due to buffer overrun. A passive, real-time monitor enables unobtrusive evaluation of experiment progress -- indicating number of connections, number of retransmissions, protocol efficiency, and total data sent [MIL85].

The experiments divide naturally into three sets: 1) throughput profile, 2) delay profile, and 3) multi-application profile. Measured results for each set are discussed in the following sections.

---

*Although MAP requires use of a token passing bus media access control technique, the object of this study is transport layer performance on an unloaded local network, and so the use of a CSMA/CD local network does not invalidate the study.

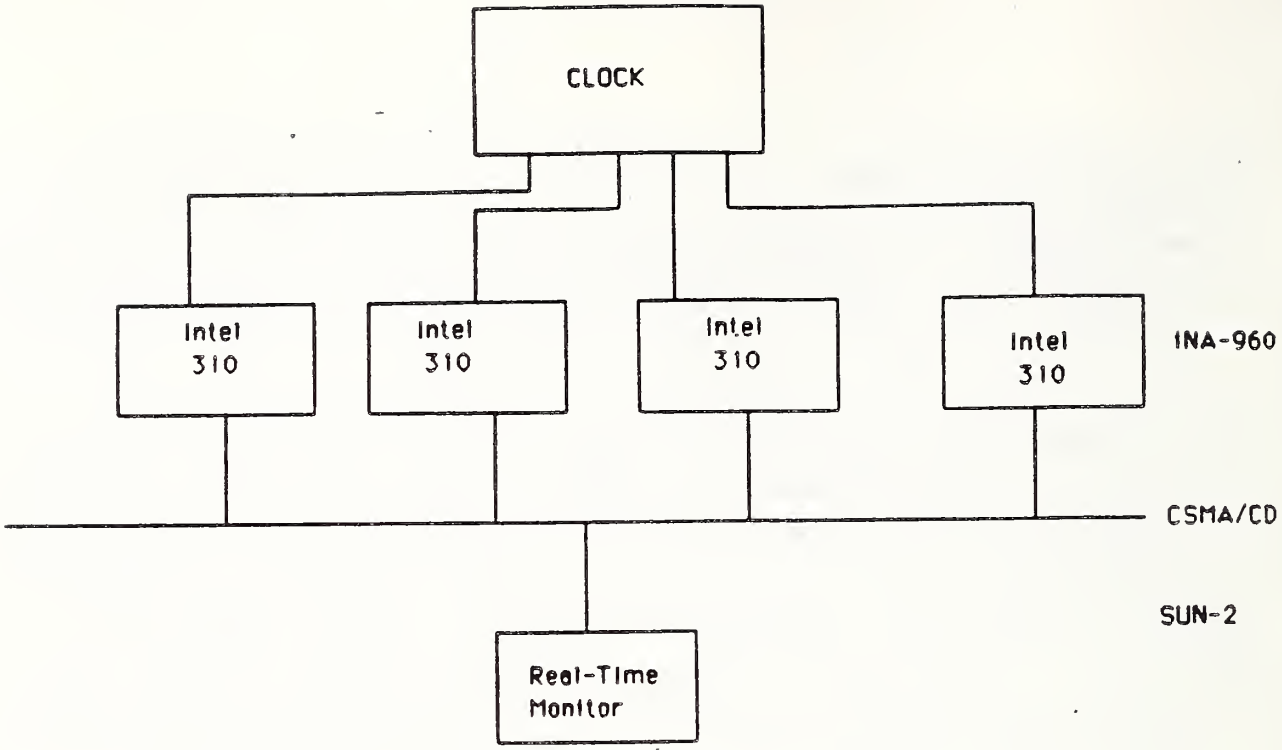


FIGURE IV-1. EXPERIMENT NETWORK TESTBED

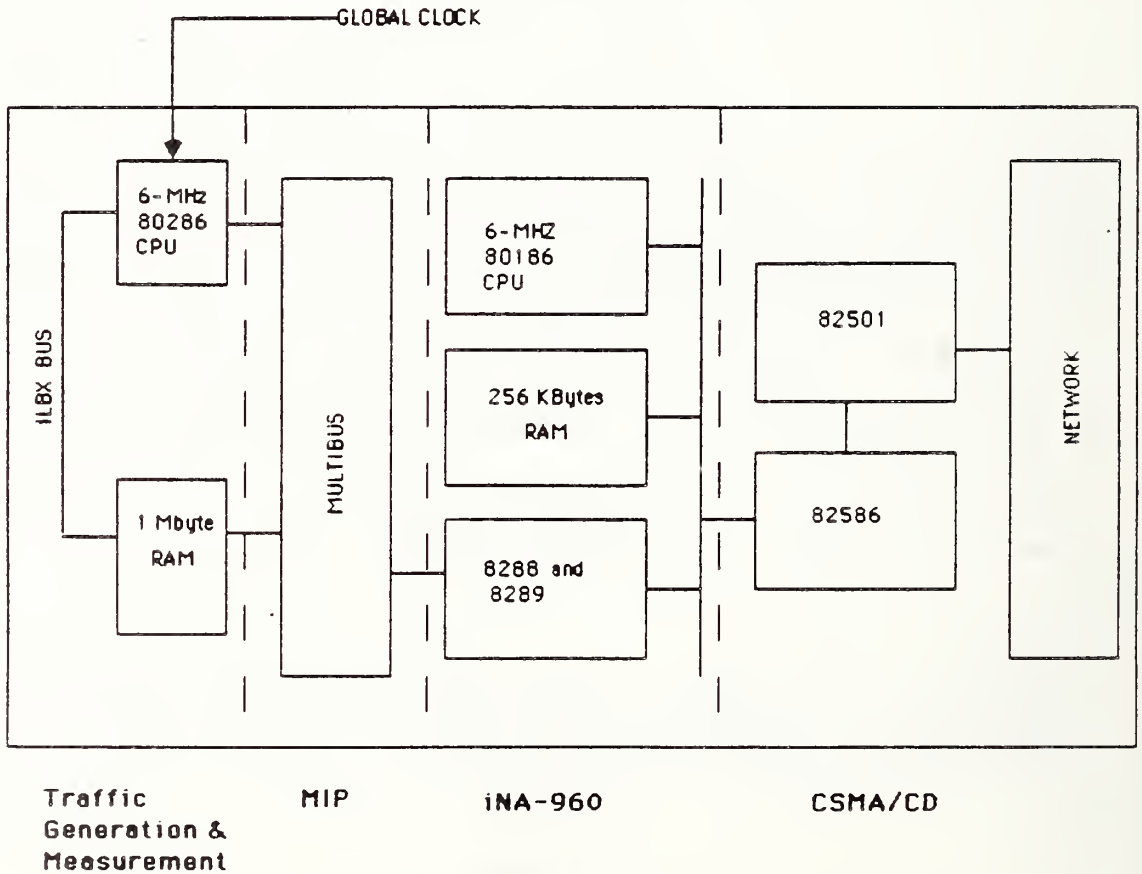


Figure IV-2. Architecture of an Intel 310 Network Node

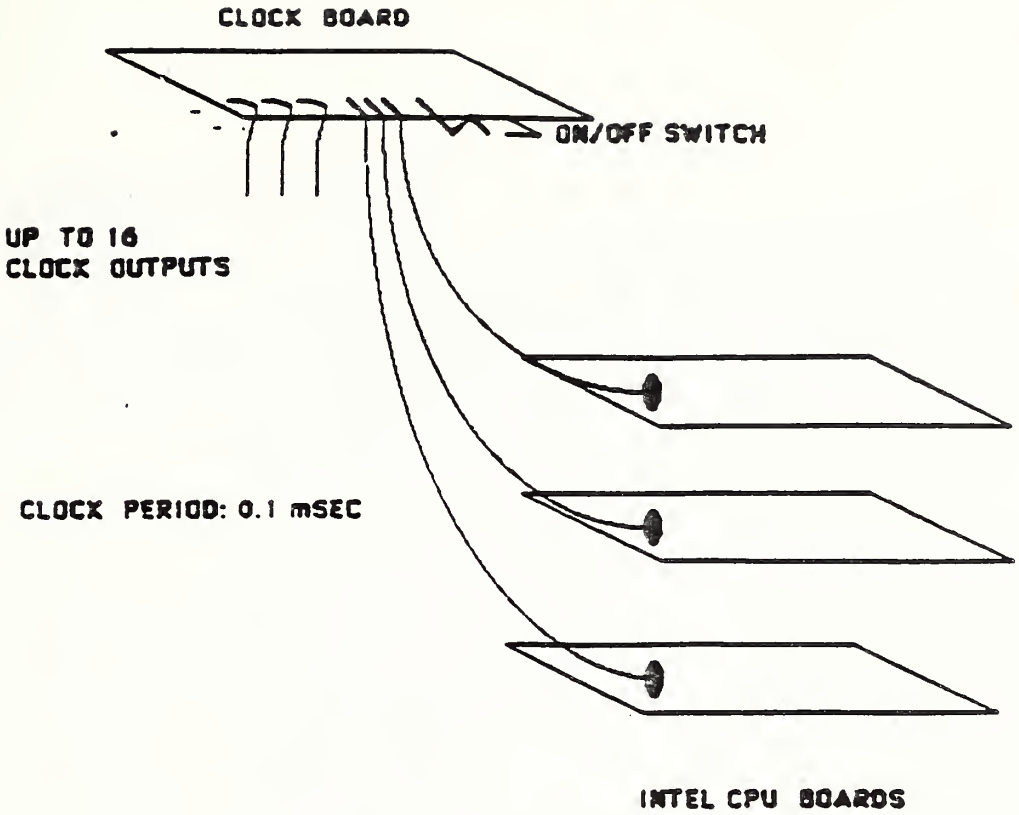
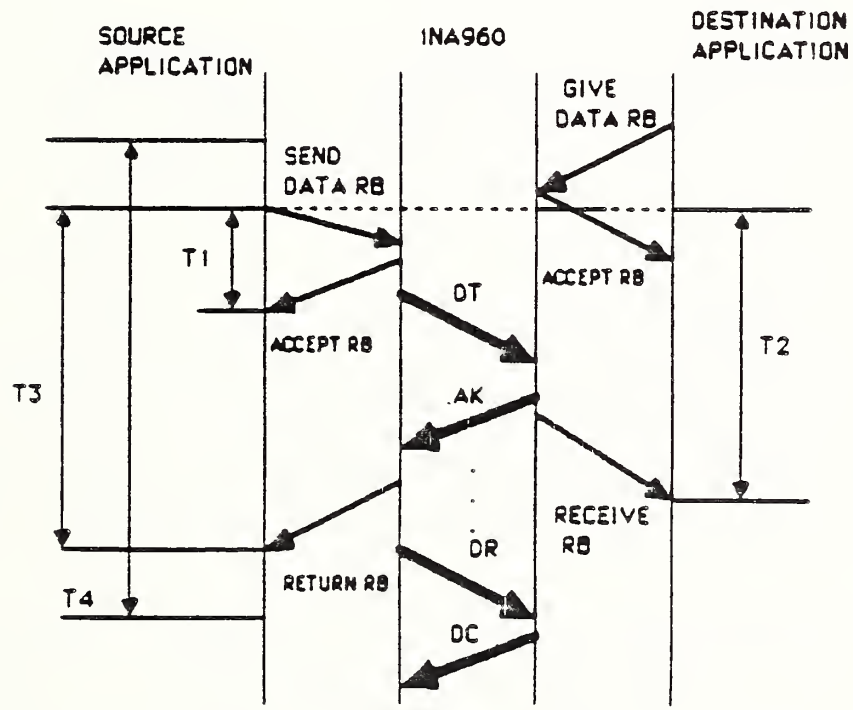


FIGURE IV-3. THE GLOBAL SYNCHRONIZED CLOCK SYSTEM



T1: RB accept delay
T2: User message delay
T3: RB return delay
T4: Experiment duration

FIGURE IV-4. USER DELAYS MEASURED

Table IV-1. Traffic Generation Variables

Application Priority
Inter-message Delay
Duplex or Simplex Data Flow
User Message Size
Total Data Transferred
Number of Connections
Number of User Buffers

B. Throughput Profile

The throughput profile shows total measured throughput under a variety of conditions. Simplex and full-duplex data flows are considered and a flow control problem is discussed. User message sizes are always 10K octets. To achieve the best throughput the value for the adaptable retransmission timer parameters had to be increased as the number of connections increased at each node. This adjustment is required because the apparent round-trip time increases as the load increases in each node.

Simplex Transfer

Figure IV-5 shows the total throughput measured during simplex data transfer, between two Intel 310 systems, as the number of buffers per connection is varied. Measures are shown for one, two, three and four transport connections. The minimum values for the adaptable retransmission timers used for each experiment are given below (Table IV-2). Throughput ranged between 60 Kcps and 108 Kcps. With only two buffers per connection end-point available, throughput is increased (Figure IV-5) by adding connections because unused CPU capacity is available within the system. Once four buffers are available per connection, the unused capacity is reduced and the overhead associated with connection scheduling becomes evident. Little throughput difference was observed between three and four connections.

Full-Duplex Transfer

Figure IV-6 shows throughput measured when the experiments were repeated using full-duplex data transfer. Retransmission timer values used are shown in Table IV-3. Throughput ranged between 90 Kcps and 104 Kcps.

Flow Control Problem

During the throughput experiments a problem was found with the combination of the OSI transport protocol operating over a type 1 class 1 logical link control protocol. The problem is illustrated using the two throughput curves shown in Figure IV-7. One curve (single sender) shows a pair of identical machines engaged in a two-connection simplex data transfer. As the number of transport buffers per connection increases, the throughput increases toward 104 Kcps. No matter how many transport receiver buffers are offered, the receiver's link buffers cannot be overrun because only two machines are involved and both machines have identical processing capabilities.

The second curve on Figure IV-7 (two senders), indicates what happens if the sending machine is faster than the receiver or if two machines are sending to one receiver. As the number of transport receive buffers per connection increases, the throughput decreases toward 35 Kcps. The lost throughput occurs because the transport flow control window, a direct reflection of the number of receive buffers, allows the link level buffers of the receiver to be overrun, invoking transport layer retransmission procedures. Use of link layer flow control would solve this problem; otherwise, transport layer buffer decisions must be made by accounting for link layer buffer conditions.

Table IV-2. Retransmission Timer Values for Simplex Throughput

Connections	Minimum(Secs.)	Starting(Secs.)
1	.256	.512
2	.512	1.024
3	.819	1.638
4	1.024	2.048

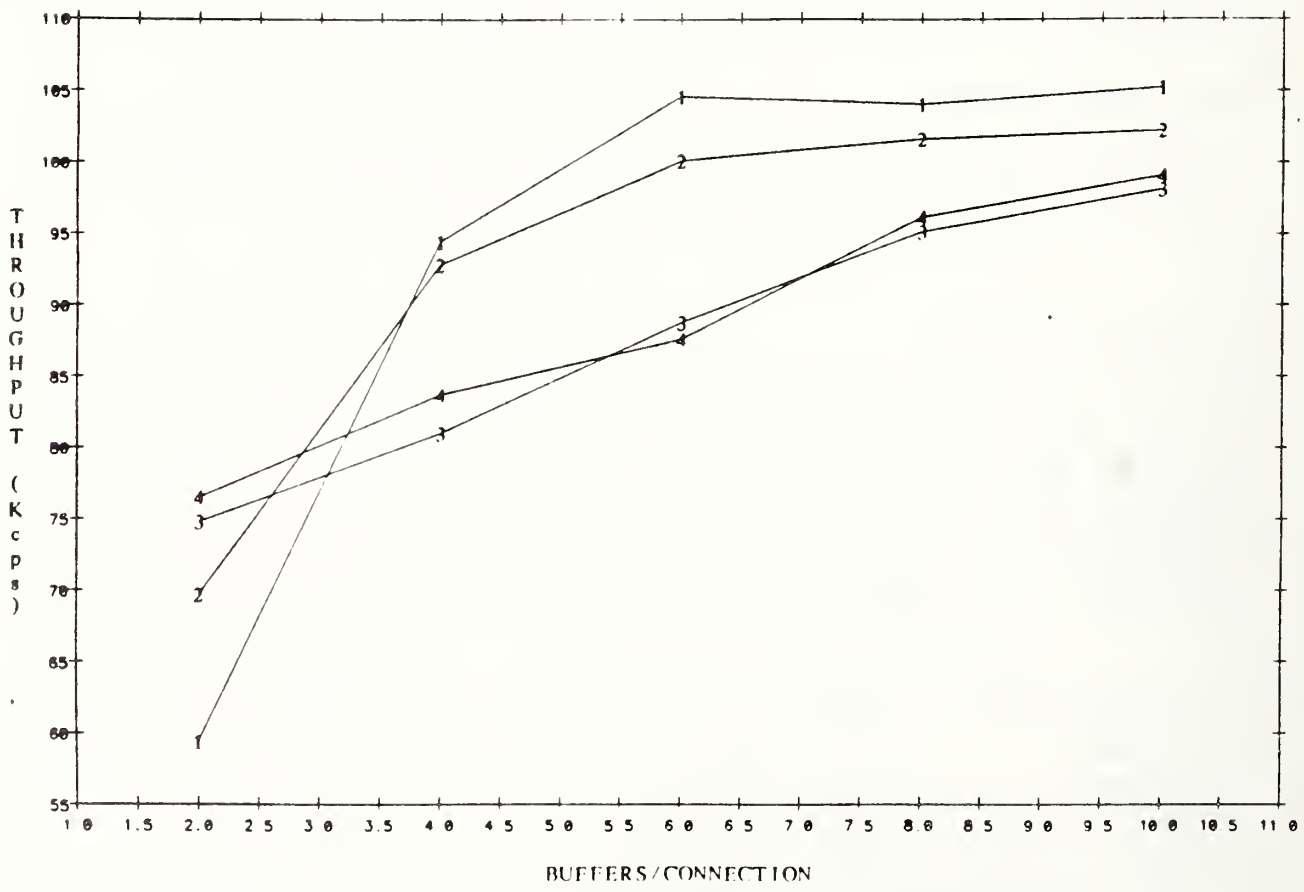


FIGURE IV-5. SIMPLEX THROUGHPUT PROFILE

Table IV-3. Retransmission Timer Values for Full-Duplex Throughput

Connections	Minimum (Secs.)	Starting (Secs.)
1	.512	1.024
2	1.024	2.048
3	1.638	3.276
4	2.048	4.096

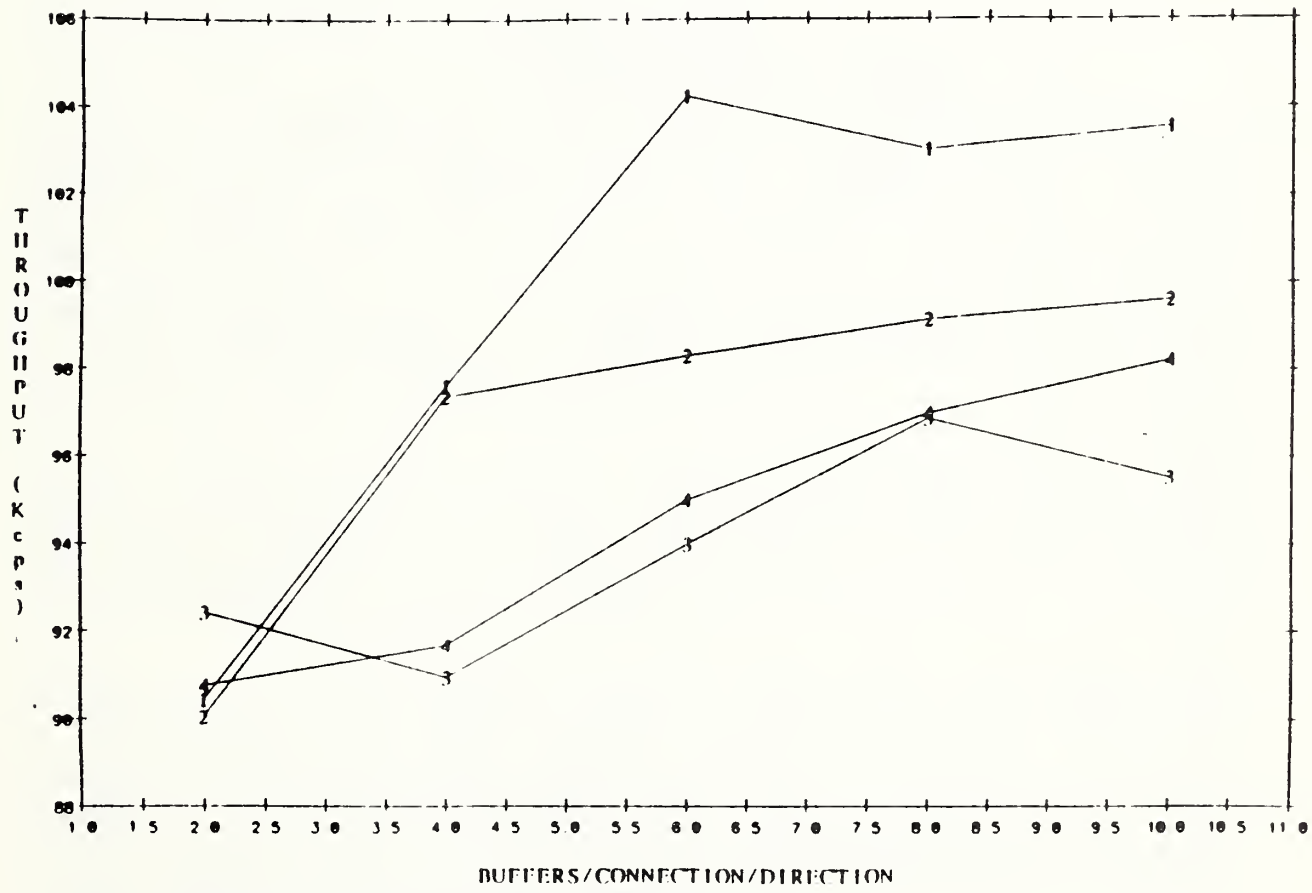


FIGURE IV-6. DUPLEX THROUGHPUT PROFILE

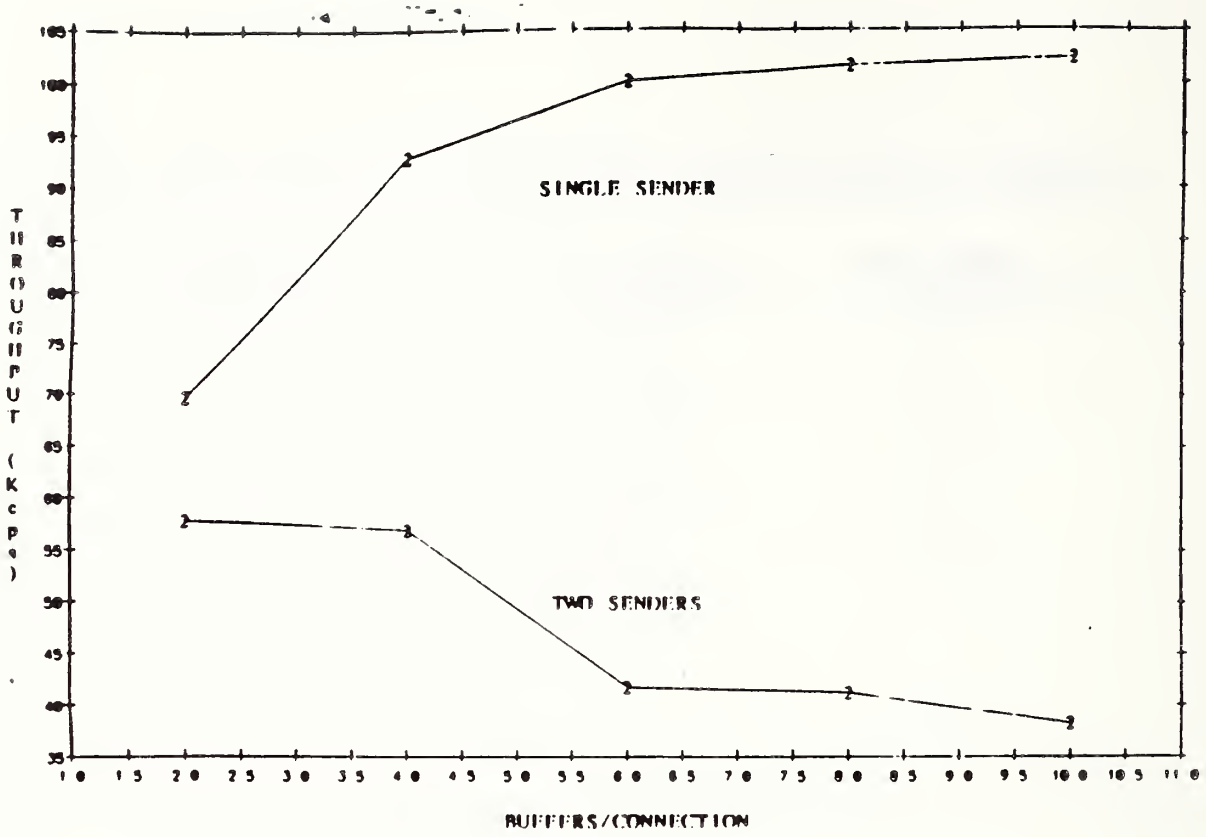


FIGURE IV-7. RECEIVER OVERRUN

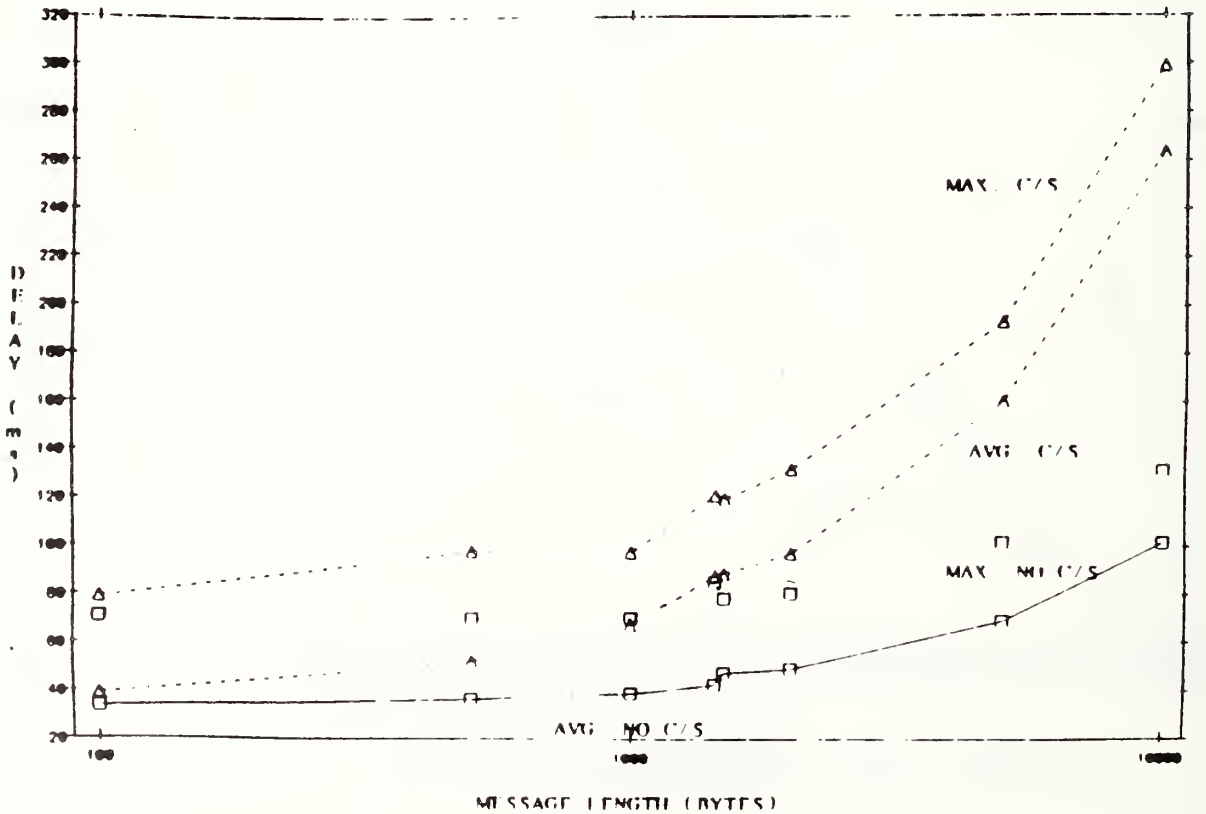


FIGURE IV-8. ONE-WAY DELAY PROFILE

C. One-Way Delay Profile

This section presents a profile of one-way user message delays (T2 in Figure IV-4) measured under a variety of conditions. In all of the delay experiments, the user message size is varied between 100 and 10,000 octets. However, when a user message is large, protocol segmenting is required because each link packet will hold only 1500 data octets. The sending user on each connection submits one message and waits for an acknowledgement indication before submitting the next message. This stop-and-wait operation limits the overall load on iNA-960 during the delay experiments.

Single Connection Delays

Figure IV-8 presents measured one-way delays with and without checksum enabled. The message transfers occur over a single connection in a single direction. The receiver allocates three transport receive buffers so that no delay is incurred for closing and reopening the transport flow control window. The lowest delays obtained occur with 100-octet user messages and no checksum, 33.5 ms average and 70.5 ms maximum. The addition of the checksum raises the lowest delays to 38.4 ms average and 78.8 ms maximum. As expected, the effect of the checksum on delay is more significant as the message size increases.

Multi-connection Delays

Figures IV-9 and IV-10 illustrate the effect of multi-connection traffic on one-way delays. For the results in Figure IV-9 the receive buffers are limited to one per connection. Thus, the effect of closing and reopening the transport flow control window is evident. The lowest one-way delays are obtained with a single connection and 100-octet messages, 45.3 ms average and 88.6 ms maximum. This means that, on the average, 11.8 ms is required to handle reopening of the transport flow control window (comparing Figure IV-9 with Figure IV-10). In the maximum case, 18.1 ms is required. While this overhead increases delay at small message sizes, it serves to reduce the one-way delay as the message size increases. Forcing the extra delay to reopen the window on each connection reduces the increase in traffic intensity normally associated with larger user message sizes.

Figure IV-10 illustrates the same experiment with three transport receive buffers allocated to each connection. As the load increases for two, three, and four connections, the average and maximum one-way delays increase significantly. The upper bounds on one-way delay in the previous case were 443.2 ms average and 904.5 ms maximum. The upper bounds in this experiment are 1395.3 ms average and 2076.0 ms maximum.

An increase of this magnitude is almost certainly due to a queuing delay incurred at the receiving user program. The user program submits empty receive buffers to and accepts filled received buffers from iNA-960. As configured, iNA-960 gives a higher priority to processing of transport operations, including passing filled receive buffers to the user, than to accepting empty receive buffers from the user. Therefore, a user's receive queue grows during periods when the user program is blocked waiting for iNA-960 to accept an empty receive buffer. This effect is demonstrated by an

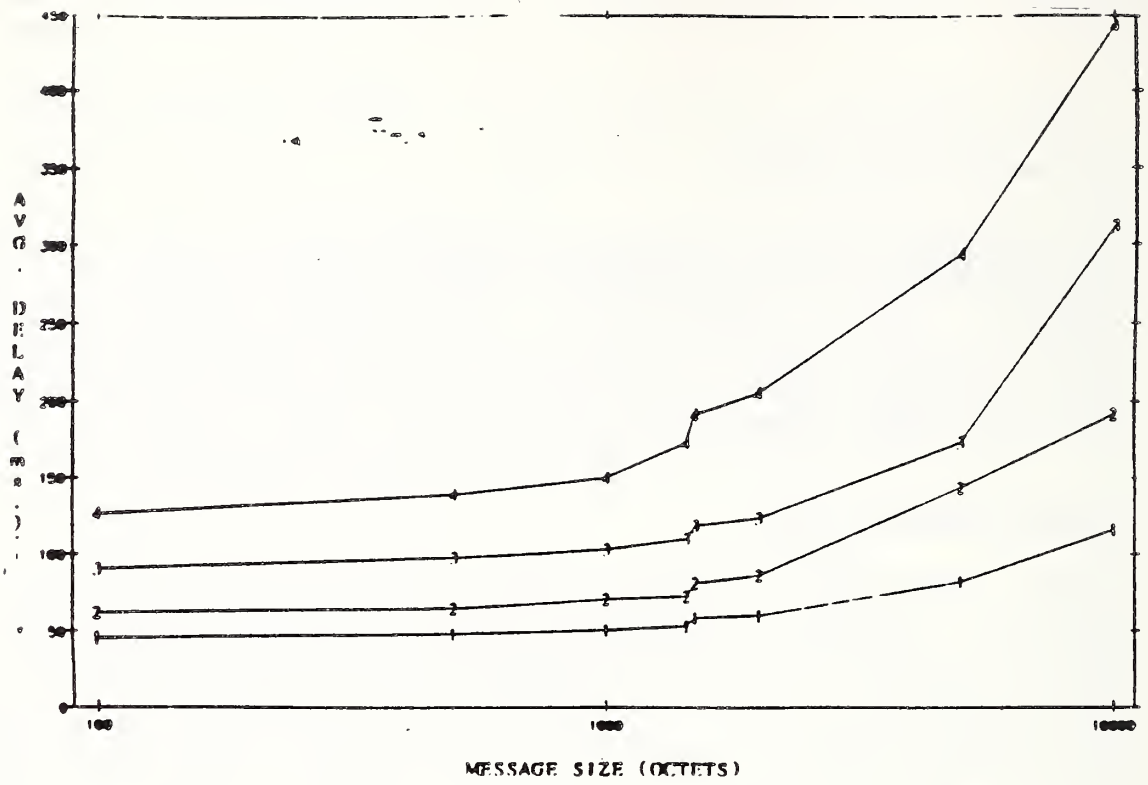


FIGURE IV-9. MULTI-CONNECTION DELAY PROFILE WITH TAUT FLOW CONTROL

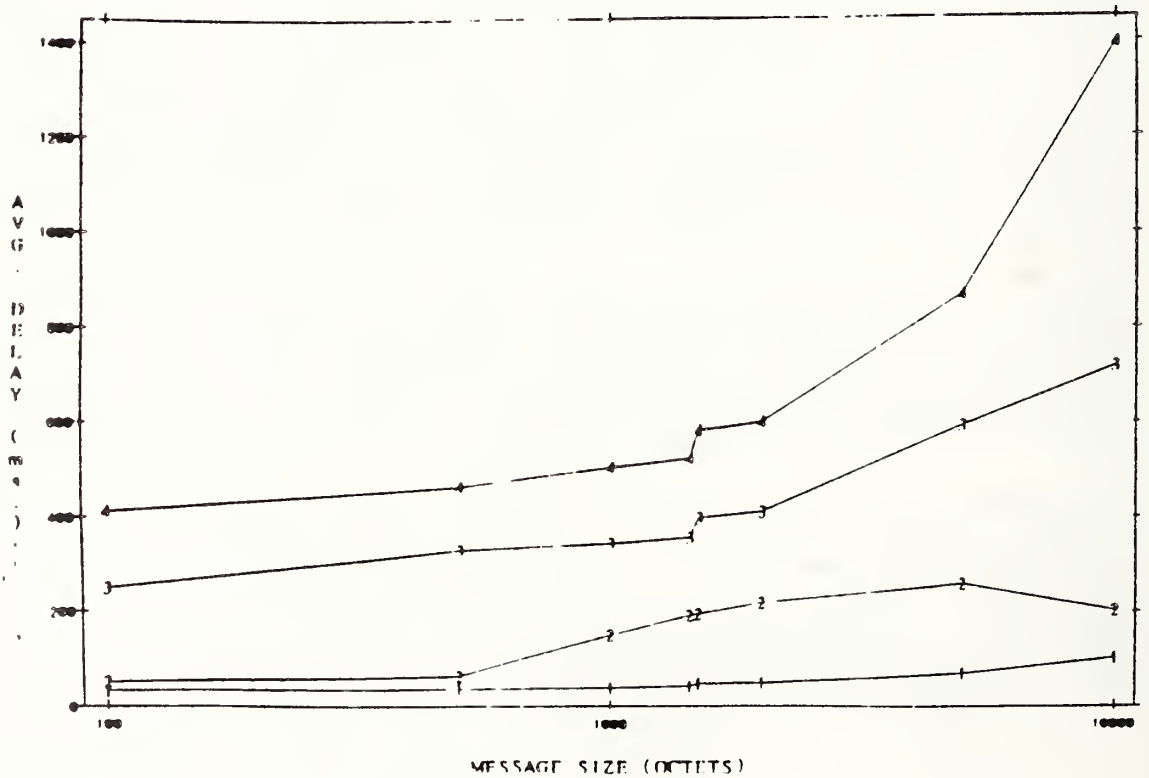


FIGURE IV-10. MULTI-CONNECTION DELAY PROFILE WITHOUT TAUT FLOW CONTROL

increasing request block accept delay (T1 in Figure IV-4) as iNA-960 traffic intensity increases. This effect might be reduced if the MIP task on the 186/51 is run at a higher priority than the iNA-960 task.

D. Multi-Application Profile

The next set of experiments involves a pair of traffic generation tasks on each of two Intel 310 systems. The first pair of tasks is generating bulk data traffic. The second pair of tasks simulates a status report application, submitting messages at a rate sustainable by the system so that no queuing delay is included. The load caused by the bulk data transfer is controlled by varying, between 400 octets and 40K octets, the size of transmit and receive buffers. Status report messages are fixed at 100 octets. The operating system priority of the status reporting task is higher than that of the bulk data task.

Figure IV-11 shows the experiment results when the data flow for both connections is in the same direction. The abscissa plots throughput of the bulk data transfer. The ordinate plots the average one-way delay for status report messages. Ideally the status report message delays (average and maximum) will be kept near the lowest delays available from the system. These target delays are superimposed on the graph with dashed lines.

The results show that the status report delays increase in a pattern similar to that seen when message sizes increase (Figure IV-10, two connections) though the status report messages do not increase in size. Also note that the lowest average and maximum delays are twice the ideal. These results represent unacceptable behavior for applications requiring real-time response. The only control mechanism available in the OSI transport standard is the allocation of buffer space. Therefore, iNA-960 does not provide priority scheduling for transport layer connections and the MIP implementation contains no multi-queue mechanism. Thus, the operating system task priority is not complemented by necessary control mechanisms in the communication system.

Figure IV-12 gives the results of the same multi-application experiment except that status reports and bulk data flow in opposite directions. These results show the lowest average delay is five times the ideal, while the lowest maximum delay is three times the ideal. Although these results are much worse than for the simplex case, the delays do not rise much as the bulk data throughput increases.

V. Conclusions

OSI protocol standards, as now specified, do not provide adequate mechanisms for guaranteeing real-time performance for selected connections or messages. This weakness in the standards is demonstrated by the iNA-960 multi-application profile where high throughput transport connections dominate the available resources forcing up the delay on all connections.

The performance measurements reported suggest limits to the traffic that can be served by first generation OSI protocol implementations. For iNA-960, a typical OSI transport layer product, an upper bound on throughput of 108 Kcps

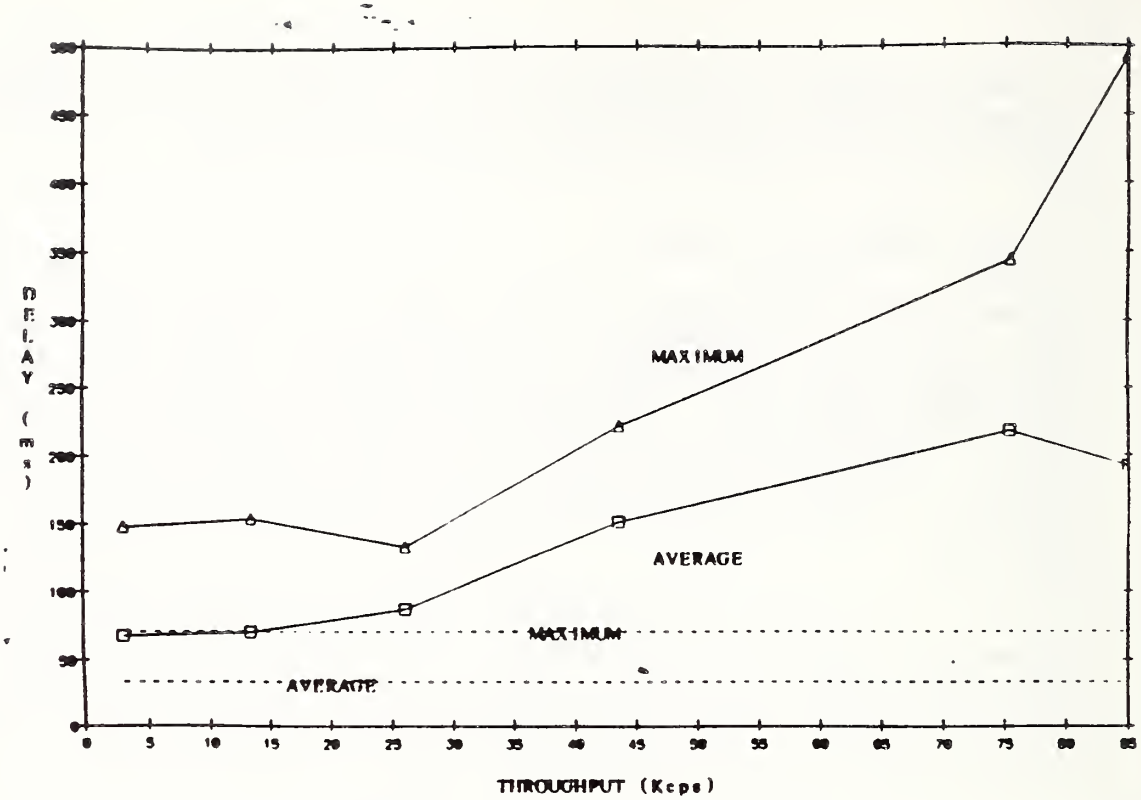


FIGURE IV-11. DUAL-APPLICATION (SIMPLEX)

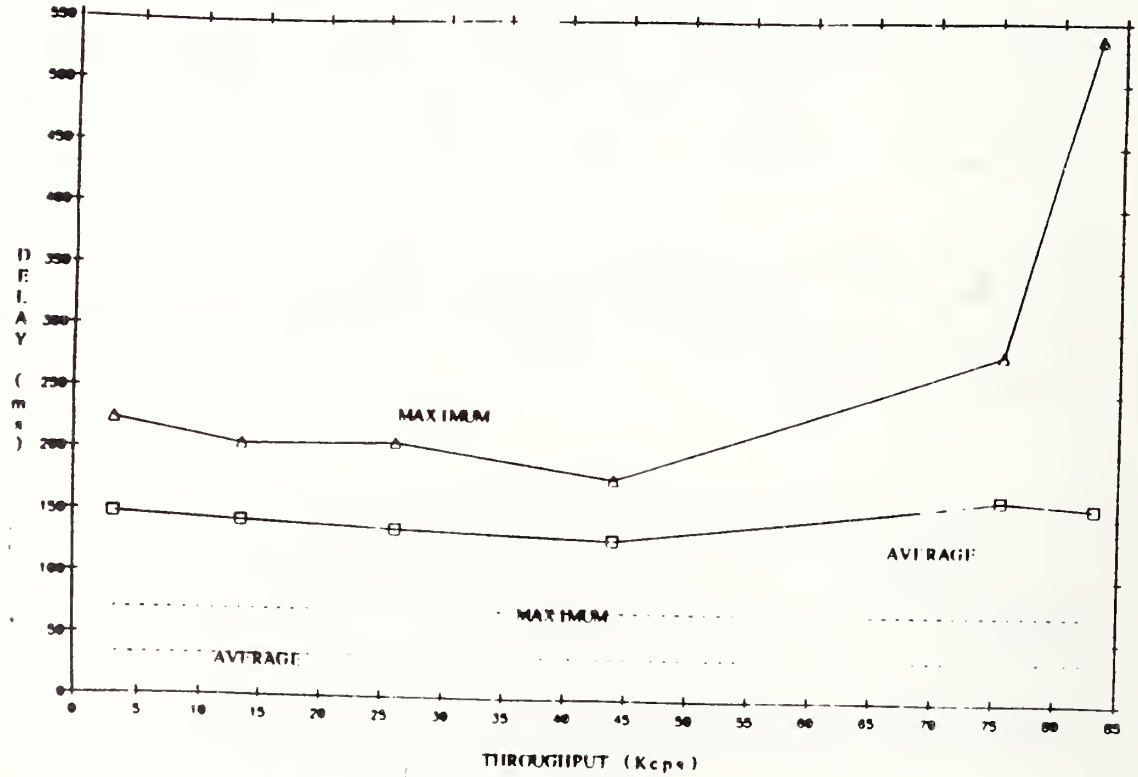


FIGURE IV-12. DUAL-APPLICATION (DUPLEX)

and a lower bound on delay of 33.5 ms were measured. The measured performance decreased as the number of connections increased. A flow control problem may occur when an OSI transport protocol is used over a type 1 class 1 logical link control protocol.

As long as "full" MAP cannot meet the complete hierarchy of performance requirements for factory communications solutions like EPA are inevitable. The EPA, with fewer protocol layers and a potential for application access to priority queues, has better properties for achieving real-time communication. Although the real-time performance of EPA is expected to be much better than with "full" MAP, this has not been demonstrated. The LLC 3, with inherent conflicts between token passing and contention, may prove difficult to implement successfully and may not provide, deterministically, fast data transfer between devices from multiple vendors. MMS may possibly turn out to be an important performance bottleneck. The performance of EPA needs to be investigated in a manner similar to that reported here for OSI protocols.

References

[ALB81] J. Albus, et al., "Theory and Practice of Hierarchical Control," Proceedings of IEEE 1981 Fall Compcon, National Bureau of Standards, 1981.

[CCI84] Presentation Syntax, CCITT Recommendation X.409, 1984.

[EIA85] Manufacturing Message Format Standard, EIA Standard 1393A, 1985.

[IEE84] Logical Link Control, IEEE Standard 802.2, 1984.

[IEE85] Token-Passing Bus Access Method, IEEE Standard 802.3, 1985.

[INT83] Guide to Using the iSBC 286/10 Single Board Computer, Intel Corporation, 14627-001, Santa Clara, CA, 1983.

[INT84A] iNA 960 Programmer's Reference Manual, Intel Corporation, 122193-001, Santa Clara, CA, 1984.

[INT84B] iSBC 186/51 COMMputer Board Hardware Reference Manual, Intel Corporation, 122136-002, Santa Clara, CA, 1984.

[ISA72] Proway Local Area Network Standard, ISA dS 72.01.

[MIL85] K. Mills, J. Gura, and C.M. Chernick, Performance Measurement of OSI Class 4 Transport Implementations, NBSIR 85-3104, January 1985.

IEEE 802.4 Conformance Test Project

January 15, 1987

Hsin-Hwai Chen
General Motors
Advanced Engineering Staff
GM Technical Center
Manufacturing Building A/MD-39
30300 Mound Road
Warren, Michigan 48090-9040
(313) 947-0558

Wayne A. Mack
VANCE Systems, Inc.
Dulles Business Park
3901-V Bonanza Boulevard
Chantilly, Virginia 22021
(703) 471-9402

ABSTRACT

Conformance testing of OSI network protocols provides end users with a level of assurance that multi vendor products will interoperate. The token passing bus protocols defined by the IEEE 802.4 standard, layers one and two of the OSI protocol reference model, requires a specialized media access control (MAC) test facility to adequately test vendor implementations. The test facility includes a programmable MAC machine, a data collector, a test controller, a headend remodulator, and the vendor supplied interface mechanism. This paper describes the test system and the test methodology and scenarios used with the test system to demonstrate MAC conformance.

Test suite generation, for conducting tests on the test system, is based on a hierarchical design structure. Using the IEEE 802.4 protocol standard and the test philosophy discussed in this paper, a suite of tests, known as the conformance test scenario, are developed. The test scenario encompasses the entire test system, providing individual test scripts for each of the components of the test system. The test scripts are implemented in the test languages of the test system components and translated into executable binary codes by the appropriate compilers.

Within the conformance test scenario, four testing phases are identified: 1) frame formation testing, 2) data transfer testing, 3) access control machine (ACM) state transversal testing, and 4) parameter adherence testing. The frame formation tests verify the functionality of the MAC transmit machine. The data transfer tests verify the operation of the MAC sublayer services which support the logical link control (LLC) sublayer. The ACM state transversal tests verify the functionality of the medium access control mechanism including token passing, token use, ring maintenance, error detection, and error recovery. The parameter adherence tests verify that all programmable variables can be set by the station management entity and manipulated by the ACM state machine properly.

A methodology for developing the ACM state transversal test scenarios is also described in this paper. To achieve completeness, the ACM state machine function is categorized into seven areas. In each area, all transitions and all possible combinations of inputs and internal variables which can cause state transitions are covered. All changes of the ACM state variables caused by the execution of state transitions are verified by the tests.

Additionally, this paper describes the test procedures that are used to execute the test scenario using the test system.

Introduction

IEEE Standard 802.4 specifies a communications protocol using a token passing algorithm and a broadcast bus medium. The standard specifies the physical layer and part of the data link layer, known as the medium access control (MAC) sublayer, of the International Standards Organization (ISO) Open Systems Interconnection (OSI) seven layer reference model. The remainder of data link layer, known as the logical link control (LLC) sublayer, is specified by a separate standard - IEEE Standard 802.2.

General Motors has selected the IEEE 802.4 protocol for use in its Manufacturing Automation Protocol (MAP) because of its deterministic characteristics. Since the token passing bus protocol is both a complex and a new and developing technology. GM has sponsored a cooperative research associate program at the National Bureau of Standards (NBS) since 1984. This program is known as the IEEE 802.4 Conformance Test Project. The goal of this program is to develop confidence and competence in the IEEE 802.4 token bus technology prior to its use in large factory installations.

One of the key activities involved with achieving this goal is IEEE 802.4 MAC-Sublayer conformance testing. These conformance tests will be used to verify that the protocol implementations of various local area network vendors conform with the specification. This provides assurance that various vendors' equipment will interoperate within a network and that the network will operate in the expected manner. The intent of the MAC-Sublayer Conformance Test Project is to develop the test system. methodology, scenarios and procedures required to demonstrate IEEE 802.4 MAC-Sublayer conformance.

Objectives

There are several interrelated objectives for MAC-Sublayer conformance testing. These objectives establish a philosophy which is carried throughout the test system and procedures for conformance testing. These objectives are:

1) All tests shall be performed in a manner which is non-intrusive to the vendor's implementation. The vendor implementation being tested will be allowed to operate as a stand-alone entity, reducing the risk of errors being induced by the test equipment.
2) No special interfaces to the vendor's device will be required. The control of parameters, LLC data requests, and LLC data interpretation shall only be achieved through a vendor supplied mechanism. The results of tests may only be determined through the observation of IUT outputs under the influence of an emulated bus test scenario.
3) The conformance tests shall be kept independent of the adjacent layers. Only the MAC-Sublayer is being tested and not the LLC Sublayer, Physical layer, or Station Management.
4) The test methodology and procedures shall be described through a high level language such that the descriptions can remain independent of the test tools, yet provide sufficient detail to implement the test bed without ambiguity.
5) The tests shall be easily upgradable to support future revisions of the IEEE 802.4 standard.
6) Conformance testing will be developed with interoperability as a goal.

MAC-Sublayer Architecture

The IEEE 802.4 MAC-Sublayer provides the mechanism for accessing and moving data, reliably, over the physical medium of the network. The MAC-Sublayer supports interfaces to the Physical layer, the Logical Link Control sublayer, and Station Management. The Physical layer resides below the MAC-Sublayer and provides a broadcast communications medium in a bus topology. The Logical Link sublayer resides above and is supported by the MAC-Sublayer. Station Management supports local administrative services between the MAC-Sublayer and its manager, providing the means and methods for initialization, network parameter reading and setting. address validation and setting, network membership control, monitoring and reporting of changes to the MAC entity's parameters, and supporting the exchange of MAC service data units for station management peer communications.

Internally, the architecture of the MAC-Sublayer is partitioned into five machines as illustrated in Figure 1. These machines are: 1) the Receive Machine, 2) the Transmit Machine, 3) the Interface Machine, 4) the Access Control Machine, and 5) the Regenerative Repeater Machine.

The Receive Machine accepts MAC-symbols from the physical layer, integrates the MAC-symbols into MAC-frames, checks the validity of the frames, passes valid data frames to the Interface Machine and valid control frames to the Access Control Machine, and indicates the current bus condition.

The Transmit Machine accepts MAC-frames from the Access Control Machine. appends the appropriate amount of preamble, the frame delimiters and frame check sequence, decomposes the MAC-frames into individual MAC-symbols, and passes the MAC-symbols to the physical layer.

The Interface Machine acts as an intermediary between the MAC-sublayer machines and the Logical Link Control (LLC) sublayer, and between the MAC-Sublayer machines and Station Management. It interprets and generates service primitives for incoming and outgoing MAC-frames. It also performs address recognition on data frames passed to it by the Transmit Machine.

The Access Control Machine coordinates access to the shared bus with the other stations on the network and limits its own access to the bus to guarantee deterministic operation. It performs the functions of token passing. token use. ring maintenance, ring initialization, ring fault detection, and ring fault recovery. It also accepts data frames from the Interface Machine and passes the frames to the transmit machine at the appropriate times.

The Regenerative Repeater Machine is an optional component of the MAC-Sublayer architecture which is only present in special repeater stations. Its function is to connect electrically or frequency separate bus segments into an extended logical bus network. A broadband headend is a special case of a regenerative repeater.

MAC-Sublayer Test System

The conformance test system architecture defined by the MAC-Sublayer Research Project for broadband testing is illustrated in Figure 2. This test system architecure describes the structural relationship among the individual components of the test system. These components are: 1) the Implementation Under Test

(IUT), 2) the Test Controller, 3) the Programmable Media Access Controller (PMAC), 4) the Data Collector, 5) the Modems, and 6) the Headend Remodulator.

The IUT is the vendor's product which is to be tested. To remain non-intrusive, the interface to the IUT is limited to the Physical layer interface and a vendor interface. The vendor interface is a vendor supplied mechanism which 1) allows the MAC-Sublayer programmable parameters to be controlled in the IUT, 2) invokes an IUT send by issuing a MA_DATA.request to the MAC-Sublayer, and 3) detects data reception by the IUT via a MA_DATA.indication issued by the MAC-Sublayer.

The Test Controller coordinates the actions of the PMAC, Data Collector, and IUT. Additionally, the Test Controller performs the analysis of the various tests and determines the results.

The PMAC emulates various token bus scenarios which may occur within a token bus network. The PMAC is primarily used as a stimulus, exercising the IUT and causing it to output various MAC-frames.

The Data Collector provides the main observation point for MAC-Sublayer conformance testing by monitoring the activities of the bus, storing all or select series of MAC-symbols occurring on the bus, and time stamping each of the stored symbols. This provides a log of the bus activity during conformance testing.

The Modems and Headend Remodulator provide the Physical layer for the test system. The Modem associated with the IUT is vendor supplied such that the testing is non-intrusive to the implementation.

Test Suite Generation

Having defined the test system architecture for conformance testing, the next step is to define the methodology for developing and implementing the conformance test suite. The hierarchical design structure that was adopted by the IEEE 802.4 Conformance Test Project for the conformance test suite is illustrated in Figure 4. This architecture identifies three levels of test implementation: 1) test scenarios, 2) test scripts, and 3) test code. At the top level of this architecture are the test scenarios.

The test scenario is a set of abstract test definitions and is generated using various sections of the IEEE 802.4 standard and applying the test objectives and philosophy previously stated. The test scenario encompass the entire MAC-Sublayer conformance test system and is written in a formal high level language.

Using the test scenario and the architectural description of the MAC-Sublayer test system, individual test scripts are created for each component of the test system; the test controller, the PMAC, the Data Collector, and the IUT. Each of the test scripts contains multiple tests, which correspond on a one to one basis with the tests defined in the test scenario. The test scripts are written in languages specific to each of the test system modules. Where multiple test modules are required to allow a component to perform an entire test, the modules are referred to as test script segments.

The test scripts are translated into test codes specific to, and executable by the individual MAC-Sublayer test system components. There is a one to one relationship between test script segments and test code segments.

Test Phases

Based on the architecture and the test generation methodology described above, five test phases have been identified for the conformance test suite: 1)Frame Formation Testing, 2) Noise Burst Detection Testing, 3) ACM State Transversal Testing, 4) LLC Data Transfer Testing, and 5) Parameter Adherence Testing.

Frame Formation Testing verifies the Transmit Machine. Frame formation testing requires that the MAC-frames generated by the IUT contain the following fields:

1) A preamble which consists of a series of pad_idle MAC-symbols. The preamble is primarily used by the receiving modem to acquire signal level and phase lock.
2) A Start Delimiter which indicates the end of preamble and the start of the frame.
3) A Frame Control field which defines the frame type (i.e TOKEN, DATA, etc.)
4) A Destination Address.
5) A Source Address which must be the same as the address of the transmitting MAC entity.
6) A Data Unit containing 0 to 8191 octets of LLC data.
7) A Frame Check Sequence field octets containing the CRC-32 value calculated for the Frame Control, Destination Address, Source Address and Data Unit fields.
8) An End Delimiter which indicates the end of the frame.

Noise Burst Detection Testing verifies the Receive Machine. This test requires that the IUT reject invalid frames. The IEEE 802.4 specification defines a MAC-frame which meets any of the following conditions as being invalid:

1) It is identified as such by the physical layer.
2) It is not an integral number of octets in length.
3) It does not consist of a Start Delimiter, one Frame Control field, two properly formed address fields, one Data Unit field of appropriate length (dependent on the bit pattern specified in the Frame Control field), one Frame Check Sequence field, and an End Delimiter, in that order.
4) A CRC-32 computation, when applied to all octets between the Start Delimiter and End Delimiter, fails to yield the correct remainder.

ACM State Transversal Testing verifies the Access Control Machine. The IEEE 802.4 standard defines the Access Control Machine (ACM) as a finite state machine as depicted in Figure 5. The functionality of this finite state machine is actually much more complicated than what is represented in this diagram, because the same input may trigger different transitions depending upon the current values of ACM variables. Also, transitions which change variable values will affect later state transitions. A list of the inputs to the ACM, the parameters used by the ACM and the internal variables used by the ACM is given in figure 6. The methodology used in developing the ACM state transversal tests is as follows:

1) The ACM is divided into eight functional areas: the CLAIM TOKEN algorithm, the PASS TOKEN algorithm, the USE TOKEN algorithm, the RING MAINTENANCE algorithm, the RING ENTRY algorithm, the RING EXIT algorithm, the RING COLLAPSE RECOVERY algorithm, and the

OTHER_HEARD transition.

2) Each transition emanating from the idle state is assigned to one of the seven areas, depending upon the context required to cause that transition.
3) Every transition is to be tested.
4) Every possible combination of inputs and variables that can cause a transition are to be tested.
5) In some cases, because not all of the state variables are defined for a state, it is possible that the ACM enters the same state with different values of state variables. It is not necessary to emulate all possible legitimate variable variations.
6) For those transitions that may have iterative occurrences such as ring entry, ring maintenance and claim token processes, three cases are tested: the minimum case, the maximum case and one intermediate case.
7) The change of some variable inside the ACM state machine may not be immediately observable after the transition that causes the change occurs. To verify that the variable is changed correctly, the IUT will be induced to some state where that variable will cause a subsequent transition.
8) To emulate a particular token bus environment, the test system may need to send out a sequence of invalid MAC-frames (noise_bursts).
9) The minimum delay before a station can receive a frame after passing a frame such as token, who follows, solicit_successor, or request_with_response data frame is zero.
10) The minimum delay for two consecutive frames is 2 microseconds.
11) Only legitimate events can occur.

LLC Data Transfer Testing verifies the services provided by the Interface Machine to the LLC-Sublayer. This test verifies the following capabilities:

1) the ability to send a data frame containing 516 octets of data,
2) the ability to receive a data frame containing 516 octets of data,
3) the ability to send a frame and insert the appropriate value in the SA field,
4) the ability to receive a broadcast frame,
5) the ability to receive a group addressed frame, and
6) the ability to send and receive a frame addressed to one's self.

Parameter Adherence Testing verifies the services provided by the Interface Machine to Station Management. These services include the setting of various parameters. This test verifies that the following parameters can be set by Station Management:

1) the station address,
2) the set of group addresses accepted by the station,
3) the slot_time,
4) the hi_pri_token_rotation_time,
5) target_rotation_time(4)
6) target_rotation_time(2),
7) target_rotation_time(0)
8) target_rotation_time(ring_maintenance),
9) the ring_maintenance_timer_initial_value,
10) the min_post_silence_preamble_length,
11) the in_ring_desired flag.

12) the max_retry_limit, and
13) the max_inter_solicit_count.

Conclusion

A conformance testbed has been implemented at the Institute for Computer Sciences and Technology of National Bureau of Standards to demonstrate the test methodology described here. This testbed has shown that not only is conformance testing at the MAC-Sublayer feasible, but that the testing methodology is similar to that which is used at the higher protocol layers. Thus the wealth of experience that has been gained in conformance testing at higher layers (such as the transport layer) can be utilized during the conformance testing of the MAC-sublayer in order to reduce test development time, effort and risk.

References

IEEE Standards for Local Area Networks: Token-Passing Bus Access Method and Physical Layer Specifications, ANSI/IEEE Std 802.4-1985, October, 1985.

MAC-sublayer Functioning Partitioning

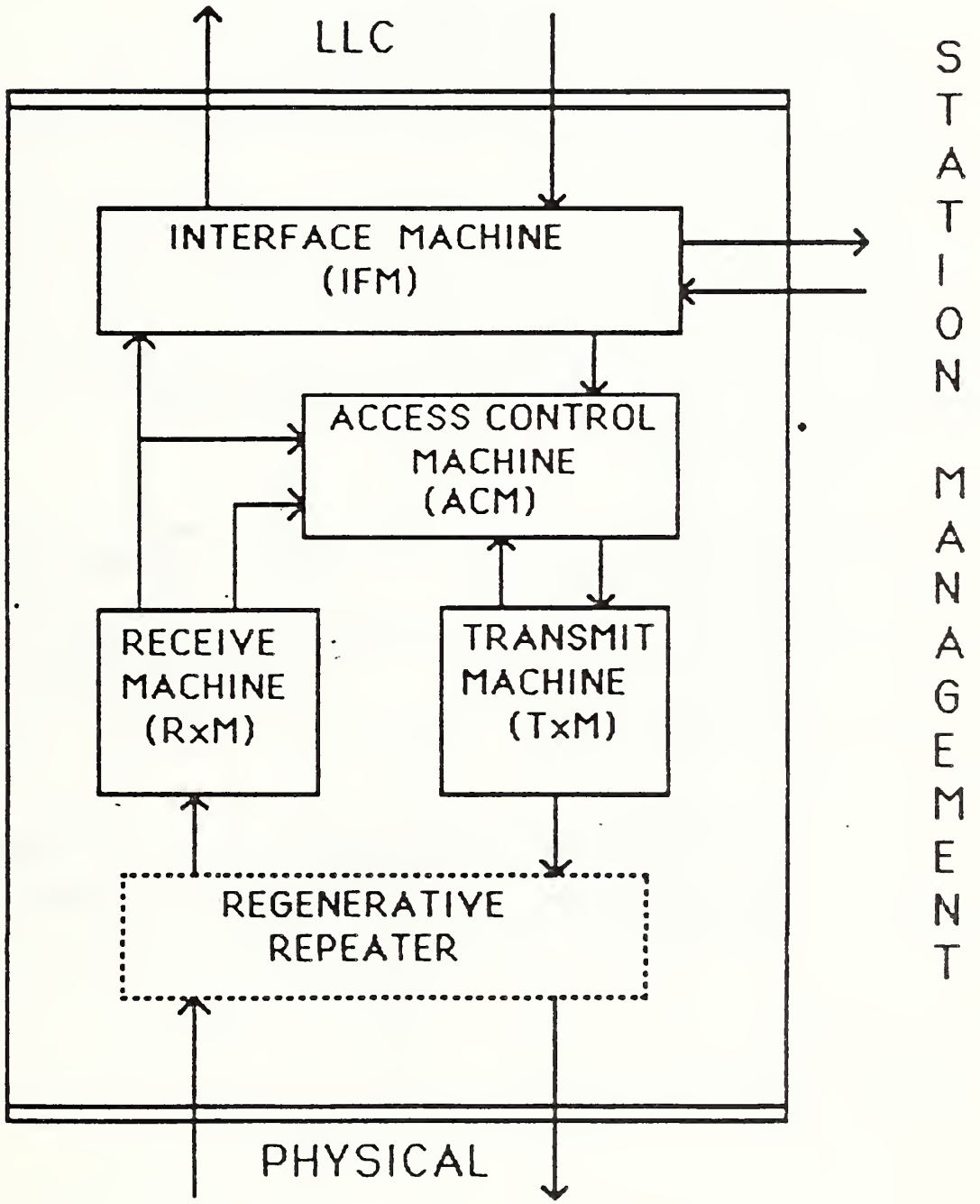


FIGURE 1

Test Harness

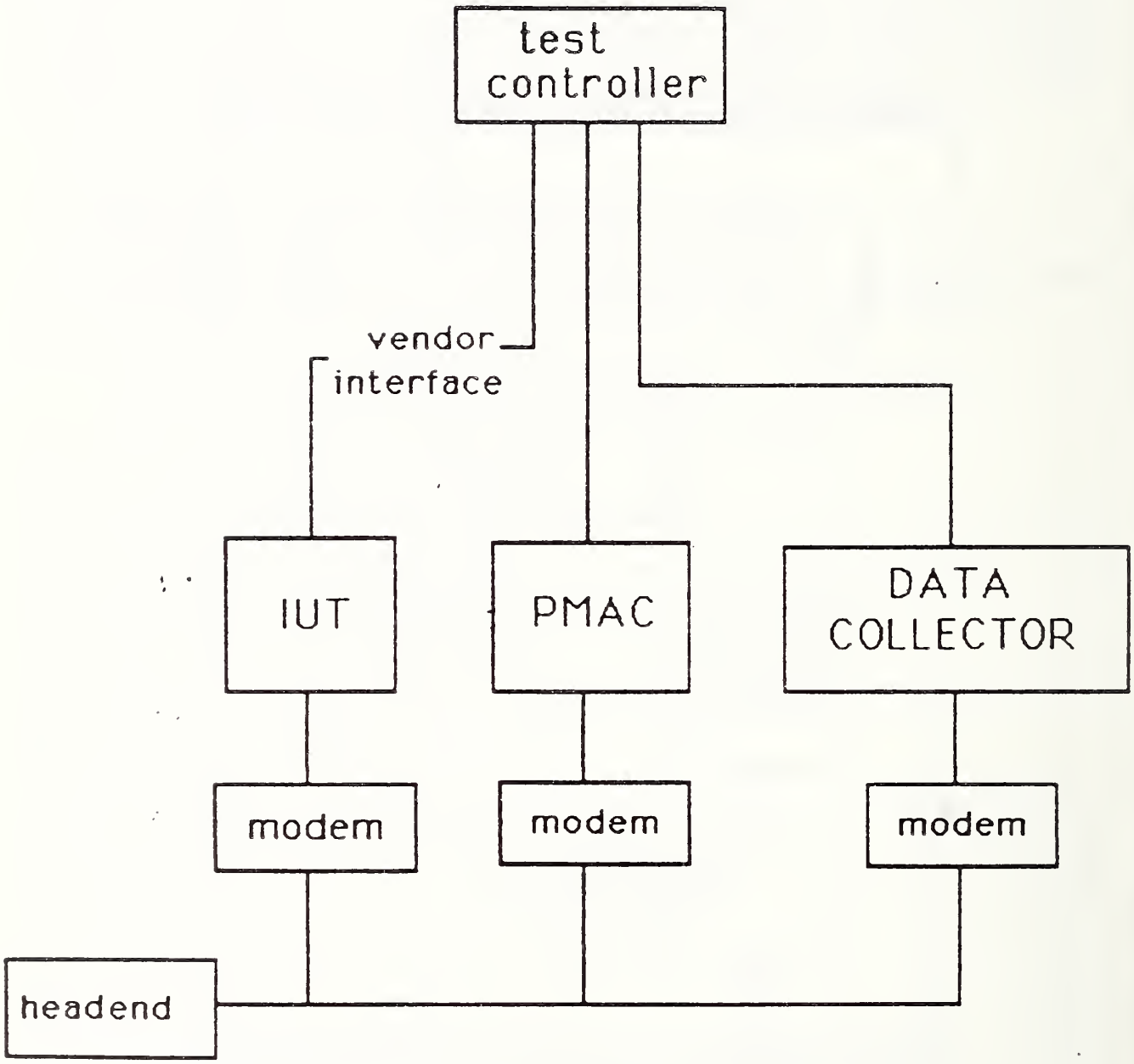


FIGURE 2

PMAC Functional Partitioning

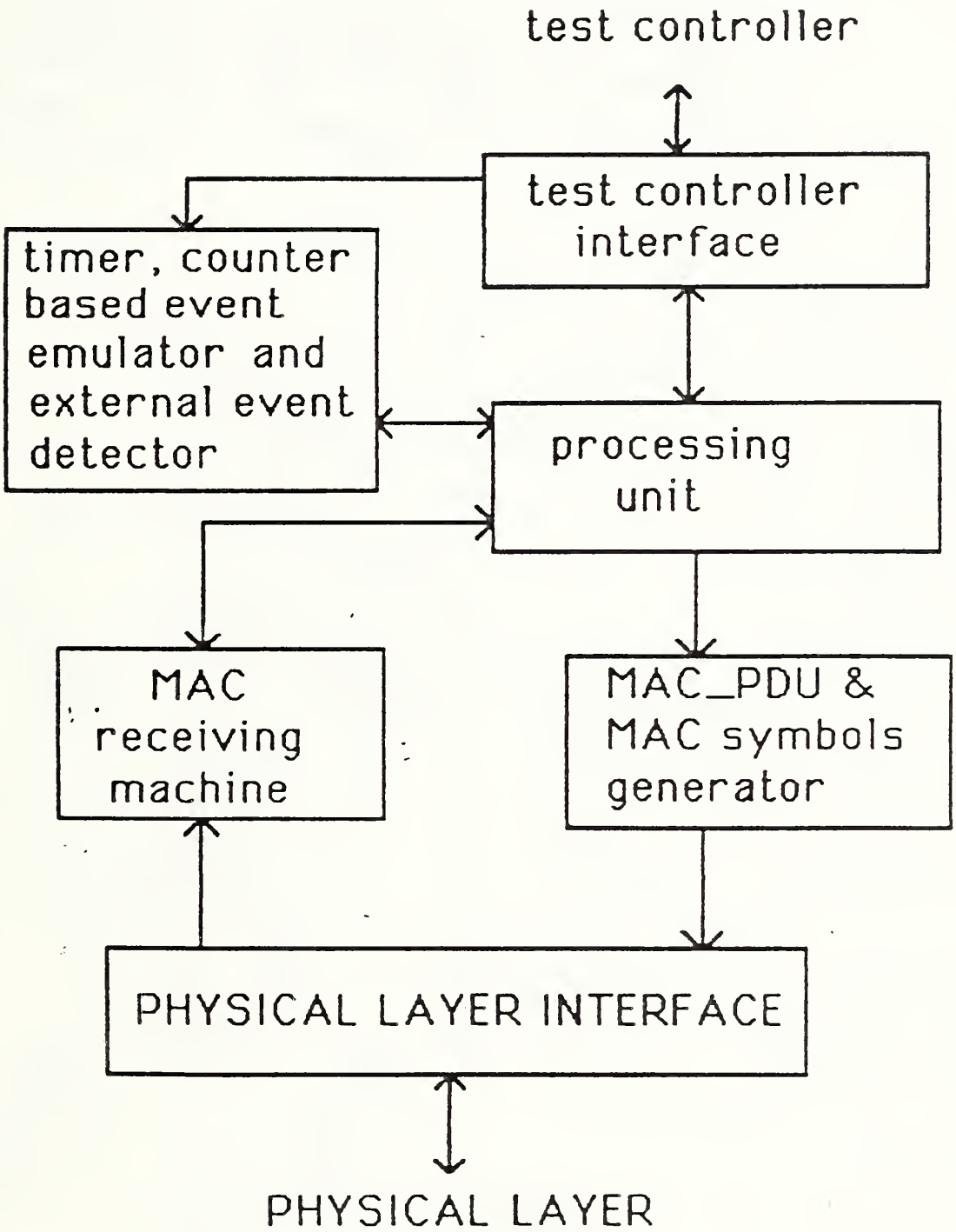


FIGURE 3

Test Code Generation

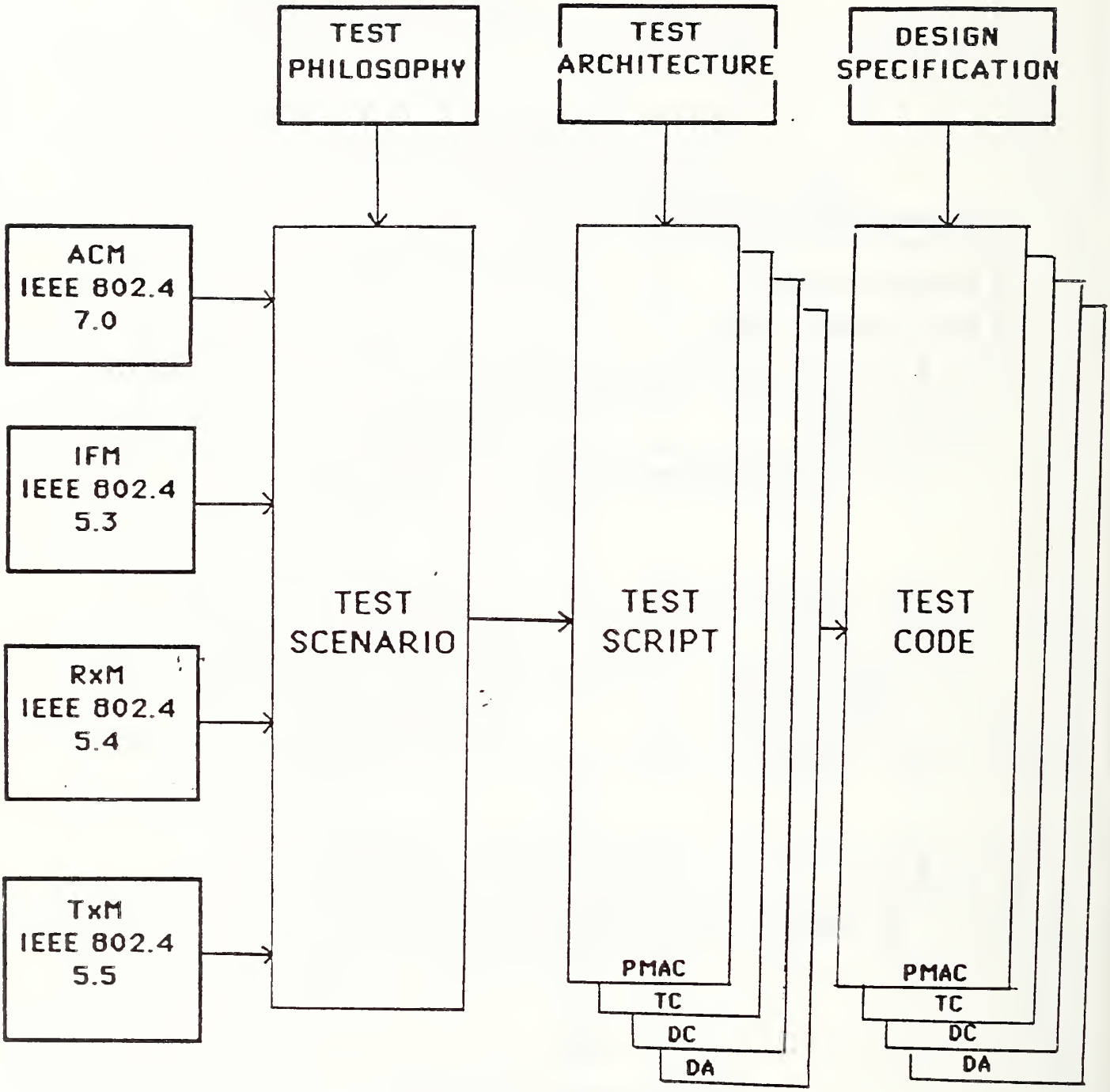
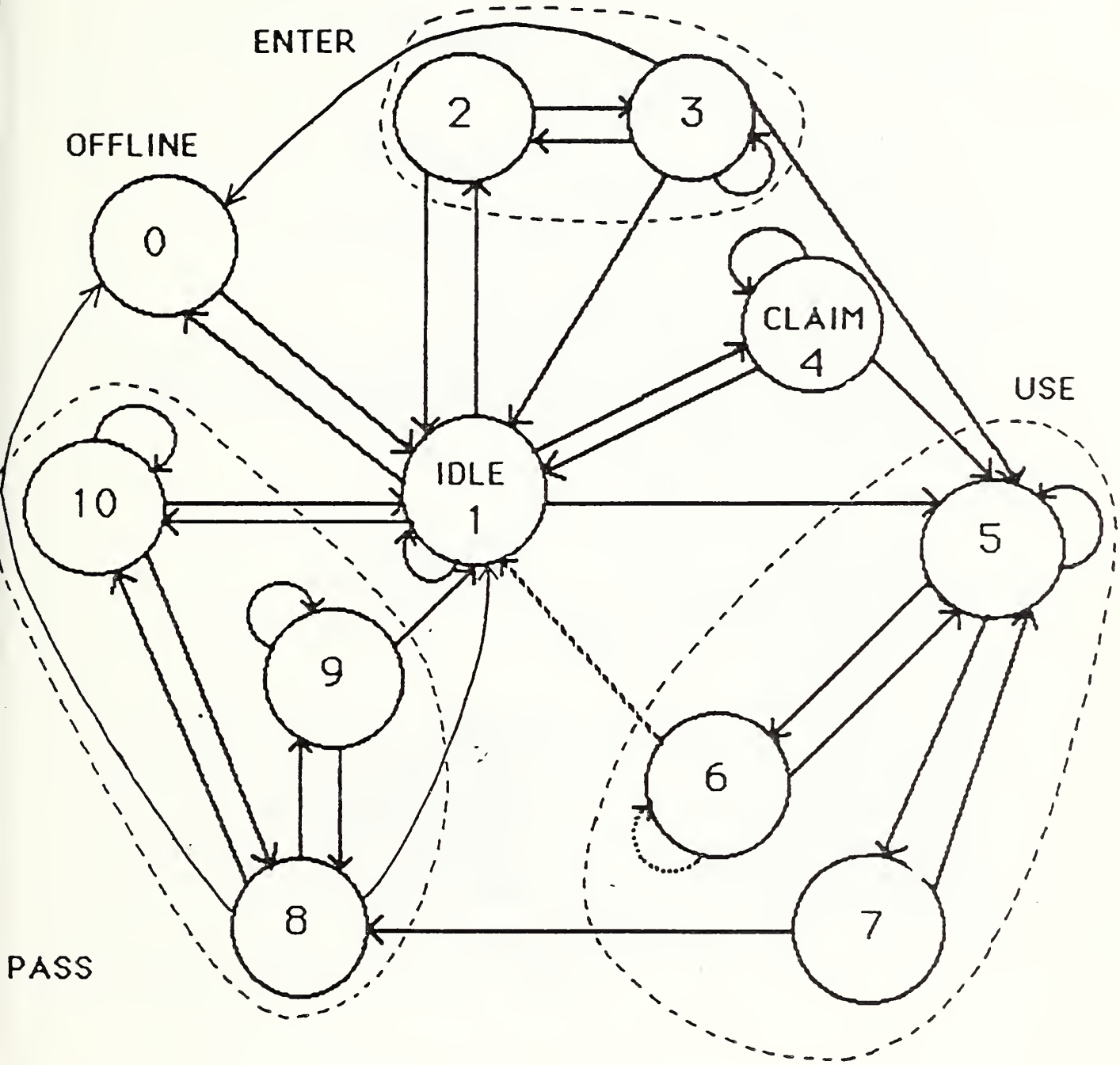


FIGURE 4

ACM Finite State Machine Diagram



0-OFFLINE
1-IDLE
2-DEMAND_IN
3-DEMAND_DELAY
4-CLAIM_TOKEN
5-USE_TOKEN
6-AWAIT_IFM_RESPONSE
7-CHECK_ACCESS_CLASS
8-PASS_TOKEN
9-CHECK_TOKEN_PASS
10-AWAIT_RESPONSE

FIGURE 5

Inputs provided by the receive machine:

rx_protocol_frame	rx_data_frame
noise_burst	bus_quiet

Inputs provided by the Interface Machine:

any_send_pending	signal_received

Inputs provided by Station Management:

power_ok	SM_ACTION.initialize

Parameters programmable by Station Management:

station address	slot_time
hi_pri_token_rotation_time	target_rotation_time(access class)
ring_maintenance_timer_init_value	min_post_silence_preamble_length
in_ring_desired	max_retry_limit
max_inter_solicit_count	

Internal ACM Timers:

bus_idle_timer	claim_timer
contention_timer	token_pass_timer
response_window_timer	token_hold_timer
token_rotation_timer(access class)	

Internal ACM variables:

NS	NS_known
access_class	in_ring
sole_active_station	lowest_address
just_transmitted	claim_pass_count
contention_pass_count	resolution_pass_count
inter_solicit_count	remaining_retries
transmitter_fault_count	first_time
echoes_expected	heard_successor

Figure 6

Clock Synchronization on the Factory Floor

Walter Gora and Ulrich Herzog

Chair for Computer Architecture and Performance Evaluation

University of Erlangen-Nuremberg, West Germany

*Satish K. Tripathi**

System Research Center,

UMIACS, and

Department of Computer Science

University of Maryland

College Park, MD 20742

Abstract

One important issue for the coordination of flexible manufacturing systems (FMS) in an automated factory is the synchronization among the manufacturing processes based on a common clock. This paper describes the synchronization requirements on the factory floor and discusses several clock synchronization algorithms, their theoretical bounds, and the results of joint work at Maryland and Erlangen. The measurement results based on the implementations of the synchronization algorithms on local area networks at Maryland and Erlangen are presented. For hierarchical LANs an algorithm is developed and its behaviour is simulated.

1. Scope - The PAP model factory

Reduction in costs, diversification and customization of products, and shortening downtime of manufacturing and assembly equipment are some of the basic requirements for the "factory of the

* This research was supported in part by National Science Foundation under grant 01R-85-00108 and by Alexander von Humboldt-Stiftung.

future". Many companies have installed new technologies, such as CAD (computer aided design), CAM (computer aided manufacturing) or CAP (computer aided planing). However, normally connection between different computer equipment and the automated devices is very limited.

The flexible fabrication of customized products requires a continuous exchange of information and materials in the whole system. Therefore, the computer intergrated manufacturing (CIM) concepts can be understood as the combination of highly flexible manufacturing and assembly cells together with specific computer hosts to a whole working system. Characteristic for such a system is the extensive use of the computer aided technologies for planing, design, testing, and controlling. Furthermore, the versability of the all components in the whole system must be stressed [Dupo82].

At the University of Erlangen-Nuremberg project PAP ("Project for Flexible Automated Production Systems") was initiated in 1985 as a common effort of both the industrial engineering and computer science departments, supported by government and industry. The goal of the PAP research project is to build a highly flexible model factory during the next five years at the campus of the university (Figure 1).

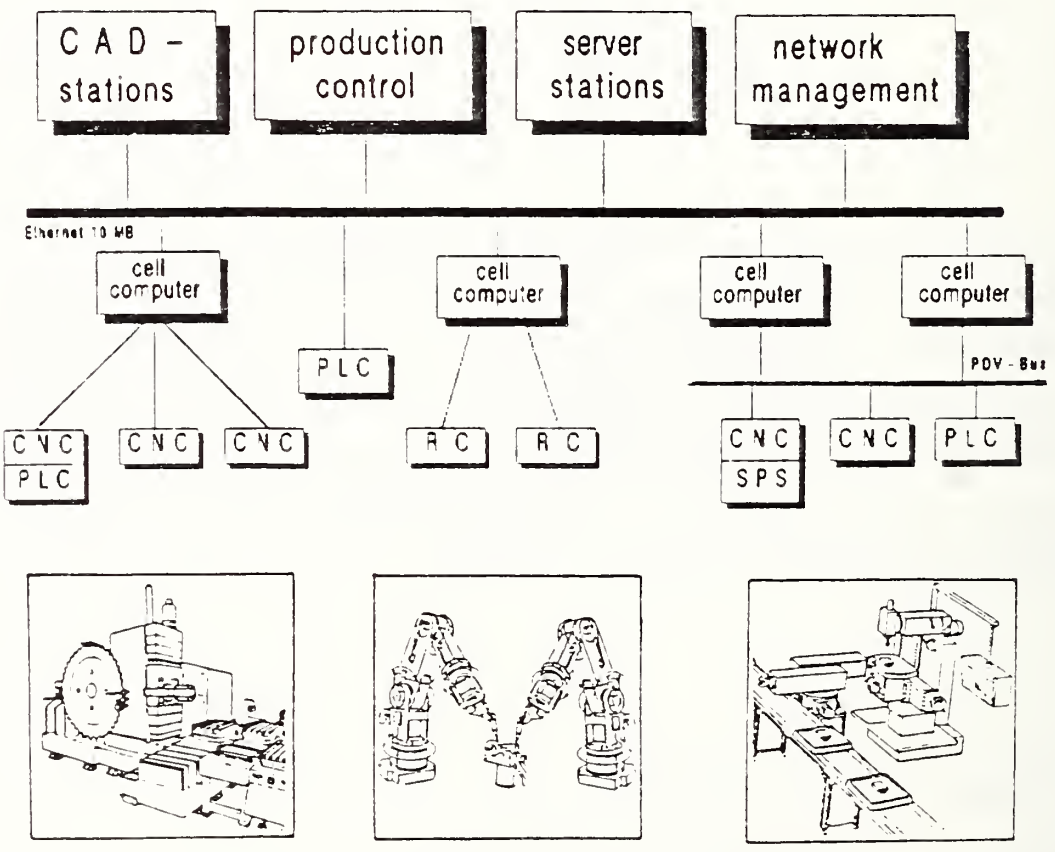


Figure 1. The model factory of the PAP project

Important issues for the communication among different devices are a unique communication architecture and standardized protocols. For the PAP model factory the MAP protocols (*Manufacturing Automation Protocols*) [GM85, GM86] are selected and being implemented.

In 1980 Generals Motors initiated the MAP project, a common effort of various manufacturers and users. The goal of the MAP project, still under development, is to establish commmunication and application protocols on the factory floor based on the International Standards Organization's (ISO) seven layer model for Open Systems Interconnection (OSI).

The requirements for the PAP communication system (Figure 1) are derived from different fields and conditions:

(1) Between several CAD working stations the exchange of graphic information must be provided in many formats.

(2) For the mechanical engineering part of the PAP project, the communication system must connect the automation equipment as programmable logic controllers (PLC), robots, and numerically controlled machines (NC) directed by a computer to a flexible manufacturing cell (FMC).

(3) It is also essential that information about a product can be stored, transferred, and used by different machines at various places in a manufacturing and production system. Therefore, a complex network between the automation equipment and special purpose computers like print servers, database hosts, programming stations, and planning systems must be provided.

(4) Distributed control systems must support realtime response, when a part of the production system is down. Thus the control systems must have an immediate access to the manufacturing cells for their status.

(5) Several tasks must be provided by network management. One task is controlling and supervising the whole communication system. Also statistics about events in the distributed system and performance analysis are requirements for the network management.

In an error free environment the production can easily be planed and synchronized. But in reality, inherent system uncertainties and unforeseen situations must be handled [Hatv84]. One important problem on the factory floor is the electronic noise generated by the automation equipment, which can interfere with network transmission. If the exchange of information between the

manufacturing cells is delayed or interfered with, the whole production can be slowed down or interrupted.

An extensive cooperation and synchronization between the flexible manufacturing cells, therefore, leads to an improved overall system performance. E.g., in case of a failure, the control system must determine, in real-time, which part has to move to which cell, when, how, and to what manufacturing process. Therefore synchronization in an automated factory environment is essential for organizing and distributing shared resources such as material, tools, and information.

The synchronization between different manufacturing and information processes depends on an accurate synchronization of the individual clocks. Precise clock synchronization is also of importance for performance measurement and evaluation in automated production systems. In addition, the network management tasks need a global time base for the correlation of events, for the testing of communication protocols and for the diagnosis of failures in a distributed system.

In the MAP architecture, which is the basic communication concept for the PAP project, clock synchronization is handled by network management, but only a granularity of one second for the individual clocks is provided. Indeed, the synchronization of the different manufacturing systems depends on the specific application.

For process control the exchange of information must be provided under real-time conditions and therefore the necessary accuracy must be in the range of a few milliseconds. Applications like hardware measurements in a distributed system need a higher precision of few nanoseconds and can only be realized with specific hardware solutions.

In the following we first overview useful algorithms on clock synchronization, the theoretical bounds, and the results of the common work at Maryland and Erlangen. "Useful" means reasonable in both, implementation and communication costs in a heterogeneous and hierarchical system like an automated plant. In section 3 we present the results of the experiments in detail.

2. Clock Synchronization Algorithms

Before stating and analyzing the chosen algorithms in detail, we first give an classification scheme and criteria for the comparision of different clock synchronization algorithms.

2.1 Classification

Since Lamport's article *"Time, Clocks, and the Ordering of Event in a Distributed Environment"* [Lamp78] various methods for time synchronization are discussed in literature. The algorithms for clock synchronization can adopt centralized or distributed schemes which can be classified in the following order:

(1) **Central-dictatoric**: A central controller directs participating processes to synchronize their clocks. Fault tolerance and redundancy can be provided by election mechanism for the central controller [Rica81], but normally failures of the central component cause a breakdown of the clock synchronization. One example of this class of schemes is the GPS (Global Positioning System) receiver [Borc85] which is directed by four navigation satellites. The achievable clock accuracy over the entire world is in the range of nanoseconds, but the cost of one receiver is about $ 5000.

(2) **Central-democratic**: A dedicated processor (*time master*) periodically computes the general network time by a voting mechanism with the individual controllers (*time slaves*). Such a scheme must provide some kind of mechanism to pass on the leadership to other processes in case the master node fails. The *TEMPO* algorithm developed by Gusella and Zatti [Guse83] is an example for such a scheme.

(3) **Decentral-democratic**: All of the time-processes execute an identical procedure to achieve clock synchronization. Based on local time and the received timestamps from the other time-processes the clock will periodically be computed and, if necessary, set. Most existing clock synchronization algorithms use a distributed approach for fault-tolerance [Lamp82] [Halp83] [Marz83] [Lund84] [Srik86]. These algorithms, in general, are more difficult to implement because more coordinations among the processes are required. Agreement protocols may also be used to achieve better synchronization [Schn86].

Central dictatoric schemes are mostly realized by special hardware and therefore often ca not be used in a heterogeneous factory environment because of the installation costs. The algorithms presented here are based on software mechanisms and can be realized inside of the automation equipment and in a heterogeneous computer network environment.

2.2 Basic Requirements

For a comparison of the clock synchronization algorithms the following criteria are used:

- **Accuracy:** What clock accuracy can be achieved after the resynchronization? The necessary accuracy depends strongly on the application and ranges from nanoseconds to seconds (see previous section). If the clock synchronization requirements are only in the range of seconds or hundred of milliseconds "low cost" algorithms should be used.

- **Communication costs:** How many messages must be sent per resynchronization? Another important question is, what relative load is caused by the rsynchronization algorithm onto the communication system?

- **Computing costs:** How much processor load results from the clock synchronization algorithms? One important issue for this criteria can be that the time-processes should not disturb or block other application oriented tasks.

- **Complexity of the algorithm:** How sophisticated is the algorithm to implement in a heterogeneous network. It is significant that the amount of time for the implementation of clock synchronization depends strongly on the complexity of the algorithm.

- **Fault tolerance:** In which manner will failures of the communication system, the controller or the time-process be managed? It is desirable that the abort or the controlled ending of a time-process has no or a only a minimal influence on other time-processes. The reconnection or reloading of one time-process must also be supported by the chosen mechanism.

- Flexibility: Algorithm for clock synchronization should be able to adapt to different configurations of the network and the machines. E.g., different granularities of the individual clocks must not have a strong influence on the achievable clock accuracy in the whole system. Furthermore, integration of new systems into the whole process of clock synchronization should be easy.

Of course there can be more criteria depending on the field of application. At the end of this section a comparison of the following algorithms is presented.

2.3 TEMPO

The *TEMPO* algorithm [Guse83] uses a central-democratic scheme in which a dedicated process (*master*) polls each *slave* process to measure the clock difference between the hosts on which the two processes are running. After the master process has evaluated the time offsets between its clock and all the slaves' clocks, it computes a network average clock skew using a fault tolerant averaging function. Described in a simplified manner, the algorithms works as follows:

- For getting the *difference* between each slave clock and the master clock, the master transmits a message with its timestamp to the slave process. Upon receiving this message, the slave process records the time of reception and computes d_1, as difference between the reception time and the transmitted master timestamp. Now the slave repeats the same procedure by sending the first difference d_1 and its actual timestamp. The master receives this message and computes analogous d_2. The difference between the two clock can now be obtained by $(d_1-d_2)/2$. The basic method is discussed in [Elli73] and assumes that the transmission time is constant and has a symmetric distribution function. In the case that such a symmetric distribution is not appropriate a systematic error will appear.

  The transmission time in a local area network is usually variable and therefore the above described step of the *TEMPO* algorithm will be repeated several times. To eliminate high variation in communication delays, a minimum-delay message is chosen on each direction from the N-times a slave is polled.

- The *global net time* will be computed based on the differences of the individual clocks. For the computation of the global net time the mean value, the median, the mean value without extreme values, and the minimum values can be used.

- Each processor receives this global net time from the time master process and sets its individual clock within a given minimum interval.

The accuracy of the *TEMPO* algorithm depends mainly on the estimation of the transmission time and on the assumption of an unique distribution function in both transmission directions.

Detailed implementation and simulation results for the *TEMPO* algorithm conducted at Erlangen and Maryland will be presented in section 3.2. For hierarchical local area networks, as in the factory environment, an extended version of *TEMPO* (ETEMPO) was developed at Maryland and is discussed in the following section.

2.4 ETEMPO

A hierarchical LAN such as in automated factories can be divided in several "sub-LANs". To see how the *TEMPO* algorithm can be extended to hierarchical LAN, let us first consider the case of a two-level hierarchical LAN where several sub-LANs are connected through a second-level LAN (E.g. figure 1).

In this two-level hierarchical LAN, the clocks in each of the low-level LANs can be synchronized with the *TEMPO* algorithm. Since each low-level LAN has a master process to direct local synchronization, these "low-level masters" can be considered spokespersons for their networks and may elect a "master of masters" among themselves to be in charge of synchronization of their clocks, again using the *TEMPO* algorithm. Once the clocks of the low-level masters are synchronized through this "high-level synchronization", local clocks of a low-level LAN may be asked to synchronize with their master's clock.

There is little procedural difference between the high-level and the low-level synchronization except that a low-level master may wish to synchronize local clocks regularly during the idle period between the two high-level synchronizations. For the high-level synchronization, the high-level master polls the low-level masters (they are now the slave processes of the hierarchical LAN) to estimate the clock skews the same way as done for a single LAN. The time offsets are again averaged, and correction for each low-level master is computed.

In case a low-level master process fails, the slave processs elect a new master for the low-level LAN. The new master then joins the other low-level masters to synchronize their clocks. If the failed process is the high-level master process, the other low-level masters will elect a new high-level master to be in charge of the high-level synchronization. However, if a failure occurs in the communication lines and a low-level network is disconnected, the clocks in that network will not be synchronized until repair to the communication lines is completed.

When a low-level master corrects its clock due to the high-level synchronization, it has three options as to whether to propagate this adjustment to other clocks in its network. First, it can ask all local clocks to immediately reset to their master's time. Secondly, it may choose not to propagate the adjustment to local clocks but waits until the next low-level synchronization round to synchronize the local clocks. Thirdly, it can ask the other clocks to adjust by the same amount.

It is likely that each low-level LAN may synchronize their clocks more frequently than the high-level synchronization. The clock synchronization model for a two-level hierarchical LAN can theoretically be extended to a hierarchical network of several layers. The basic principle is that at each

layer up, one synchronizes the master clocks of the subnetworks of the current layer, and propagates the clock correction down to its subordinate networks.

2.5 Interactive Convergence Algorithm CNV

CNV [Lamp85] is a decentral-democratic algorithm, where the global time is divided into intervals. At the end of such an interval the time-processes compute the value of the clock using the following algorithm:

(1) The differences between all clocks will be investigated. The chosen method has a maximal error ϵ for the computed differences.

(2) The extreme values of the computed differences will be set to zero and the mean value of all differences is added to the individual clocks.

The accuracy after the resynchronization is determinable by the formula $(6 \times f + 2) \times \epsilon$, under the assumption that 2/3 of the f processors are working correct. If the difference of the clocks can be kept with an error of 40 milliseconds and two of seven processors are working wrong, the maximal achievable accuracy is equal to 560 milliseconds.

The major advantage of CNV is its fault tolerance against failures of nodes. In a factory communication environment it can hardly be used, because of the enormous communication traffic which will conflict with the information flow for the manufacturing processes. Another disadvantage is the assumption that the individual clocks are initially synchronized to approximateley the same value. Under this restrictive assumption clock synchronization can hardly be achieved.

2.6 Modified Optimal Clock Synchronization mOCS

In a complex heterogeneous network like an automated plant it is often necessary to synchronize various devices with different clock granularities. One problem with the $TEMPO$ algorithm is that the computable differences of the individual clocks are restricted by the granularity of the "worst" clock.

With OCS (Optimal Clock Synchronization) [Srik85] not only clocks are synchronized, their rates of drift from the real time can also be minimized to the drift rates of the underlying hardware clocks. thus "optimal accuracy" is achieved. The main advantage of the Optimal Clock Synchronization algorithm for a heterogeneous network is that no computation of clock skews or differences between

two clocks is needed. A modified *OCS* algorithm *mOCS* was developed at Erlangen, for the use in broadcast networks. This type of network (e.g. Ethernet) or other bus systems are normally used as backbone network in office or factory communication.

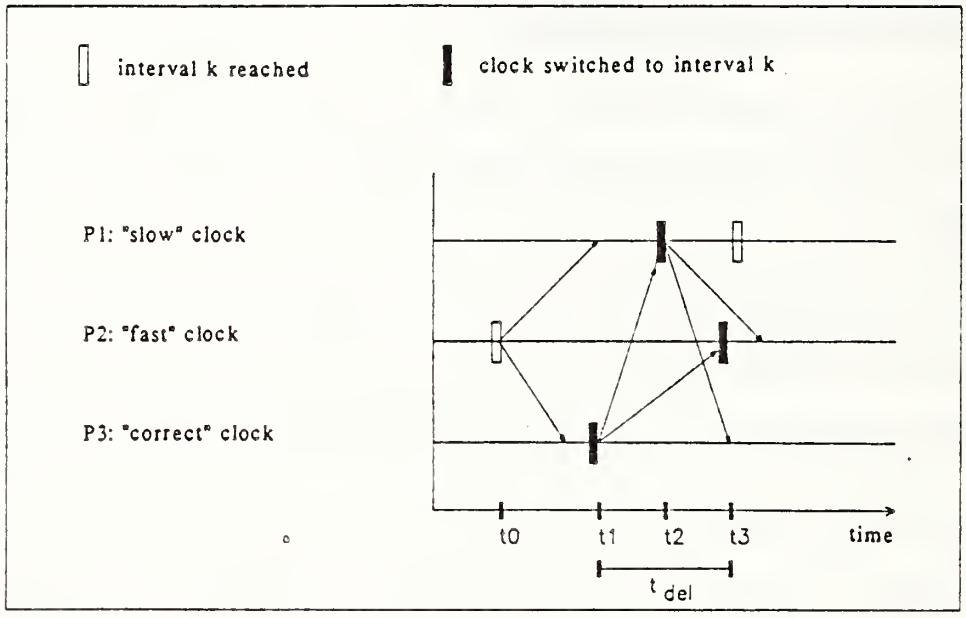


Figure 2. The *mOCS* algorithm

The time on each time-process is divided into continously numbered resynchronization intervals of length p. The algorithm synchronizes the clocks in "rounds", where the number of the interval and the round is the same. If one clock reaches the beginning of the *k.th* interval, a broadcast message will be sent to all time-processes, including itself (Fig. 2: P2 at t0, P3 at t1).

The messages contain a sender signature and the interval number. Every time-process counts the arriving *round* k-messages. If a time-process has enough messages received it knows that a sufficient number of other processes are ready to begin a new time interval (Fig. 2: P3 at t1, P1 at t2, P2 at t3). Subsequent to this, it sets its clock to the value of $k \times p$ and the other time-processes do the same.

Because of the switching of the first "correct" process to a new interval it is clear that the other "correct" processes do the same after the maximum transmission delay and begin the *kth* time interval.

Under the assumption of n time-processes in the network the treshold for beginning a new interval should be $\frac{n}{2}+1$ messages, in the case of signed, $\frac{2}{3}n+1$ messages, in the case of unsigned. The theoretical background in the case of faulty processors and the mimimum knowledge which is

necessary for achieving clock synchronization is discussed in [Dole84, Halp84].

The accuracy of *OCS* and *mOCS* depends on the maximum transmission time. In [Srik85], it is proved that the maximum difference between two clocks after resynchronization is not greater than the maximum transmission time. The advantage of this algorithm is the low system load, only counting of messages is necessary. In *mOCS* the number of messages is reduced even further. Each process has to sent only one message per resynchronization interval. Under the assumption that after a sufficient amount of messages are received, no new message is sent, the number of messages can be reduced to $n/2+1$.

The *mOCS* algorithm can be extended towards a general scheme. The processes are ordered into two classes, where the members of the first class are running the original algorithm. The time-processes of the second class accept only the impulses of switching to a new interval, but do not send messages. This separation into two classes can be seen as a fault tolerant version of a central clock, if the members of the first class can constitute a sufficient number of correct running clocks. If there is only one "correct running" clock, then this extended version of *mOCS* is the same as a central controller.

The main advantage of the algorithm is that no offsets or clock skews computations are needed. However, it may lead to a substantial error in the case of large transmission times. If the estimation of the clock skews results in an smaller error than the maximum transmission time, algorithms like *TEMPO* should be used.

2.7 Comparison

The accuracy of the algorithms discussed above are hard to compare theoretically and advantages are application dependent. Note that methods with a central clock (radio sender, central "tick" by satellite) provide a accuracy in the range of milliseconds to nanoseconds, but need a hardware integration into the processor.

Algorithms that compute the differences between the clocks (e.g. *TEMPO*) have to estimate the transmission times or delays. Methods like *mOCS* or a central clock that need no clock skews computation achive an accuracy of nearly the maximum deviation between the real and the estimated transmission time. If the central clock is realized by a radio sender, then the influence of the transmission time and so the inaccuracy can be reduced..

The number of messages per resynchronization is another criteria for comparing the clock synchronization algorithms. A central clock needs only one message while *mOCS* requires between

$n/2+1$ and n messages. For the *TEMPO* algorithm the amount of messages depends on the number of processors, n, and the number of polling, p, by the master process. This can be expressed by $(2\times p+1)\times n$. The fault tolerant algorithm *CNV* transmits $2\times p\times n\times(n-1)$ messages to achieve a global agreement. Table 1 shows the number of messages for realistic networks produced per resynchronization interval. In the case of *TEMPO* and *CNV* a polling rate of 10 is assumed.

number of nodes	*number of messages*		
	TEMPO	mOCS	CNV
10	210	5-10	900
50	1050	25-50	24,500
100	2,100	50-100	99,000
1000	21,000	500-1000	9.99×10^6

TABLE 1. Messages per resynchronization interval

The fault tolerant aspect is supported by distributed algorithms like *mOCS* and *CNV*, which recognize failures or breakdown of time-processes. But even in the case of faulty processes in a system, *CNV* has no influence on practical clock synchronization, because of the achievable low accuracy. Methods that use a central controller tolerate faults of the "slaves", but if the central component breaks down, special mechanisms must be provided to elect a new "time master".

Based on our experiences with theoretical and experimental results at Erlangen and Maryland we feel that:

- mechanism that use a central dictatoric controller achieve good results, if the central clock has a high accuracy and the timestamps can be transmitted in a fast way to the "slaves".
- *mOCS* should be chosen, if the desired accuracy is not better than the maximum transmission time and where a granularity of one second or more (either in reading or setting the clocks) is acceptable.
- *TEMPO* can be used in "middle sized" networks, but it assumes a high granularity for the computing of the clock skews and for correcting the clocks. Networks with unpredictable transmission times or a high variance can also be synchronized by *TEMPO* quite good.
- For hierarchical LANs, *ETEMPO* is a good candidate for clock synchronization, but further study has to be done for configurations of this type. General configurations need also basic research and new algorithms with respect to defined criteria have to be found.

3. Experiments

3.1 Environment of the Implementation

The experiments described here are performed on 10 MB-Ethernets at Erlangen and Maryland. At the time of the measurements the PAP model factory was not completed, therefore for the practical results in Erlangen the University's education network is chosen.

In Erlangen, the *TEMPO* algorithm is implemented on five UNIX V.2 hosts (3 Perkin Elmer 3205, 1 Perkin Elmer 3210, 1 CADMUS 9230). Measurements are made with the intention to get realistic parameters for a general simulation model. The private communication protocol uses the data link layer of the ISO/OSI reference model. The simulation program was runs on an INTEL 310 under XENIX for systematic study of the TEMPO and the *mOCS* algorithms under different conditions (distribution time, fault rate, drift rate, etc.).

The implementation in Maryland is on three hosts running UNIX 4.3 BSD (2 VAX 11/750, 1 VAX 11/780). The measurements made in Maryland are taken for a program which emulates on each of the hosts a sub-LAN. On the UNIX 4.3 BSD machines the datagram socket is used because the communication protocol supporting the other socket type, the stream socket, would usually incur long communication delay, especially when messages are lost and retransmission are necessary.

3.2 TEMPO

The implementation of the *TEMPO* algorithm on the local networks in Erlangen and Maryland allows us to make measurements over several parameters. The results presented here confirm the measurements in [Guse83]. The parameters experimented are:

(1) the interval between resynchronization,

(2) the number of times a master polls its slaves in estimating clock skews,

(3) the maximum round-trip communication delay allowed,

(4) the system load during experiment period,

(5) the initial clock difference.

3.2.1 Measurements at Maryland

At Maryland we study the clock skews measured at each synchronization round to determine the effectiveness of the algorithm in keeping clocks synchronized. In this experiments, clock adjustment is always rounded up/down to the nearest multiple of five milliseconds.

Among these parameters, only the interval between resynchronizations shows a significant influence on the distribution of clock skews. For example, in three experiments where synchronization takes place every two, four, and eight minutes, respectively, the distributions of clock skews between the hosts Tove (VAX 11/750) and Gyre (VAX 11/750) are shown in Fig. 3. In these three experiments, the process in Tove is the master, and each experiment makes 20 synchronization attempts. The positive clock skews (Fig. 3) indicate that Gyre's clock runs faster than Tove's. It is interesting to note that with two-minute interval, the most frequently occurring clock difference is five milliseconds, whereas it is 10 milliseconds for the four-minute interval, and 15 milliseconds for the eight-minute interval. When the experiments are repeated with the process on Gyre as the master, the resulting distributions of clock skews (Fig. 4) are almost a mirror image of Fig. 3. This suggests that the synchronization algorithm is independent of the placement of the master process.

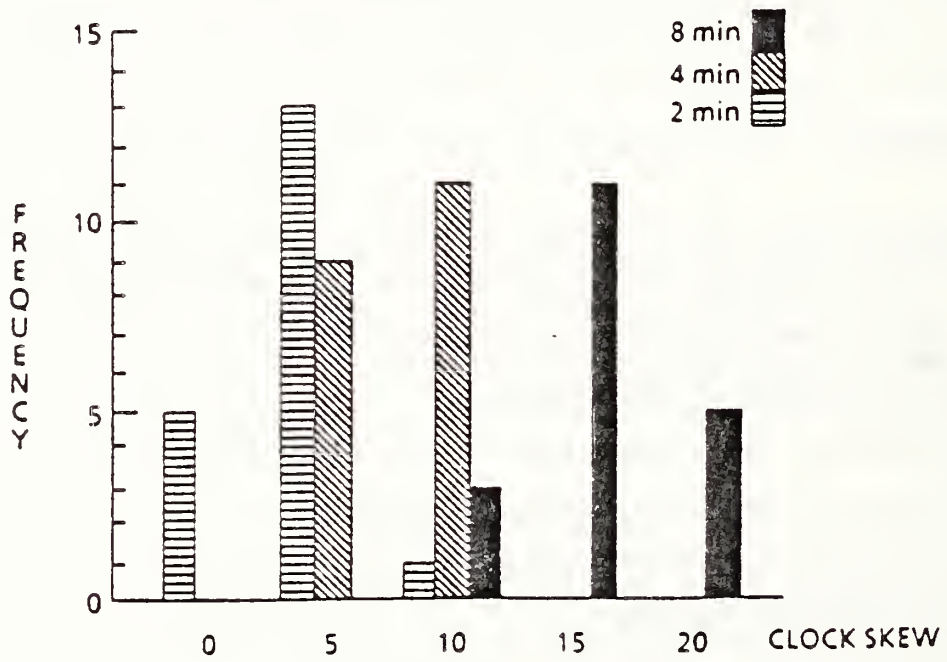


Figure 3. Histograms of clock skews (in ms.) between hosts Tove (Master) and Gyre at synchronization interval of two, four, and eight minutes. The abscissa is the frequency of occurence.

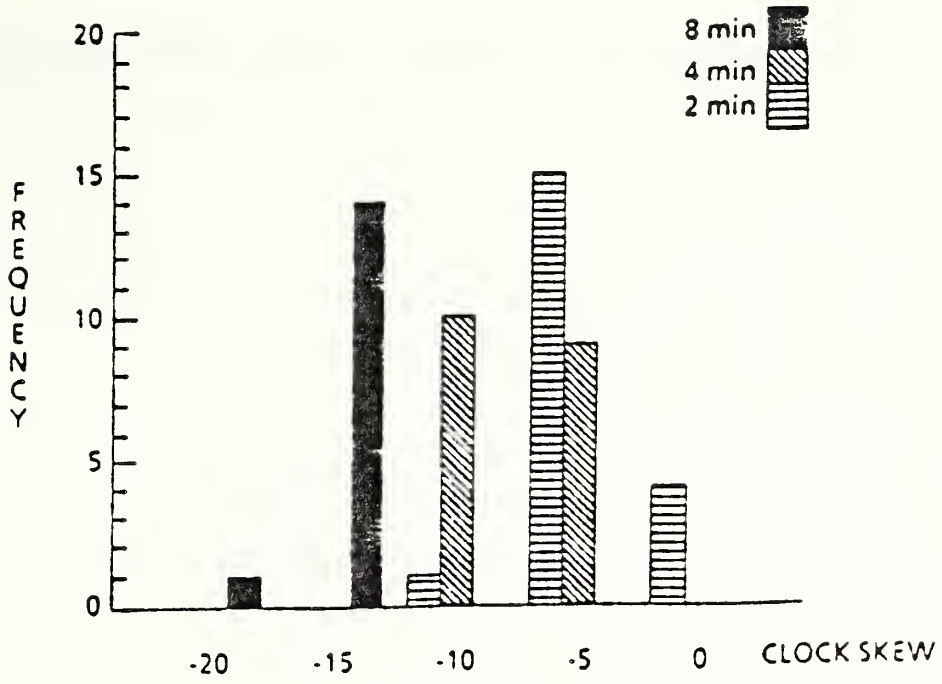


Figure 4. Same as Figure 3 except for hosts Gyre (Master) and Tove

To measure clock skews accurately, there should be little variation in the communication delays. Though the variation in the message delays is difficult to measure, we suspect it is not high in our system, because most round-trip delays are shown to be under 20 milliseconds (Table 4.1) even if the maximum round-trip delay is set to 40 milliseconds.

Round-trip communication delay and its frequency count					
Hosts	Limit	Round-trip Delays (in ms.)			
		10	20	30	40
Mimsy/Tove	20	7	68		
Mimsy/Tove	40	19	48	6	2
Mimsy/Gyre	20	16	59		
Mimsy/Gyre	40	14	55	4	2

TABLE 2. Round-trip communication delay and its frequency count

On the local network at Maryland, clock adjustment is performed gradually; during every tick interval of 10 milliseconds, a correction of one millisecond is made, if necessary. Therefore, the time it takes to establish initial synchronization depends on how clock correction is actually implemented. It may seem surprising that the measured clock skews are not sensitive to different system loads. However, because high-priority Internet ICMP time-stamping facility is used by processes to read the clocks of the other hosts directly, delays in awaking idle slave processes do not interfere with the timestamps transmitted,

thus, clock skews computed from these timestamps do not seem to be affected by different system loads.

3.2.2 Measurements at Erlangen

Between the "time master" and its "slaves" the following mean values and variances of transmission time are observed on the local area network of the Erlangen University (Table 3).

slave hosts	*transmission time*	
	mean value [msec]	*variance* [msec]
slave 1 (faui01)	67	64
slave 2 (faui02)	54	54
slave 3 (faui05)	55	30
slave 4 (faui50)	97	25
total	68	50

TABLE 3. Transmission times on the LAN at Erlangen

As an initial "time master", a Perkin Elmer 3210 was chosen because this host was not very heavily loaded. The slave hosts faui01, faui02 und faui05 are of the same type (Perkin Elmer 3205), but slightly different transmission times are observed. In the case of faui05, the transmission times with the lowest variance are observed, about 90% of the messages were transmitted in an interval of 45 to 65 milliseconds.

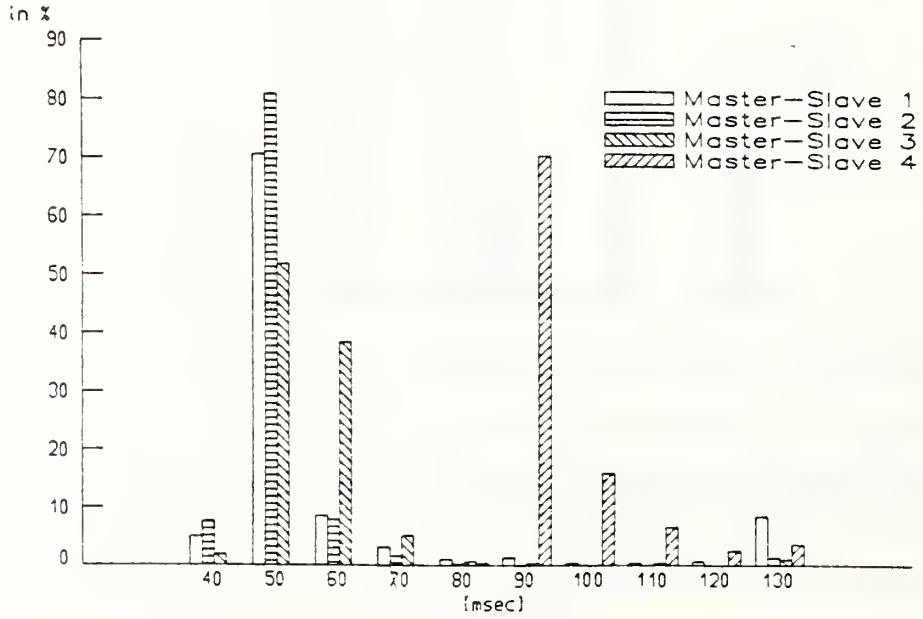


Figure 5. Distribution of the transmission times at Erlangen

Host faui01 also shows transmission time around 50 milliseconds, but also quite a few delayed messages (9%) with a transmission time over 120 milliseconds were registred. This may be caused repeated transmissions and high system loads. Host faui50 is a CADMUS 9230 machine with a 68010 processor. This node has significant higher transmission times, the messages are normally (86%) measured in an interval of 90 to 100 milliseconds.

3.2.3 Simulation of TEMPO

The simulation is performed with 5 processors (clocks). It is assumed that no communication errors occur. When a message arrives and the receiver is not ready, the message is lost.

The independent parameter considered for the simulation is the process-to-process transmission time. In a local area network like the Ethernet, the transmission time has a standard distribution function. Object of study was the effect of changing the mean and the variance of the transmission time distribution.

Table 4 shows the achievable accuracy after the resynchronization in respect of the simulated parameters*. For the scenarios S1-S5 the received minimum clock offsets were taken for the estimation of the clock skews.

scenario	polling rate	transmission time [msec]	accuracy mean value [msec]	accuracy variance [msec]
S1	3	N(10,2,5,15)	2.3	0.87
S2	3	N(100,40,20,2000)	42	18
S3	10	N(100,40,20,2000)	25	8.0
S4	10	N(100,100,20,2000)	22	11
S5	S1 with additional 20 msec receiving time for one host (unsymmetric transmission time)		11	1.0
S6	S2, but mean value taken for computing the clock skews		78	18
S7	S3, but mean value taken for computing the clock skews		50	18

TABLE 4. Accuracy of *TEMPO* (simulation model)

---

* N(a,b,c,d) means standard distribution (mean value, variance, minimum, maximum)

It is significant that if the mean value of all computed clock skews is used, the achievable accuracy decreases. Using the minimum clock skews for the estimation of the transmission times there is only a minimal error for this parameter.

An unsymmetric distributed transmission time leads to a systematic error. This error is nearly half of the difference of the transmission times for both direction and can easily arise in a heterogenous network of processors with different interrupt structures, scheduling strategies or system loads.

By increasing the number of polls a higher accuracy is achieved. Figure 6 shows the accuracy in respect to the variance for the transmission time and the number of polls.

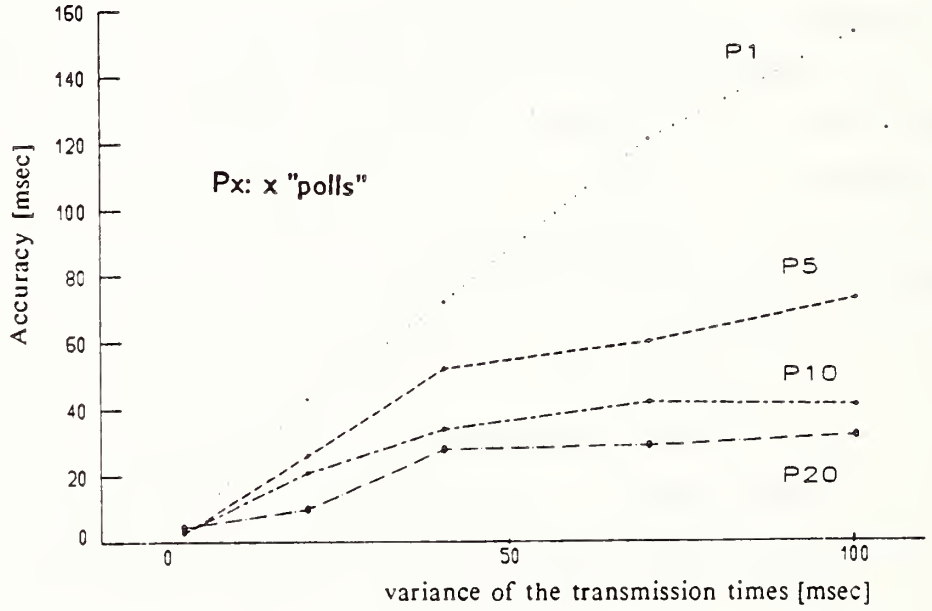


Figure 6. Influence of variance and polling rate (*TEMPO*)

Indeed, the accuracy increases with a high polling rate, but it is not linear. Normally, the number of polls should be between 5 and 15, only in the case of an extreme variance a higher rate should be chosen.

3.3 ETEMPO

For the hierarchical network, the experiments are organized into two groups. In one group, the high-level synchronization interval is fixed and it is experimented with different combinations of low-level synchronization intervals. In the second group, the low-level synchronization interval is fixed and the high-level synchronization interval is varied. Table 5 summarizes the set-ups of these experiments.

Experiment Number	High-Level Interval (in min.)	Low-level Interval (min.)			Max. Sync. Count	Length (in hrs.)
		Gyre	Mimsy	Tove		
H1	10	2	2	2	12	2
H2	10	4	4	4	12	2
H3	10	8	8	8	12	2
H4	10	2	8	4	12	2
H5	10	2	2	4	12	2
H6	10	2	2	8	12	2
H7	40	4	4	4	20	13
H8	60	4	4	4	20	20
H9	80	4	4	4	20	26

TABLE 5. Experiment Set-ups for High-level Synchronization

3.3.1 Measurements on Low-level Synchronization Intervals

In a hierarchical network, each low-level LAN may use different synchronization interval for low-level synchronizations. In Experiments H1-H6 (Table 5), the intervals of the high-level synchronization are kept at ten minutes while the low-level synchronization intervals are allowed to vary from two to eight minutes. For experiments H1-H3, the distributions of clock skews between low-level masters Gyre and Tove are shown in Figure 7.

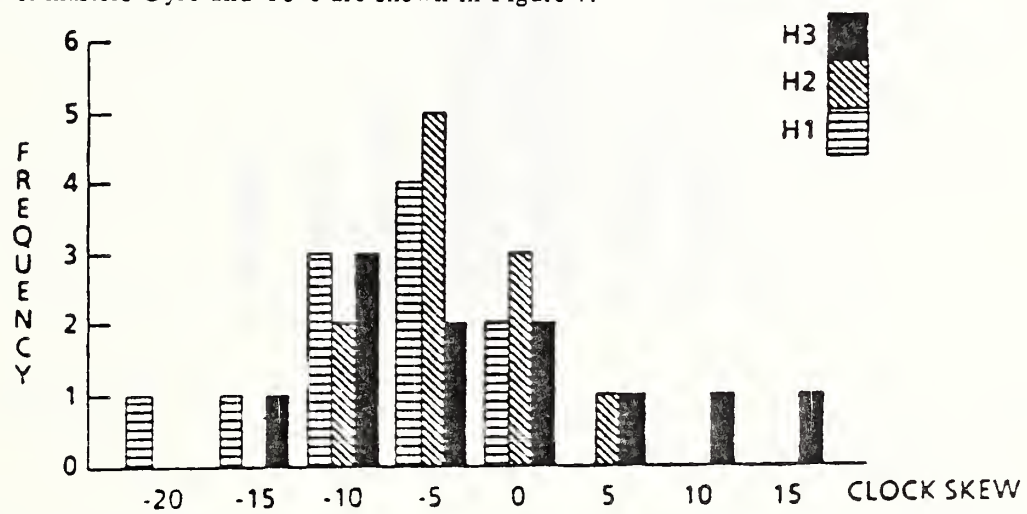


Figure 7. Histograms of clock skews between low-level masters Gyre and Tove of Experiments H1, H2, H3

The distributions are difficult to compare because there does not exist a characteristic pattern that can be identified with each of the three experiments. These results are not surprising considering that after a high-level synchronization, there are two factors causing clocks to deviate. One factor is the drifts

between clocks over the ten-minute interval between high-level resynchronizations. The other is the total corrections made on each of the two clocks during the same period from low-level synchronization within the individual LAN. More results are presented in [Chan86].

3.3.2 Measurements on High-level Synchronization Intervals

In Experiments H7-H9, the low-level synchronization interval is kept at four minutes for all low-level LANs, while the high-level synchronization interval varies from 40 to 60 to 80 minutes. The distributions of clock skews between low-level masters Tove and Gyre are shown in Fig. 8. It is observed that longer interval between resynchronization does in fact cause wider clock seperation. However, due to the fact that the high-level synchronization interval is significantly longer than the low-level synchronization interval, the drift factor in this case thus plays a more important role, and results in a clearer distribution pattern than earlier measurements.

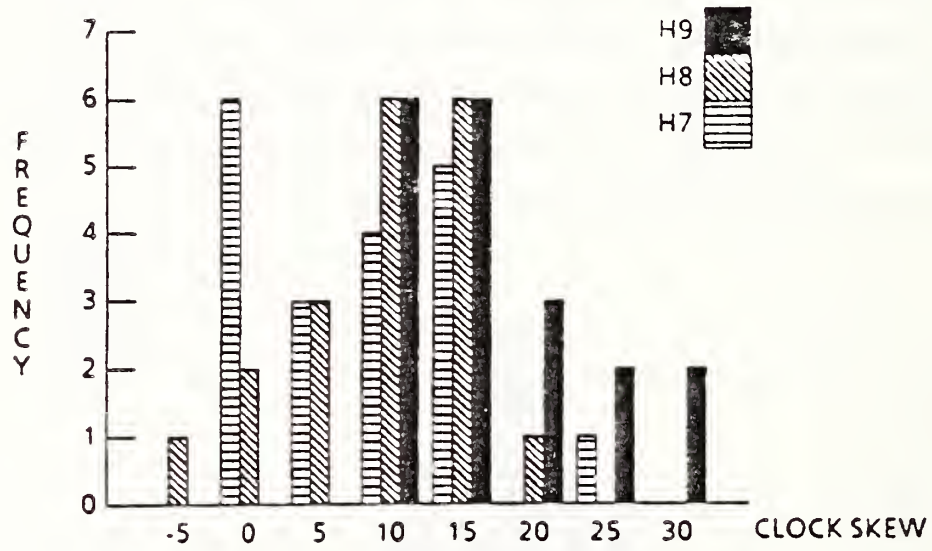


Figure 8. Histograms of clock skews between low-level masters Tove and Gyre of Experiments H7, H8, and H9

3.4 mOCS

The achievable accuracy of *mOCS* depends only on the transmission time, and therefore, in the simulation model for *mOCS* the distribution function of the transmission times is examined. Table 6 shows that for the education network at the University of Erlangen (Ethernet, UNIX hosts) *mOCS* is

not a good choice as *TEMPO*. The *TEMPO* algorithm provides a better accuracy in this case [Herb86].

scenario	transmission time [msec]	accuracy mean value [msec]	accuracy variance [msec]
O1	N(10,2,5,15)	9.4	3.1
O2	N(100,40,20,2000)	69	27
O3	N(100,100,20,2000)	104	52

TABLE 6. Accuracy of mOCS (simulation model)

4. Final remarks

Synchronization of clocks on a factory floor is an important problem. Hardware solutions for such synchronization may be too expensive. In this paper we have presented software solutions to these problem. A comparative evaluation of various algorithms has been presented. Some of the algorithms have been implemented at Erlangen and at maryland. Experimental results have also been discussed.

The next step of the research will be the implementation of the studied algorithm in the PAP model factory, so that precise measurements for a heterogeneous automation environment can be done. Combining *mOCS* and *TEMPO* may give a better synchronization algorithm and this needs further investigation.

5. References

[Borc85] Borcherding, K.; Bultmann, D.; Franck, H.; Rehmert, H.: "NAVSTAR GPS-Empfaenger, Entwicklungsprogramm Phase I", (NAVSTAR GPS-receiver, development program), Report: BMFT-FB-W85-002, 1985

[Chan86] Chang, S.H.; "Clock Synchronization in a Hierarchical Network", M.S. thesis, University of Maryland, Aug. 1986

[Dole84] Dolev, D.; Halpern, J.: "On the possibility and impossibility of achieving Clock Synchronization", Proceedings of the Sixteenth Annual ACM, Symposium on theory of computation, Washington. April 30-Mai 2, 1984

[Dupo82] Dupont-Gatelmand, C.: "A survey of flexible manufacturing systems", J. Manufacturing Syst., vol. 1, no. 1, 1982, pp. 1-16

[Elli73] Ellingson, C; Kulpinski, R.J.: "Dissemination of System Time", IEEE Trans. on Commun., no 21, pp. 605-624, May 1973

[GM85] General Motors: MAP Specification, Version 2.1, March 1985

[GM86] General Motors: MAP Specification, Version 2.1.A and 2.2, August 1986

[Guse83] Gusella, R.; Zatti, S.: "TEMPO - Time Services for the Berkeley Local Network", Report No. 4CB/CSD 83/163, University of California, Berkeley, December 1983

[Halp83] Halpern, J.; Simons, B.; Strong, R.: "An efficient fault-tolerant algorithm for clock synchronization", IBMR RJ4094, 1983

[Halp84] Halpern, J.; Moses, Y.: "Knowledge and Common Knowledge in a Distributed Environment", Proceedings Third Annual ACM, Vancouver, Canada, August 1984

[Hatv84] Hatvani, J.: "Intelligence and cooperation in heterarchic manufacturing systems" in Proc. 16th CIRP Int. Seminar Manufact. Syst., 1984, pp. 1-4

[Herb86] Herberth, H.: "Zeitsynchronisation in lokalen Netzen", (Clock Synchronization in Local Area Networks), Diploma-Thesis, University of Erlangen-Nuremberg, October 1986

[Lamp78] Lamport, L.: "Time, Clocks, and the Ordering of Events in a Distibuted Environment", Comm. of the ACM, vol 21, no 7, pp. 558-565, July 1978

[Lamp82] Lamport, L.; Shostak, R.; Pease, M.: "The Byzantine Generals Problem", ACM Trans. Program. Lang. Syst., vol 4, no 3, 1982

[Lamp85] Lamport, L.; Melliar-Smith, P.M.: "Synchronizing Clocks in the Presence of Faults", Journal of the ACM, vol 32, no 1, pp. 52-78, January 1985

[Marz83] Marzullo, K.; Owicki, S.: "Maintaining the Time in a Distributed System", Proceedings of the Second ACM Symp. on Princ. of Distr. Computing, 1983

[Morg84] Morgan, C.: "Global and Logical Time in Distributed Algorithms", Information Processing Letters, Volume 20, Number 4, 1984

[Rica81] Ricart, G.; Agrawala, A.K.: "An Optimal Algorithm for Mutual Exclusion in Computer Networks", Communications of the ACM, vol 24, no 1, pp 9-17, January 1981

[Schn86] Schneider, F.B.: "A paradigm for reliable clock synchronization", Technical Report 86-735, Cornell University, Dec. 1984

[Srik85] Srikanth, T.K.; Toueg, S.: "Optimal Clock Synchronization", Proceedings of the Fourth Annual ACM, Symposium on Principles of Distributed Computing, August 5-7, 1985

[Srik86] Srikanth, T.K.: "Designing fault-tolerant algorithms for distributed systems using communication primitives", Ph.D. dissertation, Cornell University, TR 86-739, Feb. 1986

Design and Simulation Tools

SIMULATION OF MAP PROTOCOL FOR USE IN COMMUNICATIONS STUDIES

by

TONY KANADI, STUDENT MEMBER

and

SHELDON GRUBER, SENIOR MEMBER

of

CASE WESTERN RESERVE UNIVERSITY

CLEVELAND, OHIO 44106

Abstract

A CAD software tool has been implemented for studies of communication capabilities of large networks which use the Manufacturing Automation Protocol (MAP). The first four layers of MAP have been simulated using the C language in a UNIX environment to ensure a large degree of portability. The response time statistics of any two stations (transmitter-receiver pair) in the network can be examined in the presence of the communication load created by any number of other stations. Messages are sent randomly with statistics such as mean number of messages per hour that are specified by the user for each station in the network. The program also creates an environment in which the token bus protocol can be learned.

1. Introduction

1.1. General Considerations

MAP is a network standard specifically created to aid the manufacturer in the quest for a factory computer network that can utilize the products of many companies without constantly incurring the costs of adapting incompatible protocols. The intent is to provide, relatively easily, a large scale computer communications network to handle the varied requirements on the factory floor. (The term computer used here includes programmable controllers, robots and other microprocessor based control systems as well as what is conventionally recognized as a computer.)

It has been recognized that this environment is somewhat hostile in that the considerable noise introduced in the network increases the error rate. Studies have been performed and technical solutions found.

One area that has not received much consideration is the communications bottlenecks that can occur when a large number of stations compete for the existing facilities even when high data rate equipment is used. Perhaps the most important reason for this oversight is that it is much too expensive to wait for the existence of a large installation in order to study the effect. Furthermore, message delays are likely to be network dependent. The cost of reworking hardware connections would be substantial, and moreover there would be no guidance on how to design the new configurations. Such a trial and error process is clearly impractical. To perform such studies, one may create the network in a special purpose hardware, or simulate the required protocols in software using a general purpose computer. The latter, while slower, is certainly the less expensive

and more flexible route. While the dominant effort was to produce a simulation of a network of stations adhering to the lower four MAP layers, a further consideration was to create this in a portable manner. C language in a UNIX environment seems to be the natural choice to support portability.

2. Simulation of the Token Bus Protocol

2.1. Discussion

To simulate a token bus protocol, multiple stations have to be created and run simultaneously. Each station has its own variables and shares some global variables with other stations. Several methods which were tried and abandoned are briefly mentioned. The final method is discussed in detail.

2.2. Method 1

This method uses a program which contains a specific function. Stations are simulated by calling that function in the program, and each time a station is created, the function is called. The function runs in an infinite loop manner until its corresponding station is deleted from the network. Deletion of a station corresponds to the return to the main program of its corresponding function.

There are several disadvantages with this approach:

1. Every time a station is created, a simulated process has to be created and a stack area has to be provided for that simulated process. The stack area has to be partitioned to accommodate each new station. Since the actual stack size is not dynamically determined, the number of stations possible is limited.

2. As the program hops from one station to another, the stack pointer as well as the program counter have to be adjusted accordingly.

3. The two functions above, partitioning and adjusting stack pointer & program counter have to be done in assembly language, thus limiting the portability of the program.

Mostly because of the latter, this approach was abandoned.

2.3. Method 2

Another method could use the multi-tasking capability of the UNIX system. Each station is simulated by an operating system process. This provides every station with enough stack area and partitioning is not needed here. A parent process exists to keep track of all the stations (processes) that have been created. Synchronizing all stations is done by signaling back and forth between parent process and each station process sequentially. Global variables are stored in files, and interprocess communication is done by using UNIX facilities. This approach was tried successfully on an IRIS UNIX operating system, but carries these disadvantages:

1. Creation of a station means creating a process which runs a certain program. Even with a relatively small program, the creation of many stations will use so much memory space that it becomes impossible to have reasonably large number of stations for the simulation.

2. Addition of a station considerably slows down the simulation.

3. A significant number of interprocess communication results in the loss of some of the signals sent between processes which, forces the sending process to resend the signals. This kind of real operating system signaling is both complicated and not fast enough for this simulation.

These drawbacks caused a search for a better method and culminated in the final choice below.

2.4. Chosen Method

This method is based on the following observations:

1. Even though all stations run "simultaneously", they can be simulated by running them sequentially which results that only one station is active at any time. A single process is therefore sufficient since no two processes ever run simultaneously.

2. Each station is represented by its parameters in the process. An element of a linked-list which contains all the variables of a station can therefore represent the station itself.

3. The protocol is implemented by having only one set of functions which processes all elements of the linked-list (stations) sequentially. All the stations use the same protocol; in other words all

the stations call the same functions which represents the protocol. Processing one station means that the function processes the variables of that station (an element of the linked-list). This process involves changing the values of some of the variables in the station. A time signal for all stations corresponds to the function processing the variables of all the stations in the network sequentially. So every time variables of station one is processed, station one is said to be time signalled. The time itself is updated after one pass of processing all stations in the network. The fact that the same function processes all the stations sequentially implies that the function shall never pause in the middle of processing a station. If that situation were to occur, the program would become deadlocked. This leads to the final observation below:

4. If a function of the protocol contains a pause situation, the function can always be divided into two or more separate functions. For example the function "a()" below can be divided into two separate functions in addition to updating function "states()" as illustrated below:

```
states (ptr)
stat_ptr ptr;
{
        boolean x;
             .
        if (x) a();  /* if x is true then calls a */
             .
}

a()
{
          .
        p = 1;
        sleep (10);
        q = 3;
         .
}
/*-----------------------------------------*/
states (ptr)
stat_ptr ptr;
{
        boolean x;
        static boolean start=TRUE;
         .
         .

        if (sleep_timer) {
                sleep_timer --;
                if (!sleep_timer)  start = FALSE;
                /* sleep timer equals zero */
        }
        else if (x && start) a();
        else if (x) after_a();
/* timer just goes off, execute the second half*/
             .
}

a()
{
          .
        p = 1;
        sleep_timer = 10;
}

after_a()
{
        start = TRUE;
        q = 3;
         .
}
```

By using the above observations, a hierarchical link list can be used to represents multiple stations that might exist in the network. The hierarchical structure conforms to the addressing structure by having regions which contain segments which in turn contain stations. Each station is represented by an element of stat_struct type. A segment represents a logical ring while a region groups several segments together. This hierarchical structure is illustrated in Figure 1. It can be seen from the figure that stations which reside on the same segment share the same bus variable. This bus simulates the physical medium of the network.

The simulation of the token bus protocol requires two major operations:

1. Configuring the network which includes adding or deleting of stations or modifying existing stations.

2. Running the simulation by employing an infinite loop function that traverses every station sequentially, and pass the station pointer to the protocol function. Each pass of all stations in the network corresponds to the smallest unit of time in this simulation, which is an octet_time. The program can be ended by having a time limit to the simulation, say 100,000 octet_times.

This simulation, however, is not very helpful since nothing appears on the screen during the execution, and there is no way to check that the simulation program runs correctly. In order to make sure that the simulation program runs correctly, ten windows are created on the screen to monitor dynamically the ACM state of any stations. Since at most ten stations can be monitored at any time, commands to move the windows from a set of stations to another are provided. Each station in the network has a boolean variable have_wind which is set to TRUE when the station is being monitored, and FALSE otherwise. If a station's have_wind is TRUE, the station has an access to a window on the screen and it uses that window to show the ACM state as well as some other information about the station. As time progresses, the state of the station might change and the screen is updated accordingly. As the monitoring windows move within a segment, between segments or between regions, the variable have_wind of affected stations are adjusted accordingly. These windows also serve as a learning tool for people who want to understand the transitions of the ACM states dynamically. Figure 2 shows a screen dump of the screen windows. Screen manipulation uses the "curses" utility package which is common to all UNIX systems. This package is not terminal dependent and

therefore maintains the portability feature of the simulation program.

This method eliminates the disadvantages of the previous two methods and has the following advantages:

1. The program can run virtually as many stations as desired. Addition of a station merely corresponds to an addition of a linked-list element in the network configuration.

2. The program is relatively fast since it is a single process with respect to the operating system. It runs hundreds times faster than that of the second method discussed.

3. The program is portable to any UNIX system. It can run in any UNIX system with little or no modification. The only requirement is that the UNIX system support the "curses" utility package, a standard feature of UNIX systems.

4. It can be ported to any system which supports the C language by modifying functions which relate to screen manipulation. These functions do the most basic things such as cursor positioning and reverse video.

The only inevitable weakness of this program appears when considering the real transmission speed of the broadband coaxial cable. The transmission speed might be as fast as 10 Mbits/second. Compared to the real system, the simulation runs much slower. The simulation, nevertheless, produces response time in real time units.

2.5. Implementation of MAC Layer "Machines"

MAC layer consists of four main "machines" and an optional one. The four machines are Interface Machine (IFM), Receive Machine (RxM), Transmit Machine (TxM) and Access Control Machine (ACM). The optional machine is Regenerative Repeater Machine (RRM) and is not implemented in this simulation. The implementation of the four main machines is the focus of this sub-section.

2.5.1. Interface Machine (IFM)

IFM interfaces MAC sublayer with its upper layer. It provides the upper layer with two main services, accepting data to be sent to other stations and acknowledging incoming data destined for the station. The IFM uses two primitives to support those services, MA_DATA.request and MA_DT.indication. There is another primitive provided by the MAC sublayer, MA_DATA.confirmation which is not important to and omitted from this simulation. This primitive confirms to the upper layer the success of the last MA_DATA.request primitive.

2.5.1.1. MA_DATA.request

This primitive abbreviated as ma_dt_req in the simulation uses four queues (send_q) to store data from the upper layer. Each queue stores data with a certain priority. Data with priority 0 and 1 are stored in send_q[1], data with priority 2 and 3 in send_q[2], priority 4 and 5 in send_q[3], and priority 6 and 7 in send_q[4]. Each send_q is a first-in-first-out (FIFO) queue. The flow chart of this primitive is presented in Figure 3.

2.5.1.2. MA_DATA.indication

This primitive abbreviated as ma_dt_indic in the simulation acknowledges incoming data by passing the pointer to the data to the upper layer. The incoming data is received from the bus the Receive Machine (RxM) discussed later.

2.5.2. Access Control Machine (ACM)

This machine is discussed at length in [3]. It is the "brain" of the MAC layer by communicating with other ACMs of other stations which share the bus. The function of this machine is to keep the token moving in the logical ring. When the station has the token, the ACM machine checks the send_q queues to see if there are any data to be transmitted. If data exist and time is still available, the data will be removed from the

corresponding send_q and transmitted to destination station by calling primitives provided by the Transmit Machine. When time expires, the ACM transmits the token to its successor by sending the token control frame. The simulation follows the document closely by implementing all the functions specified in the document.

Each station in the network has a set of timers which is created and freed as a station is created and deleted from the network. All timers created are inserted in a timer queue. Each timer in the queue is updated periodically by reducing the values left on the timer. If the value becomes zero, the boolean field "expired" is set to TRUE. Timers can either have octet_time or slot_time time interval. If the interval is octet_time, the timer value is decremented every time signal; otherwise the timer value is decremented every slot_time time signals. If slot_time equals three octet_times, a timer with slot_time interval is decremented every three time signals.

Once a logical ring is established, a "steady-state" operation of ACM is presented in Fig. 4.

2.5.3. Transmit Machine (TxM)

This machine accepts the MAC pdu from the ACM and transmits it to the bus. In real implementation, each bit of data is converted to corresponding MAC_symbols. The transmit machine is implemented in this simulation by using three primitives: send_frm, send_complete_frm and rem_frm. Before discussing these three primitives, a closer look to the bus structure and its relationship with stations transmitting data into it is presented below:

2.5.3.1. Bus Structure

The type declaration of the bus is taken unchanged from the program shown below:

```
typedef struct {
    /* No of stats currently sending */
    int          no_of_senders;
    /* next state of the bus, one octet after */
    bus_cond     next_cond;
    /* current state of the bus */
    bus_cond     cond;
    /* length of data, if not collided */
    int          length;
    /* pointer to control type data */
    char       * data;
    /* pointer to upper layer data */
    n_pdu_typ * lay3_data;
    /* source address of sender */
    m_addr       SA;
    /* destination address */
    m_addr       DA;
    /* frame check sequence, not implemented*/
    fcs_typ      fcs;
    /* llc data or control type data */
    frm_typ      FT;
    /* priority if FT = llc, otherwise
       type of control type data */
    frm_sub_typ FC;
    /*transmission error, not implemented*/
    boolean      err_occurred;
} bus_struct, * bus_ptr;
```

The bus_cond refers to one of the following conditions: quiet_bus, busy, collided_bus and complete_frm. Quiet bus refers to the fact that no station is transmitting. Busy means only one station is transmitting since the last quiet_bus condition whereas collided_bus means more than one station are transmitting since the last quiet_bus condition. Complete_frm is a condition which refers to the fact that one station has successfully transmitted its complete data into the bus. The conditions which its transitions are shown in Fig. 5.

2.5.3.2. Transmit Machines conditions

Each station has four conditions with respect to the TxM: not_sending, sending, complete_send and after_sending. Fig. 6. shows the transitions of these conditions.

2.5.3.3. Send_frm

In addition to all the timers which exist for the ACM, an additional (send_timer) is provided for this simulation with octet_time interval. As with any other timer, it is updated when a time signal arrives. Send_frm first checks the condition of the bus. If the condition is not quiet_bus, it means that some other station is transmitting and the function cancels the transmission of data and gives up the token, since some other station must also have a token (multiple tokens); otherwise send_frm sends the MAC pdu, updates the bus next_cond and starts the count down send_timer with the length of the data transmitted. Notice that the next_cond field of the bus is the one updated. This enables stations to transmit data simultaneously within a timing signal pass. At the end of the pass, the bus' "cond" is set to bus' "next_cond".

This primitive does not put the data it is sending on the bus. It just marks the bus to indicate to other stations that a station is transmitting. This is an efficient scheme since it avoids putting data onto the bus only to have it discarded because of collision.

2.5.3.4. Send_complete_frm

When the send_timer's value becomes one, function send_complete_frm is called. This function checks the state of the bus. If the bus is busy, it means that no collision condition exists and this function will put the station's data on the bus and change the state of the bus to complete_frm. If the bus is in collided_bus, the state of the bus is not changed. After this function is called, the station's condition is updated to complete_send as shown in Figure 6. Since it is possible to send data to itself, this function also checks the destination address of the data. If the destination address matches station's own address, the IFM is activated by calling the function ma_dt_indic to indicate arrival of incoming data.

2.5.3.5. Rem_frm

This function is called when the send_timer changes from non-zero value to zero, thus showing that the sending period is over and the station should not transmit any longer. Rem_frm decrements the number of senders in the bus. If the number becomes zero, Rem_frm set the bus condition to quiet_bus. Rem_frm also frees the mac_layer pdu and its corresponding data unit which are created by the ma_dt_req function in the Interface Machine.

2.5.3.6. Addition and deletion of stations

Addition of a station to the network is done simply by creating a station element, initialize all variables, creating and initializing all necessary timers, and inserting the new element into the linked-list structure.

Deletion of a station then corresponds to deleting the station element from the network as well as freeing all the timers associated with it. Since deletion can be done dynamically, it is possible that a station might be deleted when it is actually transmitting data to the bus. Function "aft_stat_rem" is therefore provided to adjust the bus in case of a deletion of an actively sending station. This function also frees timers.

2.5.4. Receive Machine (RxM)

This machine is represented by function check_bus. This function checks the condition of the bus at every timing signal (octet_time). It sets four boolean variables bus_quiet, noise_burst, rx_data_frm and rx_prot_frm. If the bus condition is either busy or collided_bus, the station's bus_quiet is set to FALSE to indicate that transmission is occurring in the bus. If the bus condition is complete_frm, check_bus copies incoming data from the bus to the station's incoming buffer, sets station's bus_quiet variable back to TRUE and sets either rx_data_frm or rx_prot_frm to TRUE depending on the type of the data. All four boolean variables are used by the ACM machine and three of them can only be set back to FALSE by the ACM machine. The rx_data_frm is read by both ACM and IFM machines but can only be set to FALSE by the IFM machine.

Finally, if the bus shows a quiet_bus condition, variable bus_quiet of the station being visited is checked. A value of FALSE of bus_quiet variable implies that the station has previously heard either a busy or collided bus. This shows that some other station(s) have transmitted data to the bus, but the data is either never completed or a collision has occurred since the bus never reaches the complete_frm state. A summary of the operation of RxM is presented in Fig. 7.

2.5.5. Miscellaneous

Since this simulation runs dynamically, user input from the keyboard is checked continuously. The keyboard input is checked by polling the input buffer every time a timing signal pass to all stations was generated. Function check_input does the checking of the input buffer. Polling the input buffer after every pass,

however, slows down the simulation substantially. This polling and the display can be disabled by pressing a signal interrupt on the keyboard.

When a command such as addition of a station is generated from the keyboard. The program might request more information from the user consequently. The user might also want to cancel the command in the middle of interaction with the program. The program provides this service by providing a user interrupt signal on the keyboard.

Segments can have different transmission speed, the default is 10Mbits/second. Segments which have the default speed are timed every signal pass, segments which have lower speed are timed less often. For example, stations which reside on 5 Mbits/second segments are timed every two signal passes; thus lowering the transmission speed to half of those stations which operate on default speed.

3. Simulation of the Network Layer

3.1. Discussion:

Since LLC sublayer is not implemented in this simulation, the next upper layer after the MAC sublayer is the network layer. Having implemented the MAC sublayer, the simulation is capable of transmitting data between stations which reside on the same segment. This chapter deals with the transmission of data between stations which reside in different segments. This layer accepts layer_4 pdu from transport layer, divides the layer4_pdu into some smaller layer3_pdu's if necessary, and provides routing algorithm so that the layer_4 pdu can be sent from any station to any other station in the network. It also indicates to transport layer the arrival of layer_4 pdu's from other stations.

This layer provides a primitive (function) to transport layer. This function, n_unit_dt_req, accepts layer_4 pdu, creates as many layer_3 pdu's as necessary, and sends each of layer_3 pdu's by calling function send_pdu. Send_pdu routes the layer_3 pdu to destination station by calling the routing function and ma_dt_req function provided by the MAC sublayer. The flow-chart of n_unit_dt_req and send_pdu combined together is shown in Fig. 8.

Transmission of data between stations which reside on two different segments is done through a bridge. The bridge acts as a pair of stations which reside on the two segments. If a station in segment one wants to transmit data to a station in segment two, it sends the

data to bridge's station on segment one. When the bridge's station on segment one receives the data, it is automatically assumed that data have been received by bridge's station which reside on segment two. Every region contains a bridge queue to keep track of all the bridges which connect segments within that region. Regions are stored in a queue of type region_q. Besides storing regions, region_q also contain a bridge queue to keep track of all the bridges which connect segments of different regions. (Inter-region bridges).

Fig. 9. illustrates bridges which connect segments within a region. The bridge_q is a field of the region_struct. Another bridge_q which queues all bridges connecting segments of different regions, is also a field of the region_q. Bridges within a region are sorted by the segment numbers of the two segments being bridged in ascending order; so bridge x which connects segments 1 and 2 is in front of bridge y which connects segments 1 and 3, and bridge y is in front of bridge z which connects segments 2 and 3. Inter-region bridges are sorted in the same manner, this time by using region numbers of regions being bridged instead of segment numbers. Since connecting every pair of segments which exist in the network with a bridge would be very expensive and unnecessary, only selected segments are connected through bridges. The main function of the network layer is to route data from sending station to destination station through intermediate bridges. A routing algorithm is therefore essential for this layer.

3.2. Characteristics of Bridges

1. Bridges are assumed to have infinite buffer space.

2. Every Bridge acts as a pair of stations which tune in to two different logical rings. Each bridge-station therefore contends for token before transmitting data in the medium.

3. Incoming data from a certain priority from one end of the bridge is queued to the other end on the same priority.

4. Incoming data is treated in FIFO basis on both directions.

3.3. Routing Algorithm

Since no routing algorithm currently exists for a network connected by bridges, a hierarchical algorithm was devised for this simulation. The algorithm uses a tree structure with variable number of children. Segments residing in a region are configured as a tree

structure in which segment one acts as the root node. Regions are also configured as another tree structure with region one acting as the root node. The tree structure of segments and regions differ only in that one structure contains segment_no while the other contains region_no. The structure of the tree is shown below:

```
typedef struct sm {
        struct sm * next;
        int         segment_no;
} seg_lvl_struct;

typedef struct seg_tree {
    int segment_no;
    /* node level in the tree, root => 1 */
    short level;
    struct seg_tree * first_child;
    struct seg_tree * siblings;
    /*pointer to older child */
    struct seg_tree * left_sibling;
    struct seg_tree * parent;  /* parent node */
    /*same level connections*/
    seg_lvl_struct  * sm_lvl;
} seg_tree_struct;
```

The declaration above is illustrated in Fig. 10.

A menu for the tree structure is provided by this program. It enables users to view the tree structure and move from one node to another with relative ease. It also shows the relationships between the current node (this node is highlighted) to any other node on the screen. A screen dump of the menu is presented in Fig. 11.

Each segment contains this structure in its declaration and each region contains a similar structure. The user decides the configuration of the tree structures. A bridge exists between a pair of parent-child relationship in the tree structure. Since it is a tree, every node in the tree structure except the root should have one parent. Children of the same parent can communicate through the parent node. The user can create a tree structure with as many levels as desired. The level of the root node is defined to be one. Once the tree structure is configured, nodes which reside on the same level can also be connected by bridges.

Connection of two regions is actually a connection of two segments of the two regions. In this simulation, segment one of each region is designated to be the communication media for all stations which reside in that region to stations of other regions.

The routing algorithm uses these two tree structures to decide which segment it will route data to. This routing algorithim is simple and yet fairly general. It enables users to configure arbitrary configurations with relative ease.

Example 1

Assume that there are seven segments (logical rings) in region one, and a ring structure is desired. The tree structure is then configured as shown in Fig. 3.1.

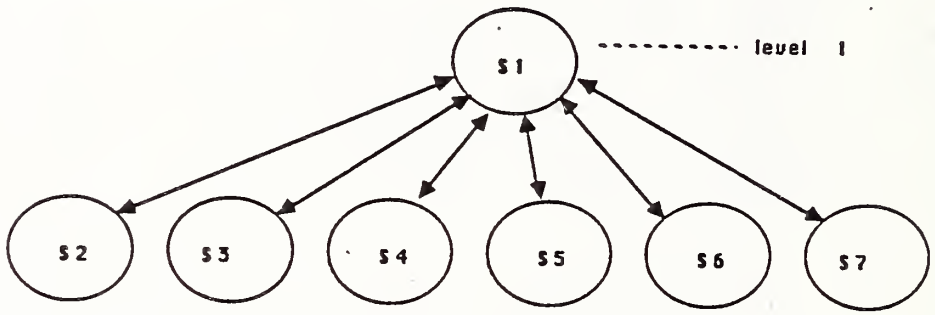


Fig. 3.1.

This configuration forces all communications between segments to travel through segment one. Let's then assume that after running the simulation it is realized that transmission time between segment two and segment six needs to be faster. A natural enhancement is to add a brige between segment two and six. This is possible because segment two and segment six are in the same region and both are of level two. This addition of a bridge causes segments two and six to communicate directly while leaving all other routes unchanged. This updated configuration is shown in Fig. 3.2.

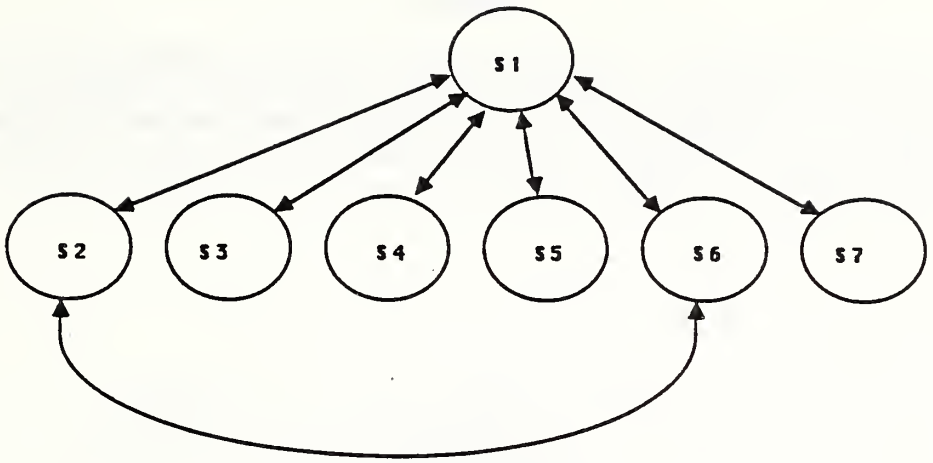


Fig. 3.2.

Example 2

A straight chain of segment can be easily configured as shown in Fig. 3.3.

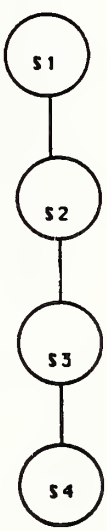


Fig. 3.3.

Example 3

Sometimes a bridge between two segments having different parent is desired. This is allowed as long as the two segments are in the same region and both of them are of the same level. Figure 3.4. shows such a configuration

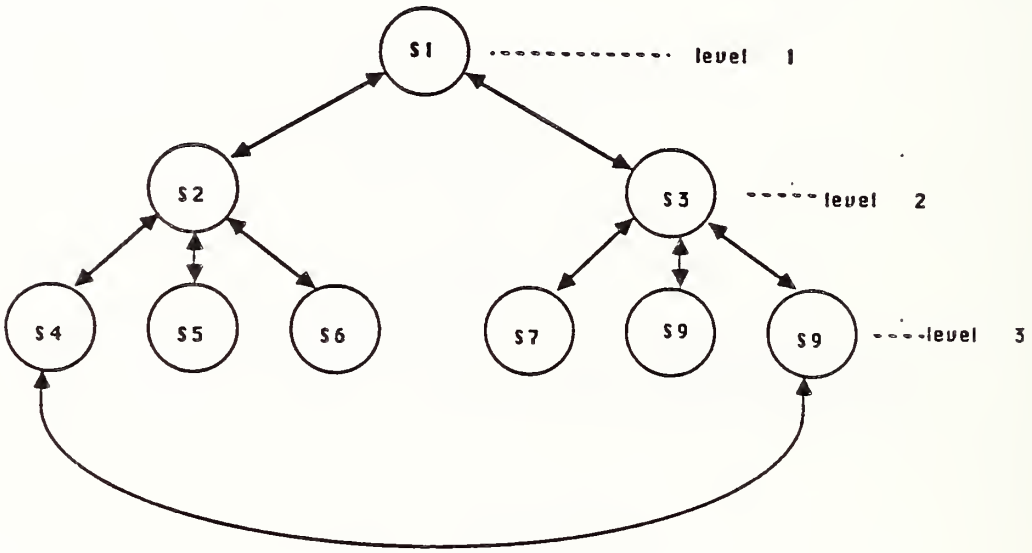


Fig. 3.4.

Example 4

This last example, shown in Fig. 3.5. illustrates a relatively large network. Routes for several segment-pairs inside a region as well as between regions are shown in Fig. 3.6. The routes from one region to another are analogous to routes between segments in a region.

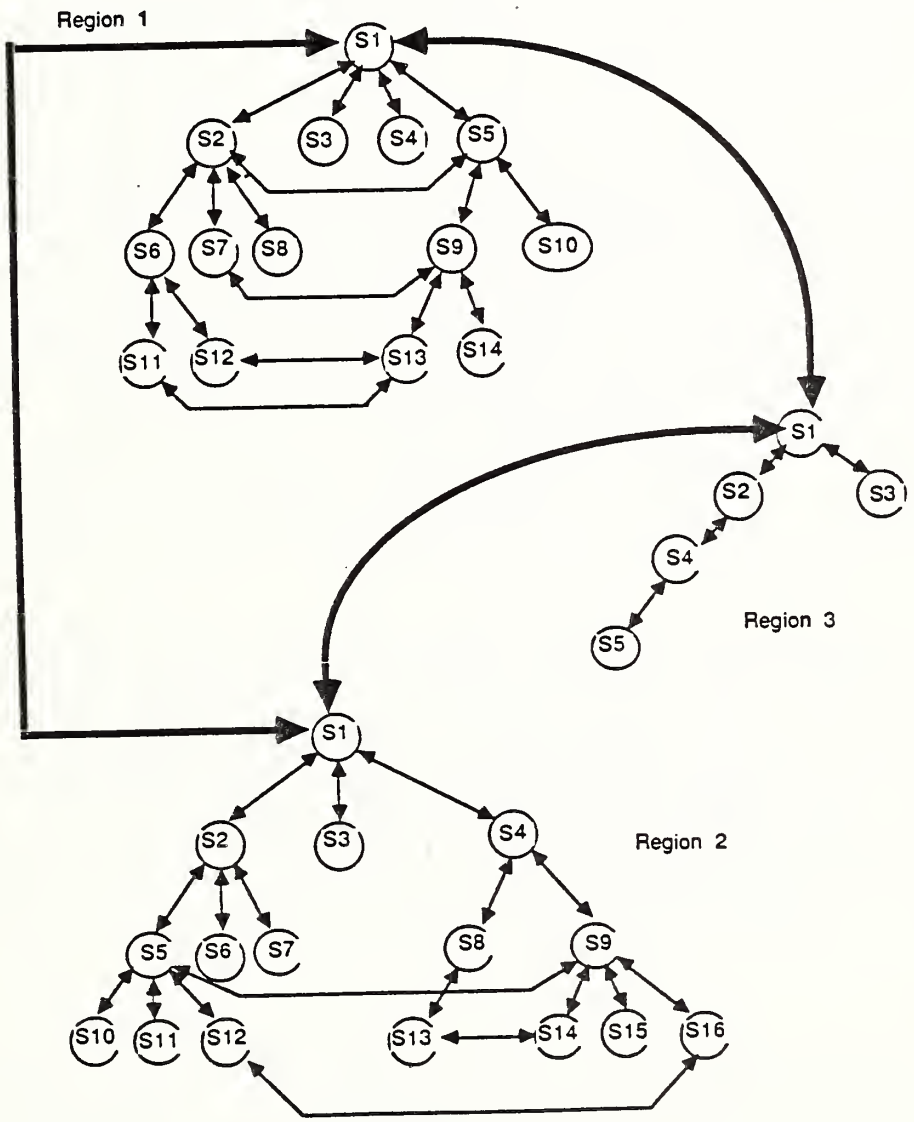


Fig. 3.5.

FROM region	segment	TO region	segment	ROUTES region	segment
1	11	1	14	1	11
				1	6
				1	2
				1	5
				1	9
				1	14
1	11	1	13	1	11
				1	13
1	7	3	3	1	7
				1	2
				1	1
				3	1
				3	3
2	10	2	15	2	10
				2	5
				2	9
				2	15
2	6	2	8	2	6
				2	2
				2	1
				2	4
				2	8

Fig. 2.6.

The flow-chart of this algorithm is presented in Fig. 12.

4. Transport Layer

4.1. Discussion

This layer is supposed to provide an end to end communication between two stations; but since this simulation program is used specifically for communication among ground floor devices, the function of this layer is tailored to serve a more specific task: generation of input messages.

This chapter also discusses briefly the mechanism to store and retrieve network configuration into/from physical storage.

4.2. Generation of Input Messages

Input messages on the network have specific source address, destination address, message length, frequency of transmission and priority. Messages with the same source address belong to the same station. Addition and deletion of messages are done dynamically by the user through the data menu. Each message is assigned a range of number depending on the frequency of the message. The higher the frequency, the higher the range of the number. This range of number is used to match to numbers generated by the random number generator described next. Keep in mind that the messages above are actually templates of messages instead of actual messages. The actual messages are created and deleted dynamically.

Having set up all message templates in the network, a mechanism to actually send the message is needed. This message generation is done by utilizing a congruential random number generator. This random generator produces uniformly distributed random numbers. When a station is timed, it calls the random number generator which returns a number. If the range of number of any message in the station covers the number generated by the random number generator, the message is transmitted; otherwise no message is sent in that time signal. The random nature of the generator makes the simulation more realistic.

If a message is transmitted, a copy of the message is stored in layer4_queue until the message is acknowledged by its destination station. when the message is acknowledged, it is freed from the queue and response time statistics of that particular message is updated. At the end of the simulation, user can get statistics of all the messages in the network. This statistics includes the number of messages sent during the simulation time, the worst response time and the average

response time of the message. The simulation program also provides a histogram of response time of any message in the network. This histogram is done by printing ASCII characters on the screen which, even though not fancy but maintains the portability feature of the program.

4.3. Save and Retrieve Configurations

Once a network configuration is established, a mechanism to save the configuration into physical storage is a must. Saving a configuration requires saving all regions in the network and their bridge connections, all segments inside a region and their connections, and all stations inside a segment and their variables. The algorithm uses the multi-children tree structure which exists for both region level and segment level. For the segment level, each segment corresponds to a node in the tree structure. The segments are stored by traversing the tree in preorder walk; that is root is stored first, then left_child node and finally the right_child node. In addition to storing all the segments which is a node in the tree structure, the algorithm also saves segments which are not connected to any other segments through bridges (free segments). For a more detail discussion of tree walk, consult chapter 3 of [4].

Retrieving a configuration from a file requires more work. The program will first ask if user wants to save the current configuration and acts accordingly. Before loading a configuration, the current configuration has to be completely deleted. This deletion requires freeing of all linked-list elements in the network. This deletion function is intricate since it has to guarantee that no pointer is left dangling. Once the current configuration is deleted, the file can be loaded by algorithm discussed in [4].

5. User Interface

5.1. Discussion

User interface plays a big role in a CAD program. The manipulation of the screen to make the program "friendly" to the users is discussed here. The user interface program does not require any conceptual understanding of any kind, but a knowledge of terminal settings and the curses utility is necessary. The program relies heavily on function check_input which checks the input buffer and decides whether number, character, number and character or arrow key is pressed by the user. The user interface is helpful to both users and programmer. It is helpful for users since without the

user interface, the program can not be used at all and, it is helpful to programmers for it shows any unexpected results instantly on the screen. The user interface takes the most time to program since it requires many nitty-gritty functions and is being used to make sure that the program runs as expected. Many extra precautions were taken in the development of this program to minimize possibility of fatal error. The philosophy behind the simulation program is to make the program simple, easy to understand, modular and adaptable to changes. The simulation program does not provide any analysis of the token-bus protocol, but it has provided a foundation for further development by providing a working token bus protocol, a simple routing algorithm and input-output capability which will be discussed in the next chapter.

This simulation program uses some rudimentary functions of the "curses" facility listed below:

1. move: This function moves the cursor to a certain position on the screen, for example "move (5,20)" moves the cursor to position y = 5 and x = 20.

2. printw: This function works like printf and starts printing from the current position of the cursor.

3. standout: This function starts a reverse video output on the screen.

4. standend: This function ends a reverse video output on the screen.

5. refresh: All the functions above do not actually change the screen appearance until a refresh command is encountered.

Curses initializes the screen by using function initscr and uses function endwin to reinitialize the screen before getting out from the program. Curses uses terminal database which is usually stored in a file called termcap. Every terminal has its own "termcap database" in that file and curses just needs to know the terminal type to access the correct database. Curses is therefore independent from any terminal type.

5.2. Implementation of Segment Menu

Even though there are several menus in this simulation, the screen manipulation of each menu uses basically the same technique. The implementation of segment menu is discussed briefly here. The source code of this function (seg_data_conn) is stored in file seg_conn.c

The segment menu shows bridges which exist within a region. It enables user to position the cursor to any intersection of two segments. If a bridge exists in the intersection, the intersection is marked, otherwise it is left blank. A screen dump of the segment menu is presented in Fig. 13. Notice that some intersections might not be used since intersection of row-segment 2 and col-segment 1 is identical to intersection of row-segment 1 and col-segment 2. In this simulation the lower half of the screen is used. Implementation of segment menu uses the following data structure:

```
/*-------------------------------------------------*/
typedef struct {
     short y_st,x_st;
     short y_length;
     short x_length;
     bridge_ptr br_ptr;
}cell_struct;
typedef struct {
     cell_struct matrix[11][7];
     cell_struct hor_header[7];
     cell_struct ver_header[11];
}screen_type;
/*-------------------------------------------------*/
```

The screen is represented by variable "screen" of type screen_type above. The fields ver_header and hor_header of screen_type correspond to windows which are used to display the corresponding segment numbers. Each element of the two-dimensional array matrix corresponds to an intersection of two segments, and contains pointer to corresponding bridge if a bridge exist between the two segments or NULL otherwise.

Some variables are crucial to the program and are discussed below:

1. Constants ROW_NO and COL_NO refers to the number of vertical windows and horizontal windows on the screen respectively. ROW_NO is set to 10 and COL_NO is set to 7.

2. Seg_cur_cell of type cell_struct points to the current window position with respect to two-dimensional array "matrix". An example of assignment to this variable is seg_cur_cell = & matrix[5][2]. The window which seg_cur_cell points to is the intersection highlighted on the screen.

3. Find_cur_cell of type boolean. This variable indicates whether seg_cur_cell already points to an intersection or not. A value of FALSE corresponds to seg_cur_cell equals to NULL and no intersection is being highlighted on the screen.

4. Seg_cur_hor of type segment_ptr points to the col_segment of the intersection being highlighted.

5. Seg_cur_ver of type segment_ptr points to the row-segment of the intersection being highlighted.

6. Seg_cur_x of type integer refers to the position of the highlighted intersection with respect to the left-most intersection. When left-most intersection is being highlighted, the seg_cur_x equals to 0. The right-most intersection to (COL_NO - 1) which is 6.

7. Seg_cur_y of type integer is equivalent to seg_cur_x but refers to the y axis. Seg_cur_y ranges between 0 to (ROW_NO - 1).

8 Hor_bridge_start and ver_bridge_start are fields of the region_struct and are used to point to the start of row-segments and col-segments being displayed on the screen.

9. Cur_pos points to current command being highlighted on the screen.

The previous screen dump with its comments should suffice to clear up any ambiguity. The flow chart of this function is shown in Fig. 14. In all, the simulation program provides seven interactive menus which are configured in a hierarchical way as shown in Fig. 15.

6. Conclusion

This simulation closely follows the token-bus protocol of IEEE-802.4; hence its resulting response time is expected to be close to the real network. By simulating the protocol very closely, however, the program runs much slower than its simulated network. Two things can be done relating to this simulation program. One is to improve the technique in simulating the time and the set of timers affected by the progression of time. Specifically when a station is transmitting a large number of bytes of data into the bus, say 10,000 octets, the simulation actually traverses the linked-list elements 10,000 times, updating the time after every pass, eventhough no stations change its ACM state within the 10,000-octet-time period.

This time-simulation can be easily enhanced within a single logical ring (segment) by increasing the time value directly by 10,000 as well as updating all affected timers; but since many segments, each transmitting different size of messages, are simulated simultaneously with the same set of timers, this enhancement is becoming rather complicated. The pay-off nevertheless, is very tempting, especially for a network which transmits mostly large chunks of data.

Another field of interest is to design a replacement communication model based on communication theories by statistical model, and use the simulation program as a benchmark to test the model. When this model is completed, it would be a powerful tool to analyze any arbitrary network without having to run the simulation.

Reference

1. "Network Layer Principles", European Computer Manufacturing Association, Final Draft #ECMA/TC24/82/18 (April, 1982)

2. "GM MAP SPECIFICATION", Version 2.1 (March, 1985)

3. "IEEE Standard 802.4 - 1984 and ISO Draft International Standard 8802/3", Draft F, IEEE P802.4/84/30 (July, 1984)

4. "Data Structure Techniques", Standish A.T. (1980)

Basic simulation of multiple "concurrent" stations

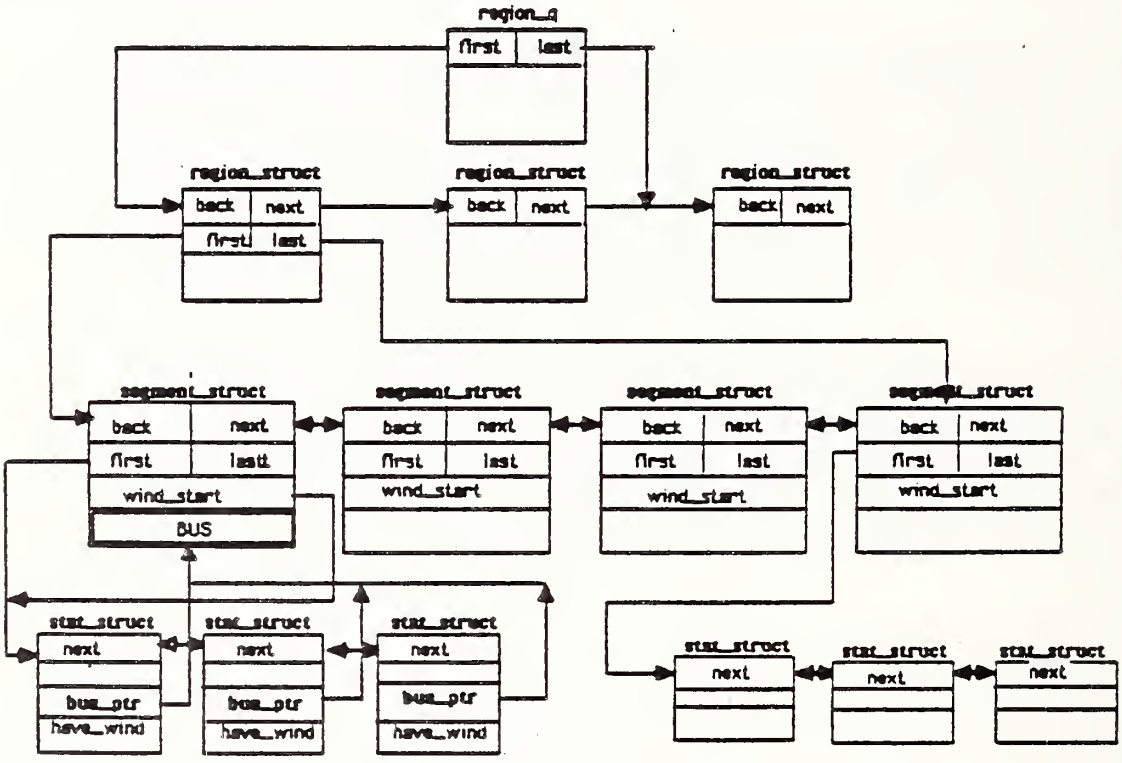


FIGURE 1

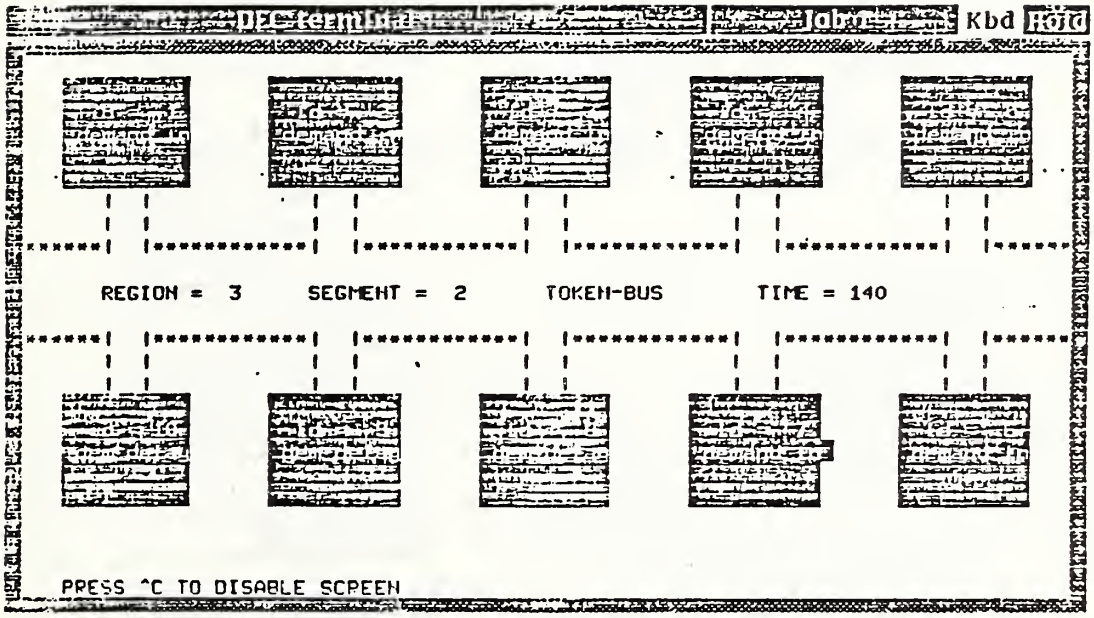

Kbd
Hold
REGION = 3
SEGMENT = 2
TOKEN-BUS
TIME = 140
PRESS ^C TO DISABLE SCREEN

Fig. 2

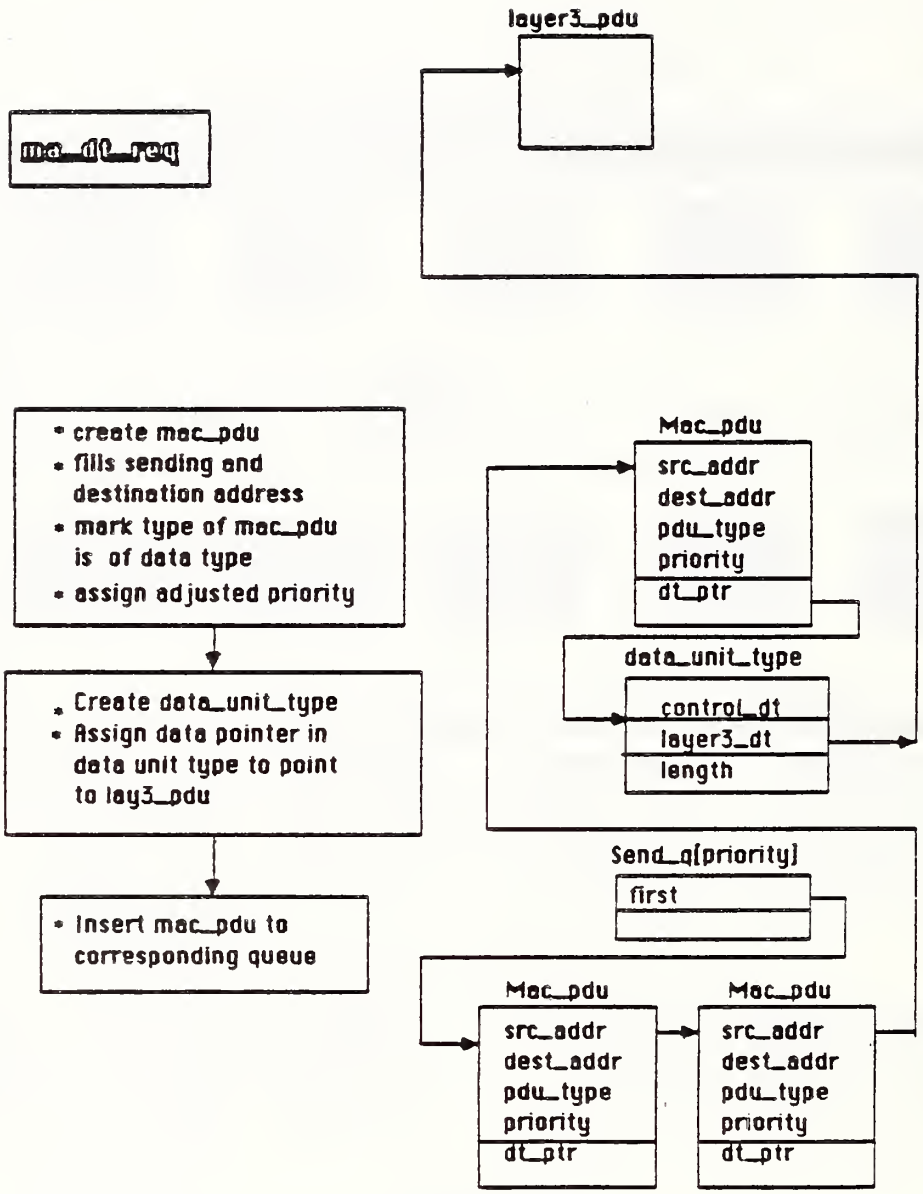

layer3_pdu
ma_dt_req
• create mac_pdu
• fills sending and destination address
• mark type of mac_pdu is of data type
• assign adjusted priority
Mac_pdu
src_addr
dest_addr
pdu_type
priority
dt_ptr
data_unit_type
control_dt
layer3_dt
length
• Create data_unit_type
• Assign data pointer in data unit type to point to lay3_pdu
Send_q[priority]
first
• Insert mac_pdu to corresponding queue
Mac_pdu
src_addr
dest_addr
pdu_type
priority
dt_ptr
Mac_pdu
src_addr
dest_addr
pdu_type
priority
dt_ptr

Fig. 3

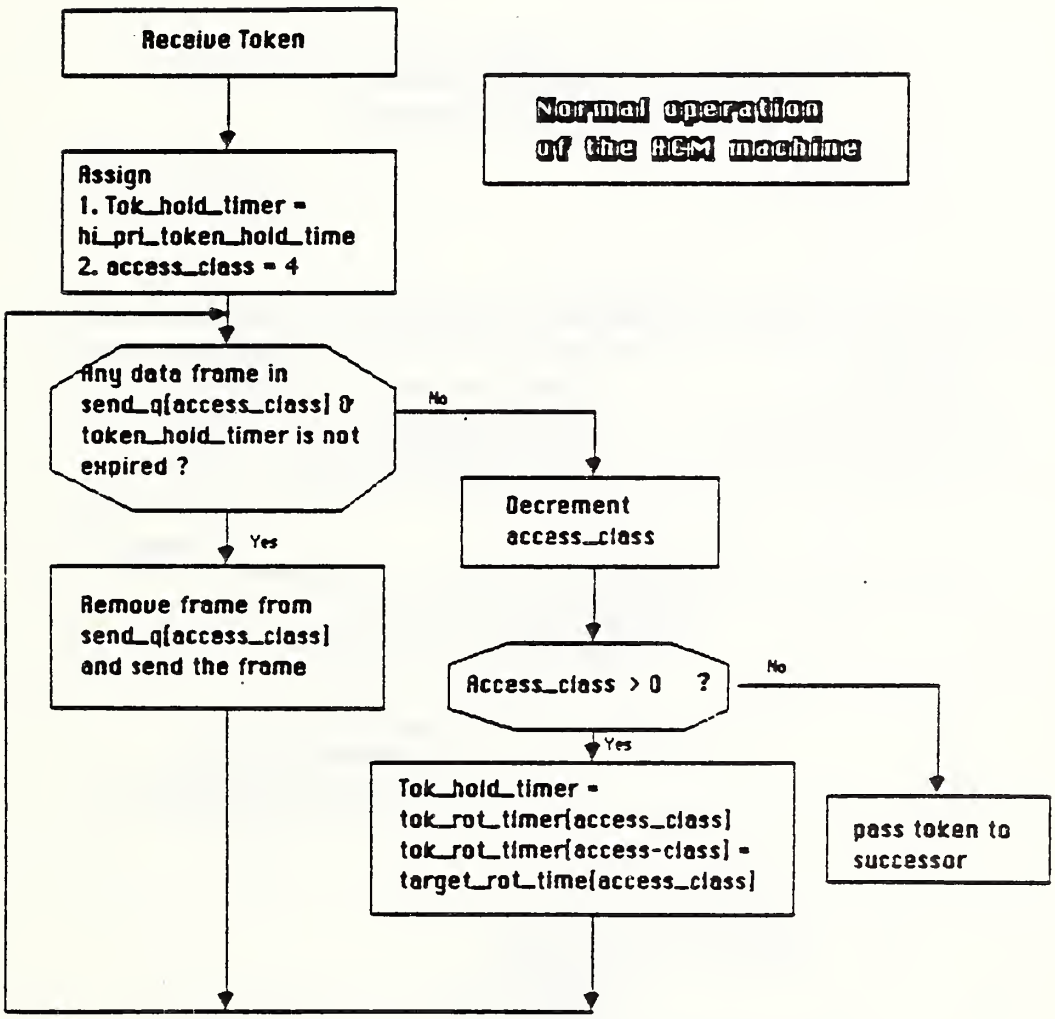

Receive Token
Normal operation of the ACM machine
Assign
1. Tok_hold_timer = hi_pri_token_hold_time
2. access_class = 4
Any data frame in send_q[access_class] & token_hold_timer is not expired ?
No
Yes
Decrement access_class
Remove frame from send_q[access_class] and send the frame
Access_class > 0 ?
No
Yes
Tok_hold_timer = tok_rot_timer[access_class]
tok_rot_timer[access-class] = target_rot_time[access_class]
pass token to successor

Fig. 4

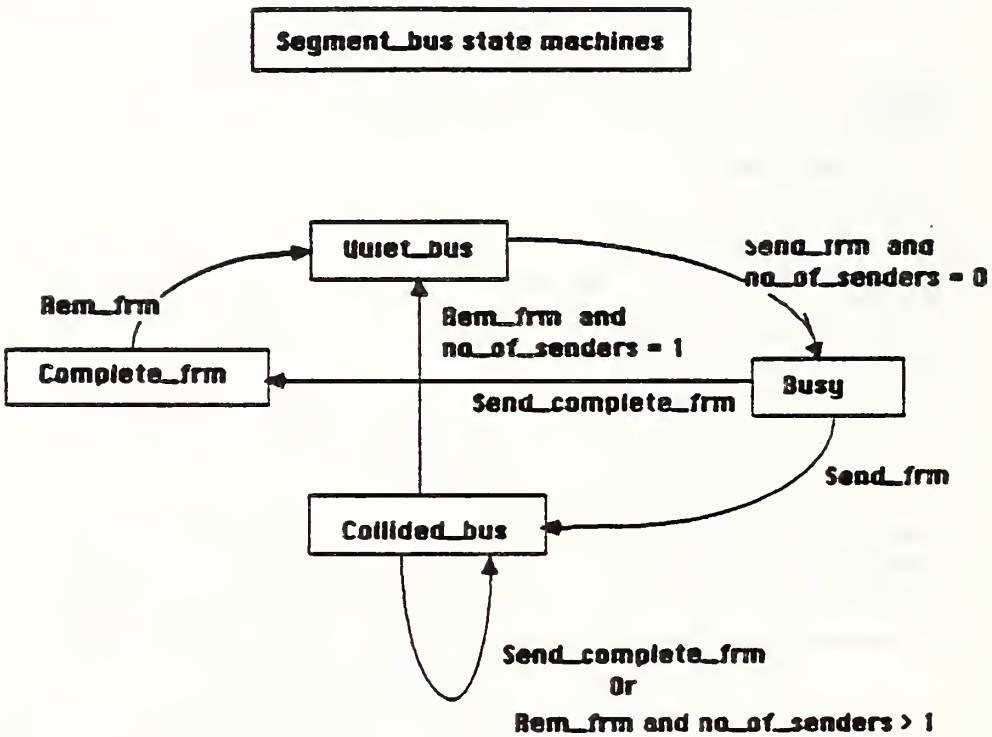

Segment_bus state machines
Quiet_bus
Send_frm and
no_of_senders = 0
Rem_frm
Rem_frm and
no_of_senders = 1
Complete_frm
Busy
Send_complete_frm
Send_frm
Collided_bus
Send_complete_frm
Or
Rem_frm and no_of_senders > 1

Fig. 5

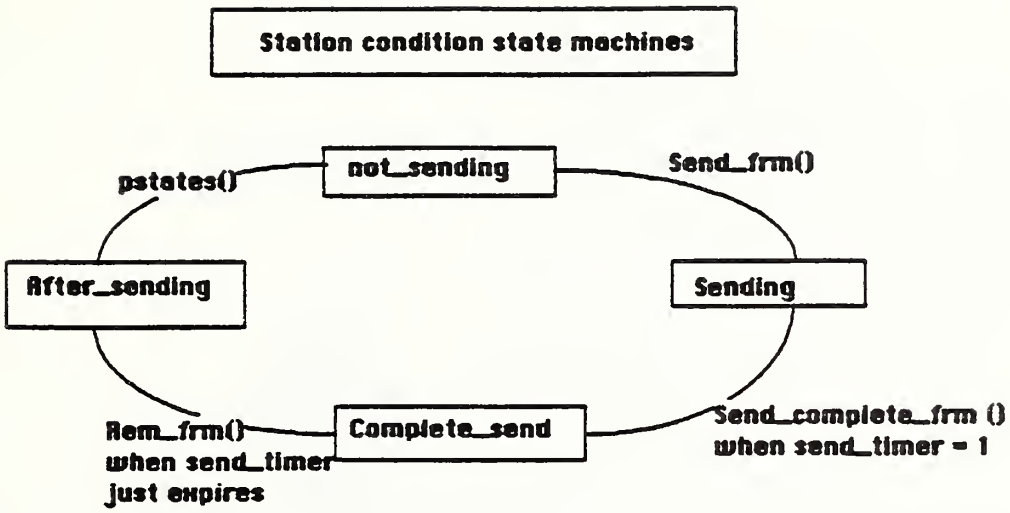


Fig. 6

Receive Machine (RxM)

Busy or collided_bus => bus_quiet = False;

Complete_frm => bus_quiet = TRUE;

=> Assign rx_data_frm or
rx_prot_frm to TRUE
depending on data type
received.

Quiet_bus => if station is not bus_quiet
then set both bus_quiet
and noise_burst to TRUE

Fig. 7

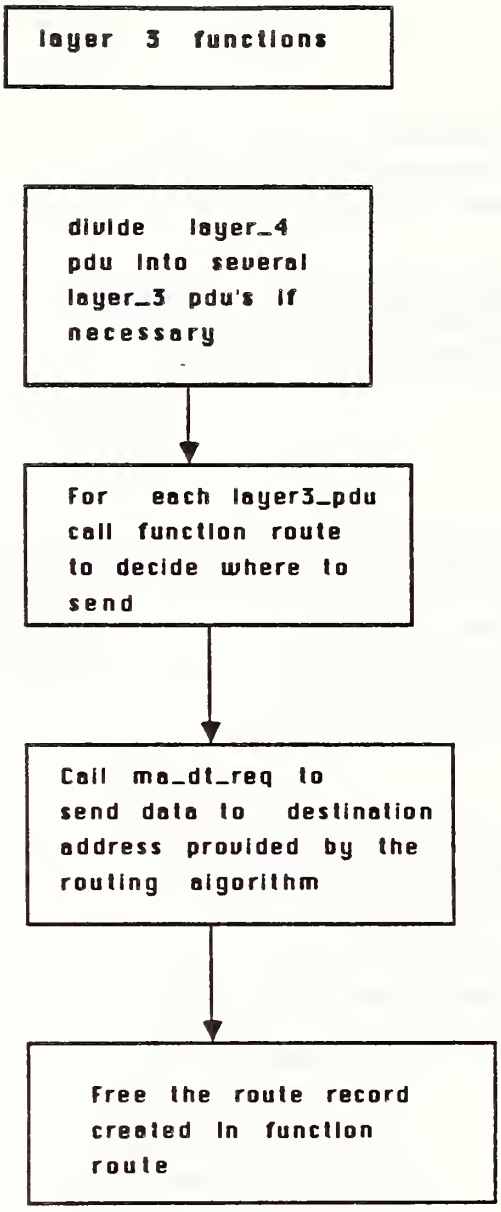

layer 3 functions
divide layer_4 pdu into several layer_3 pdu's if necessary
For each layer3_pdu call function route to decide where to send
Call ma_dt_req to send data to destination address provided by the routing algorithm
Free the route record created in function route

Fig. 8

Bridges inside a region

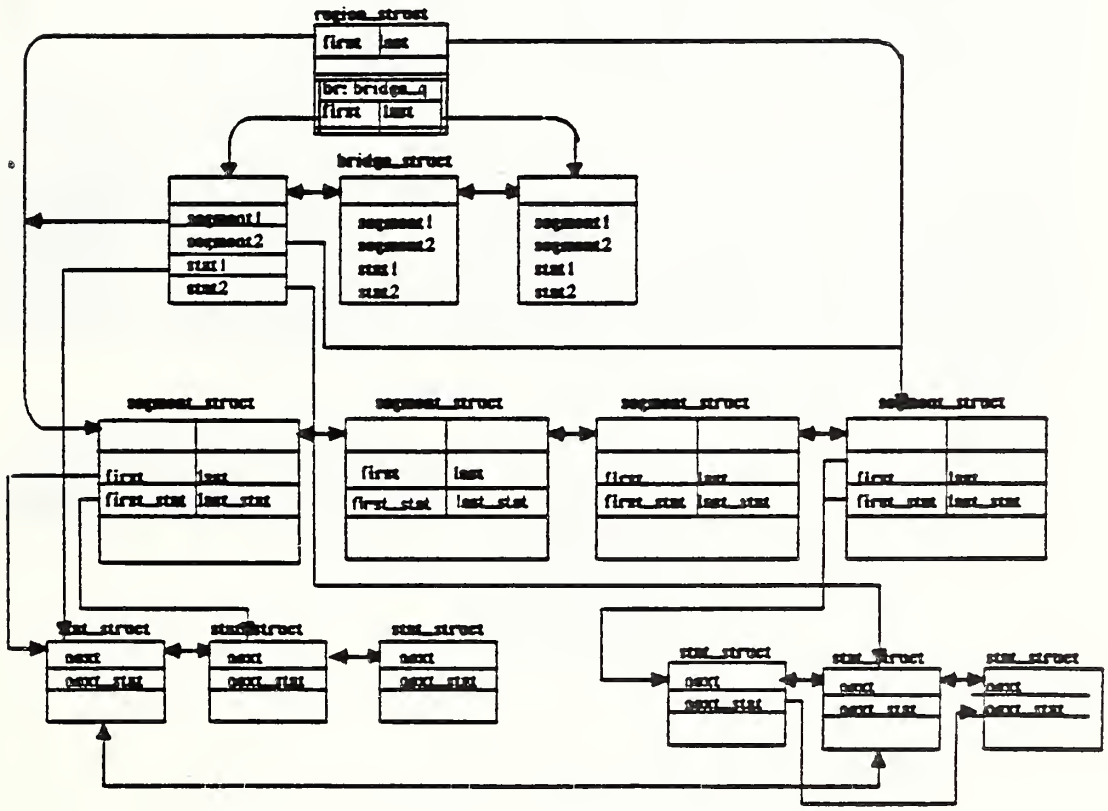


Fig. 9

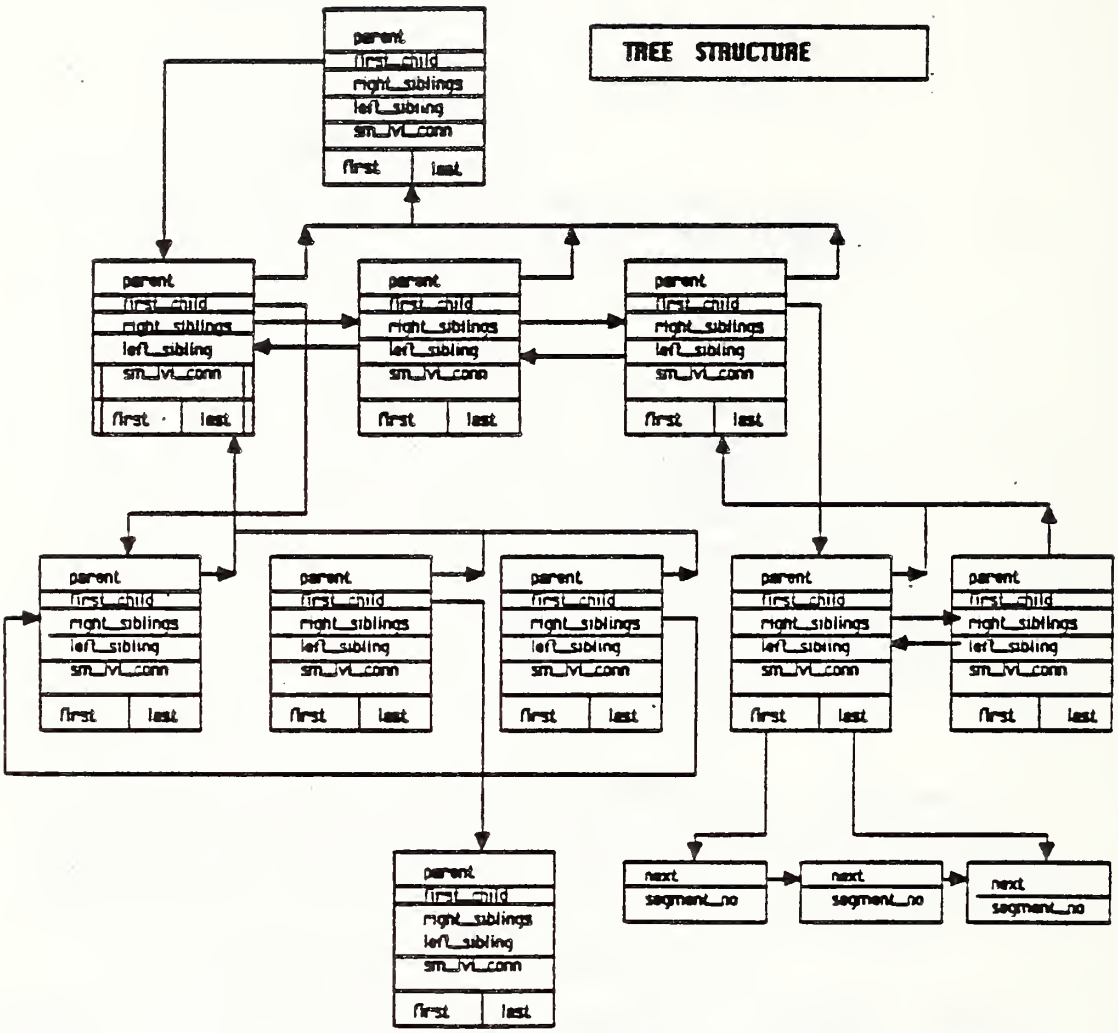

TREE STRUCTURE
parent
first_child
right_siblings
left_sibling
sm_vl_conn
first
last
next
segment_no

Fig. 10

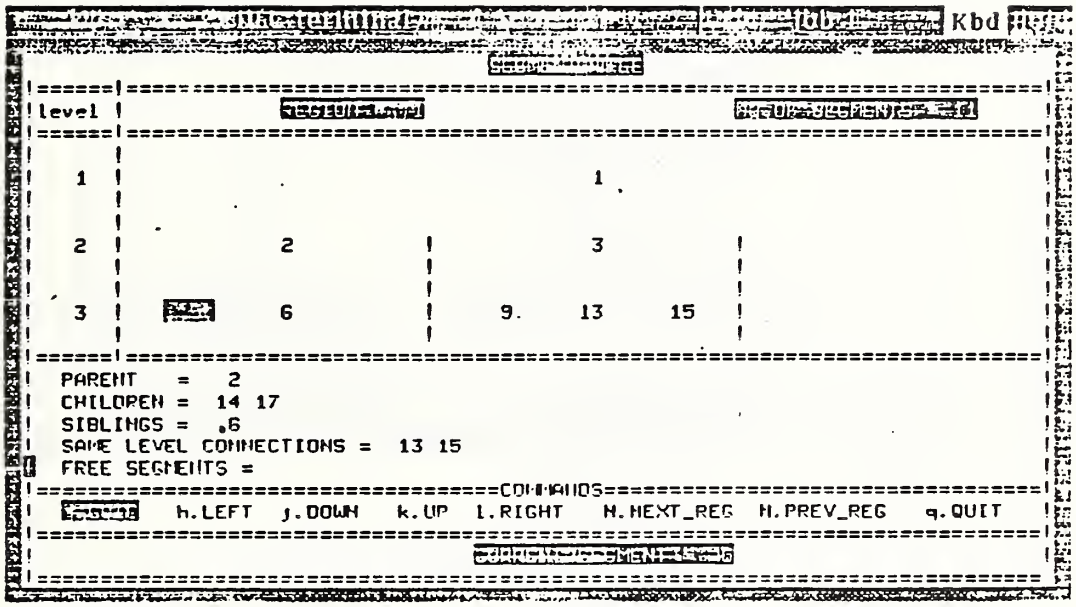

Kbd
level
1
2
3
6
9.
13
15
PARENT = 2
CHILDREN = 14 17
SIBLINGS = 6
SAME LEVEL CONNECTIONS = 13 15
FREE SEGMENTS =
COMMANDS
h.LEFT
j.DOWN
k.UP
l.RIGHT
N.NEXT_REG
P.PREV_REG
q.QUIT

Fig. 11

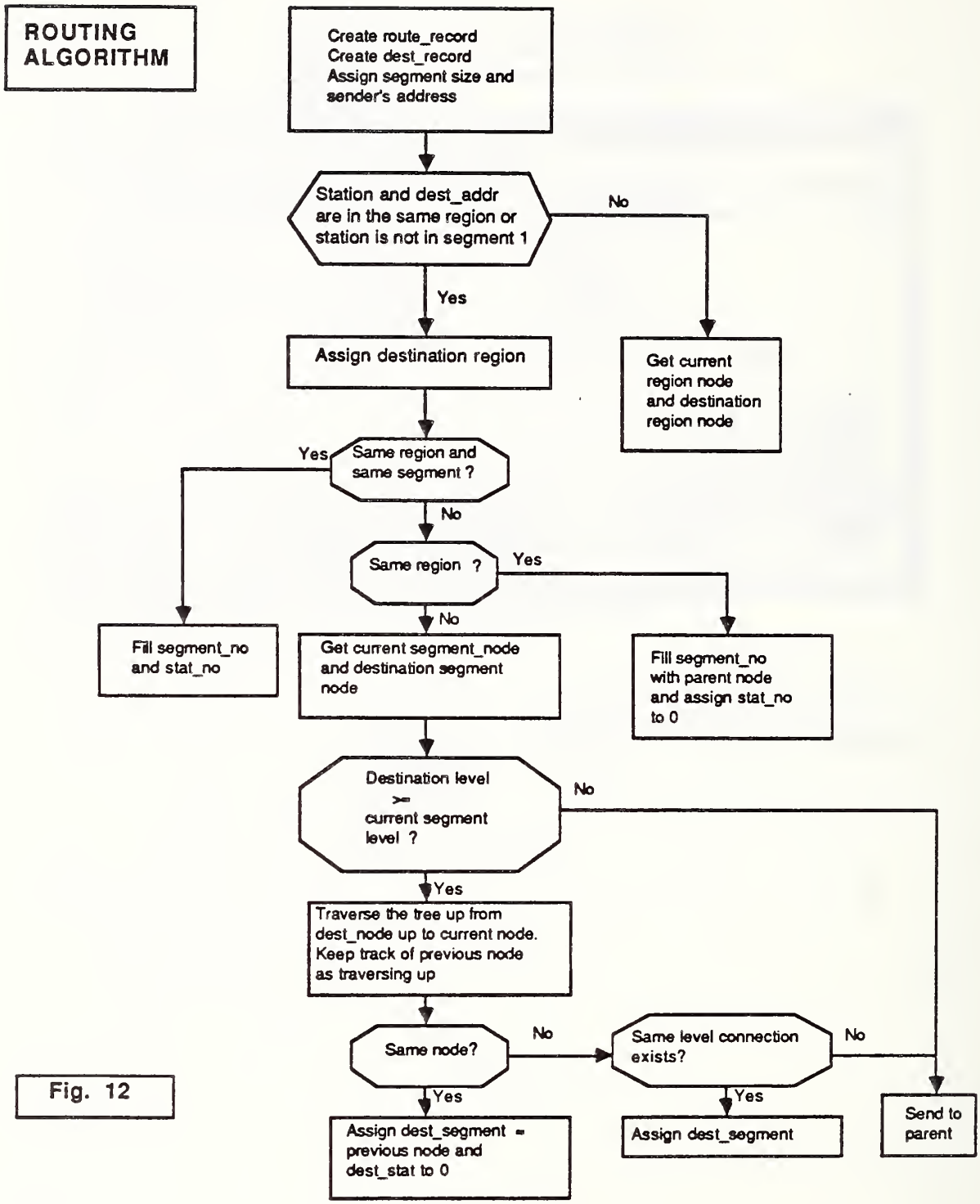
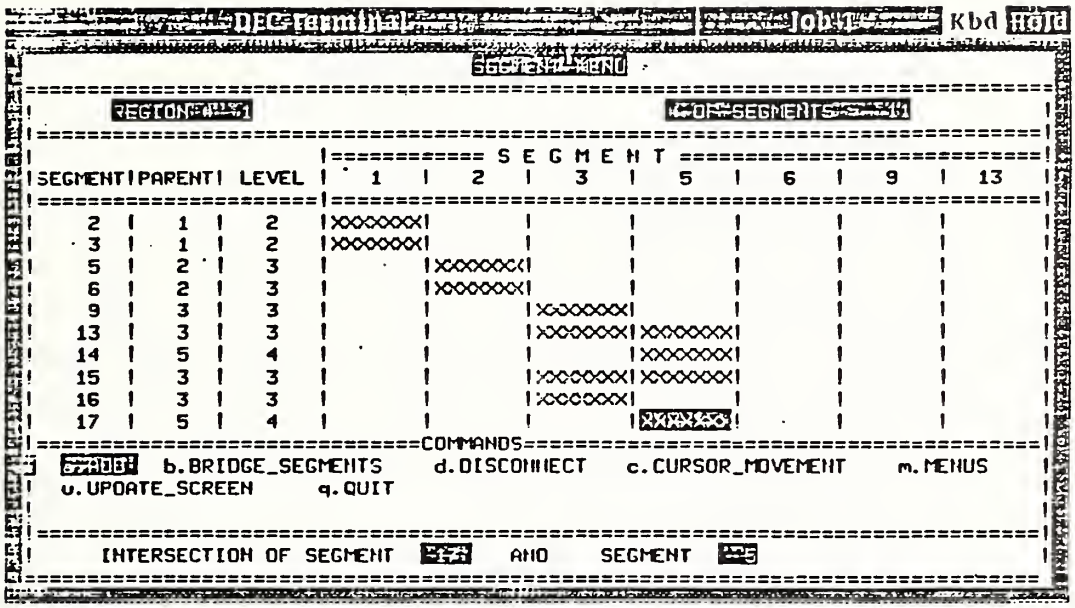

ROUTING ALGORITHM
Create route_record
Create dest_record
Assign segment size and sender's address
Station and dest_addr are in the same region or station is not in segment 1
No
Yes
Assign destination region
Get current region node and destination region node
Yes
Same region and same segment ?
No
Same region ?
Yes
No
Fill segment_no and stat_no
Get current segment_node and destination segment node
Fill segment_no with parent node and assign stat_no to 0
Destination level >= current segment level ?
No
Yes
Traverse the tree up from dest_node up to current node. Keep track of previous node as traversing up
Same node?
No
Same level connection exists?
No
Yes
Yes
Assign dest_segment = previous node and dest_stat to 0
Assign dest_segment
Send to parent

Fig. 12




```
Seg_cur_hor points to Segment 5     Cur_y_pos = 9
Seg_cur_ver points to Segment 17    Cur_x_pos = 3
Hor_bridge_start = Segment 1        Cur_command = 'c'
Ver_bridge_start = Segment 2
```

Fig. 13

SEGMENT MENU

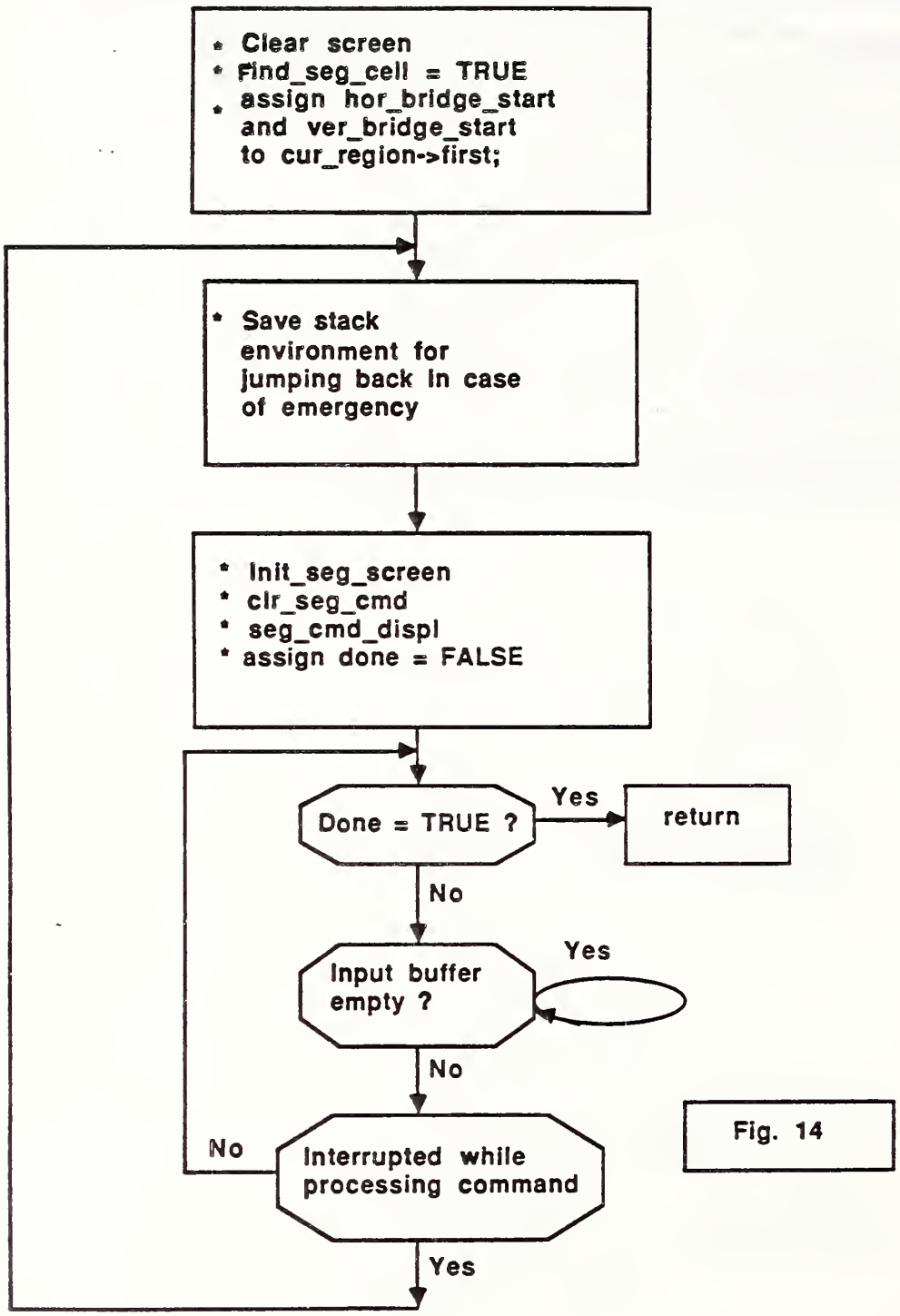


Fig. 14

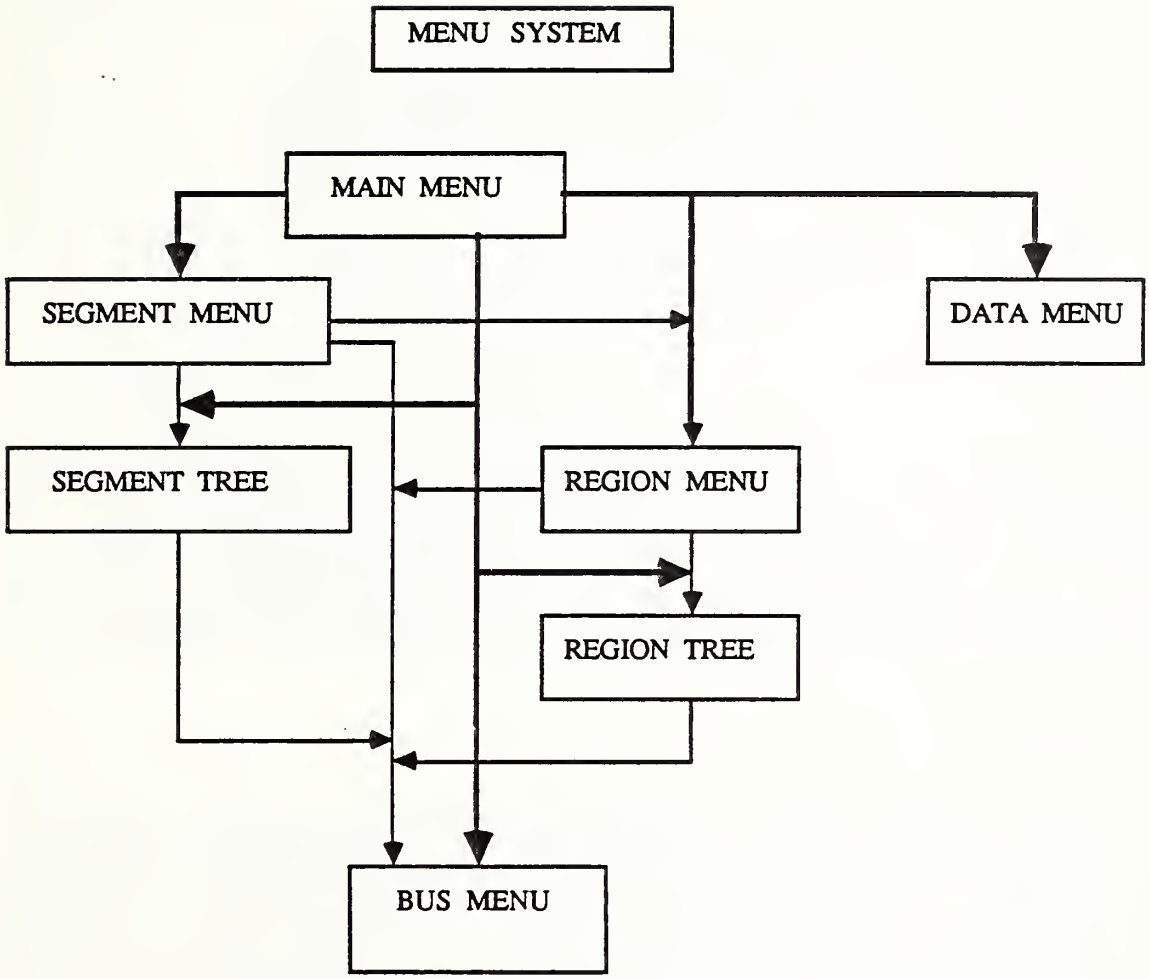

MENU SYSTEM
MAIN MENU
SEGMENT MENU
DATA MENU
SEGMENT TREE
REGION MENU
REGION TREE
BUS MENU

Fig. 15

Implementing a Factory MAP Network Simulation Tool Using the Ada[1] Programming Language

W. L. Schultz, Senior Member IEEE
In Don Bae, Student Member IEEE
Asheem Chandna, Student Member IEEE
Farrokh Khatibi, Student Member IEEE

Center for Automation and Intelligent Systems Research
Case Western Reserve University
Cleveland, Ohio 44106

Abstract

A methodology for the implementation of a factory scale MAP network simulation tool is presented. The methodology is centered on the task type replication facility and the task rendezvous facilities of the Ada programming language. The result allows physical network entities to be directly represented as program code modules in a form that is highly readable and easy to understand.

Introduction

Stations on a MAP[2] network are autonomous. This requirement associates a great deal of complexity in each station's inherent operational capabilities. Despite the deterministic nature of this complexity, the interactions among a large number of these stations is difficult to describe due to the sheer magnitude of their interaction possibilities. Thus, simulation offers a means for evaluating different MAP architectures, operating modes, and topologies prior to physically installing an actual network.

In order to simulate a large factory network, issues of: (1) topology, (2) interconnection of separate physical segments, (3) station mode of operation (Mini-MAP or Full MAP), and (4) traffic patterns (size and frequency of messages) must be taken into account. The representation and specification of these issues to a simulation program are problems in their own right. It is our intent to utilize the capabilities of an engineering graphics workstation for the convenient specification

---

1. Ada is a trademark of the Ada Joint Project Office, United States Department of Defense.

2. Manufacturing Automation Protocol (MAP) Standard, Version 2.2, General Motors Corporation, 1986.

and display of network simulation inputs, status during running, and results. However, this paper deals with the use of Ada for representing and simulating autonomous network entities, that is, components of the network.

Thus, we have split the job of factory MAP network simulation into components of: (1) user interface and (2) protocol implementation and simulation. This paper presents an approach to the latter problems using the several special facilities, and specifically the tasking mechanism, of the Ada programming language. In addition, we propose an alternative interim user interface method -- direct use of the Ada language.

In implementing the MAP protocol we have started with the lowest layers and are working toward higher layers following the incremental modeling approach described by Yeh[3] and further discussed by Rahimi and Jelatis[4] . In Ada cultural terms this approach would be called "stepwise refinement."

Our result is targeted primarily for use in the simulation of large factory networks. Our goal is to provide network designers and researchers with a new, more complete tool. Newcomers to the field should find the structure and details of our implementation helpful in furthering their understanding of MAP network issues.

Design Goals

A complete Factory Network Planning Tool must have a "toolkit" of network components including interconnection devices, message generators, and measurement tools accessible to the user. The toolkit is required to be compact in the sense that the simulation code necessary to represent a factory network must be reasonable -- it must fit in a computer available to the user. Speed is also an issue for practical simulators and is incorporated in these design goals as a call for good coding practices.

Using the packaging and tasking concepts and mechanisms of the Ada programming language[5] , the toolkit can be realized as a

---

3. Yeh, J. W., "Simulations of Local Computer Networks," Proc. 4th Conf. on Local Computer Networks, Oct. 22, 1979, pp. 56-66.

4. Rahimi, S. K., and Jelatis, G. D., "LAN Protocol Validation and Evaluation," IEEE Journal on SAC, Vol. 1, No. 5, Nov. 1983.

5. Gehani, N., Ada, An Advanced Introduction, Prentice-Hall, Inc., Englewood Cliffs, New Jersey, 1983.

convenient set of devices that can be connected together to form factory networks. The goal then is to realize each network entity as a Ada data structure, procedure, function, or task in such a way as can be easily replicated and interconnected by means of straight forward Ada program statements. The highly readable nature of the Ada Language will facilitate the checking and explanation of Factory MAP Network configurations so specified. The possibility to produce a highly readable implementation of the toolkit modules invites their careful and clear presentation to serve as a benchmark for other implementations.

Simulation Approach

Our approach relies heavily on the ability of the Ada Language to replicate MAP stations represented as concurrent tasks. This approach is similar to that taken by Powers and Nute[6] in their simulation of a CSMA/CD Network. The major entities of our system are (1) stations and (2) buses. We have kept our code nearly parallel to the procedures and data objects described in the appropriate standards documents. Where possible, we have used exactly the same names given to these entities by those documents. Careful use of reentrant procedures has minimized the additional memory required when a task is replicated. The main Ada program serves to specify the number of stations and buses, their particular interconnection, and the nature and type of communications among them.

Figure 1 illustrates a straight forward factory network plan composed of two different types of buses: the broadband backbone bus segment and several broadband or carrierband bus segments bridged to the backbone bus. Each bus, the backbone segment and each of the several bridged bus segments, has a separate logical token ring. It is expected that the number of stations on each bridged bus segment will be between 20 and 30, that the number of such segments connected to the backbone segment will be approximately 30, and that 20 additional stations (main frame computers, etc.) will be connected directly to the backbone segment. This yields an approximate size for systems to be simulated of roughly 750 stations and 30 buses.

Figure 2 illustrates the four main types of station present within a factory network. Each of these station entities has many functions in common with other stations; in particular, layer 2 is common to all and layer 1 chooses 1 of 2 possible media choices. In a similar fashion, layers 3, 4, and 5 supply common functionality to all stations using them. This leaves

---

6. Powers, W. S., and Nute, T., "Implementing a Simulator as a Set of Ada Tasks," Proc. of the Eastern Simulation Conference on Simulation in Ada, Norfolk, Virgina, March 3-8, 1985.

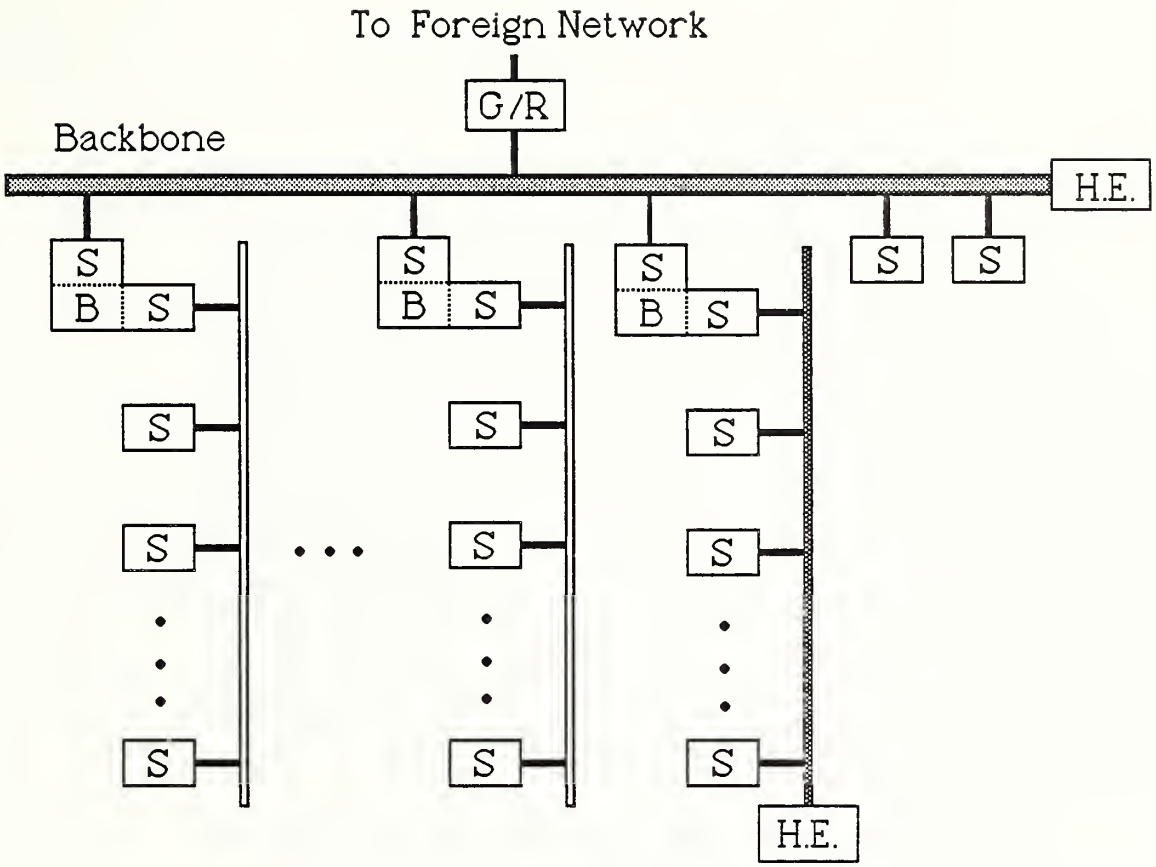


Figure 1. A Typical Factory Network
(Station and Bus Entities)

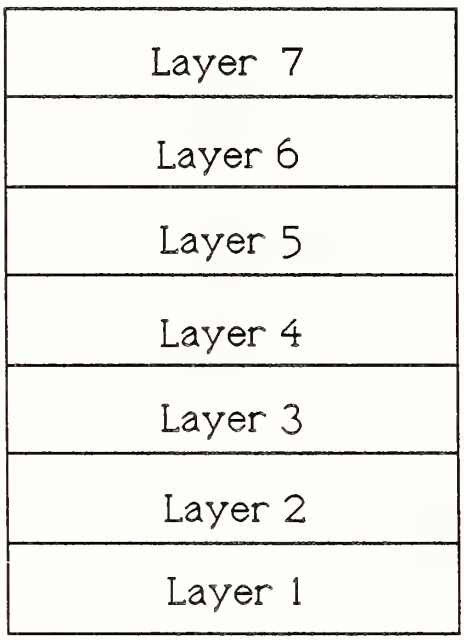


Figure 2a. MAP Station Entities
Full MAP Station

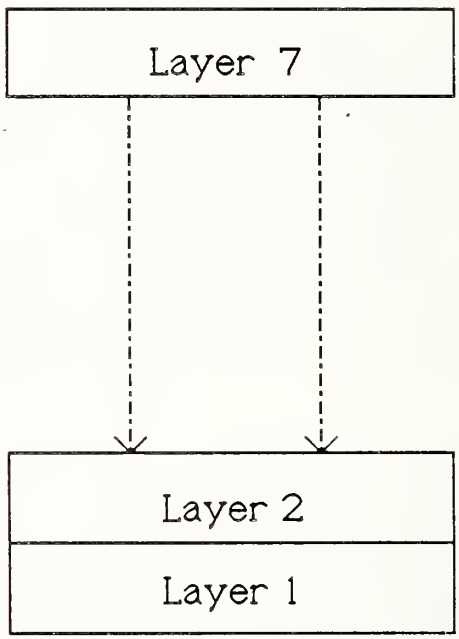


Figure 2b. MAP Station Entities
Mini-MAP Station

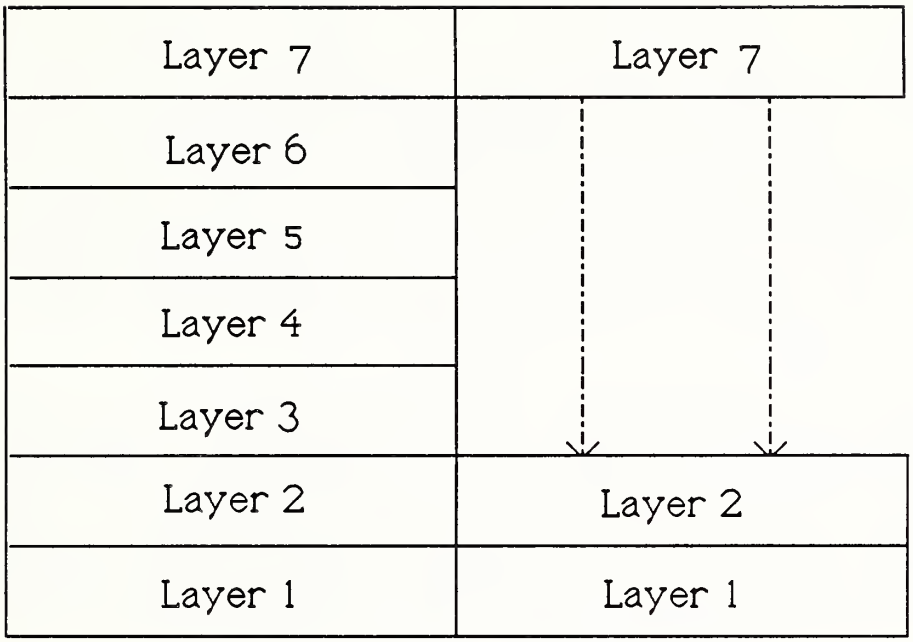


Figure 2c. MAP Station Entities
EPA - MAP Station

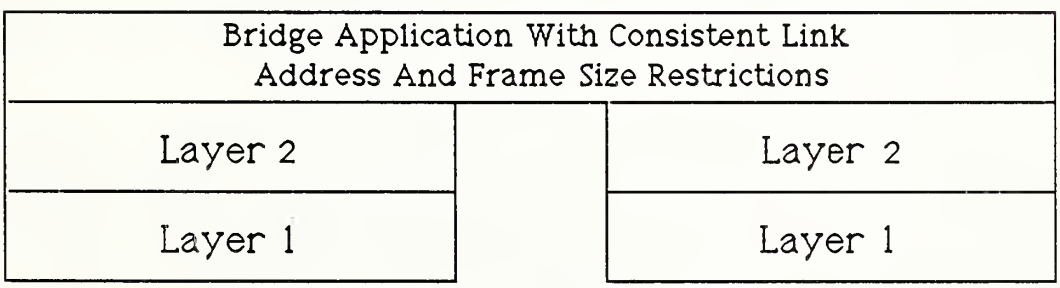


Figure 2d. MAP Station Entities
Two Segment Bridge

layers 6 and 7 as the only services that may have functional dependencies on the operating system or machine architecture.

Figure 3 shows the layers of a station implemented as "packages" of code connecting to its higher and lower layer neighbor by means of well defined procedural and data object "entries." In this representation the procedural entries are "sockets" while the adjoining layer procedure calls are "plugs" into the procedure entries represented by the "sockets." Data objects are represented by the formal parameters supplied by the calling layer. This representation, called a structure diagram, is fully described by Buhr[7] ; we use it throughout the remainder of this paper to indicate the organization of the code in our implementation.

Figure 4 (a through c) shows the detailed interconnections among the physical[8] (layer 1), MAC[8] sublayer (layer 2), LLC[9] sublayer (layer 2), and the station management functions. The connections shown are all that should be visible to other layer entities; they need not and should not access structures internal to these layers directly. Thus, by refining each layer, as shown for the MAC sublayer in Figure 5, attention for design and implementation can be focused on the issues within a particular layer.

Once implemented, a station must be replicated and connected to its copies. The vehicle for this connection is an entity, the bus, which serves to schedule, synchronize, and pass information among the several stations to be connected to that bus. In large factory systems, of course, there are many buses. Thus, the bus entity must be replicated in order to represent a large network; scheduling and synchronizing buses is accomplished by means of a global clock. Since buses do not pass information directly to other buses, the interconnection of buses must be done explicitly among stations connected to different buses. That is, a bridge station must be used to interconnect 2 buses.

In summary, our approach to the simulation of a factory MAP network is to implement a template for each of the network entities. These templates are then replicated and interconnected as needed to represent the configuration required. This technique rests on the Ada task typing facility to replicate the required network entities as autonomous, concurrent, yet

---

7. Buhr, R. J. A., System Design with Ada, Prentice-Hall, Englewood Cliffs, New Jersey, 1984.

8. ANSI/IEEE Std 802.4-1985, Token-Passing Bus Access Method and Physical Layer Specifications, IEEE, New York, 1985.

9. ANSI/IEEE Std 802.2-1985, Logical Link Control, IEEE, New York, 1984.

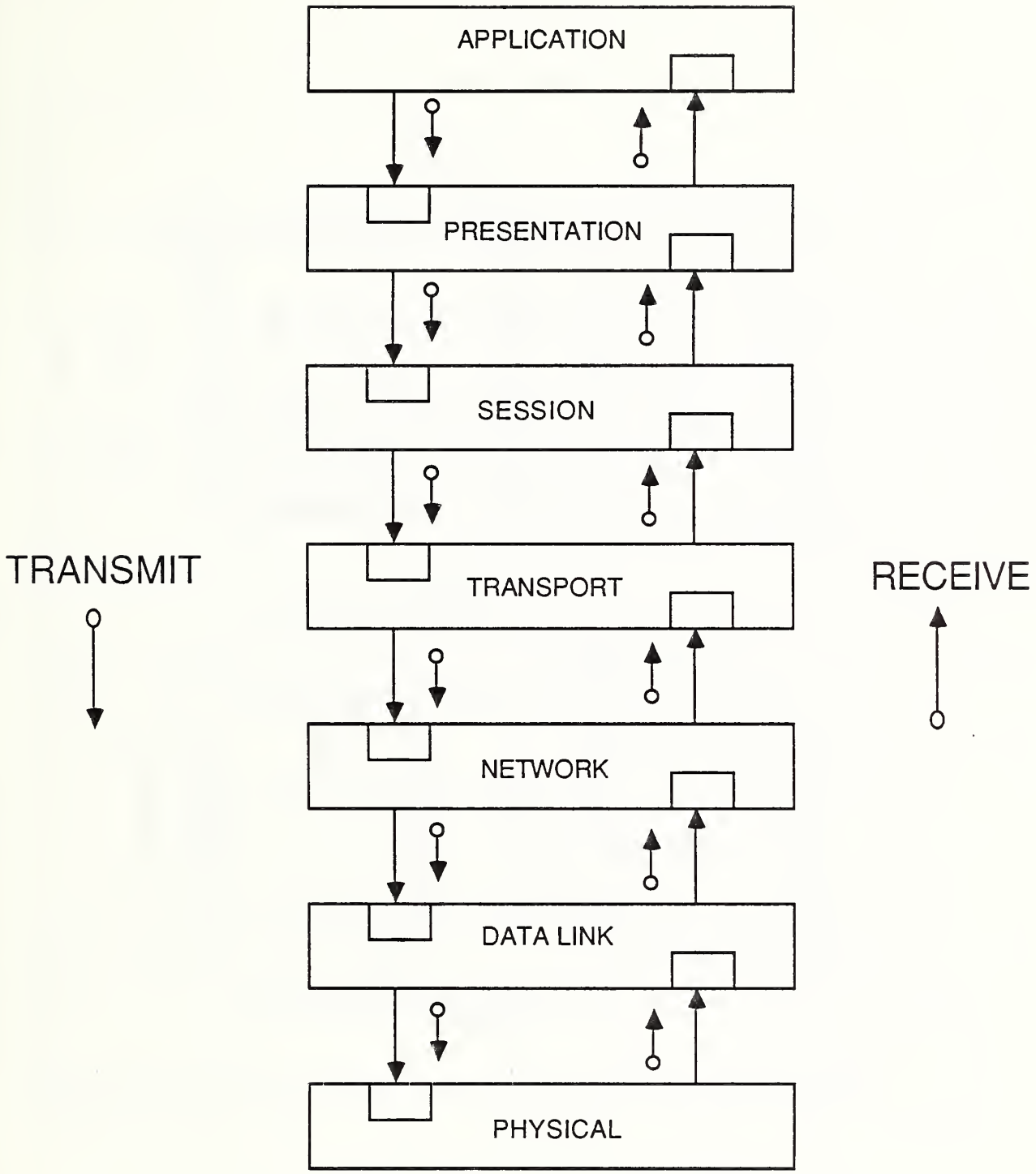


Figure 3. Simplified Structure Diagram for a Full MAP Station

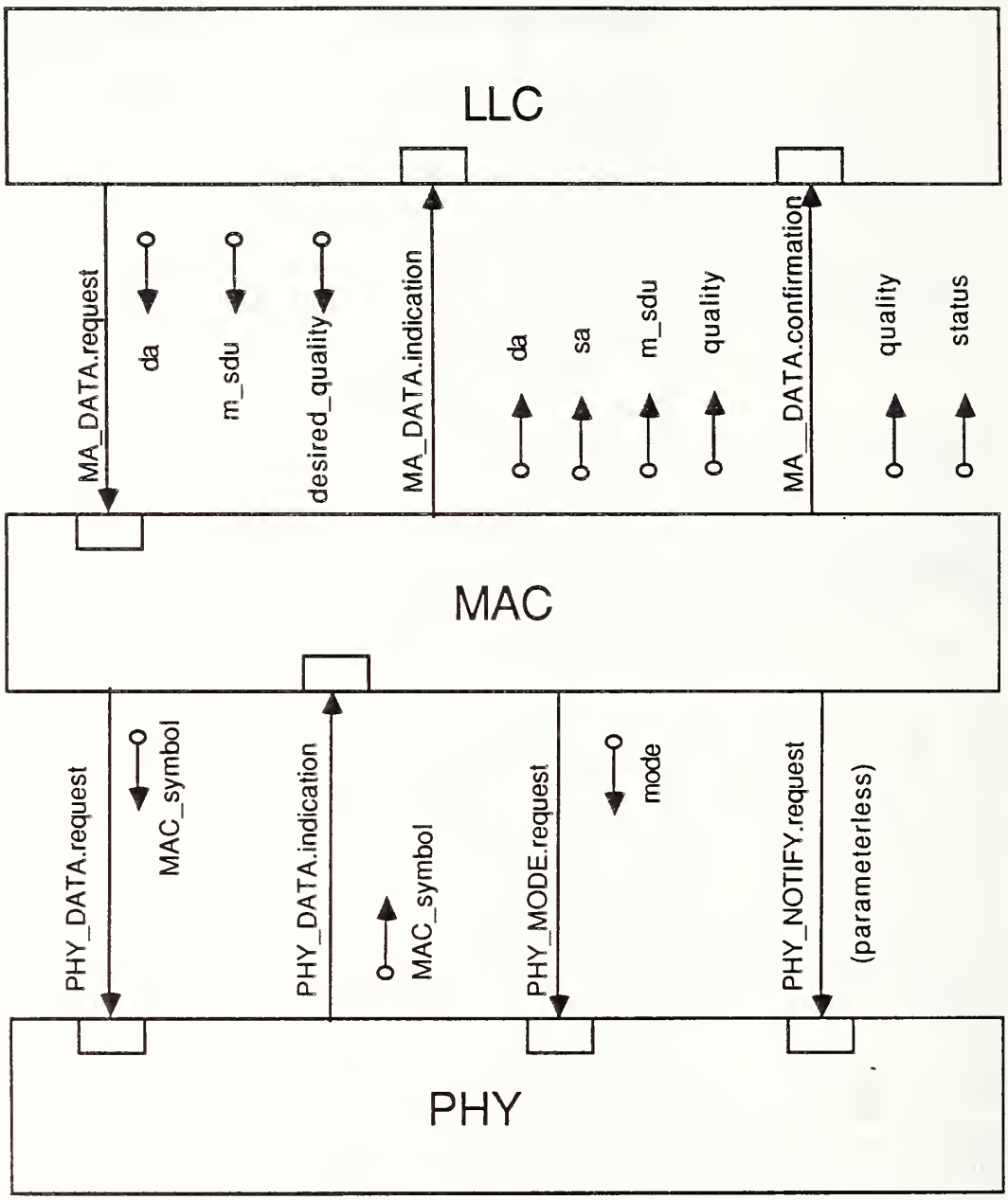


Figure 4a. Detailed Structure Diagram
Lower MAP Layers

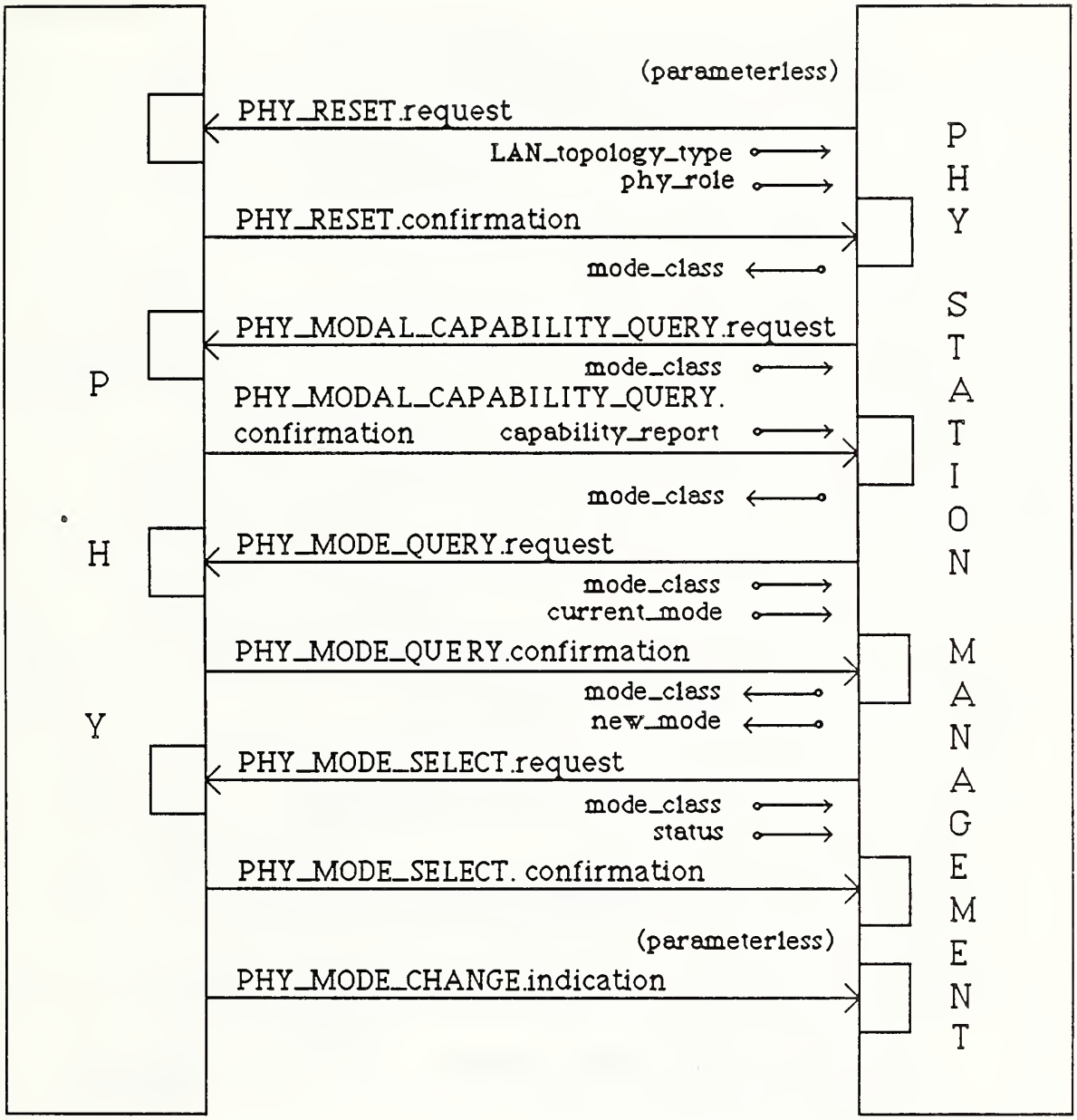


Figure 4b. Detailed Structure Diagram
Physical Layer to Station Management

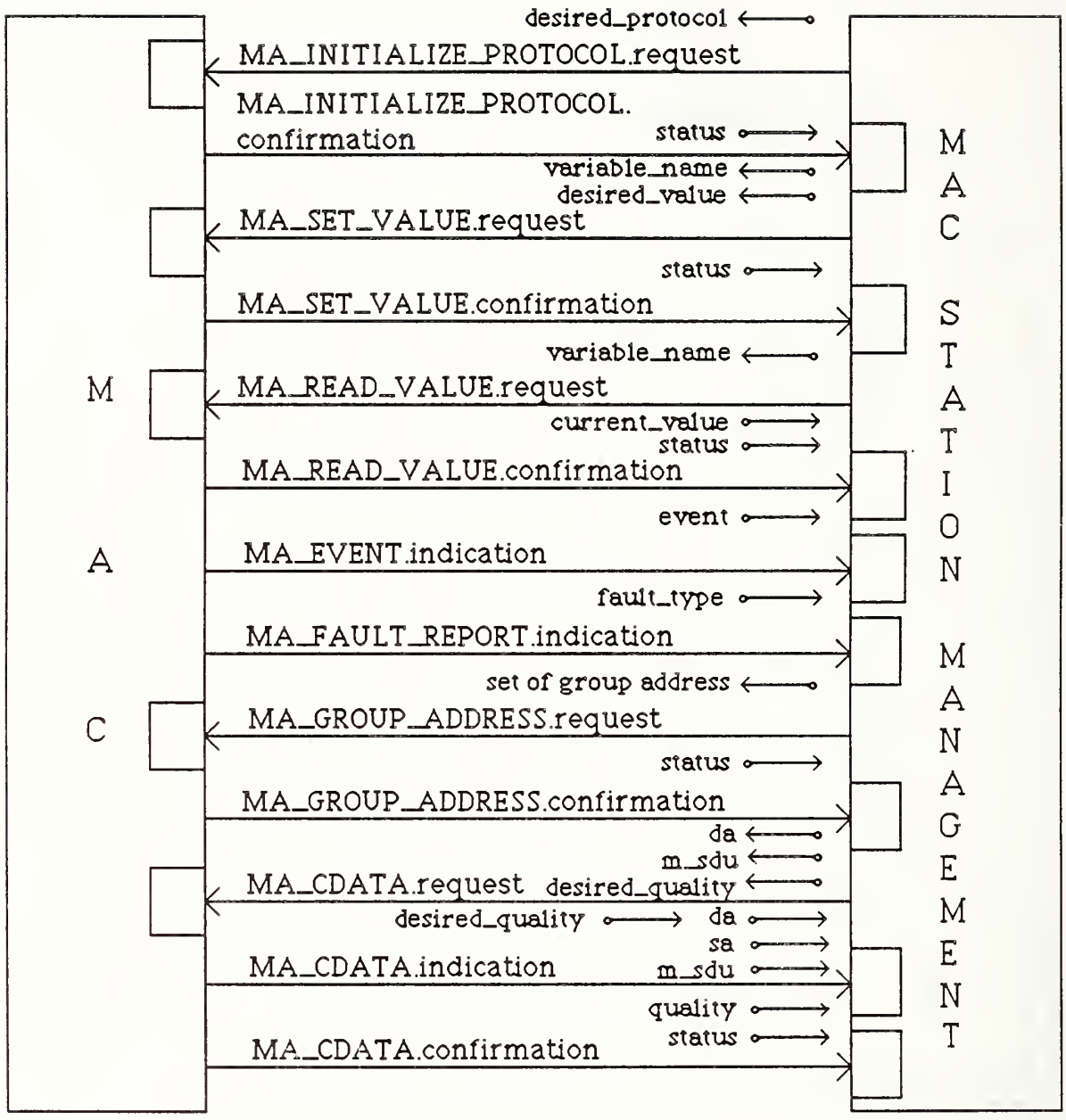


Figure 4c. Detailed Structure Diagram
MAC Sublayer to Station Management

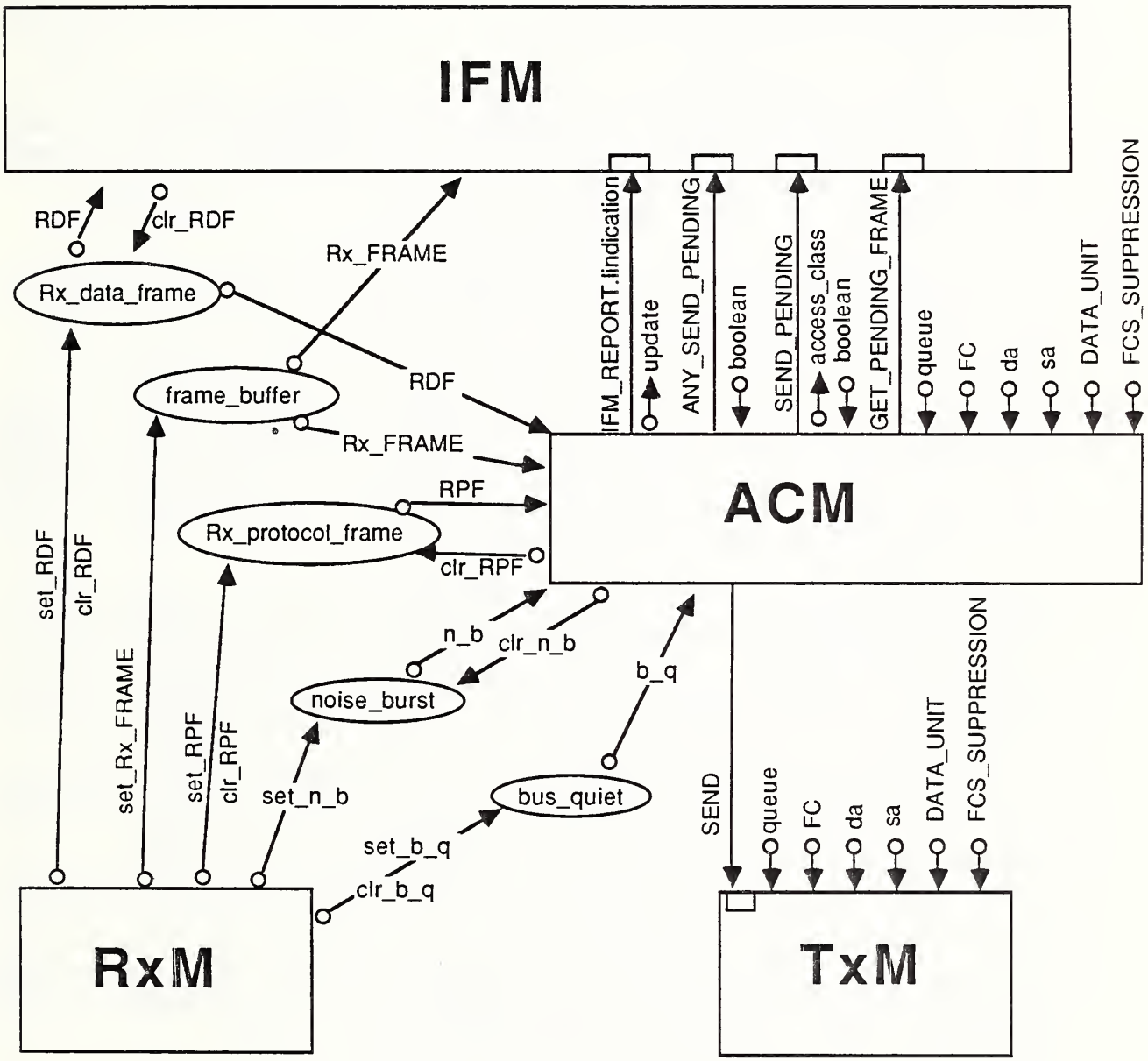


Figure 5. Structure Diagram For
MAC Sub-layer Internals

communicating tasks. Ada is the only presently available computer programming language in wide-spread use that supports this methodology.

Specifics in the Use of Ada

Each of the network entities simulated, the bus and the station, were implemented via Ada's task type facility. Replication of these entities is done by simply providing a set of unique names for the purpose of declaring tasks of type bus or type station in the same way that one might declare data variables in another language.

The replication of a task has some side effects, however. As might be expected, the memory consumed by 10 replicated tasks is about 10 times that consumed by one task -- a problem for a system intent on supporting 750 such tasks. Fortunately, Ada provides facilities for the coding of reentrant procedures. By coding the procedural parts of the station and bus tasks as reentrant procedures, one copy of these procedures serves all of the replicated tasks. With this programming refinement, only the variables local to the task and its minimum of program statements are replicated. This, at the present state of development, saves about 2500 lines of Ada code per station.

The needs for scheduling, synchronization, and mutual exclusion during data exchange are supported by means of Ada's rendezvous mechanism. In simple terms, this mechanism is the syntactical equivalent of a procedure call whereby a calling task names the required entry on a receiving task and provides access to its data areas for the exchange by specifying them as parameters of this "entry" call. The receiving task accepts a call at its entry using the syntactical equivalent of a procedure definition. For the time that the receiving task remains within the accept statement, the formal parameters of the accept statement provide direct access to the caller's data area -- the receiving task is free to read or modify the caller's data area as required. After completing the accept statement, both tasks are free to resume their activities in concurrent fashion. While the calling task was waiting for the receiving task to accept and process its call, however, it was prohibited from further activity. Thus, this rendezvous mechanism provides synchronization and mutual exclusion during data exchange implicitly. It also provides a convenient means for scheduling.

The convenient graphical technique provided by Buhr[7] is used to illustrate the relationship among tasks provided by the rendezvous mechanism. Figure 6 shows the relationship between bus tasks and station tasks. While some detail has been omitted for clarity, the diagram clearly indicates which tasks initiate an interaction with another task. In this illustration, the data to be received (or transmitted) by each station task is the current bus state and data on a particular bus.

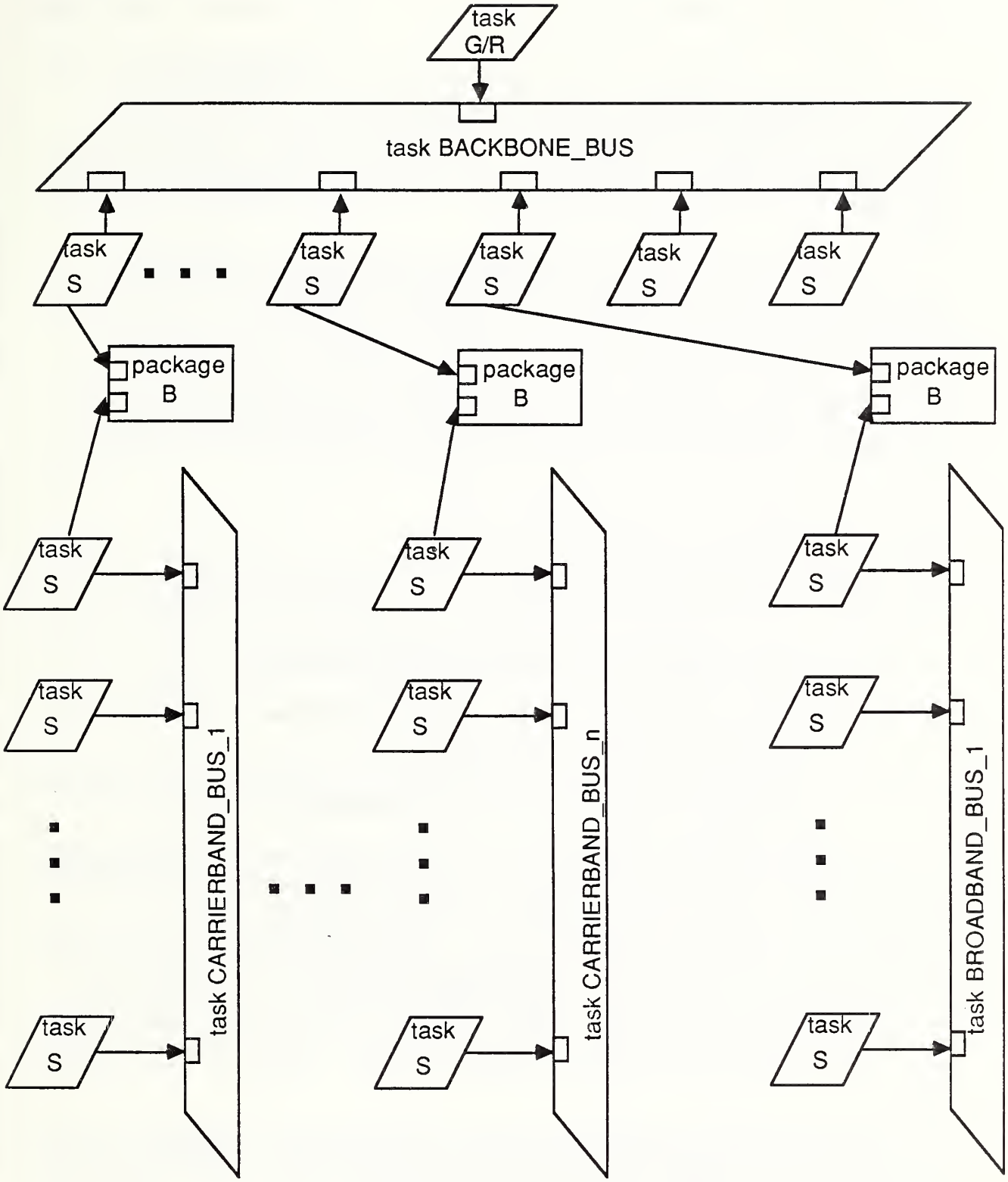


Figure 6. Structure Diagram of a Typical Factory Network.

Figure 7 presents a detailed view of the modular components used within a station. Figure 8 illustrates the use of this reentrant package of MAC sublayer components to implement one station including the connection to its bus task. As mentioned earlier, an important benefit of the reentrant package structure is memory savings when the package is shared among many station tasks.

In conclusion, the Ada language provides the facilities required for replicating concurrent tasks and for synchronizing communications between them. In addition, other Ada facilities, which permit the visibility or scope of a data or procedural entity to be limited to only those portions of code which need to use it, help to improve one's confidence in the reliability of the simulation. The fact that entities in the physical network are directly represented by code entities make the description and discussion of a network so simulated more easy to understand.

Partial Verification of the Simulator

Verification of each significant coded module proceeded as described in the following steps:

Step 1: The minimal test cases required to completely verify the module were deduced and documented.

Step 2: The expected results for each of the test cases were calculated and documented.

Step 3: Each of the test cases were simulated and the results compared with the calculated results.

To date we have verified all of the major simulator modules except for the ACM of the MAC layer which has been only partially verified due to its many states used only during error conditions. Partial verification of the ACM has been accomplished using the sequence of transitions and state changes required for the network to initialize the logical token ring. For this case, a varying number (from 2 to 20) of stations were switched on at exactly the same time in the state of "desiring to be in the ring" or "having frames to send." The results obtained from this simulation were the same as those expected (see step 2 above).

Figure 9 shows a graph of the time, in octet times, required for the logical ring to become fully established. Two cases are presented: (1) each station only desires to be in the ring but has no messages to transmit and (2) each station desires to be in the ring and has a short message to transmit. As expected, the transmission of a message by each station increases the time required to initialize the ring.

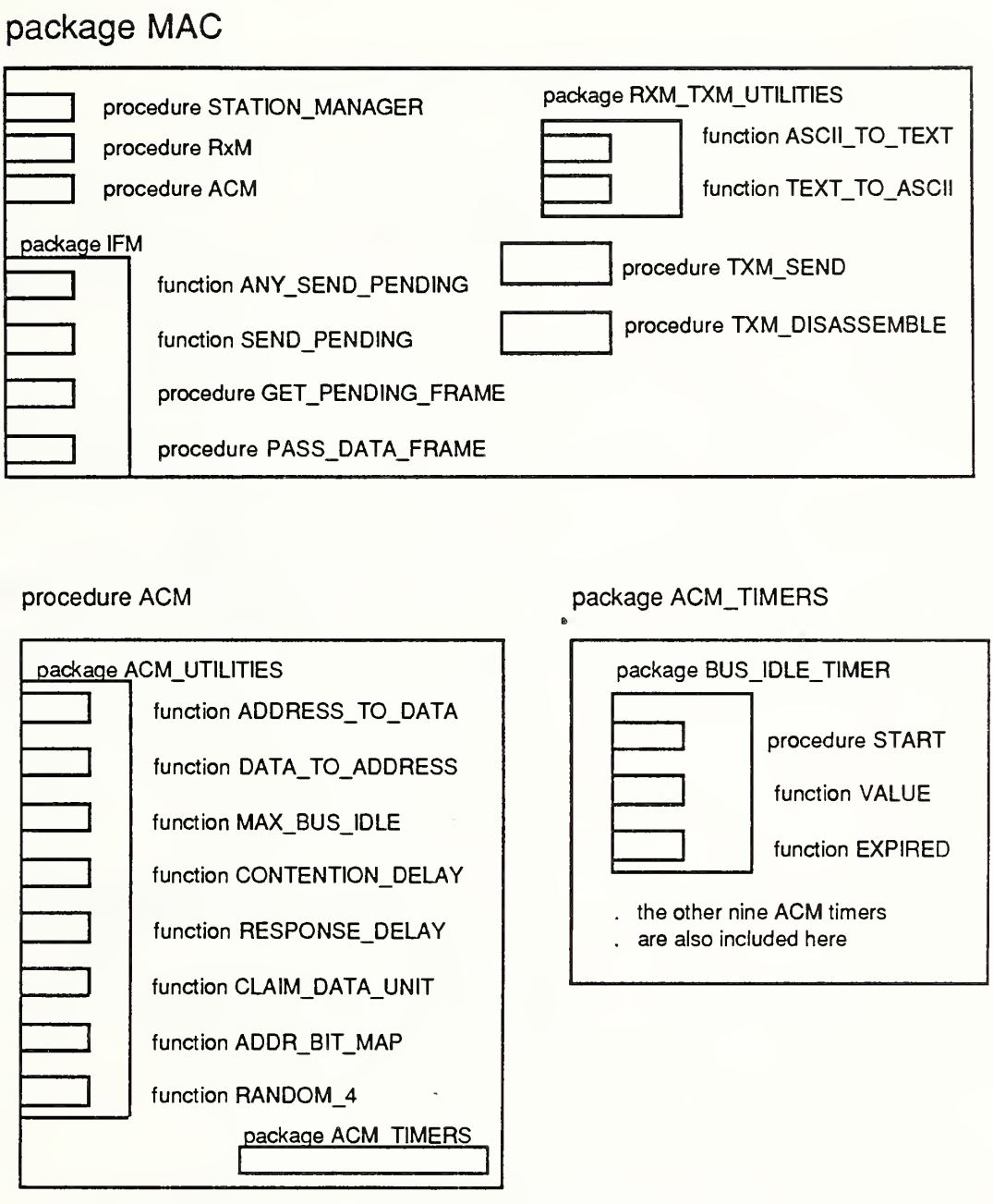


Figure 7 Structure Diagram of Station Support Package (reentrant code shared among stations)

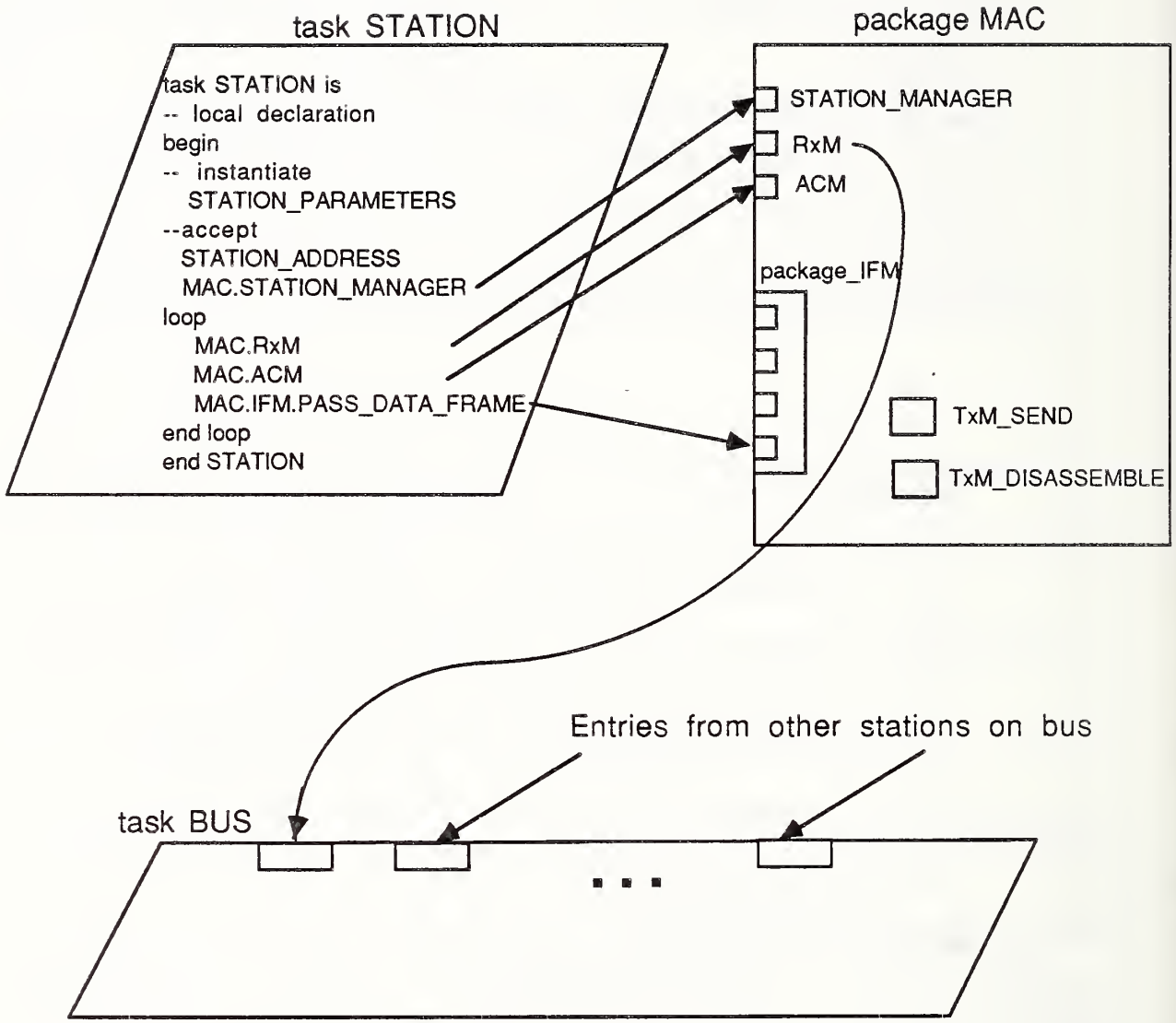


Figure 8. Station-Bus Interaction Diagram

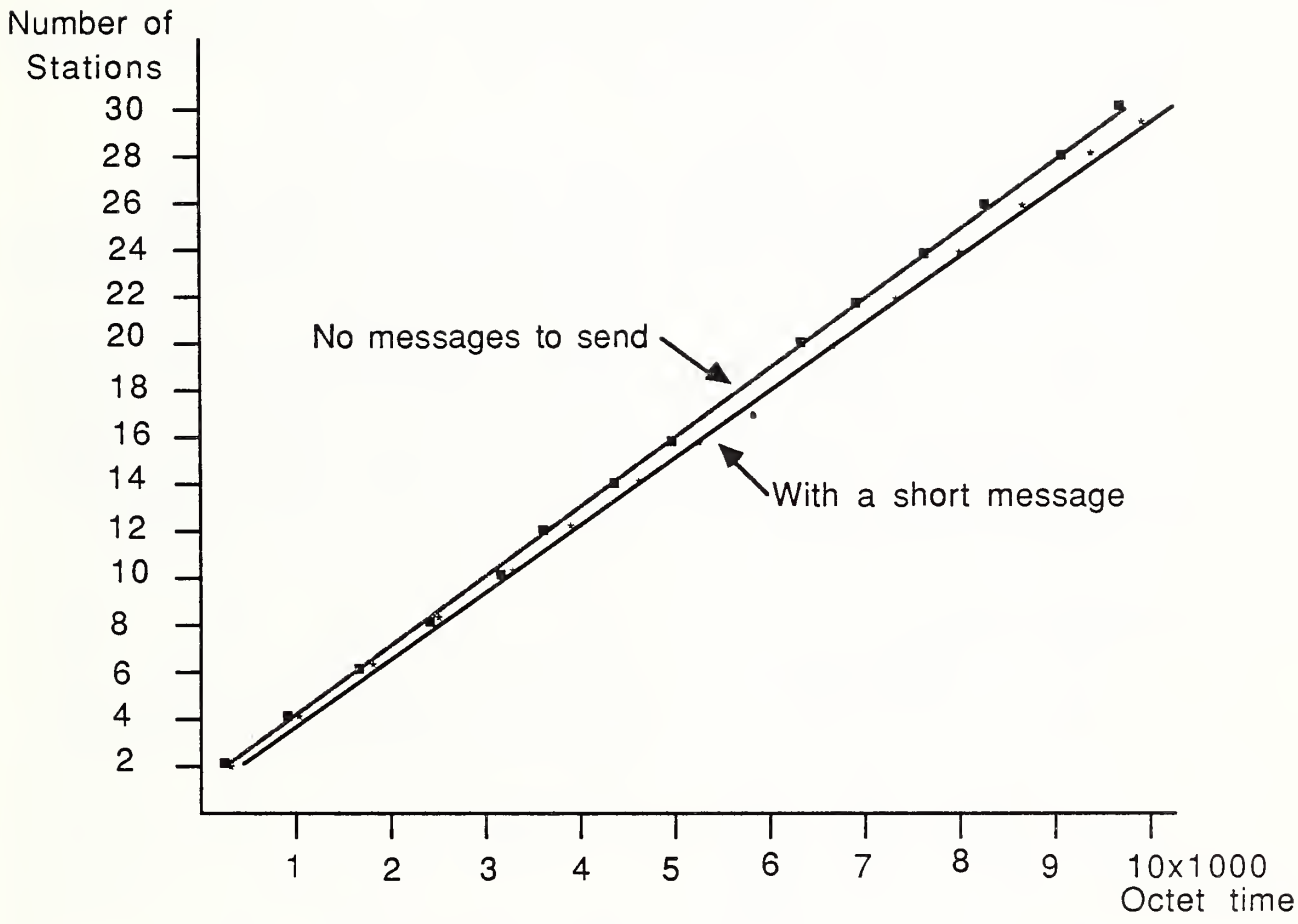


Figure 9. Ring Initialization Time

Conclusions

This presentation has focused on the Ada programming language as a vehicle for simulating factory scale MAP networks. Our simulation tool, though incomplete, shows that physical entities such as network stations, buses, bridges, and head ends as well as the processing software within a station can be represented as Ada package and task modules. Further, task modules for different network entities may be interconnected in a straight forward fashion by means of the task rendezvous mechanism to represent the network to be simulated.

Ada can serve not only to specify the modular network entities but also as the breadboard on which modules are interconnected, message traffic is generated, and network performance in measured. Ada is readable. Ada can be written in a clear fashion to promote confidence in simulation results.

The Ada programming language was designed for the production programming of large embedded computer systems. Such systems typically range upwards in size from 100,000 lines of code. To date, the size of the code produced by this project is an order of magnitude below this lower bound. In short, there is ample room to grow in the Ada environment and in the utilization of this methodology.

Acknowledgements

The authors gratefully acknowledge the General Motors Corporation, C-P-C Group, for the their support of this work.

SERVER NETWORKS FOR THE DESIGN AND ANALYSIS OF FACTORY COMMUNICATION SYSTEMS

Lawrence Zeidner
Dept. of Manufacturing Engineering
Boston University
110 Cummington St.
Boston, MA 02215

ABSTRACT

Server networks are generalized networks of asynchronous processes communicating along software-programmable communication links according to specific protocols either within the confines of a single mainframe or across several physical machines. The advantages of server networks in simulation, modeling, and production systems stem from the modularity of each server as a separate data-driven software element within the entire system. As modular elements whose communication connections with the rest of the network are clearly defined, servers can be exchanged for direct comparison. Their modularity streamlines the software development effort. In the field of manufacturing, server networks can be applied to the scheduling and routing problems posed in flexible manufacturing systems if technology transfer occurs from the field of computer operating systems. In much the same way that computer operating systems, such as VM/370, present the user with an apparent virtual machine through time slicing and scheduling, these operating systems for the factory floor present each workpiece with an apparent virtual factory through the management of routing and scheduling. The advent of inter-process shared variables in the APL2 programming language presents a working environment well-suited to the construction of server networks. Our research pertains to the development and application of a software methodology with which to construct server networks.

Key words: Server Networks; Software Engineering; Large-System Integration Tools; Software Modularity; Cooperative Processing; Data-Driven Control Flow (DDCF).

INTRODUCTION

Server networks offer the flexibility to model components of large systems at various levels of complexity and to choose the level of complexity most appropriate for any particular study. The tradeoff in choosing an appropriate component model is between the advantages of detailed emulation on one hand and associated computational costs and time required on the other. Because the software that models each system component is localized in its own server, it can be replaced by an alternative software model residing in a different server.

The data-driven control-flow methodology, described in Reference 1 and applied to server networks in Reference 2, makes manual construction of small networks (i.e., those consisting of only a few servers) a simple, straightforward exercise. However, the construction of large server networks is best accomplished through the use of large-system integration tools and concepts. Accordingly, a minimal set of parametrically tunable generic "utility" servers has been identified that can be combined to form a powerful server-network infrastructure.

Each server has a kernal of generic software that enables it to function as a member of the server network aside from its application software. This kernal of software will be identified.

A powerful large-system integration tool designed for server-network configuration has been built. It is an expert support system that is used to specify the servers in the network, their interconnection and protocol, and their resource allocation. It enables the system designer to examine, merge, archive, and retrieve entire subnetworks and to employ server macros through parametric specification and design rules.

INFRASTRUCTURE

In an effort to simplify the construction and use of server networks, we have developed a set of generic "utility servers" that are described individually in Reference 2. A minimal set of parametrically tunable servers was sought rather than an exhaustive set containing every possible variation. To date, seven utility servers have been identified. They fall into five subnetwork categories: dispatching, command, status, logging, and timing.

The Dispatching Subnetwork

A server network is built by customizing a set of generic virtual machines. Although each machine has no "personality," it does have a specific set of computing resources at its disposal. The dispatcher allocates generic virtual machines from this pool to participate in specific server networks. When such an allocation occurs, the dispatcher communicates with virtual machines from the pool, telling each one what its role will be and, thus, which set of predefined software to adopt. In addition, based on the pattern it receives from the configuration console and on a set of design rules, the dispatcher builds the appropriate drive-data matrix and downloads it to the virtual machine. The drive data informs the virtual machine of the communication variables it shares with the rest of the network, of its communication protocols, and of the software it must execute upon interrupt by

each armed variable. A virtual machine connected to other virtual machines through shared variables with defined protocols, complete with a set of software that is executed in response to specific stimuli, constitutes a server. Virtual machines in the generic pool can be thought of as servers although they are connected only to the dispatcher, and have only enough software to be dispatchable. The dispatcher not only customizes the server by giving it a "personality," it also maintains contact with it so that the server can be:

- dynamically recustomized whenever necessary, and
- restored to the generic server pool when it is no longer needed for the server network.

Figure 1 illustrates the servers and shared variables that constitute the dispatching subnetwork. The configuration console shares three variables with the dispatcher.

PATT: A pattern matrix, armed by the dispatcher, that indicates shared-variable connections between the pairs of servers and that specifies the protocol (Table 1).

INST: An instruction variable, armed by the dispatcher and executed upon receipt from the configuration console.

WHOS: An unarmed allocation matrix that indicates the computational resource requirements associated with each server. The system designer specifies this information from the configuration console. The dispatcher uses this information to select server assignments.

The dispatcher shares four variables with each server that it controls.

EXSV: A drive-data matrix, armed by the server, that is used upon receipt to establish communication connections, protocols, and a pattern of behavior in response to each possible external stimulus. The EXSV matrix, together with the software it calls, constitutes the server's "personality" (Table 2).

INST: An instruction variable armed by the server and executed upon receipt to load a specific set of software or to return to the generic server pool when no longer needed.

RPLY: A reply variable, armed by the dispatcher, that informs the dispatcher of the disposition of the instructions that it has sent to the server.

WHOS: An unarmed allocation matrix that indicates the resources allocated to each virtual machine and the current server assignments. The servers use the server-assignment information to make decisions regarding their environment.

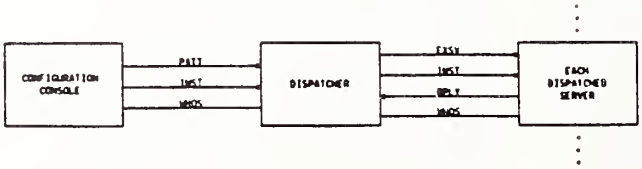


Figure 1. The Dispatching Subnetwork

ARMED	SERVER A	VARIABLE	SERVER B	ARMED
⊙	CNS1	PATT	DSP1	1
⊙	CNS1	INST	DSP1	1
1	EXC1	EXSV	DSP1	⊙
⊙	CNS1	INST	EXC1	1
⊙	DSP1	EXSV	LOG1	1
⊙	LOG1	LOGD	DSP1	⊙
⊙	EXC1	INST	LOG1	1
1	EXC1	RPLY	LOG1	⊙
⊙	EXC1	LOGD	LOG1	⊙
⊙	CNS1	LOGD	LOG1	⊙
⊙	MON1	ACTV	EXC1	1
⊙	MON1	STAT	EXC1	⊙
.	.	.	.	.
.	.	.	.	.
.	.	.	.	.

Table 1. The PATT Pattern Matrix. Each row corresponds to a shared variable. The third column indicates the variable's surrogate name. The second and fourth columns indicate the two partners who share this variable. The first and fifth columns indicate the protocol; they contain a 1 if the variable is armed on that partner's side and a 0 otherwise.

VARIABLE	PARTNER	ARMED
INST	CNS1	1
EXSV	DSP1	1
INST	LOG1	⊙
RPLY	LOG1	1
LOGD	LOG1	⊙
INST	STA1	⊙
RPLY	STA1	1
ACTV	MON1	1
STAT	MON1	⊙
INST	MON1	⊙
RPLY	MON1	1
.	.	.
.	.	.
.	.	.

Table 2. The EXSV Drive-Data Matrix. Each row corresponds to a shared variable that is shared by the server that owns this matrix. The first column indicates the variable's surrogate name. The second column indicates the partner. The third column indicates the protocol; a 1 indicates that the variable is armed for this server.

The system designer specifies resource allocation constraints and objectives at the configuration console. They are passed to the dispatcher through the WHOS allocation matrix, where an assignment algorithm chooses the appropriate virtual machine for each server. Specific assignments can be guaranteed through appropriate instructions from the configuration console, thus, overriding the assignment algorithm entirely or in part.

Multiple hierarchical dispatchers are relevant for cooperative processing among separate physical machines. While local dispatchers preside over the servers in a single physical machine, overall dispatchers group together whole machines full of servers. The advantage of local dispatchers is that they can respond to local reconfiguration needs without the necessity for inter-machine communication.

The Command Subnetwork

Human interaction with a server network involves both command and status subnetworks. The command subnetwork provides not only the expert support system, with which commands can be generated, it also provides the servers that issue and monitor the performance of these commands. In a production environment, different expert support systems would be provided for such users as production planners; purchasing, sales, and marketing agents; design, manufacturing, and quality-control engineers; machinists; and repairmen. Each expert support system must enable the user, within his scope of authority, to issue sets of commands to the production system, even though he need not be cognizant of the form of these commands, their sequence, or the precise set of servers involved. For example, when a salesman enters a new order using his expert support system, commands must be issued to adjust production scheduling, stock levels, tooling requirements, and a variety of other values within the system, few or none of which he need be aware of.

The command subnetwork consists of console, executor, and router servers. The console server presents an expert support system with which to generate tasks to be performed by other servers in the network. These tasks must be constructed, issued, and monitored for completion, often a lengthy process. The console server must remain responsive; therefore, it cannot devote itself to such time-consuming activities. Instead, the console merely communicates with the executor servers, constructing tasks for them to issue and monitor.

A console server can control any number of executor servers, each of which may serve any number of other consoles. Conflicts are avoided through the use of a router server, which gets a request for an executor's service from a console and returns the name of an available executor. This request might include the nature of the work to be done. The router would make its choice accordingly. For instance, the task might include specification of a set of servers to be notified of an event. Not all executors are connected directly to all other servers. Therefore, the router's choice might be governed by the connectivity of the available executors to the servers that will be needed to perform the task.

Figure 2 illustrates the basic connectivity of the command subnetwork. The console server shares one variable with the executor.

INST: An instruction variable armed by the server and executed upon receipt.

The executor shares two variables with each server that it controls.

INST: An instruction variable armed by the server and executed upon receipt.
RPLY: A reply variable, armed by the executor, that describes the disposition of the instructions sent earlier.

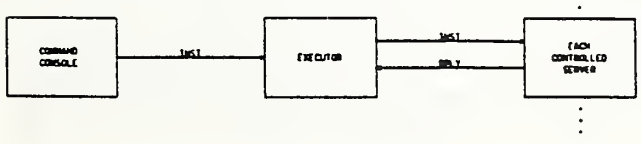


Figure 2. The Command Subnetwork

Figure 3 illustrates the connectivity of a multi-console, multi-executor command subnetwork containing a router server. The router server shares one variable with each console and another with each executor.

RQST: A request variable, armed by the router, that is used by the console to request and specify executor service and that is used by the router to indicate to the console a suitable available executor.
BUSY: An unarmed variable that is used by the executor to indicate its availability to the router and that is used by the router to reserve executor service when assigning a console to the executor.

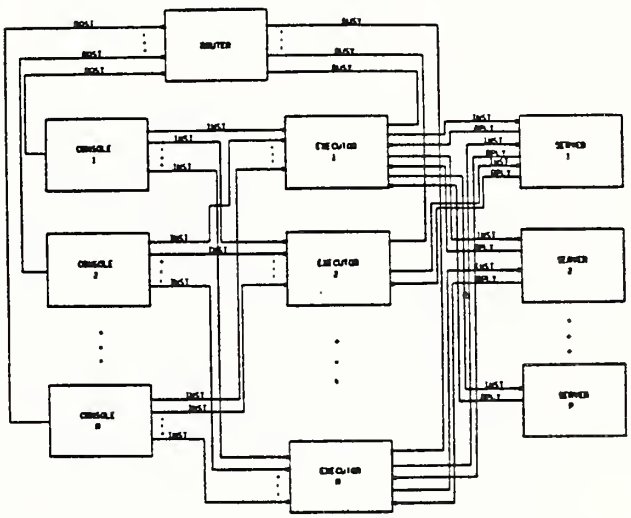


Figure 3. A Multi-Console, Multi-Executor Command Subnetwork

The Status Subnetwork

The status subnetwork provides output describing the status of the server network, the engineering system being controlled or simulated, and the progress of the simulation itself. Responses and completion of console requests and commands are displayed. Comprised of status-display servers and monitor servers, the status subnetwork is in many ways the complement of the command subnetwork.

The status-display server spends its time in one of two states: formatting the relevant data for display on its textual and graphical displays or, in a dormant state, waiting either for the arrival of some urgent message or for a regular timing interrupt to signal another output formatting cycle. The time interval between successive output cycles can be regulated to correspond to server-network activity so that excessive computation does not result from the unnecessary reformatting of unchanged data, and so that the output refresh does not cycle so slowly as to skip meaningful data changes.

The monitor server provides a service for the status display that is analogous to the service provided for the console by the executor. The monitor is used to assist the status display so that it can refresh its data quickly enough. It provides two general categories of assistance: activity queries and computational assistance.

Figure 4 illustrates the connectivity of the status subnetwork. The monitor shares two variables with each server that it monitors.

ACTV: A variable, armed by the monitored server, sent by the monitor and checked shortly thereafter for modification.

STAT: An unarmed variable that the monitored server uses to indicate the nature of its work to the monitor.

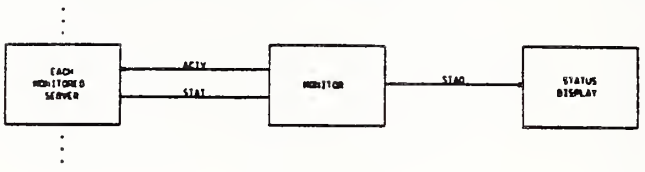


Figure 4. The Status Subnetwork

One useful type of status display data is the activity or communication status of the servers in the network. An easy way to implement this check is to send out a query to each server along a communication variable (ACTV) that triggers a trivial response function. Thus, the monitor can send out all of the queries, wait momentarily, and then gather all of the responses. If a server is busy with one function or is inactive, there will be no response. (Placing the activity variable highest in the search order for the servers assures a response from even the busiest server, unless such a server is busy responding to just one variable.) The other type of status display data gathered by the monitor is the status of each server (STAT). This is gathered by means of an unarmed variable shared between the monitor and each server it monitors. It is the responsibility of the server to keep the status variable updated. For example, in a manufacturing server network, a server controlling a numerical control (NC) machine might fill its status variable with information regarding the active workpiece, the active tooling, the progress of the present manufacturing process, or the expected time until completion.

The monitor shares one variable with the status display.

STAO: A variable, armed by the status display, that contains the status-display output information that is ready to be displayed.

Many forms of status displays that are extremely helpful to the engineer require an excessive amount of computation. The monitor server can be used to perform such computation. Furthermore, if such computation is particularly well-suited to a special-purpose computing facility, the monitor server can reside in such a machine while the status display resides elsewhere.

The Logging Subnetwork

In a typical server network, most servers are neither consoles nor status displays. Instead, they are tightly coupled networks of automatic, remotely triggered software processes. For the purposes of result validation and justification, software verification, error isolation, and computational load balancing, in-process logging is essential.

Events to be logged fall into three categories:

- engineering-application events,
- server-network communication events, and
- software-implementation events.

Within each of these categories events may be divided along another axis into:

- error reports,
- appropriate-event reports, and
- tuning-delay reports.

From the console, it is possible to selectively enable either specific logging events or whole categories of events for particular durations, as needed. Each server has embedded logging calls in its software, and the disposition of these calls depends on the particular selection of active logging events at the moment, as chosen from the console server.

Log messages accumulate locally and are only gathered and sorted by the logging server when requested. This procedure keeps the logging server from becoming overwhelmed with an enormous number of individual events and creating a bottleneck for the entire server network.

Figure 5 illustrates the connectivity of the logging subnetwork. The logging server shares one variable with each server to which it provides logging services.

LOGD: An unarmed variable in which log data accumulates locally. This variable is a FIFO stack that is filled by the server and drained by the logging server asynchronously without any need for interlocks.

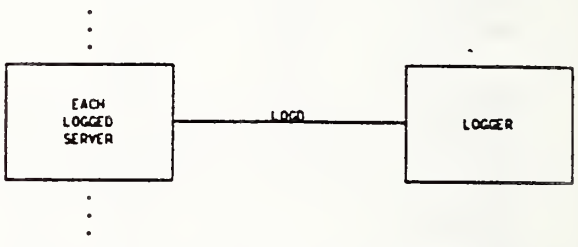


Figure 5. The Logging Subnetwork

The Timing Subnetwork

While servers are essentially asynchronous software processes, engineering and management applications frequently require that subsets of the entire server network by synchronized. Clock servers provide this synchronization while offering control through a console and visibility through a status display.

Figure 6 illustrates the connectivity of the timing subnetwork. The clock server shares one variable with each server that requires timing services.

TIME: A doubly armed variable that is used by the clock server to signal the start of each timing interval and that is used by the other servers in a simulation environment to indicate their completion of work for that interval. In a production environment, the clock sets the TIME variable in real time and the variable need not be armed by the clock because no information is passed back to it by the servers.

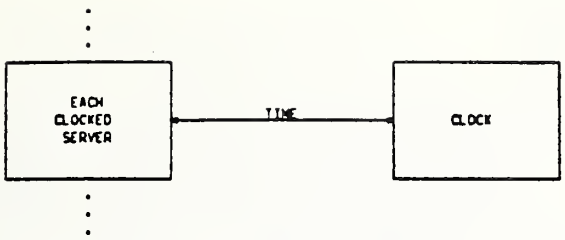


Figure 6. The Timing Subnetwork

Both discrete-time and discrete-event clocking can be provided for simulations. Discrete time clocking can be provided by a straightforward incremental clock that sends out a value over a set of doubly armed variables (TIME). These variables trigger the servers into action. When each server has finished its simulation of the interval, it sends back a value to indicate completion. The clock waits until all servers have responded, then it begins the next interval. The clock is in an excellent position to offer good, computational, load-balancing logging messages based on the order of the responses.

Discrete-event clocking can be provided by a slight variation to the discrete-time approach. When each server has finished the interval, it calculates the expected time of its next event and returns this information to the clock as its response. The clock chooses the earliest response time as the next event time. This approach, while appropriate for simulation, is inappropriate for real-time applications, which require access to a real-time clock and rely upon the server network's computational ability to keep up with the rate of actual events.

A SERVER KERNAL

As a member of a server network, each server requires a kernal of software just to allow it to respond to stimuli, to communicate with its assigned neighbors, and to be dispatchable, commandable, monitorable, loggable and clockable. An effort has been made to isolate, refine, and standardize this kernal of software so that other server software can be reliably built upon it.

Response to Stimuli

Each server has two fundamental states of operation. It spends most of its existence in an idle state waiting for one of its armed variables to awaken it when modified by a partner. In this idle state, a server consumes negligible computing resources. Whenever a server is not in the idle state it is either checking which variable awakened it or, upon isolating this variable, it is executing the functionality associated with the variable.

Naming conventions are used to simplify the association between shared variable names, their surrogate names, and the associated functionality on each partner's side. APL2 allows the partners to refer to a shared variable by two different names and to associate the two names with a third and possibly different name called a surrogate. We have chosen to use the generic name (e.g.,EXSV, PATT, or INST) as the common surrogate, since a server can have several variables with the same surrogate name shared with different partners, but it can only have one variable with each actual name. We build the shared variable's actual name by appending the partner's name onto the generic name. For example, if the clock (CLK1) shares a TIME variable with the executor (EXC5), the clock calls the variable TIMEEXC5, the surrogate name is TIME, and the executor calls the variable TIMECLK1. These names are generated automatically from the EXSV matrix.

The associated function calls are generated in a similar fashion. The function executed upon being awakened by an armed variable is given the same name as the surrogate and is called with the partner's name as the argument. Continuing the example, if the TIME variable is doubly armed, the clock executes the expression, TIME 'EXC5', upon being awakened, and the executor executes the expression, TIME 'CLK1'. By reducing the proliferation of the nearly identical software that occurs when each armed, shared variable has its own devoted function, this use of the surrogate name as the function name simplifies software development and maintenance considerably. In addition, the EXSV matrices can be based solely upon network connectivity and arming protocols, with no need to include function calls explicitly. Therefore, the PATT matrix can be constructed automatically from the connectivity and arming information, and the EXSV matrices can be built automatically from the PATT matrix. Thus, with appropriately named sets of software, the server network can be sketched at the configuration console using the configuration ESS, and automatically a functional server network is created and begins processing. Likewise, at any time it can be halted, reconfigured, and restarted.

Communication

Whenever a new or modified EXSV matrix is sent by the dispatcher, or when a new set of software is loaded, the server must use the EXSV matrix to establish communication connections and protocols. First, however, it must sever its old connections since they are no longer applicable. A RETRACT function retracts all of the previously shared variables as governed by the old EXSV matrix. A SHARE function shares all of the new shared variables and arms them appropriately as governed by the new EXSV matrix.

Dispatchability

The dispatcher must always be able to command a server to load a new set of software when it takes on a new role and to adopt a new set of communication connections when the network configuration changes. Therefore, each server must be equipped with an EXSV function that calls RETRACT to sever old connections and that calls SHARE to establish new connections and protocols. Each server also needs an INST function that takes an instruction passed in along the INST variable and executes it, sending a suitable reply back to the dispatcher.

Commandability

The INST function that is used to execute the dispatcher's commands is also used for the executor's commands. In this case, the reply is sent to the executor, but no change of software is necessary because the INST function receives an argument (as indicated above in the

section on "Response to Stimuli") and this argument indicates which partner is used as the destination for the reply. For example, if the dispatcher (DSP2) sent the instruction, the reply would be sent back through RPLYDSP2. If the executor (EXC7) had sent it, the reply would automatically be sent through RPLYEXC7.

Monitorability

The monitor uses the ACTV variable to check whether each server is still functional. The ACTV function merely modifies the value of the variable (e.g., ACTVMON4) to verify its operation. Whenever the server's software, performs a computation that takes a significant period of time, it modifies the unarmed status (STAT) variable, since it will appear nonfunctional by not responding to ACTV queries during that period. If the server encounters a software error and halts during the process, the value of the status variable can be helpful in locating the error.

Logability

The LOGD variable is unarmed so there is no LOGD function in each server; however, each server has calls throughout its software to a LOG function. The log function appends new log messages onto the FIFO LOGD stack variable.

Clockability

Each server that participates in simulation timing needs a TIME function, which notes that the clock cycle has begun. In most cases, however, it merely toggles a bit in the PREREQUISITES vector, a Boolean vector used to monitor the events that must occur before local action can be taken. When the appropriate set of bits has been set, the local action is initiated and the bits are reset.

THE CONFIGURATION EXPERT SUPPORT SYSTEM

The configuration expert support system is used by the network designer at the configuration console to graphically specify the servers in the network, the sets of software they will use, their computational resource requirements, and their communication connections and protocols. This information is used to build the variables WHOS and PATT, which are shared with the dispatcher. The server network, which runs using existing sets of software, is dispatched automatically based upon information in these two variables. Four facilities constitute the configuration expert support system: the Network Layout Facility, the Server Requirements Facility, the Server Software Facility, and the Archival Facility.

The Network Layout Facility

This textual and graphical facility enables the network designer to sketch a server network by indicating servers and their communication connections and protocols. Textually, the servers receive names (e.g., DSP2, EXC5), and the shared variables are identified by name (e.g., TIME, EXSV). This facility provides everything necessary to build the PATT matrix.

The Server Requirements Facility

This textual facility is used to indicate the specific computing resource needs associated with specific servers. For example, loggers should be given sufficient disk storage to maintain voluminous files containing past logged events. Status-display servers should be dispatched to virtual machines already connected to workstations equipped with the appropriate textual and graphical accessories. During the software-development cycle, servers under careful scrutiny should be dispatched to virtual machines already connected to workstations so that their software can be examined and possibly modified; other more stable computational servers should be dispatched to disconnected virtual machines.

In a multiple-computer server network that spans two or more physical machines, care should be taken to assign servers to virtual machines according to computational load. Certain servers, such as those doing detailed modeling of system components, may require considerable computational power during each clock cycle. If possible, these servers should be dispatched to run on the fastest machine available, and the load on such machines should be balanced so that they are not overloaded and do not perform worse than other slower machines.

These requirements are used to form the preliminary version of the WHOS matrix that is sent to the dispatcher. The dispatcher assigns servers accordingly to available virtual machines and fills in the actual assignments to form the completed version of the WHOS matrix, which it shares with the dispatched servers.

The Server Software Facility

This simple facility enables the network designer to graphically associate sets of server software with servers in the network. This allows great flexibility since alternative software sets can be compared in side-by-side tests, and software modifications can often be made and carefully integrated once a server network is already running, without necessarily restarting the entire network. The software data coupled with the computing needs constitute the preliminary WHOS matrix.

The Archival Facility

This facility provides the ability to archive subnetworks or entire networks, retrieve, copy, merge, and manipulate server networks at a macroscopic level. This is extremely useful since few server networks are built completely from scratch. It is much more common to begin either with a "starter set" consisting of an appropriate infrastructure configuration of utility servers already connected, or with a previous server network to which slight modifications are to be made. Often the only change is the version of software set associations that need to be updated to try a new experiment.

CONCLUSIONS

As engineering systems continue to increase in scope and complexity, outgrowing existing design and development tools, server networks offer significant advantages as a software methodology capable of providing large-system integration tools. Their data-driven control flow, their modularity, and their ability to be dynamically reconfigured automatically based upon graphical respecification, combine to make server networks a promising mechanism with which to address the challenges of cooperative processing among the multitude of separate elements of computing hardware that are making their way onto today's factory floor. [3-5]

REFERENCES

1. Zeidner, L.E. and Y. Hazony, "Data-Driven Control Flow of Interactive Engineering Applications," submitted for publication to IEEE Transactions on Software Engineering.

2. Zeidner, L., "Server Networks for Engineering and Manufacturing," CIMTECH, Boston, MA, 1986, No: MS86-238.

3. Zeidner, L.E., et al., "A Dynamic Simulation Manager for Robotics, Computer-Integrated Manufacturing and Communications Conference, Anaheim, CA, April 1985, No. MS85-336.

4. Zeidner, L., Hazony, Y. and A. Williams, "An Expert System Generator," submitted for publication to IASTED Control and Computers, 1986.

5. APL2 Programming: Language Reference, IBM #SH20-9229, Copyright 1985, International Business Machines.

ACKNOWLEDGMENTS

The development of server networks enjoys continual support and encouragement from Y. Hazony, CIDAM Director. Isolation and refinement of the server kernal software was performed using implementations by M. Abt, A. Williams, and W. Labossier. G. D'Angelo provided invaluable technical editing of this paper.

Analytic and Simulation Models

A Flow and Error Control Model with Phase Type Timeout Periods

by

S. Chakravarthy* , J.R. Pimentel** and K.V. Ravi**
GMI Engineering and Management Institute
Flint, Michigan 48502-2276

ABSTRACT

In this paper, we present a finite queueing model to study the transmission process at a network station. This queueing model is appropriate for modeling the flow and error control mechanisms at the higher layers of the OSI reference model such as the transport layer. The basic assumptions of the queueing model under study are as follows. Messages (packets) arrive at the source station according to a Poisson process with rate λ. At any time only N+1 new messages may be present in the system; one in the transmitting unit (service) and at most N waiting in the primary buffer for service. Any new arrivals are considered to be lost. The transmission times of the messages are independent and identically distributed random variables having a common exponential distribution with parameter μ_1. A copy of the transmitted message is kept in the secondary buffer and is removed once an acknowledgement for it is received from the destination station. The secondary buffer can store at most W messages. It is assumed that the arrival of acknowledgements at the source station follows a Poisson process with rate μ_2. A transmitted message may require a subsequent transmission if no acknowledgement for it is received within a specified *timeout* period. We assume that the timeout period follows a *phase type* distribution. The class of phase type distributions includes many well known distributions such as the generalized Erlang, hyperexponential, etc., as special cases and has a number of interesting closure properties. Using Markov chain theroy, we discuss numerically stable algorithms to compute various steady state system performance measures such as throughput, expected numbers of messages waiting for transmissions/retransmissions, acknowledgements. A number of numerical examples are presented.

---

*Department of Science and Mathematics

**Department of Electrical and Computer Engineering.

To be presented at the IEEE workshop on Factory communications. to be held during March 17-18, 1987 at Maryland.

1. INTRODUCTION

In this paper, we consider a finite capacity queueing model in which the messages (packets) arrive at the source station according to a Poisson process with rate λ. At any time at most N+1 new messages may be present; one in the transmitting unit(service) and at most N waiting in the primary buffer. Any new arrivals are considered to be lost. The transmission times of the messages are independent and identically distributed random vaiables having a common *exponential* distribution with parameter μ_1. A copy of the transmitted message is kept in a secondary buffer and is removed once an acknowledgement for it is received from the destination station. The secondary buffer can store at most W (window size) messages. It is assumed that the arrival of acknowledgements at the source station follows a *Poisson* process with rate μ_2. A transmitted message may require a subsequent transmission if no acknowledgement for it is received within a specified *timeout* period, which is assumed to be random.

This queueing model is well suited for modeling the flow and error control mechanisms at the higher layers of the OSI reference model. For example, at the transport layer message delays are caused due to one or more of the following :

- variable size of the messages.
- different paths used by the messages.
- congestion in the network.
- availability of the protocol resources.

Hence, the model presented in this paper can be used to study the performance of the transport layer protocols in which retransmission timeout periods cannot be considered to be constants. For more details on the transport layer, we refer the reader to Stallings [7] or Tanenbaum [8].

We assume that the timeout period follows a *phase type* distribution (PH-distribution) with representation $(\underline{\alpha}, T)$ of order m. PH-distributions and PH-renewal processes were introduced by Neuts [4]. The class of PH-distributions includes many well known distributions such as generalized Erlang, hyperexponential, etc., as special cases and has a number of interesting closure properties. Also the class of PH-distributions is dense in the class of all distributions on $[0,\infty)$. A detailed discussion of the properties of PH-distributions and their uses in stochastic modeling may be found in Neuts [5].

Using Erlangian timeout periods, Sethi and Ghosal [6] considered a finite queueing model to study the transmission process at a layer, such as data link layer, in which the timeout periods are constants. While it is possible to have more than one acknowledgement during a timeout period in their model, we assume in our model that each message has its own timeout clock and is triggered only when it is at the head of the queue in the secondary buffer. However, with minor modifications our model can be used to study various schemes of the acknowledgement process. Their analysis of the model is completely different from ours. It should be noted that the coefficient of variation for Erlang distributions is less than 1 and hence their model is appropriate, with higher order Erlangs, in cases where the timeout periods are almost constants.

Thus, the system under study consists of a transmitting unit, a primary buffer to store the incoming new messages; a secondary buffer to store the transmitted messages waiting for an acknowledgement and an auxiliary buffer to store the messages that are to be retransmitted. The messages are transmitted on a first come first served basis. However, a message that requires a retransmission has a *non-preemptive* priority over the messages waiting in the primary buffer. The timeout period is active as long as there is at least one message in the secondary buffer. This system is described pictorically in Figure 1 below.

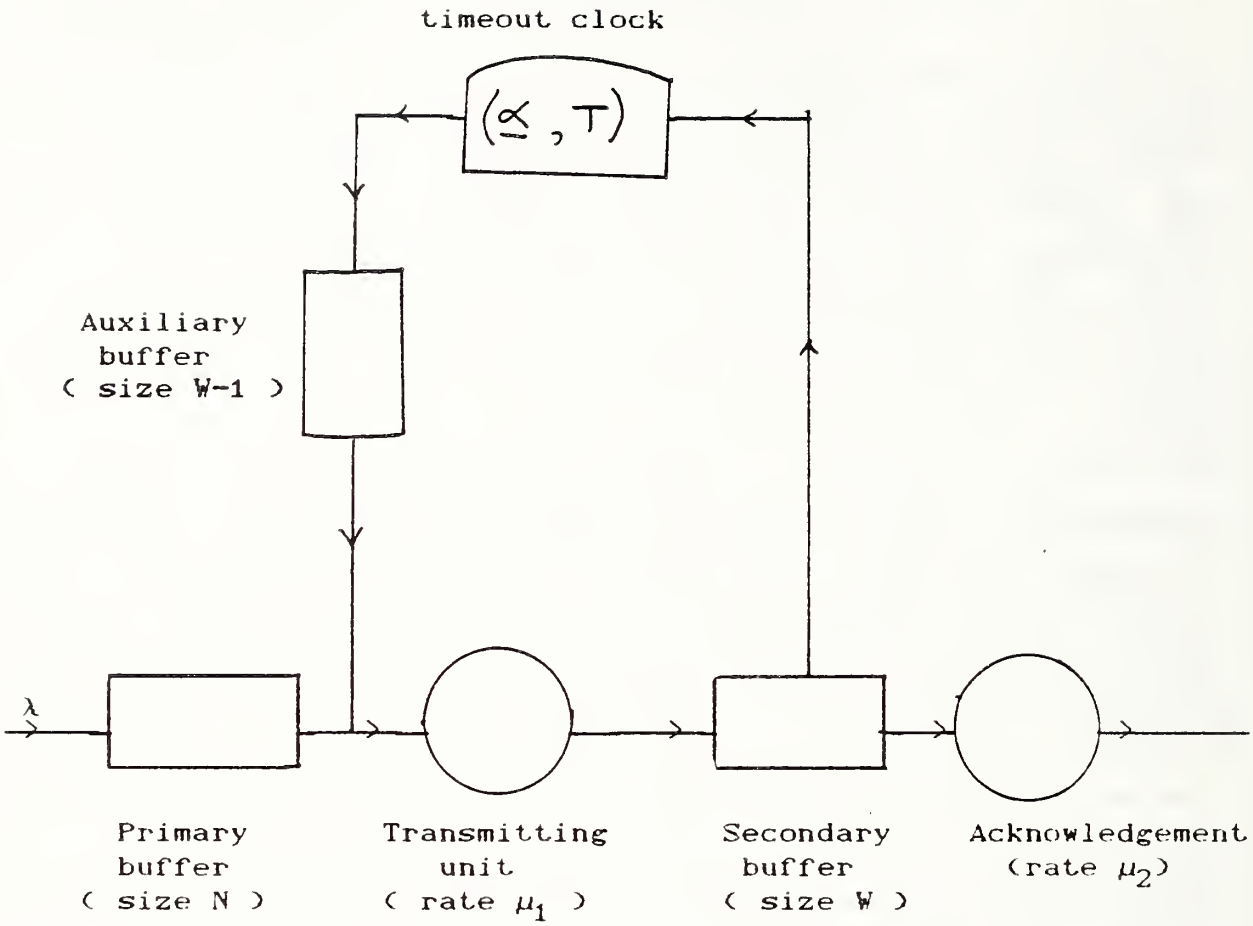


Figure 1

The paper is organized as follows. In Section 2, we briefly review the basic definitions and preliminaries of PH-distributions that are needed in the paper. The description of the Markov process governing the system under study and the generator of the Markov process for a specific case of the model are given in Section 3. The algorithmic procedure for computing the various steady state performance measures of the model is discussed in Section 4 and some numerical examples are given in Section 5.

2 PHASE TYPE DISTRIBUTIONS

In this section, we review the basic definitions and preliminaries of PH-distributions that are needed in the paper.

We consider a continuous time parameter Markov chain (MC) on the states (1, 2, ..., m, m+1) with infinitesimal generator $\hat{Q}$ given by

$$\hat{Q} = \begin{vmatrix} T & T^{O} \\ \mathbf{0} & 0 \end{vmatrix} , \qquad (1)$$

where the m x m matrix T satisfies $T_{ii} < 0$, for $1 \leq i \leq m$, and $T_{ij} \geq 0$, for $i \neq j$. The column vector T^{O} is such that $Te + T^{O} = \mathbf{0}$, and the initial probability vector of $\hat{Q}$ is given by $(\underline{\alpha}, \alpha_{m+1})$, with $\underline{\alpha}\, e + \alpha_{m+1} = 1$. We assume that T^{-1} exists so that the states 1,2,..., m are all transient and absorption into the state m+1, from any initial state, is certain. The probability distribution F(.) of the time until absorption in the state m+1, corresponding to the initial probability vector $(\underline{\alpha}, \alpha_{m+1})$ is given by

$$F(x) = 1 - \underline{\alpha} \exp (Tx)\, e, \quad \text{for } x \geq 0. \qquad (2)$$

Definition: A probability distribution F(.) on [0, ∞) is a PH-distribution if and only if it is the distribution of the time until absorption in a finite MC of the type defined in (1). The pair ($\underline{\alpha}$,T) is called a representation of F(.).

The noncentral moments ν_i' of F(.) are all finite and given by

$$\nu_i' = (-1)^i\, i!\, \underline{\alpha}\, T^{-i}\, e \ , \quad \text{for } i \geq 1. \qquad (3)$$

To avoid uninteresting complications, we assume that the PH-distribution used in this work do not have a jump at the origin so that $F(0) = \alpha_{m+1} = 0$.

We conclude this section with some examples of PH-distributions.

Examples 1. The exponential distribution with parameter λ is a PH-distribution with representation $\underline{\alpha} = 1$, $T = -\lambda$.

2. The (generalized) Erlang distribution of order m with parameters $\lambda_1, \lambda_2, \ldots, \lambda_m$, has the representation $\underline{\alpha} = (1,0,\ldots,0)$ and

$$T = \begin{vmatrix} -\lambda_1 & \lambda_1 & & & \\ & -\lambda_2 & \lambda_2 & & \\ & & \ddots & & \\ & & & -\lambda_{m-1} & \lambda_{m-1} \\ & & & & -\lambda_m \end{vmatrix}.$$

3. The hyperexponential distribution

$F(x) = \sum_{j=1}^{m} \alpha_j \, (1 - e^{-\lambda_j x})$ has the representation

$\underline{\alpha} = (\alpha_1, \ldots, \alpha_m)$ and $T = \text{diag}\,(-\lambda_1, \ldots, -\lambda_m)$, a diagonal matrix.

3. THE MARKOV PROCESS

The system outlined in Section 1 can now be described by a Markov process with $[1 + 2W(N+1) + mNW(W-1) + m(W^2+N)]$ states. The state space may be partitioned into N+4 sets of states and one state. These are denoted by Δ_1, Δ_2, **0**, **1**, ..., **N**, **N+1** and ***** and their descriptions are given in Table 1.

Table 1

State	Description
*	The system is idle.
Δ_1 = {(i',j) : 1 ≤ i ≤ N+1, 0 ≤ j ≤ W-1 }	There are i new messages in the system including the one that is currently being transmitted and j messages are waiting for retransmission.
Δ_2 = {(i,j') : 0 ≤ i ≤ N, 1 ≤ j ≤ W }	There are i new messages in the system and j messages including the one in service are to be retransmitted.
0 = {(0,j',r,k) : 0 ≤ j ≤ W-1, 1 ≤ r ≤ W-j, 1 ≤ k ≤ m }	There is no new message in the system, j messages including the one in service are to be retransmitted, r messages are waiting for acknowledgements and the phase of the timeout period is in k .
i = {(i',j,r,k) : 0 ≤ j ≤ W-2, 1 ≤ r ≤ W-1-j 1 ≤ k ≤ m } U {(i,0,W,k) : 1 ≤ k ≤ m } U {(i,j',r,k) : 1 ≤ j ≤ W-1, 1 ≤ r ≤ W-j, 1 ≤ k ≤ m }.	There are i new messages in the system, j messages are waiting for retransmissions, r messages are waiting for acknowledgements and the phase of the timeout period is in k. Either the transmitter is blocked (because the secondary buffer is full) or a new message or the one that needs a retransmission is in the transmitting unit, for 1 ≤ i ≤ N.
N+1 = {(N+1',j,r,k) : 0 ≤ j ≤ W-2, 1 ≤ r ≤ W-1-j, 1 ≤ k ≤ m }	There are N+1 new messages, j messages are waiting for retransmission, r messages are waiting for acknowledgements and the phase of the timeout period is in k.

We have found it somewhat convenient to order the sets of states lexicographically.

In the sequel the notation **e** stands for a column vector of 1's, $\mathbf{e}_i$ a unit column vector with 1 in the i-th position and 0 elsewhere and I an identity matrix of appropriate dimensions. The symbol ⊗ will denote the **Kronecker product** of two matrices. Specifically L⊗M stands for the matrix made up of the block $L_{ij}M$. For more details on the Kronecker products, we refer the reader to

Bellman [1] or Marcus and Minc [3].

The Markov process with the infinitesimal generator Q is very sparse. To display Q for the general case requires setting up a number of notations. However, we give below in Table 2 the generator for the case N = 4 and W = 3. The general form of Q is given in Chakravarthy [2]. In the sequel, we shall discuss the solution of the steady state equations and the algorithmic procedures for this case only. The discussions for the general case will be presented elsewhere.

4. ALGORITHMIC PROCEDURES

4.1 The Stationary Distribution

The stationary probability vector x of the generator Q is the unique solution to the system of equations

$$xQ = 0, \qquad xe = 1 \tag{4}$$

In view of the high order of the matrix Q, it is essential to use its special structure to evaluate the components of x. We first partition the vector x as $x = (x^{*}, \xi_1, \xi_2, x_0, x_1, x_2, \ldots, x_5)$ according to the states $*, \Delta_1, \Delta_2, 0, 1, 2, \ldots, 5$. Further partitioning the components of x as

$\xi_1 = (\xi_{11}, \xi_{12}, \ldots, \xi_{15})$, ξ_{1j} is a vector of order 3,

$\xi_2 = (\xi_{20}, \xi_{21}, \ldots, \xi_{24})$, ξ_{2j} is a vector of order 3,

$x_0 = (x_{01}, x_{02}, \ldots, x_{06})$, x_{0j} is a vector of order m,

$x_i = (y_i, z_i)$, for $1 \le i \le 4$,

with

$y_i = (y_{i1}, y_{i2}, y_{i3}, y_{i4})$, y_{ij} is a vector of order m,

$z_i = (z_{i1}, z_{i2}, z_{i3})$, z_{ij} is a vector of order m ,

and

$x_5 = (x_{51}, x_{52}, x_{53})$, x_{5j} is a vector of order m ,

we obtain from equation (4)

$$-\lambda\, x^* + \mu_2 x_{01} e = 0, \tag{5}$$

$$\lambda\, x^* e_1' - (\lambda+\mu_1)\xi_{11} + y_{11}S_1 + y_{14}S_2 = 0, \tag{6}$$

$$\lambda\, \xi_{1j} - (\lambda+\mu_1)\xi_{1,j+1} + y_{j+1,1}S_1 + y_{j+1,4}S_2 = 0, \tag{7}$$

$$\text{for } 1 \le j \le 3,$$

$$\lambda\, \xi_{14} - \mu_1\, \xi_{15} + x_{51}S_1 + x_{53}S_2 = 0, \tag{8}$$

$$-(\lambda+\mu_1)\xi_{20} + x_{01}R + x_{04}S_1 + x_{06}S_2 = 0, \tag{9}$$

$$\lambda\, \xi_{2j} - (\lambda+\mu_1)\xi_{2,j+1} + z_{j+1,1}S_1 + z_{j+1,3}S_2 = 0, \tag{10}$$

$$\text{for } 0 \le j \le 2,$$

$$\lambda\, \xi_{23} - \mu_1\, \xi_{24} + z_{41}S_1 + z_{43}S_2 = 0, \tag{11}$$

$$(\xi_{11} + \xi_{20})M_1 + x_{01}A_3 + \mu_2 x_{02} e\alpha = 0, \tag{12}$$

$$x_{02}A_3 + \mu_2 x_{03} e\alpha + \mu_1(x_{04} + y_{11}) = 0, \tag{13}$$

$$x_{03}A_3 + \mu_1(x_{05} + y_{12}) = 0, \tag{14}$$

$$(\xi_{11} + \xi_{20})M_2 + x_{02}T^o\alpha + x_{04}A_1 + \mu_2 x_{05} e\alpha = 0, \tag{15}$$

$$x_{03}T^o\alpha + x_{05}A_1 + \mu_1(x_{06} + y_{14}) = 0, \tag{16}$$

$$(\xi_{11} + \xi_{20})M_3 + x_{05}T^o\alpha + x_{06}A_1 = 0, \tag{17}$$

$$(\xi_{12} + \xi_{21})M_1 + \lambda x_{01} + y_{11}A_1 + \mu_2 y_{12} e\alpha = 0, \quad (18)$$

$$\lambda x_{02} + y_{12}A_1 + \mu_2 y_{13} e\alpha + \mu_1(z_{11} + y_{21}) = 0, \quad (19)$$

$$\lambda\, x_{03} + y_{13}A_3 + \mu_1(z_{12} + y_{22}) = 0, \quad (20)$$

$$y_{12}T^{O}\alpha + y_{14}A_1 = 0, \quad (21)$$

$$(\xi_{12} + \xi_{21})M_2 + \lambda x_{04} + z_{11}A_1 + \mu_2 z_{12} e\alpha = 0, \quad (22)$$

$$\lambda x_{05} + y_{13}T^{O}\alpha + z_{12}A_1 + \mu_1(z_{13} + y_{24}) = 0, \quad (23)$$

$$(\xi_{12} + \xi_{21})M_3 + \lambda x_{06} + z_{12}T^{O}\alpha + z_{13}A_1 = 0, \quad (24)$$

$$(\xi_{13} + \xi_{22})M_1 + \lambda y_{11} + y_{21}A_1 + \mu_2 y_{22} e\alpha = 0, \quad (25)$$

$$\lambda y_{12} + y_{22}A_1 + \mu_2 y_{23} e\alpha + \mu_1(z_{21} + y_{31}) = 0, \quad (26)$$

$$\lambda\, y_{13} + y_{23}A_3 + \mu_1(z_{22}+y_{32}) = 0, \quad (27)$$

$$\lambda y_{14} + y_{22}T^{O}\alpha + y_{24}A_1 = 0, \quad (28)$$

$$(\xi_{13}+\xi_{22})M_2 + \lambda z_{11} + z_{21}A_1 + \mu_2 z_{22} e\alpha = 0, \quad (29)$$

$$\lambda z_{12} + y_{23}T^{O}\alpha + z_{22}A_1 + \mu_1(z_{23} + y_{34}) = 0, \quad (30)$$

$$(\xi_{13}+\xi_{22})M_3 + \lambda z_{13} + z_{22}T^{O}\alpha + z_{23}A_1 = 0, \quad (31)$$

$$(\xi_{14} + \xi_{23})M_1 + \lambda y_{21} + y_{31}A_1 + \mu_2 y_{32} e\alpha = 0, \quad (32)$$

$$\lambda y_{22} + y_{32}A_1 + \mu_2 y_{33} e\alpha + \mu_1(z_{31} + y_{41}) = 0, \quad (33)$$

$$\lambda\, y_{23} + y_{33}A_3 + \mu_1(z_{32} + y_{42}) = 0, \tag{34}$$

$$\lambda y_{24} + y_{32}T^{o}\alpha + y_{34}A_1 = 0, \tag{35}$$

$$(\xi_{14} + \xi_{23})M_2 + \lambda z_{21} + z_{31}A_1 + \mu_2 z_{32}e\alpha = 0, \tag{36}$$

$$\lambda z_{22} + y_{33}T^{o}\alpha + z_{32}A_1 + \mu_1(z_{33} + y_{44}) = 0, \tag{37}$$

$$(\xi_{14} + \xi_{23})M_3 + \lambda z_{23} + z_{32}T^{o}\alpha + z_{33}A_1 = 0, \tag{38}$$

$$(\xi_{15} + \xi_{24})M_1 + \lambda y_{31} + y_{41}A_1 + \mu_2 y_{42}e\alpha = 0, \tag{39}$$

$$\lambda y_{32} + y_{42}A_1 + \mu_2 y_{43}e\alpha + \mu_1(z_{41} + x_{51}) = 0, \tag{40}$$

$$\lambda\, y_{33} + y_{43}A_4 + \mu_1(z_{42} + x_{52}) = 0, \tag{41}$$

$$\lambda y_{34} + y_{42}T^{o}\alpha + y_{44}A_1 = 0, \tag{42}$$

$$(\xi_{15} + \xi_{24})M_2 + \lambda z_{31} + z_{41}A_2 + \mu_2 z_{42}e\alpha = 0, \tag{43}$$

$$\lambda z_{32} + y_{43}T^{o}\alpha + z_{42}A_2 + \mu_1(z_{43} + x_{53}) = 0, \tag{44}$$

$$(\xi_{15} + \xi_{24})M_3 + \lambda z_{33} + z_{42}T^{o}\alpha + z_{43}A_2 = 0, \tag{45}$$

$$\lambda y_{41} + x_{51}A_2 + \mu_2 x_{52}e\alpha = 0, \tag{46}$$

$$\lambda y_{42} + x_{52}A_2 = 0, \tag{47}$$

$$\lambda y_{44} + x_{52}T^{o}\alpha + x_{53}A_2 = 0, \tag{48}$$

with the normalizing equation

$$x^* + \xi_1 e + \xi_2 e + \sum_{j=0}^{5} x_j e = 1. \qquad (49)$$

The coefficient matrices appearing in the preceding equations are given by

$$A_1 = T - (\lambda+\mu_1+\mu_2)I\ , \qquad M_i = \mu_1(e_i \otimes \underline{\alpha})\ , \text{ for } 1 \le i \le 3\ ,$$

$$A_2 = T - (\mu_1+\mu_2)I\ , \qquad R = e_1' \otimes T^o\ , \qquad (50)$$

$$A_3 = T - (\lambda+\mu_2)I\ , \qquad A_4 = T - \mu_2 I\ ,$$

$$S_i = \mu_2(e_i' \otimes e) + (e_{i+1}' \otimes T^o), \text{ for } 1 \le i \le 2.$$

By using the special structure of the Kronecker products, which arise as coefficient matrices in equations (6) through (11), we can express ξ_1 and ξ_2 in terms of quantities of smaller dimensions. The final expressions are given as

Theorem 1 : We have,

$$\xi_{11} = (\lambda+\mu_1)^{-1}\ (\lambda x^* + \mu_2 y_{11} e,\ y_{11} T^o + \mu_2 y_{14} e,\ y_{14} T^o),$$

$$\xi_{1,j+1} = (\lambda+\mu_1)^{-1}[\lambda\xi_{1j} + (\mu_2 y_{j+1,1} e, y_{j+1,1} T^o + \mu_2 y_{j+1,4} e,\ y_{j+1,4} T^o)],$$

$$\text{for } 1 \le j \le 3,$$

$$\xi_{15} = [\ \lambda\xi_{14} + (\ \mu_2 x_{51} e\ ,\ \mu_2 x_{53} e + x_{51} T^o\ ,\ x_{53} T^o\ )\ ]/\mu_1\ ,$$

$$\xi_{20} = (\lambda+\mu_1)^{-1}\ (\ x_{01} T^o + \mu_2 x_{04} e,\ x_{04} T^o + \mu_2 x_{06} e,\ x_{06} T^o\ ). \qquad (51)$$

$$\xi_{2,j+1} = (\lambda+\mu_1)^{-1}[\lambda\xi_{2j} + (\mu_2 z_{j+1,1} e, z_{j+1,1} T^o + \mu_2 z_{j+1,3} e,\ z_{j+1,3} T^o)],$$

$$\text{for } 0 \le j \le 2,$$

$$\xi_{24} = [\ \lambda\xi_{23} + (\ \mu_2 z_{41} e\ ,\ \mu_2 z_{43} e + z_{41} T^{o}\ ,\ z_{43} T^{o}\ )\ ]/\mu_1\ .$$

The equations (5)-(48) will now be recast into a form which makes them appropriate for solution by block Gauss-Seidel iteration. We evaluate the three inverses

$$B_1 = [\ (\lambda+\mu_1+\mu_2)I - T\ ]^{-1}, \qquad B_2 = [\ (\mu_1+\mu_2)I - T\ ]^{-1},$$

$$B_3 = [\ (\lambda+\mu_2)I - T\ ]^{-1}, \qquad B_4 = [\ \mu_2 I - T\ ]^{-1},$$

that are needed for the iterative procedure. In Theorem 3.2.1[5] it is shown that the above inverses do indeed exist and are nonnegative. Below, we shall give a few sample equations that are used for computational purposes. The remaining equations are not listed due to space constraint and the interested reader can get these by similar steps. From equations (5) and (12), we obtain respectively,

$$x^{*} = (\mu_2 x_{01} e)/\lambda,$$

$$x_{01} = [\ \mu_2 x_{02} e_{\mathcal{Q}} + (\xi_{11} + \xi_{20}) M_1\ ] B_3\ .$$

The purpose of this is to obtain numerically stable recursive scheme. It should be noted that the quantities that appear on the right-hand side of the above equations are all nonnegative and thus lead to stable algorithms. We solved the above (recast) equations by block Gauss-Seidel iteration. The successive solution vectors are kept within a compact polytope by forcing them to satisfy the normalizing equation (49).

4.2 Measures of System Performance

In this section we list a number of system performance measures. The *throghput* (γ) of the system is defined as the rate at which the messages are successfully transmitted. On noting that $\gamma = \lambda$ P(primary buffer is not full) or $\gamma = \mu_2$ P (at least one message in the secondary buffer), we see that

$$\gamma = \lambda\ [1 - \xi_{15}e - \xi_{24}e - z_4e - y_{43}e - x_5e\ ] = \mu_2\ [\ 1 - x^* - \xi_1 e - \xi_2 e\ ]$$

The *blocking probability* (δ) of the transmitting unit is given by

$$\delta = \sum_{j=1}^{4} y_{j3}e$$

The *system idle probability* is given by x^*. The other system performance measures that are discussed in this paper are as follows : Means and standard deviations of the numbers of messages waiting in the primary, secondary and auxiliary buffers. These quantities are easily computed from the stationary probability vector $\mathbf{x}$. For example, the mean number (η) of messages waiting in the primary buffer is given by

$$\eta = \sum_{k=1}^{3} k\ [\ \xi_{1,k+1}e + \xi_{2k}e + y_{k3}e + z_k e + y_{k+1}e - y_{k+1,3}e\ ]$$

$$+ 4\ [\ \xi_{15}e + \xi_{24}e + y_{43}e + z_4 e + x_5 e\ ].$$

4.3 Accuracy Checks

Algorithms for general PH-distributions have a powerful accuracy check, which is based on the following property. If T is an irreducible, stable matrix with eigenvalue of maximum real part $-c < 0$, and if $\underline{\alpha}$ is chosen to be the corresponding left eigenvector, normalized by $\underline{\alpha}e = 1$, then $\underline{\alpha}$ is a positive vector and the PH-distribution with the representation $(\underline{\alpha}, T)$ is the exponential distribution with parameter c.

First we obtained the numerical solution for the exponential distributions in their simple form. Next, we implemented the general algorithm, but chose the representation of the PH-distribution *so that it was in fact exponential*. The general algorithm does not utilize this fact in any manner, but the two sets of numerical results agreed very much.

A number of other accuracy checks are also available. For example, the result of Theorem 2 below may be used to check whether the computed quantities satisfy it. We conclude this section with the following theorem.

Theorem 2: We have,

$$\lambda \left[\xi_{1j} e + \xi_{2,j-1} e + x_{j-1} e \right]$$

$$= \mu_1 \Big\{ \sum_{\substack{k=1 \\ k \neq 3}} y_{jk} e \Big\} + \mu_2 \left[y_{j1} e + y_{j4} e \right] + y_{j1} T^o + y_{j4} T^o, \quad 1 \leq j \leq 4,$$

$$\lambda \left[y_{41} e + y_{42} e + y_{44} e \right] = \mu_1 x_5 e + \mu_2 \left[x_{51} e + x_{53} e \right] + x_{51} T^o + x_{53} T^o.$$

Proof : We shall outline the proof for the first two equations as the others are similar. Postmultiplying each one of the equations (12) through (17) by e and adding the resulting equations and using equations (6) and (9), we get the equation for j=1. Now postmultiplying each one of the equations (18) through (24) by e and by adding the resulting equations and using the equation obtained for j=1, and equations (7) and (10), we get the equation for j=2.

5. NUMERICAL EXAMPLES

In order to test the feasibility of the algorithm proposed in this paper, a FORTRAN code was developed and tested on a large number of examples. In this section we shall discuss a number of examples.

The purpose of the first example is to show how the system performance measures are affected due to a change in the probability distribution of the timeout period.

Example 1. Here, $\mu_1 = \mu_2 = 100$, and $\lambda = 50$. We consider four PH-distributions whose representations (α, T) are given as follows:

		α	T
I)	Erlang of order 10 with parameter 100.		
II)	Exponential	1	-10
III)	General PH-distribution	(0.3,0.7)	$\begin{bmatrix} -10 & 7.4 \\ 5.75 & -25 \end{bmatrix}$
IV)	Hyperexponential	(0.99,0.01)	$\begin{bmatrix} -25 & 0 \\ 0 & -1/6.04 \end{bmatrix}$

While each of these distributions have the same mean 0.1, the distributions are qualitatively very different. The variances of these distributions are respectively 10^{-3}, $10\text{x}10^{-3}$, $14\text{x}10^{-3}$, and $722.8\text{x}10^{-3}$. The performance measures: throughput (γ), blocking probability (δ), idle probability (x^*), expected number of messages in the primary, secondary, and auxiliary buffers denoted respectively by EPB, ESB, and EAB and their standard deviations denoted respectively by SPB, SSB, and SAB are given in table 3.

TABLE 3

Performance measures for four systems with different timeout distributions all having the same mean.

	I	II	III	IV
γ	48.6259228	48.2128816	48.0443538	47.4467966
δ	0.0567970	0.0554344	0.0551878	0.0530324
x^*	0.2486512	0.2254771	0.2172642	0.1932486
EPB	0.5881190	0.6819047	0.7178059	0.8346006
SPB	1.0273865	1.1007878	1.1265698	1.2032027
ESB	0.7880407	0.7756025	0.7714101	0.7543386
SSB	0.9652667	0.9567162	0.9543451	0.9427258
EAB	0.0000646	0.0242191	0.03406010	0.0668052
SAB	0.0080358	0.1564650	0.18576732	0.2603967

Examining table 3 we observe that as the variance of the timeout period increases, the performance measures throughput, the blocking probability and the idle probability appear to decrease, as is to be expected.

The objective of Examples 2 and 3 is to see how the various performance measures are affected by considering two PH-distributions having different means and variances but with the same coefficient of variation and by varying λ.

Example 2. Here $\mu_1 = \mu_2 = 100$. The timeout period has the PH-representation (α, T) given by

$$\alpha = (0.999, 0.001), \qquad T = \begin{bmatrix} -100 & 0 \\ 0 & -1/90.01 \end{bmatrix}$$

The mean and coefficient of variation for this PH-distribution are respectively 0.1 and 40.241521.

Example 3. Here $\mu_1 = \mu_2 = 100$. The timeout has the PH representation (α, T) given by

$$\alpha = (0.999, 0.001), \qquad T = \begin{bmatrix} -1.0000003446 & 0 \\ 0 & -1.1109876637 \times 10^{-4} \end{bmatrix}$$

The mean and coefficient of variation for this PH-distribution are respectively 10 and 40.241521.

The throughput, idle probability, and blocking probability are given in table 4.

While the distribution considered in Example 2 can be visualized as producing timer expirations of short periods with occational bursts of timer expirations of intermediate periods, the distribution in Example 3 can be visualized as producing timer expirations of relatively large periods with occational bursts of intermediate periods.

TABLE 4

Performance Measures for Examples 2 & 3 having PH-distributions with different means and variances.

	Throughput		Idle prob.		Blocking prob.	
λ/μ_1	Ex. 2	Ex. 3	Ex. 2	Ex. 3	Ex. 2	Ex. 3
0.1	9.9991553	9.9999041	.7200927	.8090761	.0002287	.0001778
0.2	19.9236208	19.9938591	.4801913	.6380620	.0030376	.0024916
0.4	36.4542506	39.6053605	.1531751	.3554163	.0247622	.0287528
0.6	43.7547136	56.3212742	.0368445	.1616858	.0477109	.0917169
0.8	45.8197577	66.7187442	.0093176	.0610520	.0585978	.1592081
1.0	46.3950986	71.4762545	.0027565	.0213765	.0630283	.2028236
1.2	46.5799154	73.3532804	.0009464	.0076382	.0649292	.2250017
1.4	46.6479653	74.0819515	.0003670	.0029046	.0658163	.2355487
1.6	46.6761243	74.3775309	.0001570	.0011868	.0662634	.2406199
1.8	46.6888927	74.5050929	.0000726	.0005195	.0665036	.2431535
2.0	46.6952100	74.5637168	.0000359	.0002419	.0666398	.2444774
5.0	46.7306207	74.6258908	.0000000	.0000001	.0668685	.2462757

Examining Table 4, we observe that

- by considering PH-distributions having the same coefficient of variation and increasing the mean significantly from 0.1 to 10, the throughput and idle probability appear to increase accordingly.

- the blocking probability however appears to decrease for values of λ up to about 38 and appears to increase for λ greater than 38. The result seems to be obvious for large values of λ. However, for small values of λ, the result is not that obvious.

- in general performance measures follow expected behavior with respect to message arrival rate. For example, increasing λ, the throughput increases asymptotically towards its saturation value.

6. REFERENCES

[1] Bellman, R. (1960). " Introduction to Matrix Analysis", McGraw Hill, New York, NY.

[2] Chakravarthy, S. (1986). " A Stochastic Model for a Computer Communications Network", working paper. Department of Science and Mathematics, GMI Engineering and Management Institute.

[3] Marcus, M. and Minc, H. (1964). "**A Survey of Matrix Theory and Matrix Inequalities**", Allyn and Bacon, Boston, MA.

[4] Neuts, M.F. (1975). "Probability Distributions of Phase Type" Liber Amicorum Prof. Emeritus H. Florin, Department of Mathematics, University of Louvain, Belgium, 173-206.

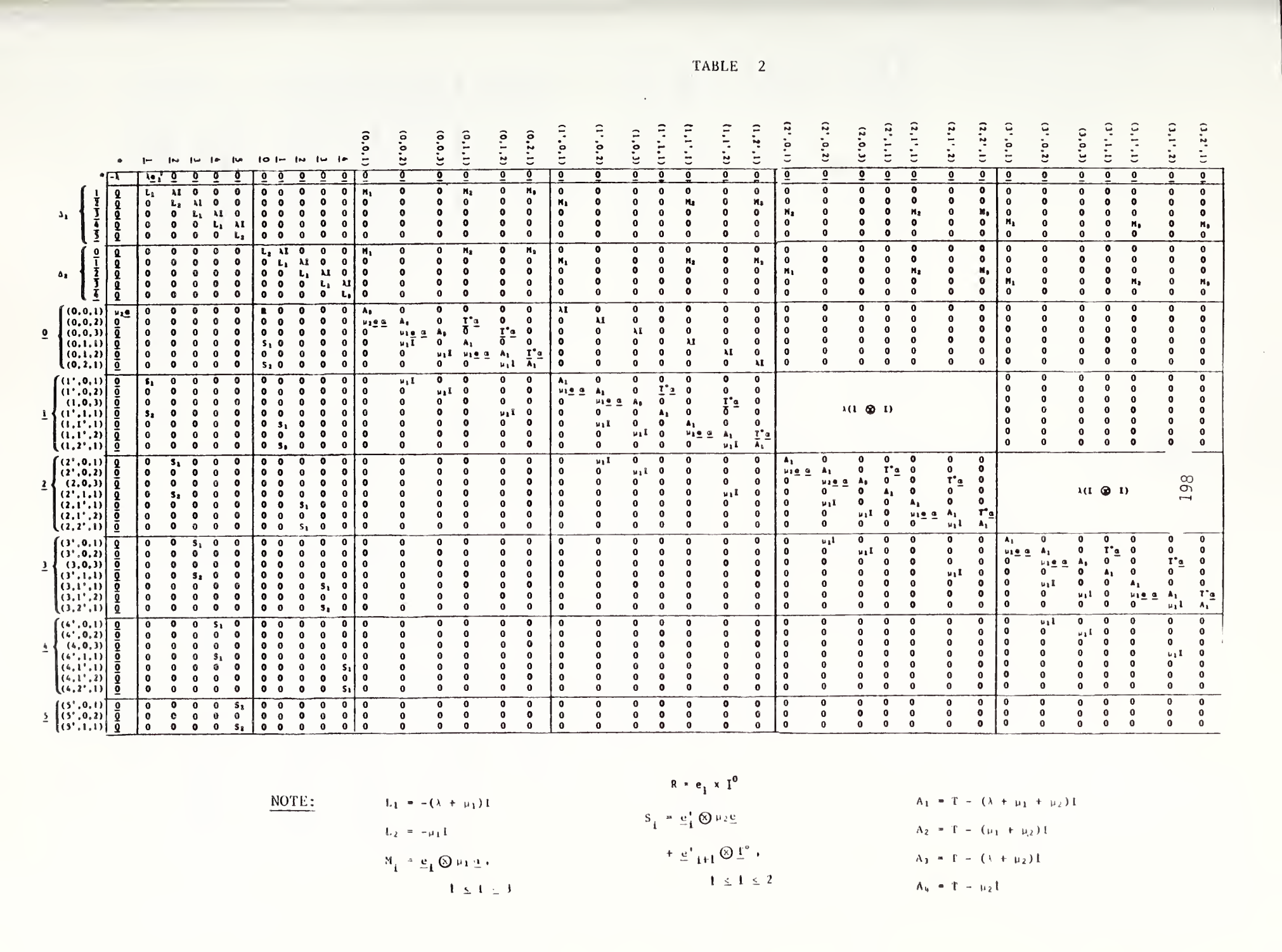
[5] Neuts, M.F. (1981). "**Matrix-Geometric Solutions in Stochastic models- An Algorithmic Approach**", The Johns Hopkins University Press, Baltimore, MD.

[6] Sethi, A.S. and Ghosal, R. (1986)." An analytical model for window mechanisms with error control", Proc. INFOCOM86, pp 162-171.

[7] Stallings, W. (1985)." **Data and Computer Communications** ", MacMillan Publishing Co., New York.

[8] Tanenbaum, A.S. (1981)." **Computer Networks** ", Prentice Hall, Englewood cliffs, New Jersey.

TABLE 2



NOTE:

$L_1 = -(\lambda + \mu_1)I$

$L_2 = -\mu_1 I$

$M_i = \underline{e}_i \otimes \mu_1 \underline{e}$,

$1 \le i \le 3$

$R = e_1 \times I^0$

$S_i = \underline{e}_i' \otimes \mu_2 \underline{e}$

$+ \underline{e}'_{i+1} \otimes \underline{I}^0$,

$1 \le i \le 2$

$A_1 = T - (\lambda + \mu_1 + \mu_2)I$

$A_2 = T - (\mu_1 + \mu_2)I$

$A_3 = T - (\lambda + \mu_2)I$

$A_4 = T - \mu_2 I$

Setting Target_Rotation_Times in an 802.4 Network

Mangala Gorur
Alfred C. Weaver
Dept. of Computer Science
University of Virginia
Charlottesville, Va 22903

The IEEE 802.4 token bus protocol for local area networks is emerging as a popular standard for factory automation applications. The enormous interest in this protocol is due to its various distinguishing features, one of which is the provision of a prioritized mechanism to access the transmission medium. The implementation of the priority scheme by any station in an 802.4 network is optional. If a station implements the priority scheme, then the objective is to allocate bandwidth to lower priority frames only after transmission of higher priority frames. Each of the lower three access_classes is assigned a target token rotation time. This goal rotation time is different for each access_class and is known as the *Target_Rotation_Time* (TRT) of an access_class. The TRT setting at each access_class decides the amount of time available to serve messages of that priority. The IEEE 802.4 token bus standard specifies only the maximum values for the Target_Rotation_Times. Hence it is essential to know how to set the TRTs to achieve a desired priority scheme.

In this paper we present an analytic model which can be used to solve for the TRT settings which will implement a user-defined priority scheme. For example, suppose that the user wants normal service at Time_Available access_class until network throughput rises to α, with $0 \le \alpha \le 1$. Then letting N be the number of stations in the network, X_T be the transmission time of a token, X_M be the transmission time of a message, and λ_s, λ_{ua}, λ_{na}, and λ_{ta} be the arrival rate in packets per second at the Synchronous, Urgent_Asynchronous, Normal_Asynchronous, and Time_Available access_classes respectively, and using the analytic model, the effective value of the TRT_{ta} is,

$$eff\,(TRT_{ta}) = \frac{N \cdot X_T}{1 - N \cdot X_M \cdot (\lambda_s + \lambda_{ua} + \lambda_{na} + \lambda_{ta})}.$$

The effective value can be higher than the actual TRT setting by up to one message transmission time. This is according to the specifications of the standard that the transmission of a message once started will go on to completion even if it runs past the expiration of the timers.

The analytic predictions show close agreement with the simulation results. Thus, given a user-defined performance target (eg., "Time_Available access_class to receive normal service until network throughput exceeds α"), we can calculate the TRT settings which will achieve that goal.

1. Introduction

The 802.4 token passing access method offers four levels of service called *access_classes* and messages can be transmitted at any of the four priorities. The four access_classes in descending order of priority are Synchronous access_class, Urgent_Asynchronous access_class, Normal_Asynchronous access_class, and Time_Available access_class. The implementation of the priority scheme by any station is optional. By suitably setting the variables associated with the implementation of the priority scheme, it can be ensured that this scheme gives preference to frames of higher priority. The amount of time available to transmit messages from the Synchronous access_class is decided by the value of the variable High_Priority_Token_Hold_Time (HPTHT). The amount of time available to transmit messages from each of the lower three access_classes is decided by the value of the Target_Rotation_Time (TRT) setting at each access_class. This setting is different for each access_class.

The TRT setting at each access_class limits the token cycle time at each access_class. In this paper we present an analytic model that can be used to calculate the values of TRT settings to obtain optimum service at an access_class until a user-defined throughput is reached.

1.1. An analytic model for determining TRTs

A problem faced by token bus network designers and operators is the selection of Target_Rotation_Times which will implement a desired priority scheme. It is possible to determine the individual TRT setting of an access_class so as to obtain maximum possible service at an access_class (that is, on the average a message gets transmitted within one token cycle) at an access_class until network throughput reaches or exceeds a user-defined threshold. For purposes of analysis presented in this paper, throughput has been defined as the number of data bits transmitted (including all address and framing bits) per bit time expressed as a fraction of the bus capacity.

1.1.1. Definitions

An active access_class at a station is termed a server. The following notation is used:

X_T ≡ duration of a token transmission (seconds/token transmission).

s ≡ the Synchronous access_class.

ua ≡ the Urgent_Asynchronous access_class.

na ≡ the Normal_Asynchronous access_class.

ta ≡ the Time_Available access_class.

λ_{ac} ≡ the mean message arrival rate at each server of an access_class,

where ac is s, ua, na or ta.

S ≡ the set of all *Synchronous access_class* servers.

UA ≡ the set of all *Urgent_Asynchronous access_class* servers.

NA ≡ the set of all *Normal_Asynchronous access_class* servers.

TA ≡ the set of all *Time_Available access_class* servers.

R ≡ set of all distinct servers on the logical ring

≡ (S, UA, NA, TA).

λ_x ≡ the mean message arrival rate at server $x \varepsilon R$ (messages/second).

$HPTHT$ ≡ the *High_Priority_Token_Hold_Time*.

TRT_x ≡ the *Target_Rotation_Time* at server $x \varepsilon (UA, NA, TA)$.

TRT_{asy} ≡ the *Target_Rotation_Time* at each server of an access_class,

where asy is ua, na, or ta.

$TCT_{x,i}$ ≡ the time from the end of the $(i-1)^{st}$ service period

until the beginning of the i^{th} service period (seconds)

≡ token circulation time as seen by server x on the i^{th} token cycle,

i.e., the time interval for which the token is away from x.

$TC_{x,i}$ ≡ time from the end of the $(i-1)^{st}$ service period until

the end of the i^{th} service period (seconds).

≡ token cycle time as seen by server x on the i^{th} token cycle.

$A_{x,i}$ ≡ the number of message arrivals at x during the interval from the end of the $(i-1)^{st}$ service period until the end of the i^{th} service period.

$Q_{x,i}$ ≡ the number of messages enqueued at server x after the $(i-1)^{st}$ service period.

$f(n)$ ≡ time required to transmit n messages (octet_times).

t ≡ residual time in the token_hold_timer (octet_times).

$eff(t)$ ≡ the effective amount of time which a server can transmit messages.

= $max(0, t + f(1) -$ one *octet_time*) octet_times.

$TS_{x,i}$ ≡ duration of the i^{th} service period.

$\overline{TS}_{ac}$ ≡ average service time available to every server of an access_class, where ac is s, ua, na or ta.

1.1.2. Assumptions

In the following section, an analytic expression for the token cycle time of an 802.4 network is derived. The derivation is based on the following assumptions about the characteristics of the network:

1) The protocol management overhead is assumed to be negligible

2) Stations are permanent members of the logical ring.

3) Each station has traffic at all four access_classes.

4) Message arrival at each station follows a Poisson process.

5) Messages from all the servers are of constant length l_M bits and take X_M seconds for transmission.

6) Token is of length 112 bits for a 10 Mbps bus and takes X_T seconds for transmission.

7) All Synchronous access_class servers have the same HPTHT setting.

8) Each station has HPTHT set to the maximum allowed value (52.43 msec for a 10 Mbps bus). Hence messages from the Synchronous access_class normally are transmitted on the same token cycle as they arrive.

9) All servers of the same priority have identical traffic.

10) All servers of the same priority have the same TRT settings.

11) The TRTs are set in the order $TRT_{ua} > TRT_{na} > TRT_{ta}$.

12) Each active access_class has N servers.

1.1.3. Service time

The amount of time available to any server on any token cycle is limited. A station starts transmitting messages when there is residual time in the token_hold_timer. If x is a synchronous server then the average service time of x can be expressed as

$$\overline{TS_x} = \min(eff\ (HPTHT), f\ (\bar{Q}_x + \bar{A}_x\ )). \tag{1}$$

where $\bar{Q}_x$ is the average queue length at x, and $\bar{A}_x$ is the average number of arrivals at x during an average token cycle. If x is an asynchronous server then the average service time of server x can be expressed as

$$\overline{TS_x} = \begin{cases} \min(eff\ (TRT_x - \overline{TCT_x}\ ), f\ (\bar{Q}_x + \bar{A}_x\ )) & \text{for} \quad \overline{TCT_x} < TRT_x \\ 0 & \text{for} \quad \overline{TCT_x} \geq TRT_x \end{cases} \tag{2}$$

where $\overline{TCT_x},\ x \varepsilon TA$ is the average token circulation time at every server at the Time_Available class, $\bar{Q}_x$ is the average queue length at x, and $\bar{A}_x$ is the average number of arrivals at x. Refer to [Gorur 86] for a derivation of token cycle time and token circulation time. Assuming an average arrival rate of messages λ_x at server x, the number of messages that have arrived while

a) the token is circulating around the ring and

b) x is being served on the i^{th} token cycle is

$$A_{x,i} = \lambda_x \cdot TCT_{x,i} + \lambda_x \cdot TS_{x,i} = \lambda_x \cdot TC_{x,i}. \tag{3}$$

Hence the average number of messages that have arrived at any server x, $x\ \varepsilon\ (S, UA, NA, TA)$, during an average token cycle can be expressed as

$$\bar{A}_x = \lambda_x \cdot \overline{TC}. \tag{4}$$

The time taken to transmit the average number of messages that have arrived at a server x can be expressed as $X_M \cdot \lambda_x \cdot \overline{TC}$, from equation (4). As long as the time to transmit all the messages that have arrived is less than or equal to HPTHT (if x is a synchronous server), or less than or equal to $(TRT_{asy} - \overline{TCT})$ if x is an asynchronous server, the average service time at server x can be expressed as

$$\overline{TS_x} = f(\bar{Q}_x + \bar{A}_x). \tag{5}$$

This is true for $\overline{TC} \leq HPTHT$ and $\overline{TC} \leq TRT_{asy}$. In the region $\overline{TC} \leq HPTHT$ it can be assumed that all the messages from the Synchronous access_class get transmitted during the same token cycle. Also in the region $\overline{TC} \leq TRT_{asy}$ it can be assumed that all the messages from that access_class and from access_classes of higher priority that have arrived during a particular token cycle get transmitted during the same token cycle. Hence $\bar{Q}_x$ can be considered to be negligible. Eliminating $\bar{Q}_x$ from equation (5) and substituting the value of $\bar{A}_x$ from equation (4), the average service time can be expressed as

$$\overline{TS_x} = \overline{TS_{ac}} = X_m \cdot \lambda_{ac} \cdot \overline{TC}. \tag{6}$$

Traffic from each access_class gets served in the order of priority, starting from the Synchronous access_class, and if time is available, lower access_classes also get service. The average service time at each of the lower three access_classes can increase with an increase in offered load from that access_class until the average token cycle time equals the TRT of that access_class. Thus the throughput which results in a token cycle time of TRT for an access_class corresponds to the throughput until which maximum possible service can be obtained at that access_class.

1.1.4. Determining TRTs

If the designer determines that the Time_Available access_class should receive maximum possible service until network throughput reaches α, $0 \leq \alpha \leq 1$, then

$$\alpha = \frac{N \cdot \Lambda_\alpha \cdot l_M}{C \cdot \overline{TC}_\alpha} \tag{7}$$

where Λ_α is the mean number of messages transmitted per station per token cycle at a throughput of α, and $\overline{TC}_\alpha$ is the average token cycle time at a throughput of α. Solving for Λ_α,

$$\Lambda_\alpha = \frac{\alpha \cdot C \cdot \overline{TC}_\alpha}{N \cdot l_M}. \tag{8}$$

From the discussion in section 1.2.3, it follows that maximum possible service at an access_class can be obtained in the region where $\overline{TC} \leq TRT_{asy}$. Hence the average number of messages that arrive during the time interval $\overline{TC}_\alpha$ (region where $\overline{TC}_\alpha \leq TRT_{ta}$) should equal the mean number of messages transmitted in

this time interval, i.e., Λ_α. Hence

$$\Lambda_\alpha = (\lambda_s + \lambda_{ua} + \lambda_{na} + \lambda_{ta}) \cdot \overline{TC}_\alpha. \tag{9}$$

Substituting for Λ_α in equation (8) we obtain the sum of λs to be

$$\lambda_s + \lambda_{ua} + \lambda_{na} + \lambda_{ta} = \frac{\alpha \cdot C}{N \cdot l_M}. \tag{10}$$

Many combinations of individual message arrival rates will yield the unique sum satisfying the above equation. Since the token cycle time can be expressed as

$$\overline{TC}_\alpha = N \cdot X_T + N \cdot X_M \cdot \Lambda_\alpha, \tag{11}$$

substituting for Λ_α from equation (9) yields

$$\overline{TC}_\alpha = N \cdot X_T + N \cdot X_M \cdot (\lambda_s + \lambda_{ua} + \lambda_{na} + \lambda_{ta}) \cdot \overline{TC}_\alpha. \tag{12}$$

Since the Time_Available access_class should receive maximum possible service until throughput reaches α, it follows from the discussion of service time in the previous section that the token cycle time at this throughput decides the TRT of the Time_Available access_class. Hence solving for $eff\,(TRT_{ta}) = \overline{TC}_\alpha$,

$$eff\,(TRT_{ta}) = \frac{N \cdot X_T}{1 - N \cdot X_M \cdot (\lambda_s + \lambda_{ua} + \lambda_{na} + \lambda_{ta})} \tag{13}$$

which is the effective value of the Time_Available *Target_Rotation_Time*.

As the offered load increases, increasing the network throughput beyond α, the token cycle time increases beyond the TRT of the Time_Available access_class. Hence service to the Time_Available access_class class gets reduced and eventually drops to zero when TCT_{ta} equals TRT_{ta} as given by equation (2). Hence the traffic from the Time_Available access_class does not contribute to the network throughput anymore. Hence the load carried by the network is contributed by the upper three classes alone.

If the designer determines that the service at the Normal_Asynchronous access_class should reach a maximum when the network throughput reaches β, $0 \le \alpha < \beta \le 1$, then

$$\beta = \frac{N \cdot \Lambda_\beta \cdot l_M}{C \cdot \overline{TC}_\beta} \tag{14}$$

where Λ_β is the mean total number of messages transmitted per station per token cycle at a throughput of β. Solving for Λ_β,

$$\Lambda_\beta = \frac{\beta \cdot C \cdot \overline{TC}_\beta}{N \cdot l_M}. \tag{15}$$

The mean number of message arrivals during the time interval $\overline{TC}_\beta$ should equal the mean number of messages transmitted during that time interval, i.e., Λ_β. Hence

$$\Lambda_\beta = (\lambda_s + \lambda_{ua} + \lambda_{na}) \cdot \overline{TC}_\beta. \tag{16}$$

Substituting for Λ_β in equation (15) we obtain the sum of message arrival rates of the upper three access_classes to be

$$\lambda_s + \lambda_{ua} + \lambda_{na} = \frac{\beta \cdot C}{N \cdot l_M}. \tag{17}$$

Of course, many combinations of individual message arrival rates will yield the unique sum in equation (17). The token cycle time in this region can be expressed as

$$\overline{TC}_\beta = N \cdot X_T + N \cdot X_M \cdot \Lambda_\beta. \tag{18}$$

Substituting for Λ_β from equation (16) yields

$$\overline{TC}_\beta = N \cdot X_T + N \cdot X_M \cdot (\lambda_s + \lambda_{ua} + \lambda_{na}) \cdot \overline{TC}_\beta. \tag{19}$$

Since service at the Normal_Asynchronous access_class should be maximum at a throughput of β, it follows from the discussion of service time in section 1.2.3 that the token cycle time at this carried load decides the TRT of the Normal_Asynchronous access_class. Solving for *eff* $(TRT_{na}) = \overline{TC}_\beta$,

$$eff\,(TRT_{na}) = \frac{N \cdot X_T}{1 - N \cdot X_M \cdot (\lambda_s + \lambda_{ua} + \lambda_{na})} \tag{20}$$

which is the effective value of the Normal_Asynchronous *Target_Rotation_Time*.

As the offered load increases, increasing the network throughput beyond β, the token cycle time increases beyond the TRT of the Normal_Asynchronous access_class. Hence service to the Normal_Asynchronous class gets reduced and eventually drops to zero when TCT_{na} equals TRT_{na} as given by equation (2). At this point traffic from the Normal_Asynchronous access_class does not contribute to the network throughput anymore, so the load carried by the network is contributed only by the upper two classes.

If a designer determines that the service to the Urgent_Asynchronous access_class should be maximized when the network throughput is γ where $0 \le \alpha < \beta < \gamma \le 1$, then

$$\gamma = \frac{N \cdot \Lambda_\gamma \cdot l_M}{C \cdot \overline{TC}_\gamma}. \tag{21}$$

Using logic similar to the previous discussion, the number of messages transmitted during the time interval $\overline{TC}_\gamma$ is

$$\Lambda_\gamma = \frac{\gamma \cdot C \cdot \overline{TC}_\gamma}{N \cdot l_M}, \tag{22}$$

and

$$\Lambda_\gamma = (\lambda_s + \lambda_{ua}) \cdot \overline{TC}_\gamma \tag{23}$$

Then

$$\lambda_s + \lambda_{ua} = \frac{\gamma \cdot C}{N \cdot l_M}, \tag{24}$$

and

$$\overline{TC}_\gamma = N \cdot X_T + N \cdot X_M \cdot \Lambda_\gamma \tag{25}$$

and

$$\overline{TC}_{\gamma} = N \cdot X_T + N \cdot X_M \cdot (\lambda_s + \lambda_{ua}) \cdot \overline{TC}_{\gamma} \tag{26}$$

Now solving for $eff\,(TRT_{ua}) = \overline{TC}_{\gamma}$,

$$eff\,(TRT_{ua}) = \frac{N \cdot X_T}{1 - N \cdot X_M \cdot (\lambda_s + \lambda_{ua})} \tag{27}$$

which is the effective value of the Urgent_Asynchronous *Target_Rotation_Time*.

As the offered load increases, increasing the network throughput beyond γ, the token cycle time increases beyond the TRT of the Urgent_Asynchronous access_class. Hence service to the Urgent_Asynchronous class gets reduced and finally drops to zero when TCT_{ua} equals TRT_{ua} as given by equation (2).

1.2. Results

In order to verify the analytic predictions, a 32 station network with each station offering service at all four access_classes, with identical traffic at all servers of an access_class, was simulated using the simulation package reported in [Summers 85] with parameters:

Bus capacity = 10 Mbps,

Size of data frames = 272 bits including framing,

Token = 112 bits,

X_T = 16.2 microseconds (including 50 bit times of propagation delay),

N = 32,

X_M = .0000272 seconds = 27.2 microseconds,

X_T = .0000162 seconds = 16.2 microseconds.

The individual message arrival rates were assumed such that they satisfied the conditions derived in the previous section. From the effective values of TRTs calculated using the equations derived in the previous section, the actual values of TRTs are given by ($eff\ (TRT_{asy}) - f\ (1) + 1\ octet\_time$), where f(1) is equal to 34 octet_times, $eff\ (TRT_{ta})$ is 790 octet_times, $eff\ (TRT_{na})$ is 852 octet_times, $eff\ (TRT_{ua})$ is 1157 octet_times, and TRT_{ta} is 0.6056 msecs, TRT_{na} is 0.655 msecs, and TRT_{ua} is 0.8992 msecs.

The TRTs were set to the actual values shown above. A series of network configurations were simulated to show the variation of different parameters with throughput. The individual message arrival rates at each access_class have been varied from one simulation to another to increase the total offered load.

It can be observed from Figure 1 that average service time at the Time_Available access_class increases until a throughput of about 0.18. As the throughput increases further, the average service time starts falling and eventually drops to zero. From Figure 2 it can be observed that the average delivery times are reasonably low until the network throughput exceeds 0.18. This shows that most messages are getting transmitted on the same token cycle as they arrive. Beyond a throughput of 0.18 the token cycle time has exceeded TRT_{ta}. This can be observed from Figure 7. The Time_Available class is getting reduced service and messages are suffering higher queueing delays, so the delivery times are increasing

exponentially in this region. This can be observed from Figure 2.

The TRT_{ta} value was calculated so as to achieve maximum possible service until a throughput of 0.18. The results of the simulation agree with the expected values very closely.

It can be observed from Figure 3 that the average service time at the Normal_Asynchronous access_class increases until a throughput of about 0.24. As the throughput increases further, the average service time starts decreasing and drops to zero. From Figure 4 it can be observed that the average delivery times are reasonably low until the network throughput is 0.24. This indicates that most messages are getting transmitted on the same token cycle as they arrive. Beyond a throughput of 0.24 the token cycle time has exceeded TRT_{na}. This can be seen from Figure 7. The Normal_Asynchronous class is getting reduced service leading to higher queueing delays, and the delivery times rise exponentially in this region. This can be observed from Figure 4.

The TRT_{na} value was calculated so as to achieve maximum possible service until a throughput of 0.24. From the above discussion it follows that the results of the simulation agree with the expected values very closely.

It can be observed from Figure 5 that the average service time at the Urgent_Asynchronous access_class increases until a throughput of about 0.44. As the throughput increases further, the average service time starts decreasing. From Figure 6 it can be observed that the average delivery times are reasonably low until the network throughput is 0.42. This indicates that most messages are getting transmitted on the same token cycle as they arrive. Beyond a throughput of 0.42 the token cycle time has exceeded TRT_{ua} and hence the Urgent_Asynchronous class is getting reduced service. This can be observed from Figure 7. Messages are suffering higher queueing delays and delivery times rise exponentially in this region, as seen in Figure 6.

The calculations showed that maximum possible service can be obtained at the Urgent_Asynchronous access_class until a throughput of 0.44. The same is illustrated by the simulation results.

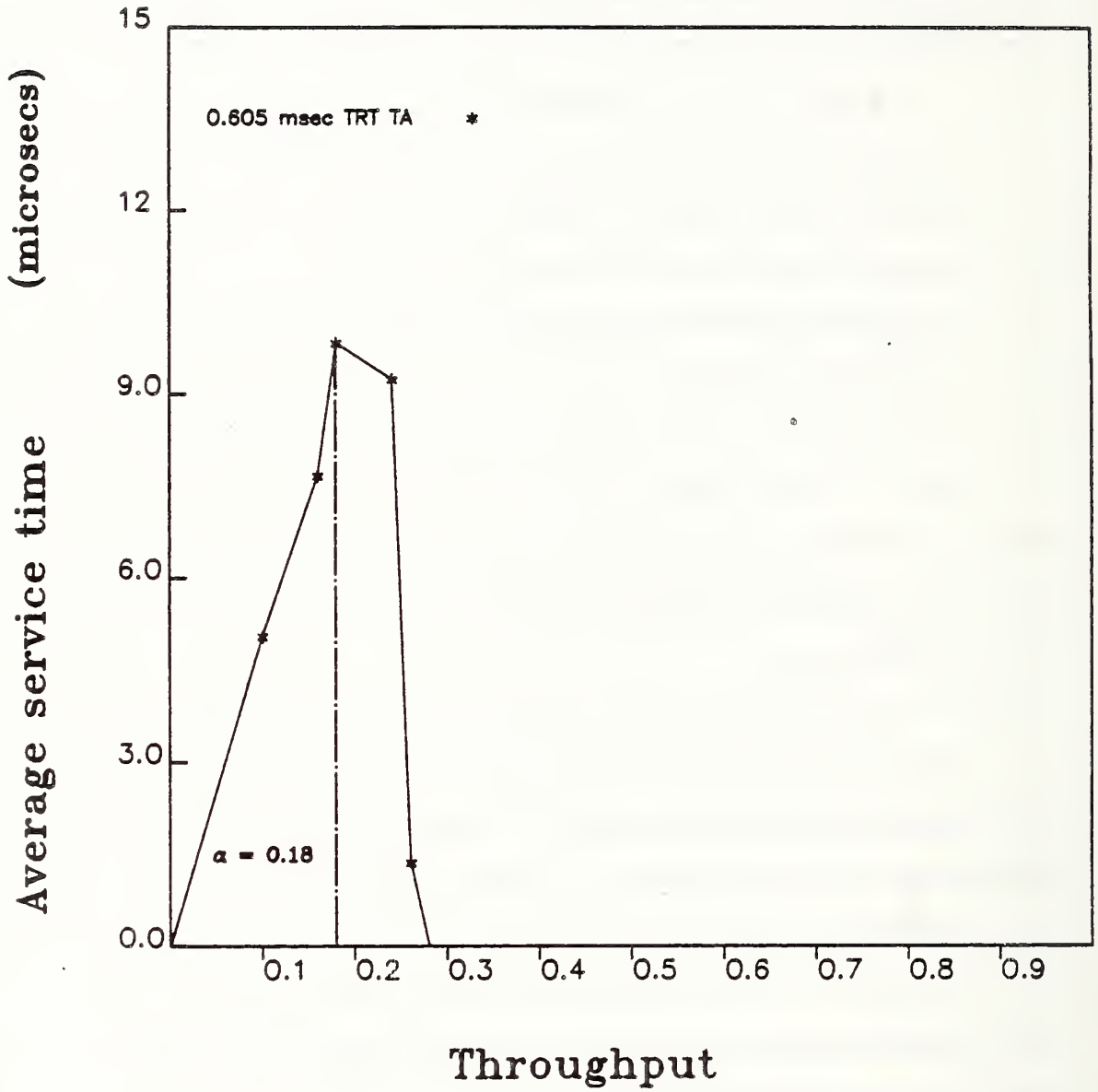


Figure 1
Average service time for Time_Available class

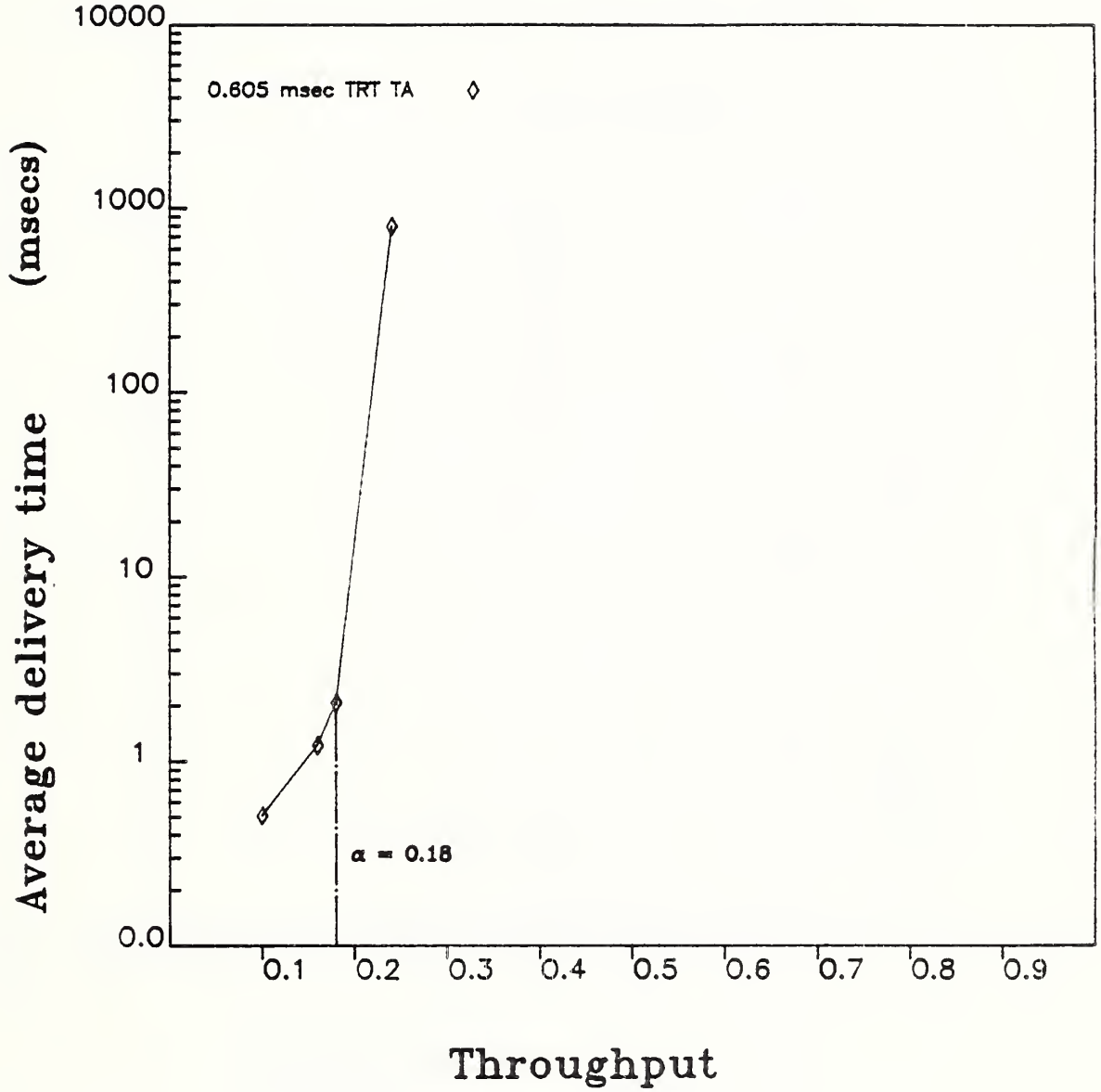


Figure 2
Average delivery time for Time_Available class

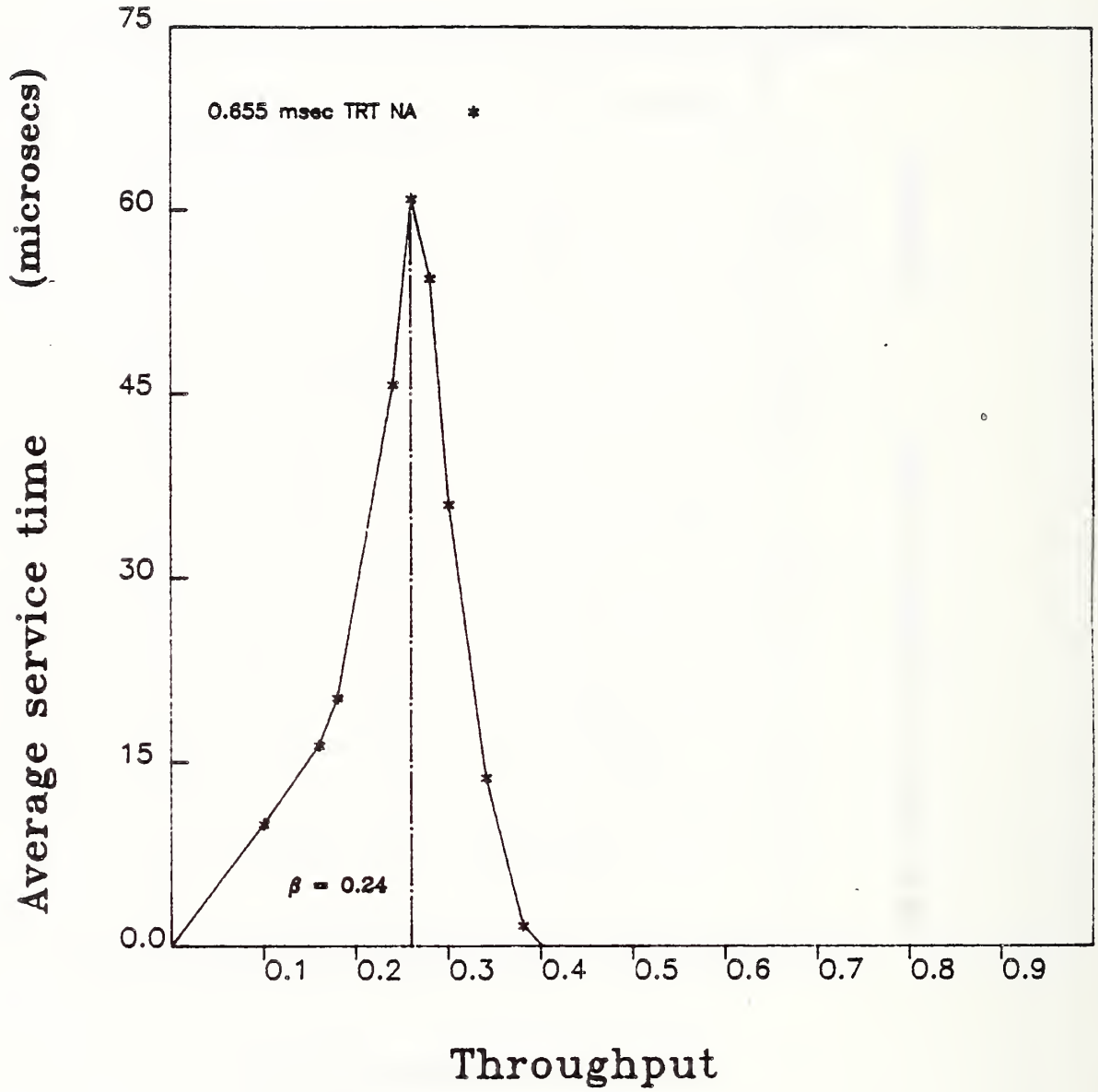


Figure 3
Average service time for Normal_Asynchronous class

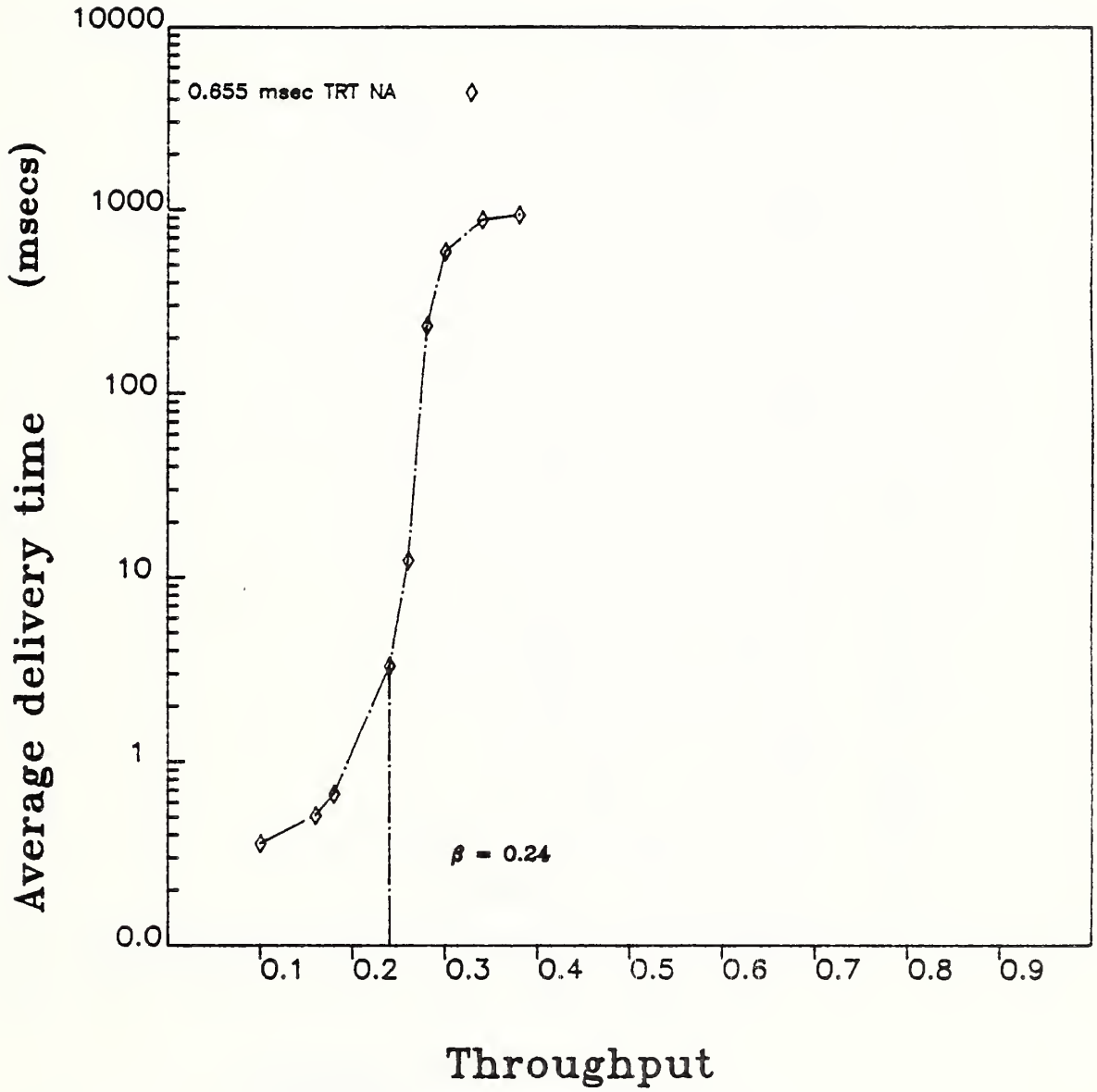


Figure 4
Average delivery time for Normal_Asynchronous class

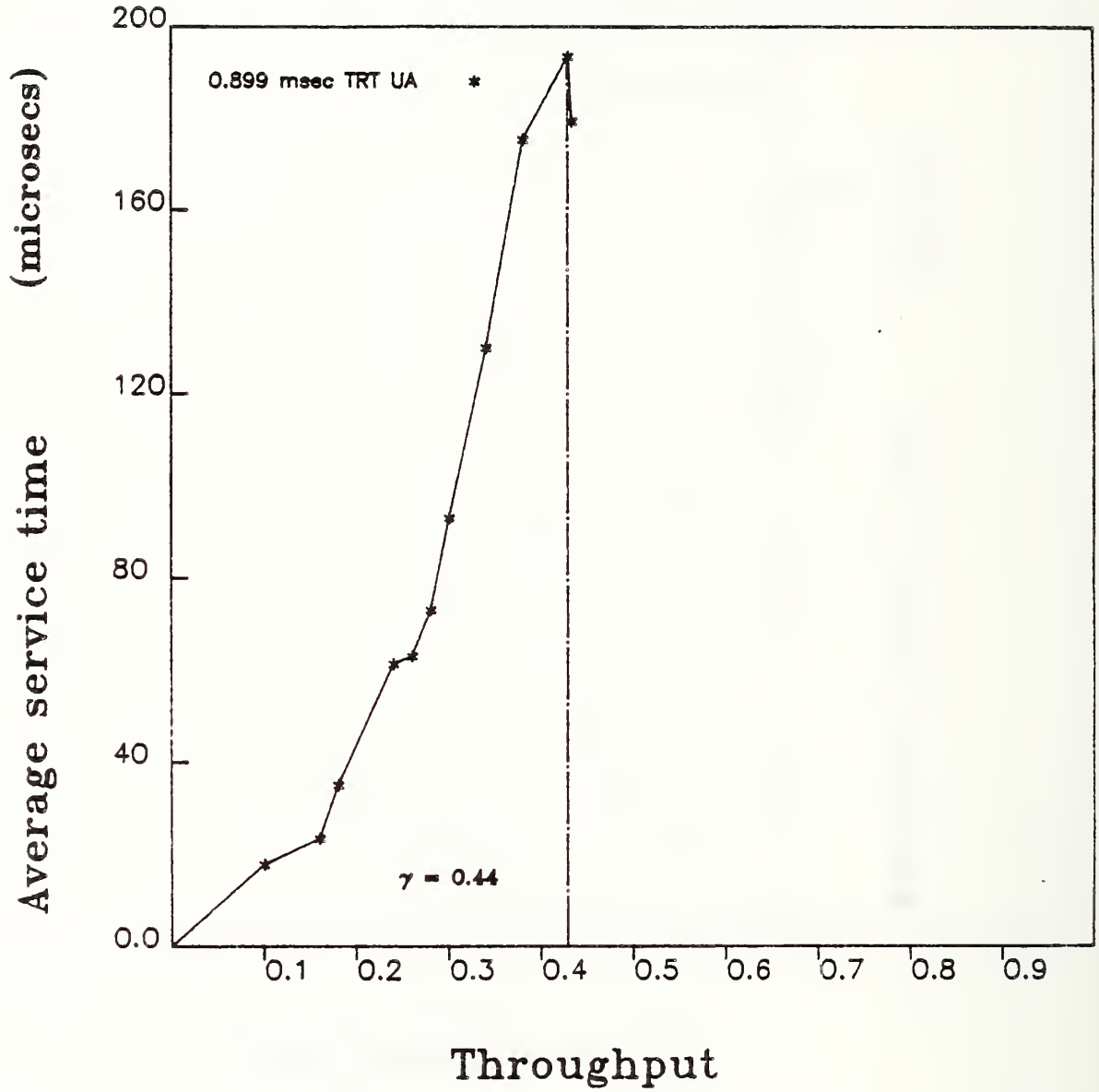


Figure 5
Average service time for Urgent_Asynchronous class

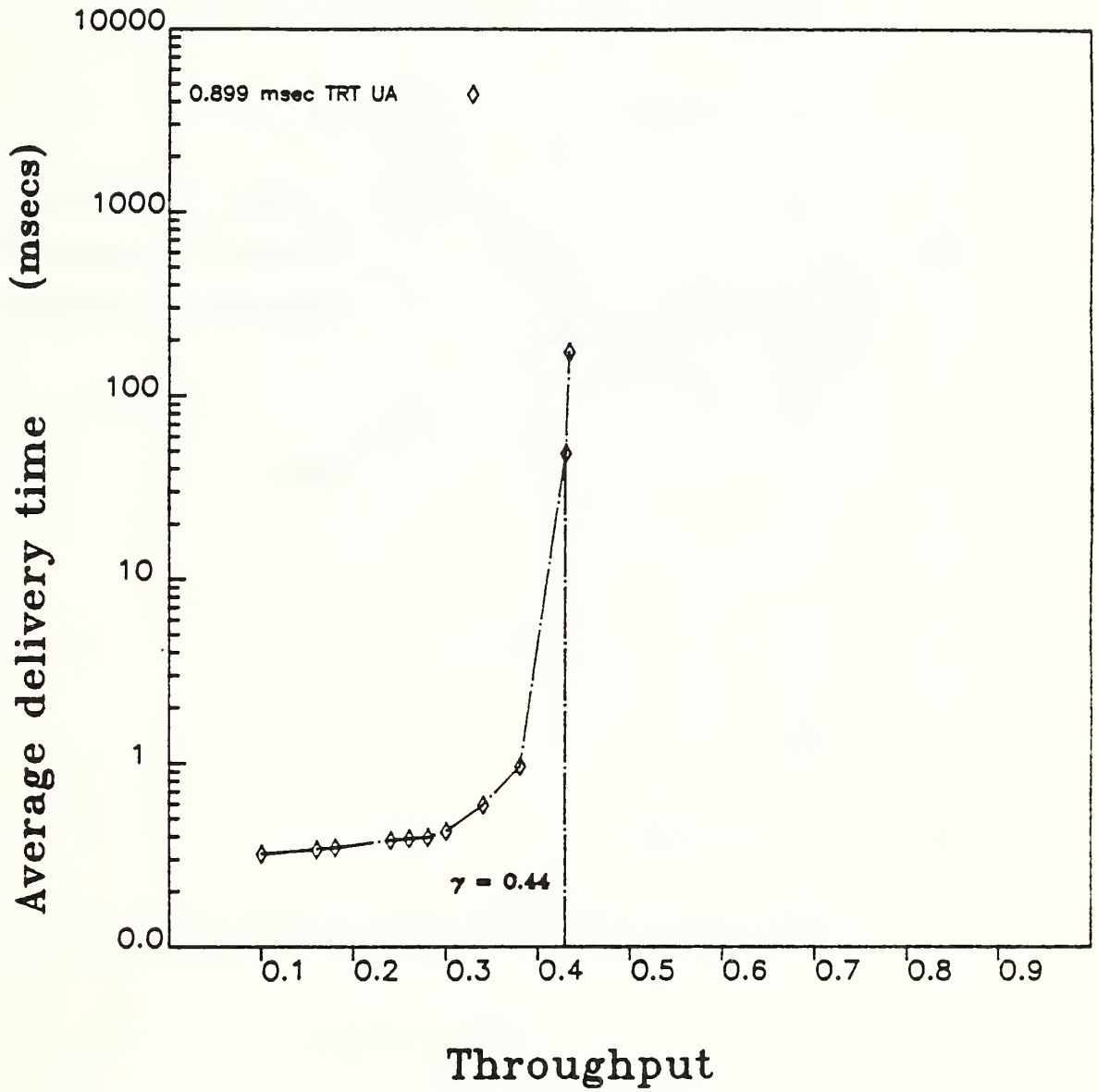


Figure 6
Average delivery time for Urgent_Asynchronous class

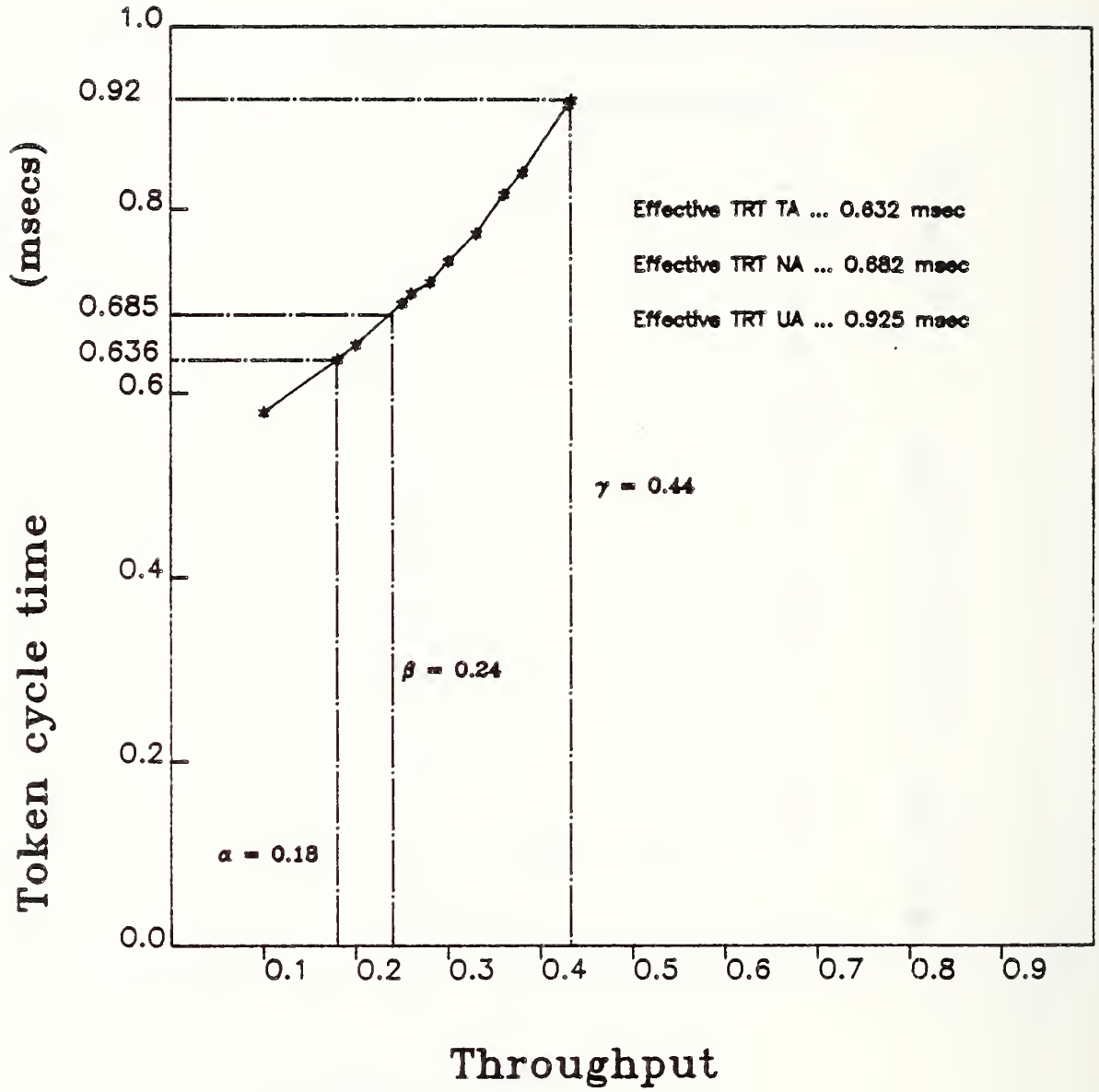


Figure 7
Token cycle time

1.3. Concluding Remarks

A static token passing bus has been studied and it has been shown that TRT settings play a crucial role in deciding the service time at the lower three access_classes. Optimum service can be obtained at an access_class until network throughput rises to the point that token cycle time equals the Target_Rotation_Time of that access_class. The analytic model developed in this study can be used to determine TRTs for any desired throughput range for a network configuration.

In this study all participating stations were assumed to be active and with identical message interarrival rates at all stations at an access_class. However the designer of a local area network has to take into consideration the communication traffic at each station at each access_class as it may be specific to each station and each access_class. Also the traffic from each station can be very bursty in nature. Nevertheless this study helped us gain a better understanding of the impact of certain parameters critical to the implementation of the priority feature.

Bibliography

[IEEE 1985]
Token-Passing Bus Access Method and Physical Layer Specifications, IEEE Standard 802.4-1985.

[Gorur 86]
Mangala Gorur and Alfred C. Weaver, *Priorities and Dynamic Ring Membership in an 802.4 Network*, Master's thesis, DAMACS Report No. 86-19, University of Virginia, Department of Computer Science, July 86.

[Summers 85]
Catherine F. Summers and Alfred C. Weaver, *Performance Studies of the IEEE Token Bus*, Master's thesis, DAMACS Report No. 85-06, University of Virginia, Department of Computer Science, May 1985.

[Weaver 85]
Catherine F. Summers and Alfred C. Weaver, *The IEEE 802.4 Token Bus - An Introduction and Performance Analysis*, Computer Science Report No. TR-85-19, University of Virginia, August 26, 1985.

PERFORMANCE MODELING OF IEEE 802.4 TOKEN BUS*

by

Anura P. Jayasumana, Member, IEEE
Department of Electrical Engineering
Colorado State University
Fort Collins, CO 80523

and

P. David Fisher, Senior Member, IEEE
Department of Electrical Engineering and Systems Science
Michigan State University
East Lansing, MI 48824

ABSTRACT

The priority mechanism of IEEE 802.4 token bus allocates the channel bandwidth among different priority classes of messages by means of a set of timers at each station. An analytical model is presented, which relates the throughput of each priority class of messages to the traffic intensities of different classes, the target token-rotation times and the high-priority token-hold time. Network is assumed to be symmetric with respect to the parameters and traffic distribution among nodes. Simulation results are used to evaluate the accuracy of the model. The model can accurately predict the behavior of token bus networks with a large number of nodes.

---

*This research was supported in part by NSF under Grant No. ECS8515824.

I. INTRODUCTION

The broadband token-passing bus scheme of IEEE 802.4 has emerged as the standard for factory Local-Area Networks due to several major reasons such as its deterministic nature, reliability at peak loads, and its support of a message based priority mechanism. The priority mechanism of the IEEE 802.4 token-bus standard allocates the channel bandwidth among different priority classes of messages by means of a set of timers at each station [1]. Typically, token-bus networks are used to connect a large number of devices together. Simulation of the IEEE 802.4 priority mechanism under such conditions requires a large amount of computer time and storage. Therefore, it is desirable to develop analytical models to predict different characteristics of this scheme. The complexity associated with the token-passing bus priority scheme makes it extremely difficult to model it using an analytical approach. But once such a model is developed, it usually offers results for a variety of load conditions and parameters at a low cost. Furthermore, analytical models show qualitative relationships between parameters and variables more clearly than do models developed by other approaches. Analytical approaches have successfully been used to model the IEEE 802.4 token bus, only with a single class of messages [2,8,9,10]. These models have been used to investigate performance of bus networks with respect to variation of parameters such as offered load, message length, signal propagation delay, token size and token holding strategies. Analytic models have also been used to address starvation and stability issues related to the IEEE 802.4 priority scheme [6,7]. This paper presents an analytical model, which predicts the throughput characteristics of token-passing networks that implement the IEEE 802.4 priority mechanism.

The model is based on the law of large numbers, which can be stated as follows [5]: A large population of demands will present a total load to a system of resources, which is equal to the sum of the average requirements of each individual demand, and further this load is a highly predictable quantity. A model developed using this law is thus capable of predicting the performance of networks with a large number of nodes.

Next section reviews the priority option of IEEE 802.4 standard. In Section III we consider the behavior of the priority scheme under specific input conditions. Section IV describes the analytical model. Simulation results are used to verify the analytical model in Section V. Limitations of the model as well as further work currently underway in this area are outlined.

II. IEEE 802.4 Token-Passing Bus Access Scheme

In the token-passing access method, a token controls the right to access to the physical medium. When a station receives the token, it becomes temporary master of the network and has the right to transmit one or more messages subject to the constraints imposed by the priority scheme discussed below. If the node does not have any messages to send, or when its token-holding time expires, the token is passed to the next station. This creates a logical ring around which the token passes. Hence, in the steady state, there will be alternate data and token transfer phases. The logical ring maintenance responsibilities such as ring initialization and lost token recovery are distributed among all token using stations of the network.

The IEEE 802.4 standard [1] provides an optional priority mechanism. The priority of each frame is indicated when the LLC sublayer requests the MAC sublayer to send a data frame. The MAC sublayer offers four levels of priority

classes, called access classes. The access classes are named 0, 2, 4 and 6, with 6 corresponding to the highest priority and 0 to the lowest. We use the term 'class H' to denote the highest priority class, and class l to denote any of the lower three priority classes. There is a separate queue for frames in each class to wait pending transmission. Any station not having the priority option transmits every data frame with the highest priority value. The object of the priority scheme is to use the network bandwidth to transmit the low priority frames when there is sufficient bandwidth available. Each access class acts as a virtual substation in that the token is passed internally from the highest access class downward, through all access classes before being passed to successor.

The priority scheme works as follows: Any station, which receives the token, initiates transmission of frames of the highest priority class in a time not exceeding some maximum time called the high-priority token-hold time (HPTHT). This high-priority token-hold time prevents any single station from monopolizing the network. After sending the high-priority frames, it starts servicing the queue of the next access class. Each of the three lower access classes at a node is assigned a target token-rotation time TTRT(l). For these three access classes, the station measures the time it takes for the token to circulate around the logical ring. If the token returns to the queue in less than the target token-rotation time, the station is allowed to send frames of that particular access class until the target token-rotation time has expired. If the target token-rotation time has expired by the time the token returns, the station is not allowed to send frames of that access class. The fraction of bandwidth that will be allocated to various classes is controlled by the target token-rotation time of each access class. The responsibility of setting these values lies with the station management.

III. PERFORMANCE OF THE PRIORITY SCHEME

In this section, we consider the behavior of the priority scheme under several specific input conditions. First, several terms used in performance analysis of local-area networks are defined. In the description below, utilization U of a network refers to the fraction of time the network spends carrying data packets. Throughput of a network S, is defined as the total number of data bits received at the destination per second (expressed as a fraction of the bandwidth). As a data frame includes header information, preamble etc., in addition to the data itself, the throughput of a network is less than its utilization. The relationship between S and U is given by

$$S = \alpha U , \tag{1}$$

where

$$\alpha = \frac{T_m}{T_m + T_{oh}} . \tag{2}$$

T_{oh} is the length of overhead portion of the data frame. It includes such components as the preamble, destination and source address fields, frame check sequence, frame control field, and delimiters. The mean length of messages (data component) of each priority class is assumed to be T_m. The components of the utilization and throughput of class i are denoted by U_i and S_i respectively. Equation (1) also holds separately for each component of the throughput. The offered load (G) is defined as the number of data bits generated by all active stations per second, expressed as a fraction of channel bandwidth. The offered load of class i is denoted by G_i.

Consider a token bus network with N nodes. Each node consists of four queues, one for each priority class of messages. Frames arrive at these queues with independent Poisson distributions. The frame lengths are assumed to be exponentially distributed. Assume that the total traffic in a given class is equally distributed among the N nodes. The high-priority token-hold time at each station is set to a constant value HPTHT. Also assume that the target token-rotation time of class 1 at each station is set to a constant value TTRT(1).

The token-circulation time T_c is defined to be the time required for the token to circulate among the N stations, in the absence of any traffic in the network. Let T_t be the mean token-passing time between two stations. This includes the station delay, the head end delay, the transmitter and receiver modem delays, the token transmission time as well as the propagation time. Bit time, the time required for the transmission of a single bit, is used as the basic time unit. All other time intervals are expressed in bit times. For a network with N nodes,

$$T_c = N T_t . \tag{3}$$

The token-rotation time is the time elapsed from the instant an access class at a station receives the token till the next instant the same access class at the same station receives the token. Let TRT denote the mean token-rotation time averaged over all the queues. The utilization of the network is then given by,

$$U = \frac{[TRT - N T_t]}{TRT} . \tag{4}$$

From Equation (1), the throughput is given by,

$$S = \alpha \frac{[TRT - N\ T_t]}{TRT} . \qquad (5)$$

This assumes error free operation and neglects the overhead associated with ring maintenance functions such as nodes switching on to and off the logical ring, and lost token recovery. In this paper, we consider the steady state operation of the network. A static logical ring is considered, in which the number of stations remains unchanged. The overhead associated with logical ring maintenance functions is neglected, i.e., error free operation of the network is assumed.

LEMMA 1: When a queue of class H is heavily loaded, its mean token-holding time is given by $(HPTHT+T_m)$. When class H is heavily loaded, $(HPTHT+T_m)N$ time units per token rotation are used to transmit class H messages.

Proof: When a station receives the token, it loads the 'token hold timer' with HPTHT, the high-priority token-hold time. The station then transmits high-priority (class H) frames until either the token hold timer expires or the frames in the queue are exhausted. When a queue is heavily loaded, i.e., when it has more frames than it is allowed to transmit, the token hold time of a queue is limited by the first condition. In this case, the station attempts to transmit one more frame, provided the token hold timer has not expired. If the token hold timer expires during the transmission of a message, the node will still complete message transmission before it transfers control to the next access class. Thus the actual token hold time of class H exceeds HPTHT. The memoryless property of exponential message length distribution

ensures that the mean length of message portion left to be transmitted after token hold timer expires is also T_m, where T_m is the mean message length. Thus under very high loads, when the station tries to send as many class H messages as possible in a given cycle, the mean token-hold time for class H is given by (HPTHT+T_m). Due to the assumption that the traffic is symmetrically distributed among the queues, when class H is heavily loaded, each of the queues can be assumed to be heavily loaded. Therefore, each queue will hold the token (HPTHT+T_m) time units. Thus in each token rotation, (HPTHT+T_m)N time units will be spent transmitting class H messages. Note that this result is not applicable to the case where HPTHT is zero. In that case, the token-hold time, and hence the throughput of class H are zero.

It is assumed that the message length includes overhead such as address and preamble. However, in practice only the data portion of a message is exponentially distributed. This could cause the mean token-hold time to differ from the above value. However, when the length of overhead segment of the message is small compared to the data portion, this result is still valid.

When class H is heavily loaded, the time required by the token to circulate among the stations and to transmit high-priority messages is given by TH, where

$$TH = [\, HPTHT + T_m + T_t \,] \, N \, . \tag{6}$$

Therefore, if the target token-rotation time of a low priority access class is larger than this value, it is possible that the messages of the lower priority class will also be transmitted, even though class H is heavily loaded[3].

LEMMA 2: The throughput of class H is related to the mean token-rotation time as follows:

$$S_H = G_H \quad \text{if} \quad TRT < TH$$

$$S_H = \text{Min}\left[G_H, \frac{\alpha[HPTHT+T_m]N}{TRT}\right] \quad \text{if} \quad TRT \geq TH \tag{7}$$

Proof: When class H is heavily loaded, $(HPTHT+T_m)N$ time units per rotation are spent transmitting class H messages. In addition, NT_t time units per rotation are required for token transmission. Thus if TRT is less than $(HPTHT+T_m+T_t)N$, class H is not heavily loaded, i.e., message transmission of class H is not limited due to HPTHT. Thus all the messages of class H get transmitted, and hence the throughput is equal to the offered load. When the token-rotation time exceeds $(HPTHT+T_m+T_t)N$, up to $(HPTHT+T_m)N$ time units per rotation may be spent transmitting messages of class H. Therefore, the maximum throughput of class H is given by,

$$S_{H(max)} = \alpha \frac{[HPTHT+T_m]N}{TRT} . \tag{8}$$

This is the maximum throughput of class H, when the mean token-rotation time is TRT. Throughput of class H reaches this value only if the total load in class H exceeds this value. When the throughput of class H has reached the value given by Equation (8), we say that the throughput of class H is limited by its high-priority token-hold time. When the offered load of class H is less than $S_{H(max)}$, all the messages of class H get transmitted. Thus, the throughput of class H is given by Equation (7).

LEMMA 3: When all the network traffic belongs to class 1 (1 = 0, 2, or 4), and class 1 is heavily loaded, the mean token-rotation time is given by

$$TRT = \frac{N}{[N+1]}[TTRT(1)+T_m+T_t]. \tag{9}$$

Note: This result is valid only when TTRT(1) is greater than the token circulation time NT_t. When this condition is not satisfied, no messages of class 1 are transmitted.

Proof: A queue is heavily loaded if it has more messages than it is allowed to transmit. Let T_t be the mean token-passing time of a network with N nodes. Consider the activity of this network when class 1 is heavily loaded. For convenience, assume that no messages are transmitted in the initial rotation. The result however is valid for any other initial token rotation. When the queue of class 1 of station 1 receives the token, x time units are left in its token rotation timer, where

$$x = [TTRT(1) - N\ T_t] \tag{10}$$

Since the queue is heavily loaded, it will transmit class 1 messages until the token rotation timer expires. When it expires, the transmission of current frame is completed, before the token is passed to the next station. Due to the memoryless property of the exponential distribution, the mean length of the message portion left to be transmitted after the timer expires is also T_m. Hence, the first station holds the token on average for $(x+T_m)$ time units as shown in Table 1. During this rotation, no other station is allowed to send messages. During the next rotation, station 2 uses $(x+T_m)$ time units. This continues for n rotations, during each of which, only a single station is

allowed to send messages. The mean token-hold time of the station transmitting messages is $(x+T_m)$ time units. In the (N+1)st rotation, the priority scheme does not allow any messages to be transmitted. It can be seen that this (N+1) token rotation pattern repeats itself. Therefore, the mean token rotation time is given by

$$TRT = \frac{[\ (x+T_m+NT_t)N + N\ T_t]}{[N+1]}. \tag{11}$$

Substituting the value of x from Equation (10), we get the result of Equation (9).

For large networks, if TTRT(1) is large compared to the token passing time and the mean message transmission time, the mean token-rotation time when class 1 is heavily loaded can be approximated by

$$TRT = TTRT(1)\ . \tag{12}$$

To keep the explanation simple, we use this value in deriving the analytical model.

LEMMA 4: When the mean token-rotation time of a network is less than the target token-rotation time of class 1 (1 = 0, 2, or 4), the throughput of class 1 is equal to its offered load, i.e.,

$$S_1 = G_1 \qquad \text{when } TRT < TTRT(1). \tag{13}$$

Proof: When the mean token-rotation time is less than TTRT(1), the class 1 queues receive the token back within TTRT(1) time units with a high probability. So they are allowed to transmit the messages of class 1. If there

are queues with class 1 frames waiting to be transmitted, these queues will utilize the time left to transmit the frames, thus increasing the value of TRT. When the load in class 1 becomes heavy, the token-rotation time will approach TTRT(1) as given by Lemma 3.

LEMMA 5: Consider the case when traffic is present only in class i, i=0,2,4,6. Then the maximum mean token-rotation time is given by

$$\begin{aligned} TRT_{max} &= TTRT(i) && \text{for } i = 0,2,4, \\ &= [HPTHT+T_m+T_t]N && \text{for } i = 6. \end{aligned} \tag{14}$$

The maximum throughput of class i is given by

$$S_{i(max)} = \alpha \frac{[\, TRT_{(max)} - NT_t \,]}{TRT_{max}} . \tag{15}$$

Proof: Follows directly from Lemma 1 and Lemma 3 (Equation 12). From this lemma it is evident that the traffic in class 1 cannot drive the mean token-rotation time above the value TTRT(1). This is true even when there is traffic in other classes.

Define P_i (i = 0,2,4,6) as

$$\begin{aligned} P_i &= \alpha \frac{[\, TTRT(i) - N\, T_t]}{TTRT(i)} && i = 0,2,4 \\ P_i &= \alpha \frac{[\, TH - N\, T_t \,]}{TH} && i = 6 . \end{aligned} \tag{16}$$

P_i is the throughput of the network when its mean token-rotation time is equal to the target token-rotation time of class i, for the lower priority classes. P_6 (P_H) is the throughput of the network when the mean token-rotation time is equal to TH. From Lemma 5, it is seen that P_i is also the peak throughput of

the network when all the messages belong to class i. These descriptions are valid provided HPTHT is not zero and each TTRT is greater than the token circulation time T_c. A more complete definition for P_H will take care of this condition by setting P_H to zero when the value of HPTHT is zero. Similarly, P_1 can be defined to be zero, when TTRT(1) is less than T_c.

Lemma 4 specifies the throughput only when the mean token-rotation time is less than the target token-rotation time of a given class. When the token-rotation time is much larger than the target token-rotation time of a given class, the probability of these queues receiving the token before the expiration of corresponding timer approaches zero. Therefore, the throughput of that class becomes negligible, i.e.,

$$S_1 = 0 \qquad \text{when TRT} >> \text{TTRT(1)}. \tag{17}$$

Equations (13) and (17) specify the throughput of class 1 (1 = 0,2,4), when the mean token-rotation time is not in the vicinity of TTRT(1). Since these results were obtained using mean value analysis, they cannot accurately predict the throughput when the value of TRT is close to TTRT(1). In developing the analytical model, we make an approximation for the throughput of class 1 in this region. This approximation can best be illustrated by an example.

Example 1: Consider a network in which TTRT(0) is less than TTRT(2) by several packet times. Assume that initially the network load consists entirely of class 0 traffic, and that this traffic is handled by the network with a mean token- rotation time TRT, which is less than TTRT(0). Now consider what happens when the load in class 2 is gradually increased. Initially, the mean token- rotation time will increase to accommodate this new traffic in class 2.

As long as TRT does not exceed TTRT(0), all the load in class 0 is handled by the network. But when TRT exceeds TTRT(0), the throughput in class 0 begins to decrease and eventually goes to zero. In developing the model, we approximate the variation of the token-rotation time by that given in Figure 1. When the mean token-rotation time reaches TTRT(0), we assume that it remains at that value until the throughput of class 0 is completely captured by the load of class 2. The corresponding throughput characteristics are also shown in Figure 1. In the region b-c, the mean token-rotation time, and hence the total throughput are assumed to be constant. In practice however, a gradual increase of TRT occurs in this region, resulting in a higher throughput for class 0.

Let I, J and K be the three lower priority classes of messages ordered according to their target token-rotation times. The TTRT's are assumed to be greater than the token circulation time T_c. If this is not the case, then no messages of that class will be transmitted and hence its throughput is zero. The model described in this paper can be used to evaluate the throughput of each class only when the target token-rotation times differ by several packet times, i.e., when

$$NT_t < TTRT(I) < TTRT(J) < TTRT(K) . \tag{18}$$

The special case wnere two or more of these have approximately the same value is considered at the end of Section IV.

In general, when the load in classes J, K, (and H, when throughput is not limited by HPTHT), is increased, the token-rotation time increases to accommodate this load. When TRT reaches the value TTRT(I), it is assumed to remain at this value until the throughput of class I is captured by classes J,

K (and H). A further increase in offered load in classes J, K (and H, when throughput of class H is not limited by HPTHT) results in increase of TRT beyond TTRT(I). A similar approximation is made for the throughputs of classes J and K when TRT is in the vicinity of their target token-rotation times. To summarize, we approximate the throughput of class l (l = 0,2,4) by,

$$\begin{aligned} S_l &= G_l \quad \text{when TRT} < \text{TTRT}(l) \\ S_l &= 0 \quad \text{when TRT} > \text{TTRT}(l). \end{aligned} \tag{19}$$

When TRT = TTRT(l), the throughput depends on the loads in other classes and their target token-rotation times as follows:

$$\begin{aligned} S_K &= [P_K - S_H]^+ && \text{when TRT} = \text{TTRT}(K) \\ S_J &= [P_J - S_H - S_K]^+ && \text{when TRT} = \text{TTRT}(J) \\ S_I &= [P_J - S_H - S_K - S_J]^+ && \text{when TRT} = \text{TTRT}(I) \end{aligned} \tag{20}$$

where $[X]^+$ is defined by,

$$\begin{aligned} [X]^+ &= 0 \quad \text{if } X \leq 0 \\ &= X \quad \text{if } X > 0. \end{aligned} \tag{21}$$

In this section, the behavior of the IEEE 802.4 priority scheme was investigated under a constrained set of input load values. In Section IV, we use these results to develop a more general analytical model for the token-bus priority scheme.

IV. MODEL DESCRIPTION

An analytical model is presented, which relates the throughput of each priority class of messages to the traffic intensities of different classes, the target token-rotation times, and the high-priority token-hold time. The model is based on the following assumptions: The traffic is symmetrically distributed among the stations. Messages arrive at nodes with independent Poisson distributions. Message lengths are assumed to be exponentially distributed. The target token-rotation time of a given class is set to the same constant value at each of the stations. The high-priority token-hold time at each station is also set to a constant value.

The behavior of the IEEE 802.4 priority scheme was considered in Section III, under a limited set of input load conditions. When there are messages in more than one class, the throughput characteristics depend on the target token rotation times, the high-priority token-hold time, and their relationship to each other. A general model has to consider the following cases separately:

(1) $TTRT(I) < TTRT(J) < TTRT(K) < TH$,

(2) $TTRT(I) < TTRT(J) < TH < TTRT(K)$,

(3) $TTRT(I) < TH < TTRT(J) < TTRT(K)$, and

(4) $TH < TTRT(I) < TTRT(J) < TTRT(K)$.

The model described below can be used to evaluate the throughput of each class only when TTRT's and TH differ from each other by several packet times. The special case where two or more of them have approximately the same value is considered later. In this paper, we present an analytical model for Case 1, i.e., when

$$TTRT(I) < TTRT(J) < TTRT(K) < TH . \quad (22)$$

Similar models have been developed for other cases [4]. Consider the mean token-rotation time of the network, when traffic is present in all the classes. The maximum possible value of the token-rotation time is TH, and this occurs when class H is heavily loaded. The condition of Equation (22) ensures that the throughput of class H is not limited by the high-priority token-hold time, except when the mean token rotation time (TRT) is equal to TH. In order to derive the throughput characteristics, we have to consider the different values for the mean token-rotation time TRT.

(i) TRT = TH

When TRT is equal to TH, the mean token-rotation time is greater than any of the target token-rotation times, i.e.,

$$TRT > TTRT(1) \qquad 1 = 0,2,4 \ . \tag{23}$$

Using Equation (19), we get

$$S_I = S_J = S_K = 0 \ . \tag{24}$$

This condition would occur only if there is sufficient load in class H to sustain the mean token-rotation time at TH, i.e., only if

$$G_H \geq P_H \ . \tag{25}$$

From Equations (5) and (16), the throughput of the network is given by P_H. Since the throughputs of the lower priority classes are zero, S_H is given by,

$$S_H = P_H \ . \tag{26}$$

(ii) $TTRT(K) \leq TRT < TH$

This condition occurs when the total load in classes K and H exceeds P_K. P_K is the peak throughput of the network when TRT is equal to TTRT(K), and is given by Equation (16). Since TRT is less than TH, from Equation (7), the throughput of class H is given by,

$$S_H = G_H . \tag{27}$$

From the inequality of Equation (22), the mean token-rotation time is greater than the target token-rotation times of classes I and J. Using Equation (19), the throughputs of classes I and J are given by

$$S_I = S_J = 0 . \tag{28}$$

According to the approximation for TRT described in Section III, when TRT is greater than TTRT(K), the throughput of class K approaches zero. When TRT is equal to TTRT(K), the throughput of class K is captured by class H. Using Equation (20), the throughput of class K is given by,

$$S_K = [\, P_K - S_H \,]^+ . \tag{29}$$

This case would occur only if there is sufficient traffic in classes H and K to drive the token-rotation time to a value above TTRT(K), i.e., when

$$G_H + G_K \geq P_K . \tag{30}$$

When using the model to calculate throughput characteristics, we first check to see whether TRT is equal to TH. If this condition is not satisfied, then we check to see whether TRT lies between TTRT(K) and TH. Therefore, we need

not include an upper bound in the above equation.

(iii) $TTRT(J) \leq TRT < TTRT(K)$

From Equation (22), the mean token-rotation time TRT is greater than TTRT(I). Therefore, using Equation (19), we have

$$S_I = 0 . \tag{31}$$

Since TRT is less than TH, the throughput of class H is not limited by HPTHT. Therefore, from Equation (7), the throughput of class H is given by

$$S_H = G_H . \tag{32}$$

We also have the condition that TRT is less than TTRT(K). Therefore, from Equation (19),

$$S_K = G_K . \tag{33}$$

The throughput of class J is thus given by,

$$S_J = [P_J - S_H - S_K]^+ \tag{34}$$

This condition can occur only if there is sufficient traffic in classes J, K and H to drive the mean token-rotation time above TTRT(J), i.e., when

$$G_H + G_K + G_J \geq P_J . \tag{35}$$

(iv) $TTRT(I) \leq TRT < TTRT(J)$

This condition occurs when none of the first three conditions occur, but there is sufficient traffic in classes I, J, K and H to drive the token

rotation time to a value higher than TTRT(I), i.e.,

$$G_H + G_K + G_J + G_I \geq P_I . \tag{36}$$

The throughputs for this case can also be obtained in a similar manner:

$$\begin{aligned} S_H &= G_H \\ S_K &= G_K \\ S_J &= G_J \\ S_I &= [\, P_I - S_H - S_K - S_J \,]^+ . \end{aligned} \tag{37}$$

(v) TRT < TTRT(I)

Since the mean token-rotation time is less than all the target token-rotation times and TH, all the messages offered to the network are transmitted. Therefore, the throughputs are given by,

$$\begin{aligned} S_H &= G_H \\ S_K &= G_K \\ S_J &= G_J \\ S_I &= G_I . \end{aligned} \tag{38}$$

The analytical model for this case, i.e., when the inequality of Equation (22) is satisfied, is summarized in Figure 2. This model however can be further simplified as follows. From Equations (16) and (22), we get the following relationship for P's:

$$P_I < P_J < P_K < P_H . \tag{39}$$

This property reduces the model given in Figure 2 into the following set of equations:

$$
\begin{aligned}
S_H &= \text{Min} (G_H , P_H) \\
S_K &= \text{Min} (G_K , [P_K - S_H]^+) \\
S_J &= \text{Min} (G_J , [P_J - S_H - S_K]^+) \\
S_I &= \text{Min} (G_I , [P_I - S_H - S_K - S_J]^+) \ . \qquad (40)
\end{aligned}
$$

In deriving the above model, it was assumed that the target token-rotation times of different classes differ at least by few packet times. The reason for this is the fact that we look at the average performance of the priority scheme. If the values of TTRT's are fairly close to each other, the distribution of the token-rotation time has to be taken in to account in order to evaluate the throughput. For example, consider the case where,

$$\text{TTRT(0)} = \text{TTRT(2)} = \text{TTRT(4)}.$$

Here, although the three target token-rotation times are identical, at a given node the access classes are served in the order of their priorities. This results in a more favorable treatment to a higher priority class, compared to a lower class. In this case, we need to look at the actual distribution of TRT in order to find the throughput of each class. However, the model can still be used to predict the total throughput of the classes for which the TTRT's are approximately the same. For example, consider the case where,

$$\text{TTRT(0)} = \text{TTRT(2)} < \text{TTRT(4)}.$$

In this case, the model can be used to evaluate the individual throughputs of classes 4, 6, and the sum of the throughputs of classes 0 and 2.

A model similar to the one shown in Figure 2 can be obtained for cases 2,3 and 4 as well. However, in these cases, the throughput of class H may become limited by the high-priority token-hold time even before the mean token-rotation time reaches its peak value [4].

V. MODEL VERIFICATION

In this section, we use performance characteristics of the IEEE 802.4 priority scheme, obtained via simulations, to verify the analytical model. A brief description of the simulator is provided in [3]. The network parameters used are shown in Table 2. The station delay in the table refers to the minimum time interval required between the end of reception of a frame by a station and beginning of a message transmission by the same station. The traffic is assumed to be symmetrically distributed among all the queues, with packet arrivals following a Poisson process. The nodes are distributed randomly along the bus. The mean token-passing time is the sum of mean propagation time, transmitter modem delay, head end delay, receiver modem delay, and the station delay. Simulations were carried out until the performance results appeared to have reached steady values. The length of a typical simulation run was about 10s. Below, we compare the results from the analytical and simulation models.

Three sets of throughput vs. offered load characteristics are used to verify the model. In each case, the total offered load is changed, while keeping the percentage of total load in each class constant. The fraction of the total offered load in each class is shown in Table 3. In Set 2 for example, each class generates 25% of the network load, i.e., the network traffic is symmetrically distributed among the four classes. The throughput vs. offered load characteristics corresponding to Sets 1, 2 and 3 are shown in

Figures 3, 4 and 5 respectively. It is seen that the values predicted by the analytical model agree very closely with those obtained using simulations. When the throughput of a class is decreasing with the offered load however, the model tends to under estimate the throughput of that class. This can be attributed to an approximation that was used in developing the model. The mean token-rotation time in this case was assumed to remain at TTRT(1), until other eligible classes capture its throughput. As shown in Figure 1, this however, is different from what happens in the actual case. Means to overcome this limitation are currently under investigation.

VI. CONCLUSION

In this paper, we investigated the performance of the priority option of IEEE 802.4 token-passing channel-access protocol. First we considered the behavior of the scheme under constrained input load conditions. Next these results were used to develop an analytical model for the token-passing priority scheme when

$$[\ HPTHT + T_m + T_t\ ] \ > \ TTRT(1), \quad \text{for all } 1,\ 1 = 0, 2, 4\ ,$$

where, HPTHT is the high-priority token-hold time, TTRT(1) the target token-rotation time of class 1, T_m the mean message length, and T_t the mean token-passing time. N is the number of stations in the network. A symmetric network is assumed, in which the traffic of each class is symmetrically distributed among the nodes, and each node has the same set of timer values. The results given in Section III can be used to obtain throughput characteristics for any other relationship between TTRT's and HPTHT [4]. In case where two or more classes have the same target token-rotation time, the model is capable of evaluating the total throughput of these classes rather than the individual throughput of each class. The model uses mean values of network variables to predict the behavior of the network. Therefore, its accuracy depends on the

number of stations in the network. Simulations show this model to be very accurate even for networks with as few as 10 nodes [4].

Simulation of large networks, to obtain the bandwidth allocated by the priority scheme to different access classes, requires a significant amount of computer resources. Models like the one described here, can be used to evaluate such characteristics accurately, with a minimal amount of computations. Although the model is based on several approximations, it provides very accurate values for throughputs of different classes. This model can be used to determine the timer values for IEEE 802.4 scheme, to meet throughput requirements of different classes of traffic present in the network.

The model is based on the assumption that all the nodes in the network have the same high-priority token-hold time, and the same target token rotation time for each access class. This however, may not be the case in many practical networks. Extension of the model to evaluate such networks is presently under investigation. Further investigation is also necessary to model the case, where traffic is not symmetrically distributed among the nodes. The model presented here does not address the issue of predicting the delays of different classes of messages. However, it provides a starting point toward developing such models.

REFERENCES

1. ANSI/IEEE std. 802.4 - 1985: Token-Passing Bus Access Method and Physical Layer Specifications, IEEE, 1985.

2. Cherukuri R., Lin L. and Louis L., "Evaluation of token-Passing Schemes in LANs," Proc. Computer Networking Symposium, 1982.

3. Jayasumana A. P., "Performance Analysis of a Token Bus Priority Scheme," To be presented at IEEE INFOCOM, March, 1987.

4. Jayasumana A. P., "A Performance Model for the IEEE 802.4 Token Bus Networks," Technical Report (Under Preparation), Department of Electrical Engineering, Colorado State University.

5. Kleinrock L., "A Decade of Network Development," Journal of Telecommunication Networks, Comp.Sci.Press, Spring 1982.

6. Nakassis A., "On the Stability of a Token Passing Network," Proc. Workshop on Analytic and Simulation Modeling of IEEE 802.4 Token Bus LAN, NBS, April, 1985.

7. Nakassis A., "Token Passing Networks and Starvation Issues," Proc. Workshop on Analytic and Simulation Modeling of IEEE 802.4 Token Bus LAN, NBS, April, 1985.

8. Sachs S. R., Kan K. and Silvester J. A., "Performance Analysis of a Token-Bus Protocol and Comparison with other LAN Protocols," Proc. 10th Conf. on Local Computer Networks, 1985.

9. Ulug, M.E., "Comparison of Token Holding Strategies for a Static Token Passing Bus, " Proc. Computer Networking Symposium, 1984.

10. Ulug, M.E., "Calculation of Waiting Times for a Dynamic Token Passing Bus, " Proc. Computer Networking Symposium, 1984.

Table 1. Mean token-holding time of class 1 queues when class 1 is heavily loaded.

	Station					
Rotation	1	2	3	4	...	N
0	0	0	0	0	...	0
1	$x+T_m$	0	0	0	...	0
2	0	$x+T_m$	0	0	...	0
3	0	0	$x+T_m$	0	...	0
.	.	.	.		...	.
.	.	.	.	.	...	.
.	.	.	.	.	...	.
.	.	.	.	.	...	.
N	0	0	0	0		$x+T_m$
N+1	0	0	0	0		0
N+2	$x+T_m$	0	0	0	...	0
.	.	.		.	...	.
.	.	.	.	.	...	.

Table 2. Network parameters used for the performance analysis.

Number of stations N		50	
Mean data frame length	T_m	1024	bits
Length of token		184	bits
Overhead per message	T_{oh}	184	bits
Bandwidth of the bus		10	Mbps
Length of the bus		1.0	km
One way propagation delay		5.0	μs
Head end delay		7.0	μs
Rx modem delay		3.0	μs
Tx modem delay		0.5	μs
Station delay		50.0	μs
High-priority token-hold time		102.4	μs
Target token-rotation times			
	Class 4	9000	μs
	Class 2	8000	μs
	Class 0	7000	μs
Mean token-passing time	T_t	83.5	μs
$(HPTHT + T_m + T_t)N$		14415.0	μs

Table 3. Percentage of offered load in class i, i = 0, 2, 4 and 6.

Set #	Class 0	Class 2	Class 4	Class 6
1	10	20	30	40
2	25	25	25	25
3	40	30	20	10

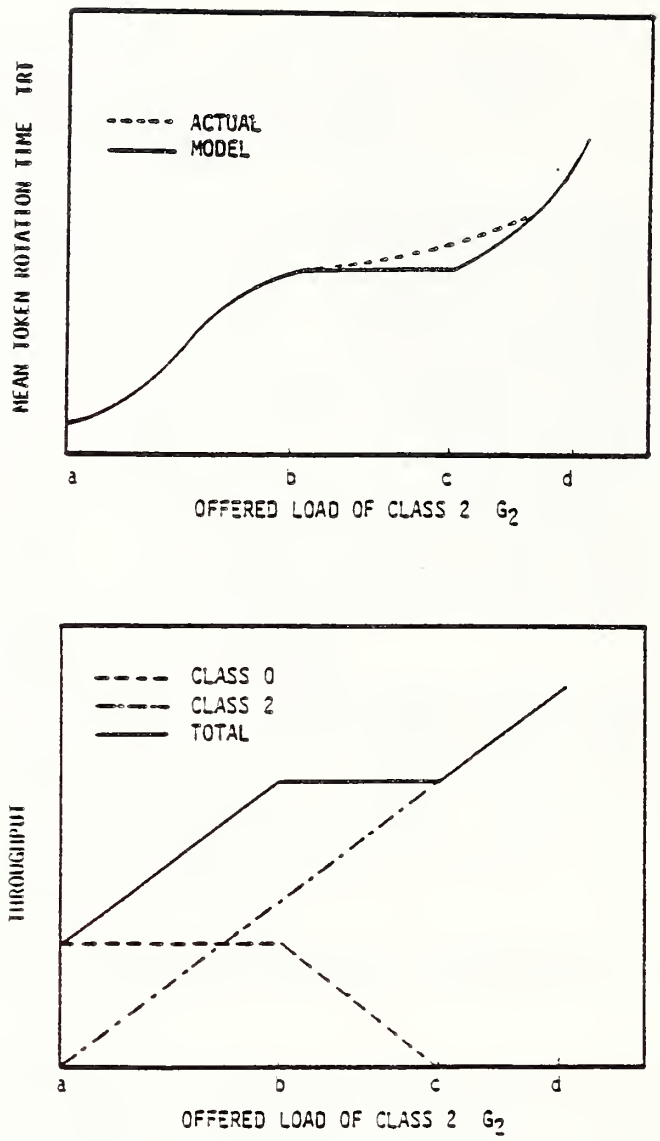


Figure 1. Variation of mean token-rotation time [TRT] and the throughput characteristics for Example 1.

```
C  ANALYTICAL MODEL FOR THE CASE
C  WHEN TTRT(I) < TTRT(J) < TTRT(K) < TH
C
      IF (G(H).GE.P(H)) THEN
C TRT = TH
      S(H) = P(H)
      S(K) = 0
      S(J) = 0
      S(I) = 0
C
      ELSE IF ( (G(H)+G(K)) .GE. P(K) ) THEN
C TH > TRT ≥ TTRT(K)
      S(H)  = G(H)
      S(K) = PLUS(P(K)-S(H))
      S(J) = 0
      S(I) = 0
C
      ELSE IF ( (G(H)+G(K)+G(J)) .GE. P(J) ) THEN
C TTRT(K) > TRT ≥ TTRT(J)
      S(H) = G(H)
      S(K) = G(K)
      S(J) = PLUS(P(J)-S(H)-S(K))
      S(I) = 0
C
      ELSE IF ( (G(H)+G(K)+G(J)+G(I)) .GE. P(I) ) THEN
C TTRT(J) > TRT ≥ TTRT(I)
      S(H) = G(H)
      S(K) = G(K)
      S(J) = G(J)
      S(I) = PLUS(P(I)-S(H)-S(K)-S(J))
C
      ELSE
C TTRT(I) ≥ TRT
      S(H) = G(H)
      S(K) = G(K)
      S(J) = G(J)
      S(I) = G(I)
      ENDIF
C
```

Figure 2. Analytical model for token-bus priority scheme when TTRT(I) < TTRT(J) < TTRT(K) < TH .

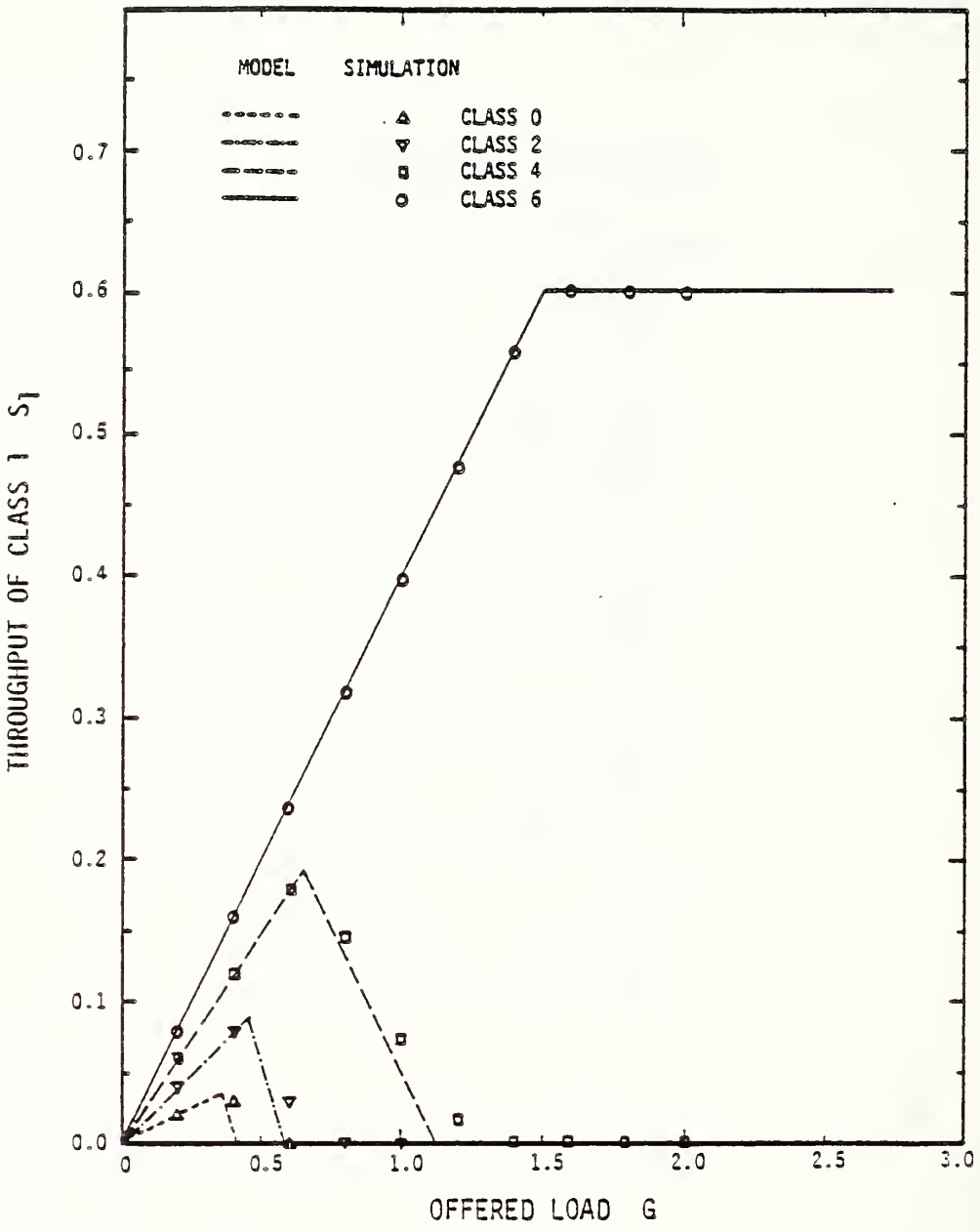


Figure 3. Throughput vs. offered load characteristics corresponding to load distribution Set 1.

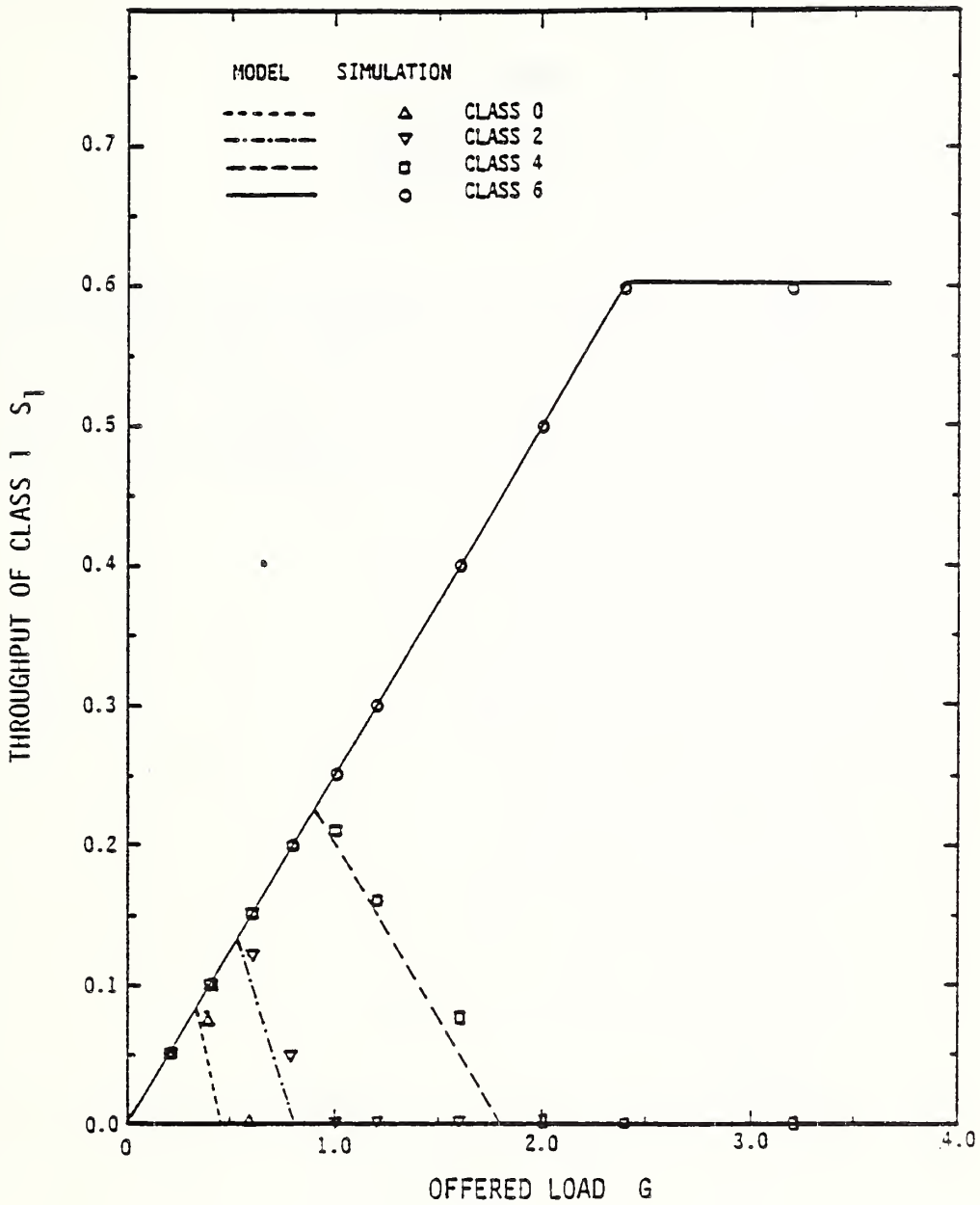


Figure 4. Throughput vs. offered load characteristics corresponding to load distribution Set 2.

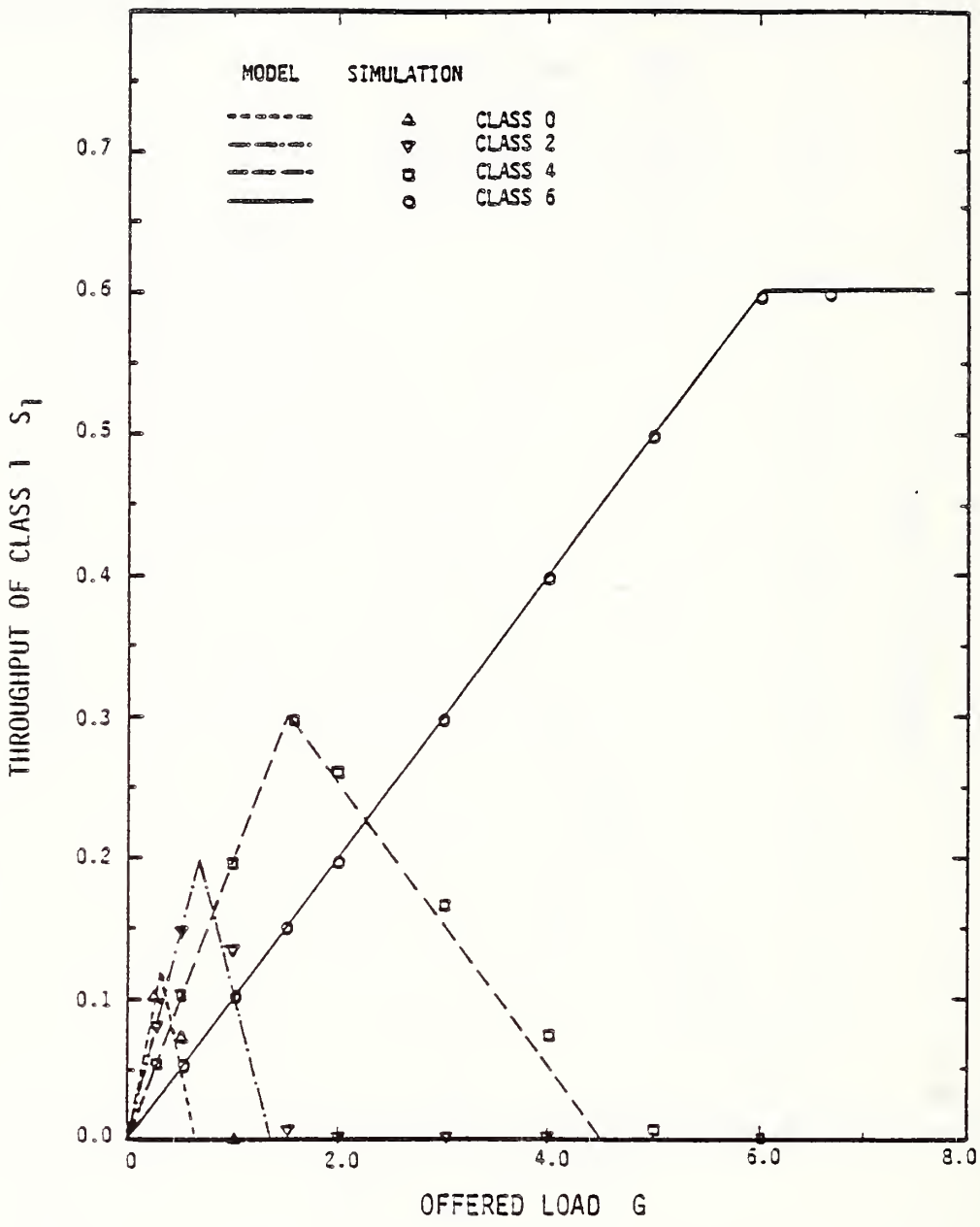


Figure 5. Throughput vs. offered load characteristics corresponding to load distribution Set 3.

Are Priorities Useful in an 802.5 Token Ring?

Jeffery H. Peden
Alfred C. Weaver
Department of Computer Science
University of Virginia
Charlottesville, Virginia 22903

The IEEE 802.5 token ring defines eight message priorities. The intent is that "high priority" messages should be delivered prior to "low priority" messages. Through a series of simulations we show that this expected behavior only occurs under a unique set of circumstances: when there are very few network stations, very short data packets, very short token hold times, and very high network load.

In the general case, we found that priorities did not markedly influence the delivery time of prioritized messages. Use of the priority system generally resulted in more overhead and longer average message delays than when all messages were carried as a single priority.

Are Priorities Useful in an 802.5 Token Ring?

1. Introduction

The 802.5 token ring protocol defines eight message priorities. These priorities are intended to ensure that higher priority messages receive better service than lower priority messages (i.e., are delivered with less delay). In this paper we examine the usefulness of the priority scheme under varying circumstances.

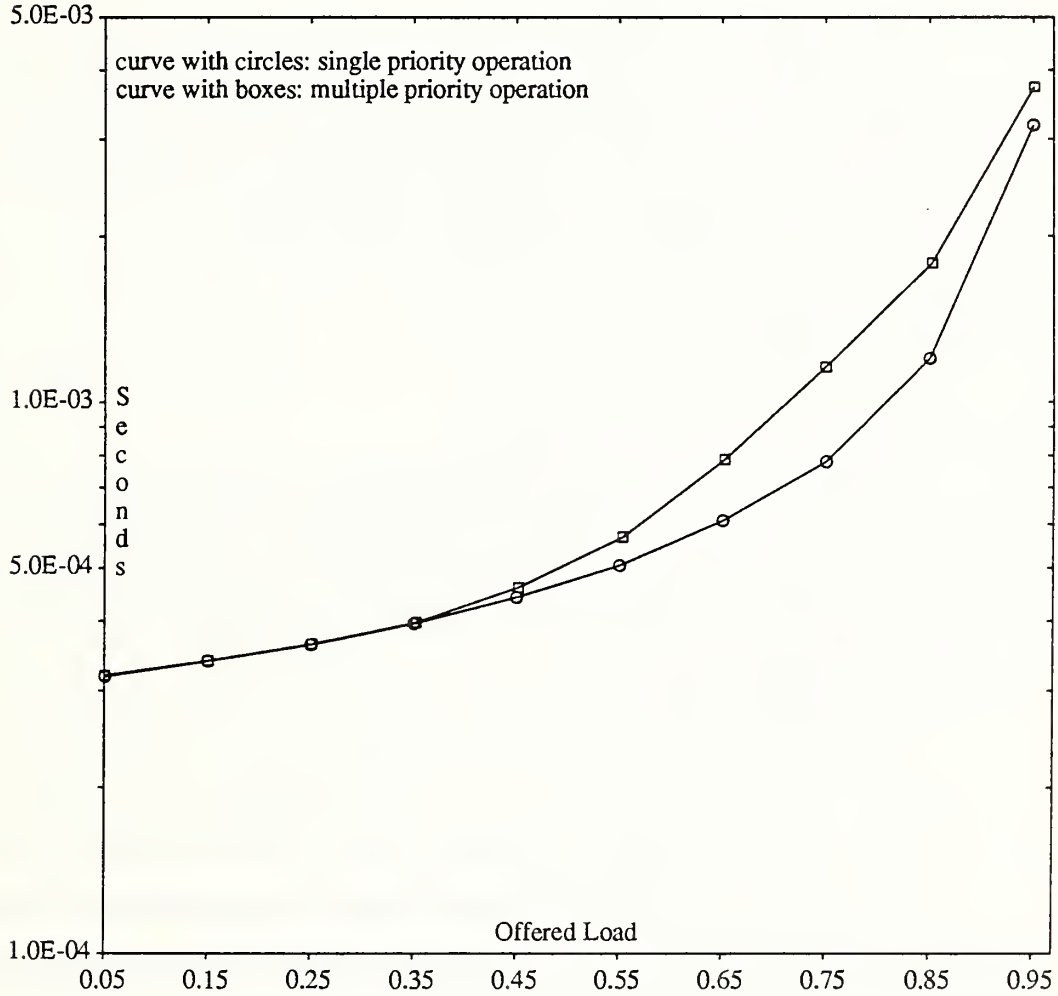
The priority scheme operates as follows: Every message is assigned a priority between 0 (lowest) and 7 (highest). When a station receives a token (the special bit sequence used to confer the right of transmission on a station) the station transmits any enqueued messages that are at or above the priority of the token, higher priority messages first. Enqueued messages that have a priority less than that of the token must wait until a lower priority token is received by the station.

As messages are repeated by non-transmitting stations, these stations may reserve the priority of the token by setting the reservation bits in the packet header so that they equal the highest priority message that they have waiting (however, the value of the token reservation may not be lowered by a station). When a station has completed its transmission, it retransmits the token at the greater of the value of the current token priority and the reservation. If a station raises the priority of the token, it is responsible for eventually lowering it to its previous value.

2. General Performance

In order to establish baseline performance for the token ring, a reference configuration was established consisting of 40 active stations, a station repeater latency of 1 bit time, a data rate of 1 Mbps, constant packet lengths of 256 bits (including framing), and an exponential arrival distribution. Studies were then made of network performance in both the single priority case and the multiple priority case (in which network load was divided evenly among the eight priorities).

Figure 1 shows the total packet delay for both the single and multiple priority modes of operation. Note the single curve for the multiple priority delay. The actual delay values shown in Figure 2a clearly



Configuration:

40 Stations
256 Bit Packets including framing
1 Bit Time Station Latency
Exponential Arrivals
1Mbps Medium

Figure 1 — Service Delay

indicate that there is no significant or consistent difference between the delay experienced by a high priority packet and that of a low priority packet. Because of this, the delay for the multiple priority case is given as the averaged delay of all eight priorities.

Also note that the delay for the multiple priority case is greater in the medium and upper load ranges than the delay for the single priority case. The actual delay values for the single priority case are shown in Figure 2b. The increase in overall delay for multiple priority operation is due to the added overhead incurred by the protocol when handling more than one priority.

3. Effects of Parameter Variation

3.1. Varying the Number of Offered Priorities

To ensure that the observed delay using multiple priorities was not an artifact of the chosen parameters, simulations were run in which the number of message priorities was varied from two to seven, and the same effect was seen. We therefore reduced the number of message priorities to only two, and varied the load distribution between them.

Simulations were run such that the fraction of load given to the low and high priorities was 0.1 and 0.9 respectively, and then varied in increments of 0.1 until the distribution of load between low and high priorities was reversed to 0.9 and 0.1 respectively. The results of these simulations indicated that the number of priorities offered to the ring, as well as the relative fraction of bandwidth assigned to each priority, had no significant impact on the delay experienced by packets. We therefore concluded that the effect was independent of the relative loading of priorities.

3.2. Varying the Number of Ring Stations and Packet Sizes

It was shown in [PEDE87] that for any particular offered load, achieved throughput for the token ring decreases as packet sizes become shorter, and that this loss of throughput is exaggerated when the number of ring stations is small. It was observed that the effects of multiple priority operation only had significant impact when the number of ring stations was less than 40. It was also shown that delay varied in inverse proportion to packet size. We therefore dealt with only those cases in which the number of sta-

tions was 40 or less, and where packets were short (on the order of 20 bytes of data).

4. Priority Effects

In this section we present and discuss the actual effects that the priority operation has on delay. Figure 3 shows that even for only 10 ring stations and packet sizes of 256 bits, there is minimal difference between the delays experienced by high and low priority packets. Figures 4 and 5 show the priority delays for ring configurations of 5 and 3 stations respectively, again with 256 bit packets. Note that even for 3 stations, where the difference between delays for different priorities is the highest, it is less than a factor of two from highest to lowest, and this does not occur until an offered load of 95% is reached (a loading difficult to achieve with only 3 stations).

To get the maximum effect from the operation of the priorities, simulations were run in which the ring was significantly overloaded for several seconds. Figure 6 shows the results observed. Note that service shutoff does not occur for the lowest priority until an offered load of over 99% is reached. Also note that the priority operation is very well behaved, in that service shutoff occurs in order of increasing priority, and that higher priorities continue to receive guaranteed delay until their service shutoff.

4.1. Effect of the Token Hold Time

The amount of time that a station may transmit is governed by the token holding timer in that station. The default value for this timer is given in the standard as 10 ms [IEEE85]. Up to this point, the effects of the priority scheme have been studied with the ring operating under the condition of the token holding timer being set to the default value. (This is a reasonable assumption to make, as it is anticipated that few users of this network will make changes in the default values of the various network parameters defined in the standard.)

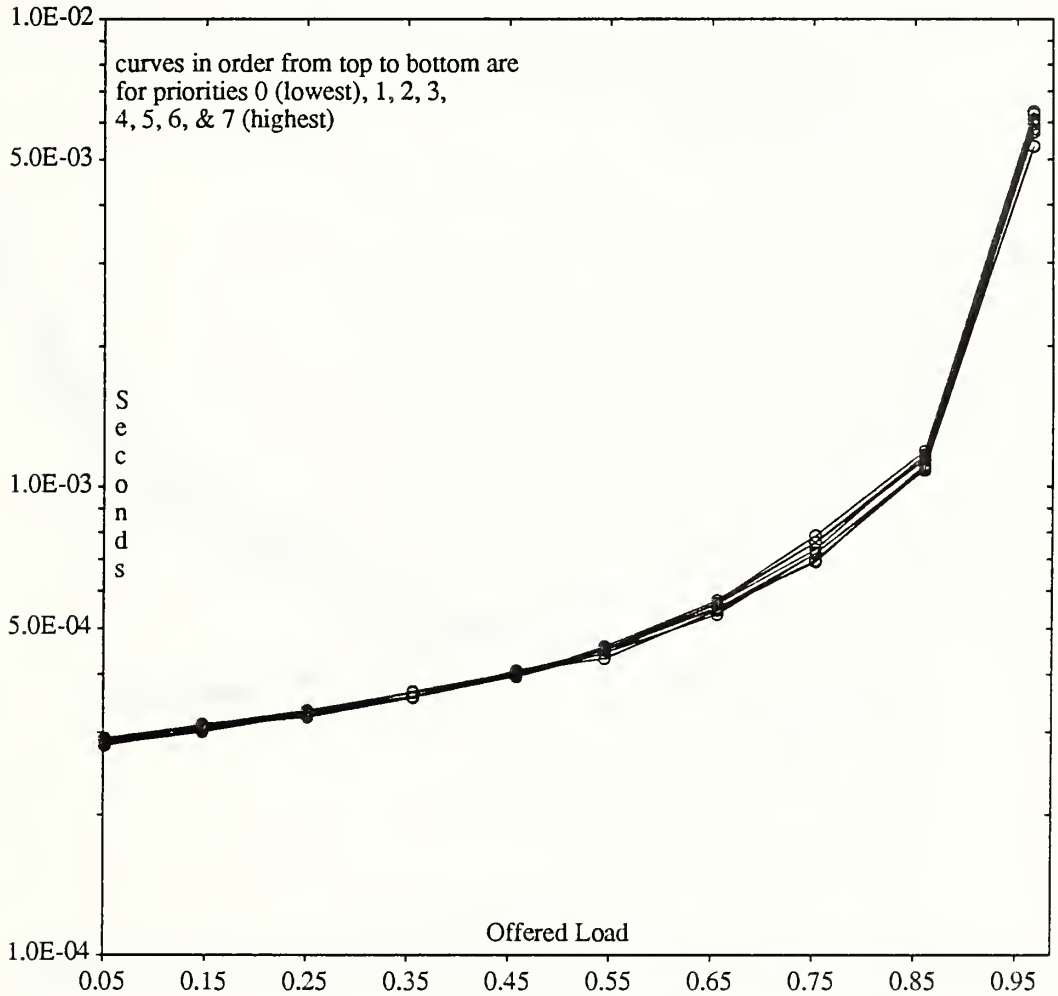
It was shown in [PEDE87] that the default setting of the token holding timer usually results in exhaustive service of the message queues. Therefore, simulations were run in which the token holding timer was set to less than the default value, in order to check the effects of the priority scheme under the condition of non-exhaustive service. Again it was seen that as the number of ring stations increases to 40 and above, the effects of the priority operation rapidly become insignificant, even when service is limited

Offered Load	Delay by Priority							
	0	1	2	3	4	5	6	7
0.05	0.320 ms	0.320 ms	0.318 ms	0.320 ms	0.318 ms	0.319 ms	0.321 ms	0.319 ms
0.15	0.342 ms	0.341 ms	0.339 ms	0.339 ms	0.340 ms	0.339 ms	0.339 ms	0.337 ms
0.25	0.364 ms	0.362 ms	0.364 ms	0.363 ms	0.365 ms	0.364 ms	0.365 ms	0.365 ms
0.35	0.403 ms	0.398 ms	0.398 ms	0.397 ms	0.396 ms	0.399 ms	0.396 ms	0.400 ms
0.45	0.461 ms	0.460 ms	0.464 ms	0.461 ms	0.457 ms	0.458 ms	0.465 ms	0.462 ms
0.55	0.581 ms	0.571 ms	0.567 ms	0.560 ms	0.570 ms	0.571 ms	0.587 ms	0.564 ms
0.65	0.825 ms	0.831 ms	0.829 ms	0.822 ms	0.823 ms	0.831 ms	0.820 ms	0.816 ms
0.75	1.17 ms	1.15 ms	1.17 ms	1.16 ms	1.16 ms	1.16 ms	1.16 ms	1.13 ms
0.85	1.80 ms	1.83 ms	1.77 ms	1.80 ms	1.79 ms	1.76 ms	1.78 ms	1.74 ms
0.95	3.76 ms	3.76 ms	3.81 ms	3.80 ms	3.77 ms	3.72 ms	3.64 ms	3.63 ms

Figure 2a — Priority Delay Table

Offered Load	Delay	Offered Load	Delay
0.05	0.318 ms	0.55	0.505 ms
0.15	0.339 ms	0.65	0.609 ms
0.25	0.363 ms	0.75	0.779 ms
0.35	0.396 ms	0.85	1.20 ms
0.45	0.442 ms	0.95	3.18 ms

Figure 2b — Delay Table for Single Priority

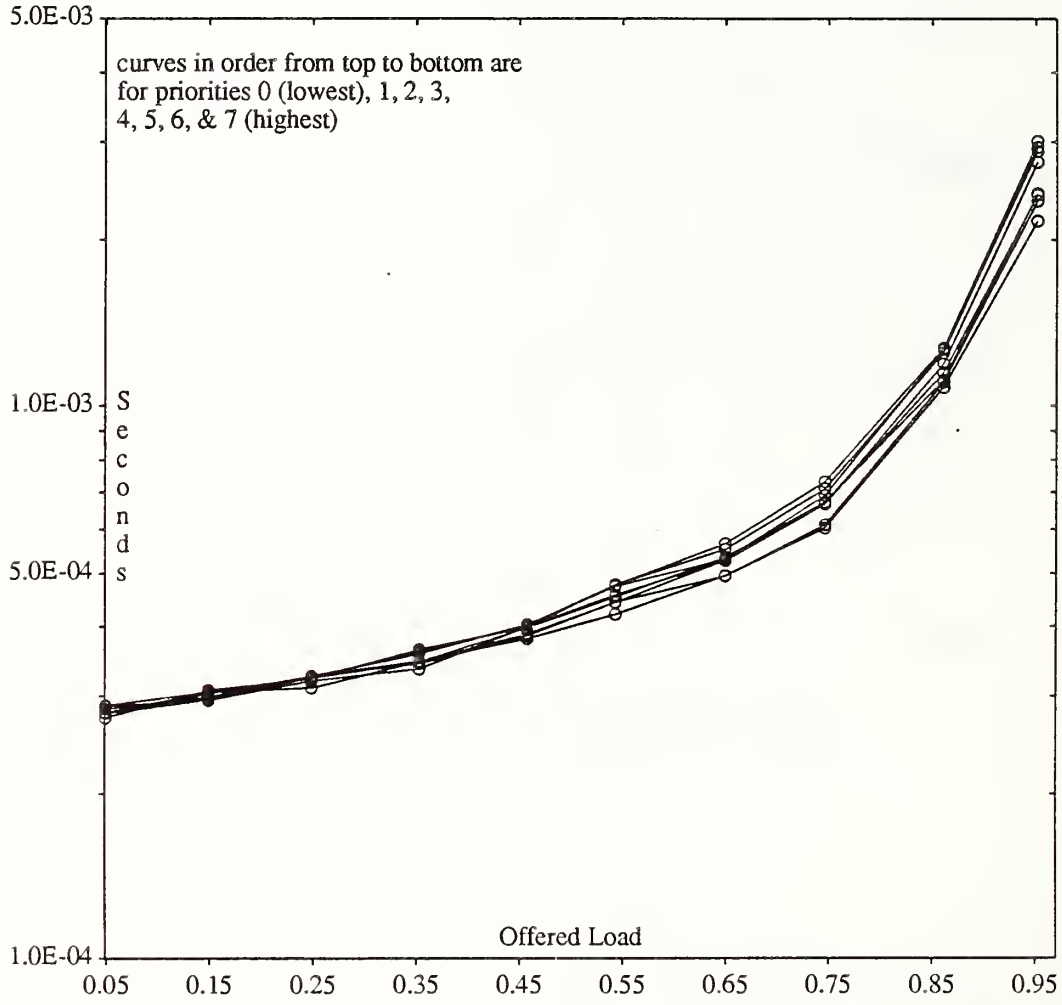


Configuration:

10 Stations
256 Bit Packets including framing
1 Bit Time Station Latency
Exponential Arrivals
1Mbps Medium

Figure 3 — Service Delay

Priority Delay for 10 Stations

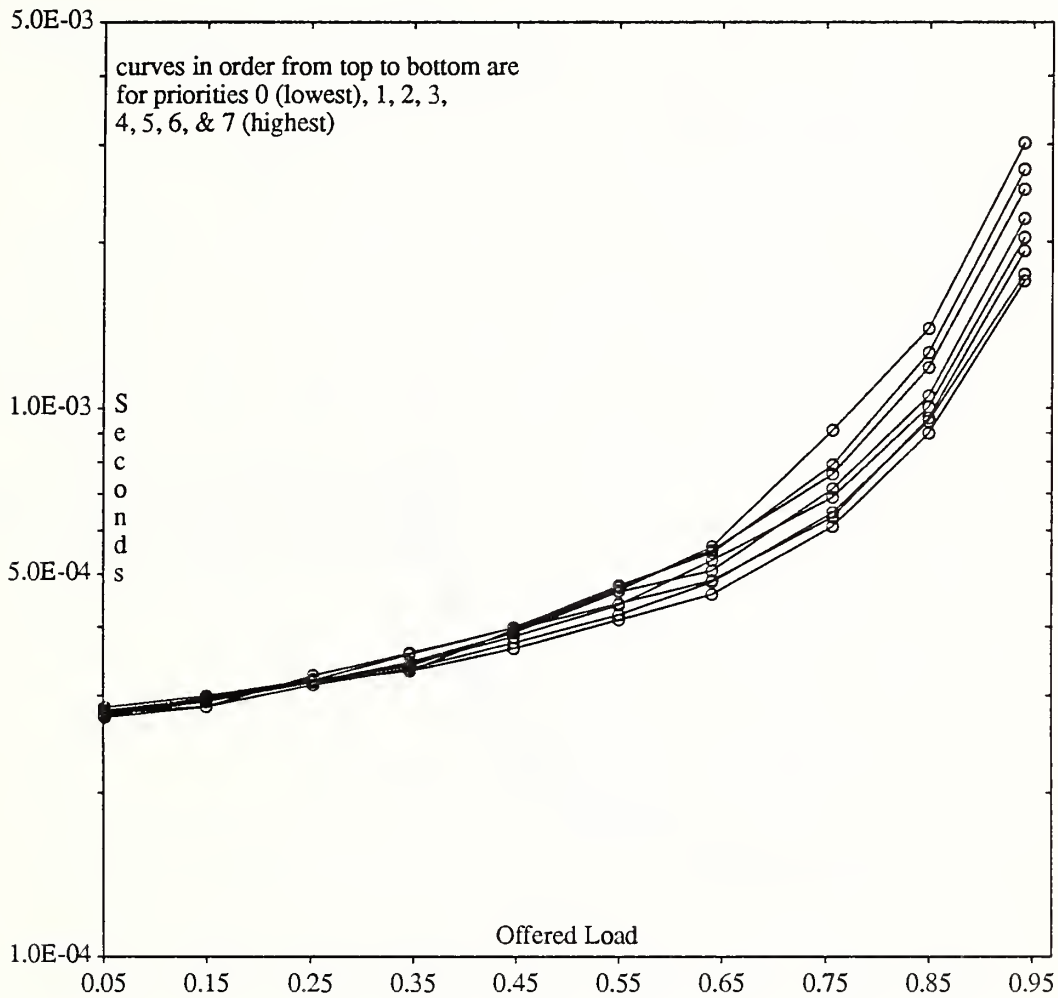


Configuration:

5 Stations
256 Bit Packets including framing
1 Bit Time Station Latency
Exponential Arrivals
1Mbps Medium

Figure 4 — Service Delay

Priority Delay for 5 Stations

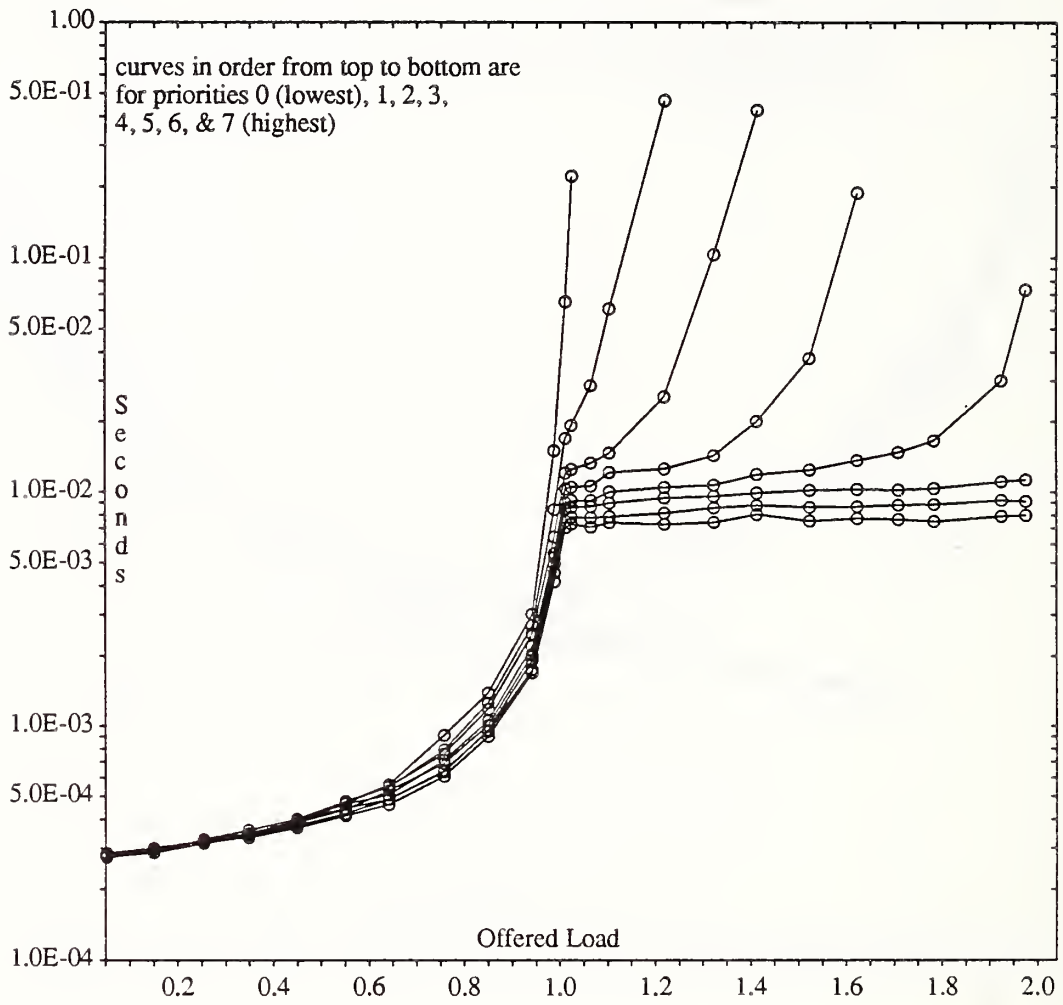


Configuration:

3 Stations
256 Bit Packets including framing
1 Bit Time Station Latency
Exponential Arrivals
1Mbps Medium

Figure 5 — Service Delay

Priority Delay for 3 Stations



Configuration:

3 Stations
256 Bit Messages including framing
1 Bit Time Station Latency
Exponential Arrivals
1Mbps Medium

Figure 6 — Priority Class Service Shutoff

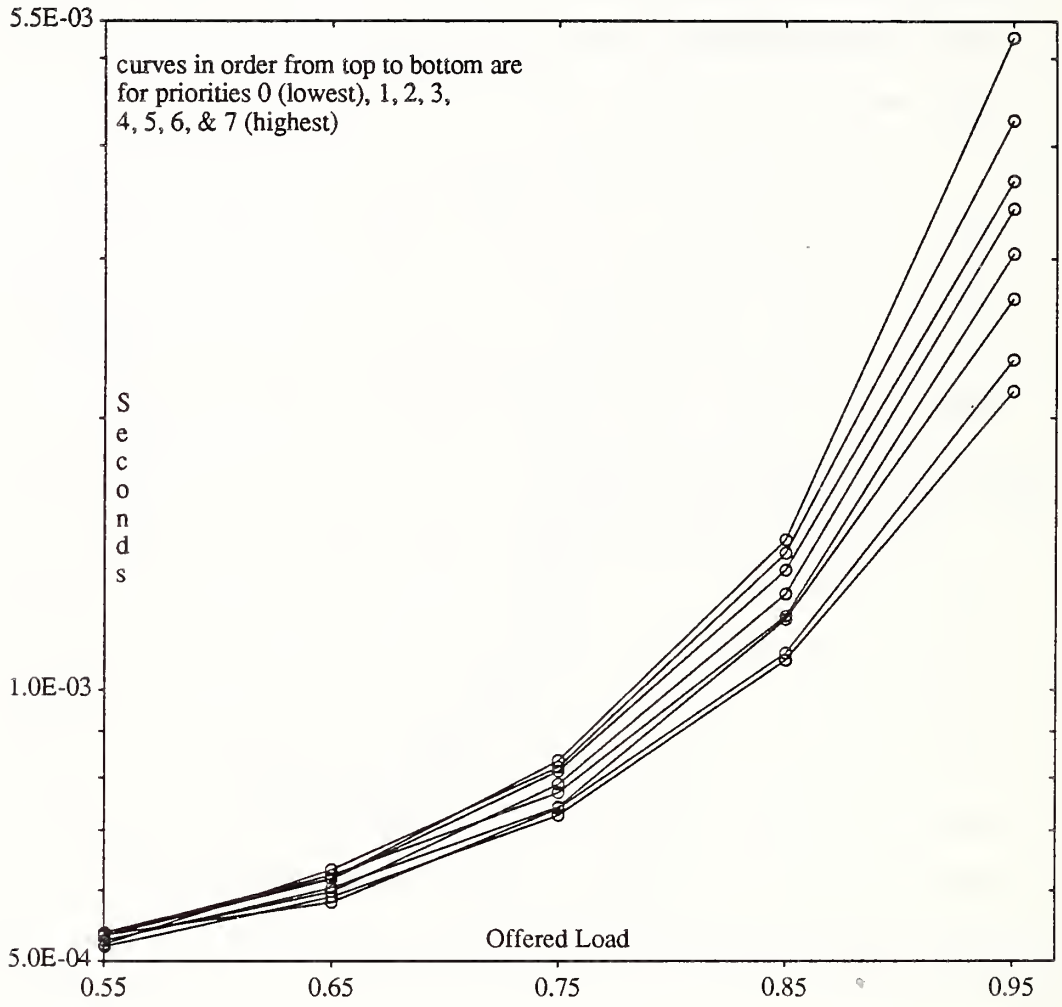
to one packet (consisting of only 19 data bytes) per token reception (note that the difference between the delay of the highest and lowest priorities is only a factor of 2.5). Figure 7 shows the service delay for 40 stations when the token holding timer has been set such that one-packet-per-token transmission results. Packets consist of 19 data bytes plus framing.

The increasing effect of the priority scheme can be seen in Figures 8, 9, and 10. Note that the graphs do not show delays for low loads. This is because there is negligible difference at low loads between the delays of the various priorities. One interesting effect is that as the number of stations decreases, the delay of the lowest priority is greater than that of a network configuration with a higher number of stations, and the delay of the highest priority is less. This is another indication that the priority scheme is operating as intended, i.e., lower priorities are taking the brunt of the increasing delay.

The ratio between the delays of the high and low priorities can be seen in Figure 11. For three stations, this ratio is about 32, for 10 stations it is about 13, and for 20 stations approximately 5.5. The actual values for the delays are given in Figure 12. Note that this chart only gives delay values for a network load of 0.95 — for loads less than this, the ratios between high and low priority delays decrease rapidly. In comparison, the delay experienced by packets in an identical network configuration with the exception that the entire network load was offered in a single priority is 5.8 ms. It should be remembered that the greatest delay will be experienced at high loads with a low number of stations, short packets, and single-packet-per-token service, so what we have been describing here is worst case behavior.

5. Conclusions

It is evident that unless very high network loads are expected in the ordinary course of events, the network performs better without the added overhead and complexity of the priority operation. The greatest effects of the priority scheme were obtained with very short packets, a low number of network stations, and single-packet-per-token transmission. The only way to limit transmission to one packet per token is to change the token holding timer such that there is only time for the transmission of one packet when the token is received. However, this will tend to reduce the token holding timer to the point where only very short packets are allowed (if only large packets are transmitted, this in itself reduces the effect

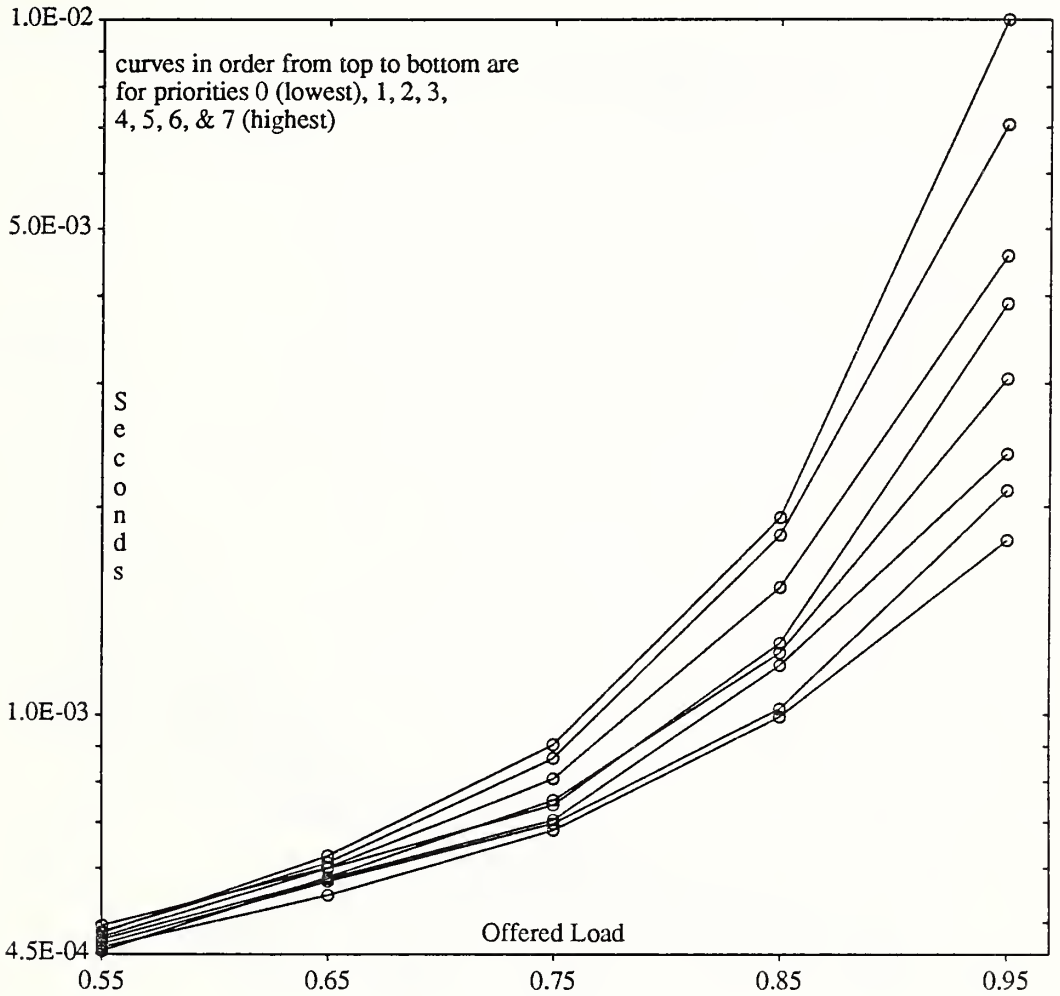


Configuration:

40 Stations
256 Bit Packets including framing
1 Bit Time Station Latency
Exponential Arrivals
1Mbps Medium

Figure 7 — Service Delay

Non-Exhaustive Service

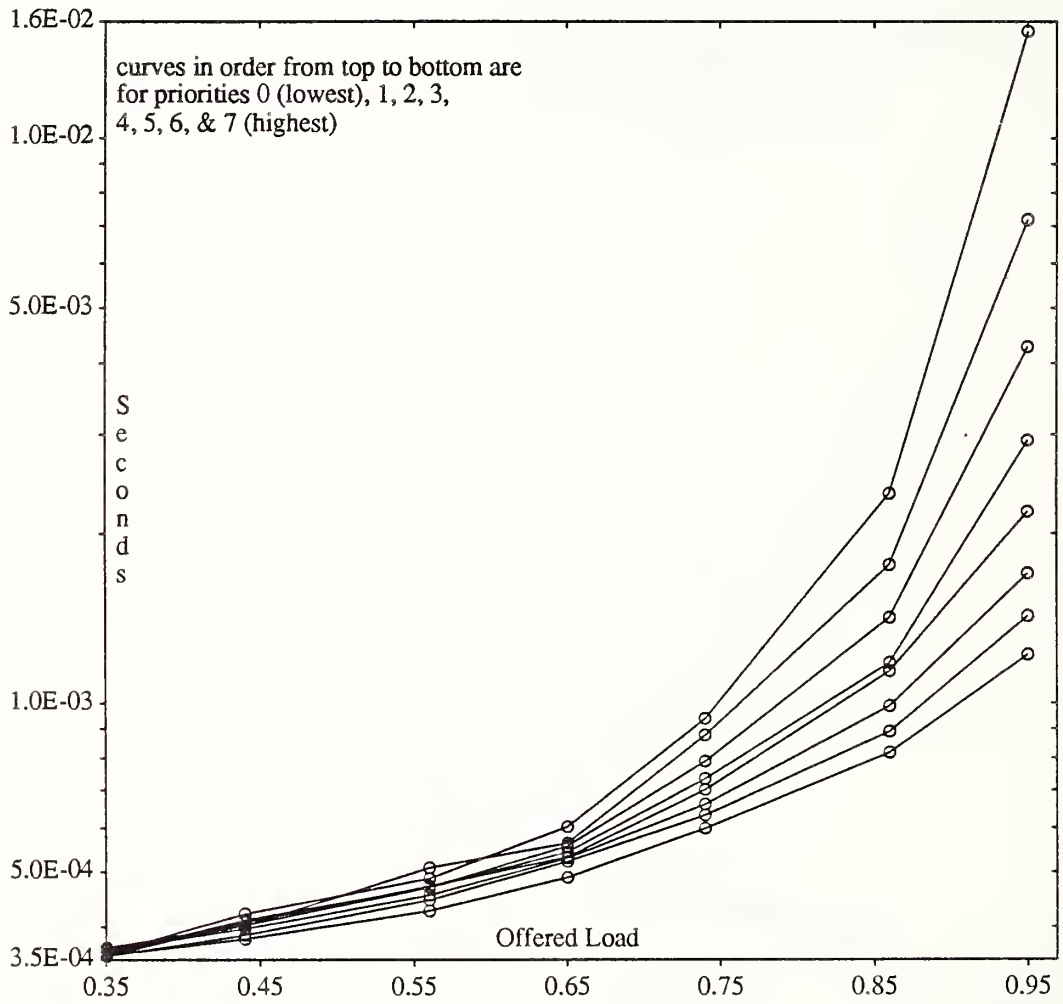


Configuration:

20 Stations
256 Bit Packets including framing
1 Bit Time Station Latency
Exponential Arrivals
1Mbps Medium

Figure 8 — Service Delay

Non-Exhaustive Service

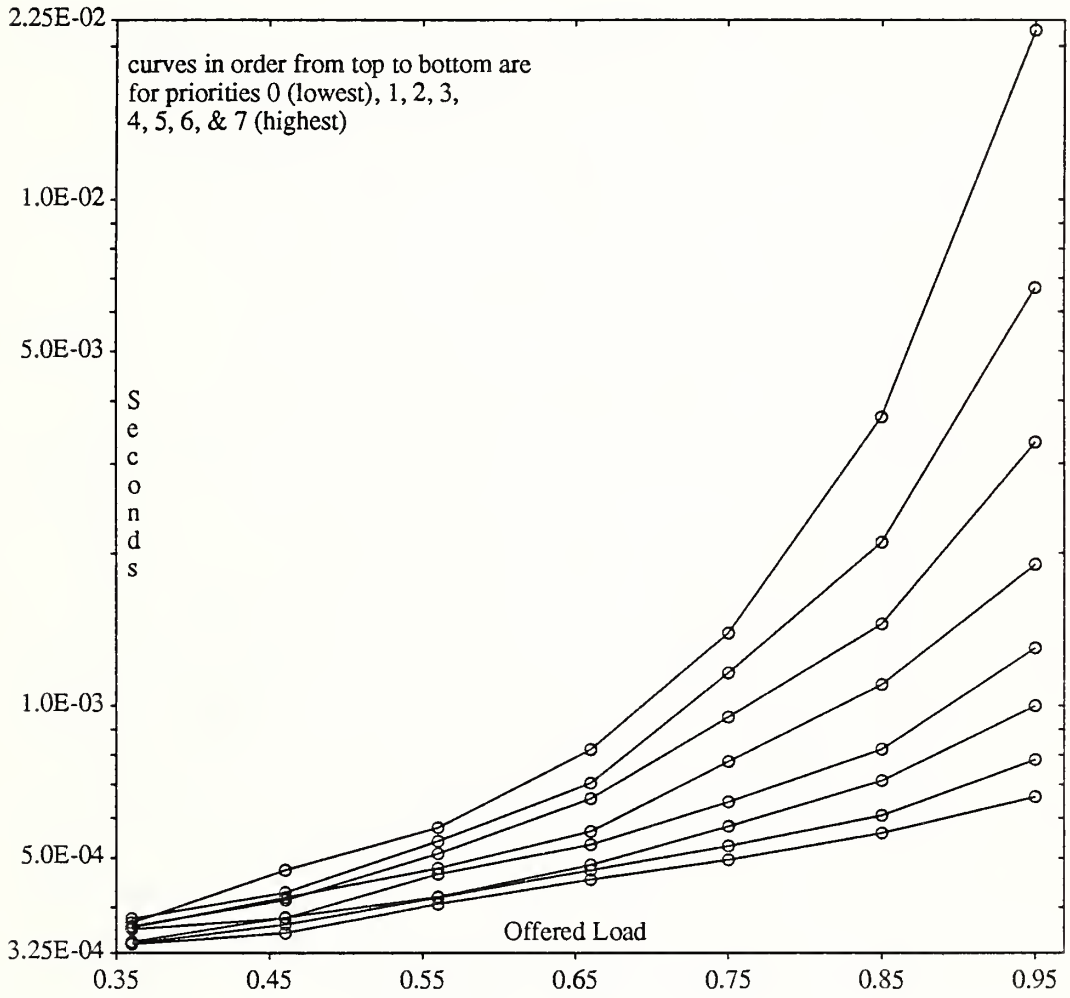


Configuration:

10 Stations
256 Bit Packets including framing
1 Bit Time Station Latency
Exponential Arrivals
1Mbps Medium

Figure 9 — Service Delay

Non-Exhaustive Service

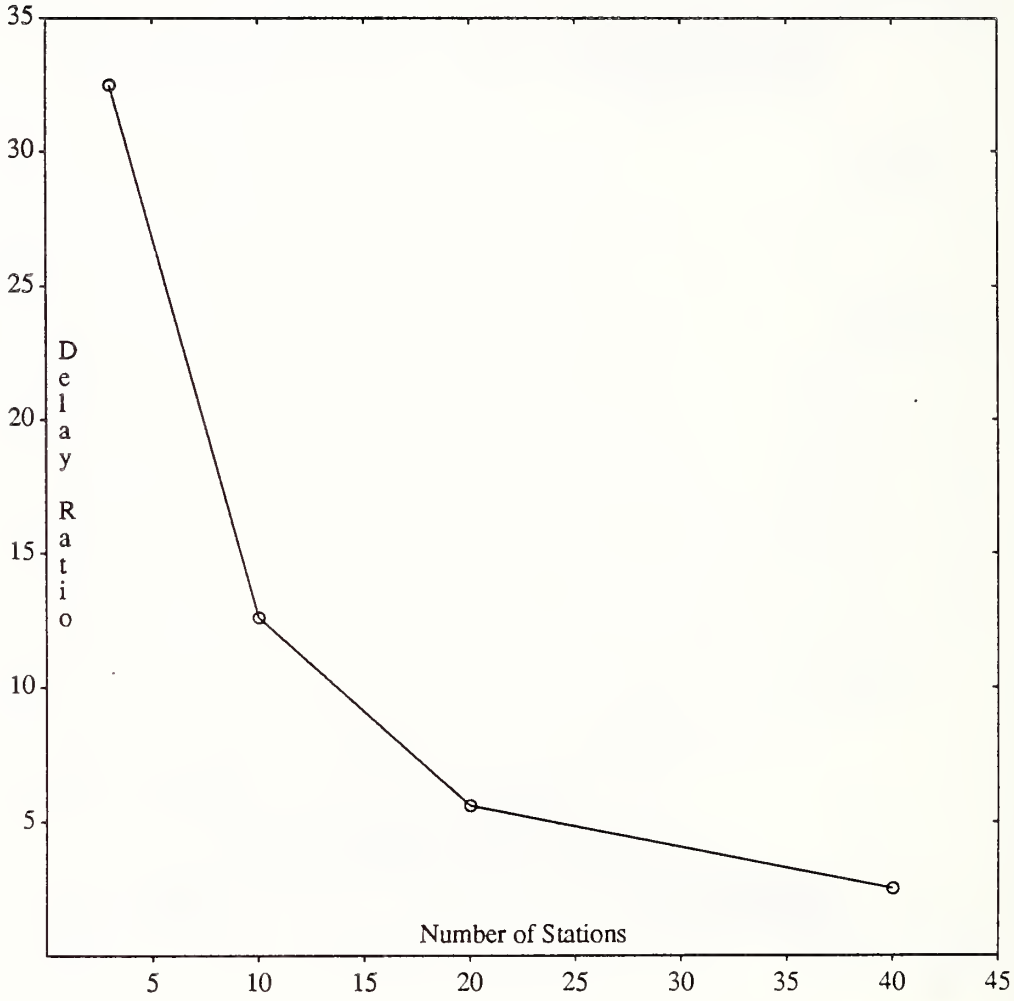


Configuration:

3 Stations
256 Bit Packets including framing
1 Bit Time Station Latency
Exponential Arrivals
1Mbps Medium

Figure 10 — Service Delay

Non-Exhaustive Service



Configuration:

Varying Stations
256 Bit Packets including framing
1 Bit Time Station Latency
Exponential Arrivals
1Mbps Medium

Figure 11 — Priority Delay Ratio

Active Stations	Delay by Priority 0	1	2	3	4	5	6	7
40	5.27 ms	4.27 ms	3.66 ms	3.41 ms	3.04 ms	2.71 ms	2.32 ms	2.14 ms
20	10.0 ms	7.06 ms	4.57 ms	3.90 ms	3.05 ms	2.38 ms	2.11 ms	1.79 ms
10	15.4 ms	7.18 ms	4.28 ms	2.93 ms	2.19 ms	1.70 ms	1.43 ms	1.22 ms
3	21.5 ms	6.69 ms	3.31 ms	1.90 ms	1.30 ms	1.00 ms	0.783 ms	0.661 ms

Figure 12 — Priority Delay Table
Offered Load of 0.95

of the priority scheme). It is also a difficult matter to generate a high network load with a small number of stations.

We conclude that, in general, priorities are not a very useful feature of the 802.5 protocol. While there are some circumstances in which priority operation is useful, they are rare. Given the added delay and overhead generated under ordinary conditions by the use of the priority scheme, and the small set of circumstances under which multiple priorities are shown to be useful, we believe that under general conditions, network traffic should be offered as a single priority, and that this will result in better service the majority of the time.

References

[IEEE85] IEEE, *Token Ring Access Method and Physical Layer Specifications*, The Institute of Electrical and Electronics Engineers, Inc., 1985.

[PEDE87] J. H. Peden, *Performance Analysis of the IEEE 802.5 Token Ring*, Master's Thesis, Department of Computer Science, University of Virginia, January 1987

Performance Analysis of an Integrated Traffic Slotted Ring

Muhammad Ali
Hermann J. Helgert

This paper describes the performance analysis of an integrated traffic slotted ring by means of simulation, as well as determining analytical models which provide best estimates for critical performance parameters. The network carries packet, voice and data circuit traffics. The parameters considered are message delays for packet traffic and blocking probabilities for voice and data traffics.

The ring network under study is capable of carrying packet, voice and data circuit traffic on a 8 Mbps slotted ring. The network, built by Laboratoire Central de Telecommunications, is currently designed for carrying X.25 traffic on the packet channel. The present study was done mainly to determine delays involved if the ring was to carry datagram traffic with lengths relatively larger than X.25 packets. Simulations were carried out to determine the performance characteristics. Analytical models were determined which provided best estimates for the message delays on the packet channel, and blocking probabilities for the voice and data circuit channels. The service disciplines considered for the packet channel were the Exhaustive, Gated and Limited service disciplines.

The ring latency,W, is a constant 125 μs, which is maintained at precisely that value by a hardware unit, irrespective of the ring size, subject to a maximum limit for the ring. The latency per station is two bytes. The 125 μs time frame is slotted as shown in Figure 1. There are a total of 128 bytes in the frame. The first byte has a pattern for synchronization. The first section constitute the packet channel, the second section is for voice circuits, with each byte slot being reserved for one voice channel. The third section is for data circuits and similar to the second section. The format for the packet channel is given in Figure 1.

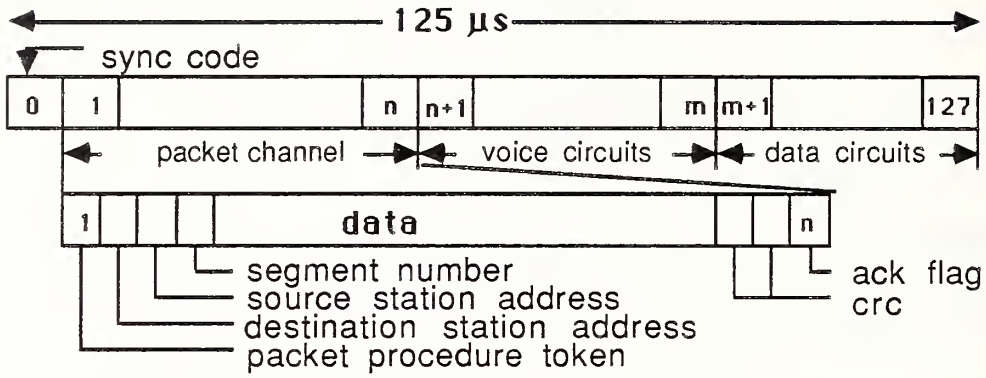


Figure 1 Frame structure of the ring

The first byte is a multiple use token. It indicates whether the packet channel is free or in use. If it is in use, different patterns indicate whether the channel content is a supervisory command, a complete message, the first segment of a message, an intermediate one, or the last one. The fourth byte indicates the number of the segment within a message. The last byte is an acknowledgement flag used by the destination to indicate acceptance of the segment or it's rejection by the destination station.

The access method employed for the packet channel could be described as demand sharing using fixed size slots, following the categorization in [1]. In the present case there is one slot(channel) for the packet traffic. When a station has a message in queue, it checks the packet channel token to determine if the channel is empty. If it is, the message, or a segment of the message is placed in the channel, and the token is set busy. The station maintains control of the channel till the message is completely transmitted. It may further maintain control of the ring depending on the service discipline. Once a station has finished using the channel, it sets the token free to indicate that the channel is available. Thus it is the source station which releases the channel and not the destination station. The channel may be occupied by the next station 'down stream' which has message(s) to send. This indicates that access control is based on a mechanism similar to hub polling. When a station wants to reserve a free channel for a voice or data circuit, it broadcasts it's intention to all other stations, so that no other station tries to use it. All stations have bitmaps indicating current utilization of the voice and data channels. This avoids overhead bits on the ring. Since the traffics in different sections do not interfere with each other, each traffic can be modelled independently.

The message arrival distribution is assumed to be Poisson for the packet channel, and identical for all stations. The message length is assumed to have a geometric distribution with a mean of 1000 bytes. Since the packet channel is generally smaller, a message may be divided into segments before transmission. The last segment of a message may be partially empty. Now the probability distribution function of message lengths is given by

$$P(y=k) = \sum_{r=(k-1)c+1}^{kc} p(1-p)^{r-1} \qquad k=1,2,....$$

where p is the inverse of mean message length
c is the segment length
k is the message length in units of segments

The walk time is considered to be constant and the sum of all walk times is W. The transmission time for each segment can be considered to be W,irrespective of it's length. Transmission error is considered negligible.

The mean and mean square message transmission times are given by

$$<m> = <y>.W$$
$$<m^2> = <y^2>.W^2$$

Smaller segments result in higher message transmission times. Using the derivations by Takagi[2] for the case of exhaustive service discipline, the mean message delay is given by

$$D_m = N[\lambda<m^2>+ w(1 - \lambda<m>)]/ 2(1 - N \lambda<m>) - W/2 + <m>$$

$$N \lambda<m> < 1$$

where N is the number of stations
λ is the mean message arrival rate at a station
w is the walktime.

For the case of gated service discpline it is

$$D_m = N[\lambda<m^2>+ w(1 + \lambda<m>)]/ 2(1 - N \lambda<m>) - W/2 + <m>$$

$$N \lambda<m> < 1$$

For the case of limited service discipline with a limit of one message serviced per visit, it is,

$$D_m = N[\lambda<m^2>+ w(1 + \lambda<m>)]/ 2(1 - N \lambda(w + <m>)) - W/2 + <m>$$

$$N \lambda(w + <m>) < 1$$

Figures 2 shows the message delay characteristics as a function of segment size. The graphs show the mean message delay results from both the analytical and simulation models, using the gated service discipline. The difference in message delays for the three service discipline is very small compared to the actual message delays, both for the analytical and simulation models. Differences in message delays due to service disciplines increase with heavy loading of the

network (values of $N\lambda\langle m\rangle$ or $N\lambda(w+\langle m\rangle)$ close to 1) , but for loadings of practical interest the differences are negligible.The delay due to the limited service discipline is the highest, while exhaustive service discipline has the lowest delay.The analytical and simulation results compare closely, for loadings of practical interest. The difference may be due to the assumptions of exponential distributions for length, as an approximation to a geometric distribution, in the simulation model.

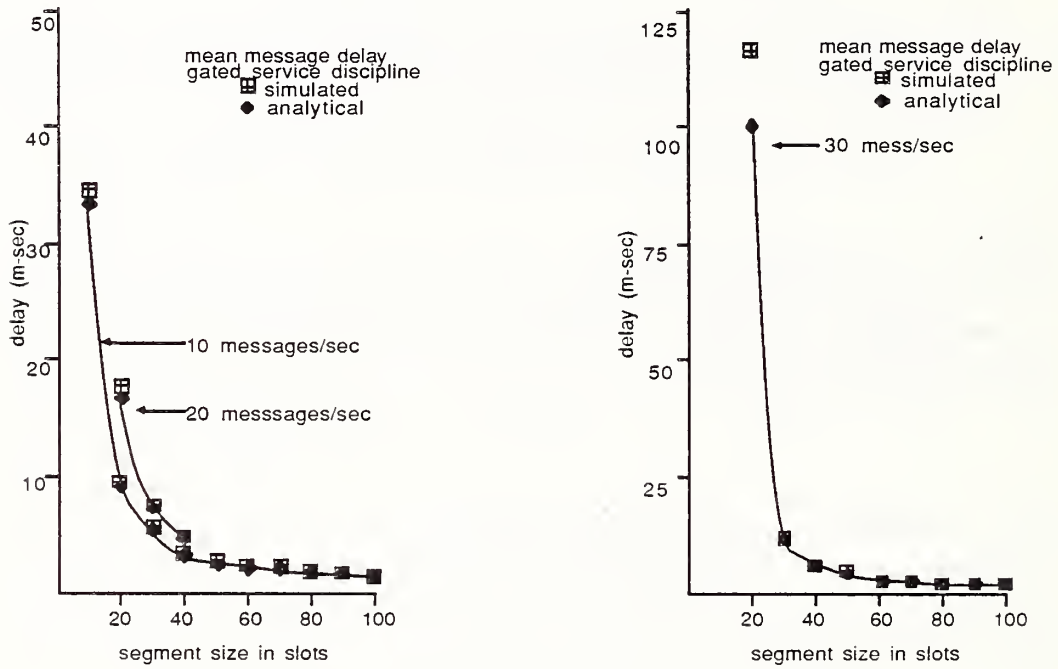


Figure 2 message delays as function of segment size

For the voice circuits it was assumed that there are infinite number of sources, which is a valid assumption if the number of sources are much greater than the number of voice channels. If the sources are not much greater in number, the approximation is still quite good [3]. The calls that are blocked are cleared, and do not return again during the same study period such as the busy hour. The call originations are characterized by a Poisson process, and the call lengths have exponential distributions[3]. The blocking probability is given by the Erlang B formula [3]

$$P_b\,(\text{voice}) = (A^c/c!) \,/\, \left(\sum_{x=0}^{c} A^x/x!\right)$$

$$A = \lambda_v L_v$$

where λ_v is the mean call arrival rate during the busy hour.

L_v is the mean length of calls

c is the voice channels allocation (slots reserved for voice circuits)

The data circuit service can be modelled as a M/M/d/d case, with the arrival process as the total for the network. The buffer is of the same size as the number of servers (channels), so the blocking probability is the probability of the buffer occupancy being d.

$$P_b\,(\text{data}) = (R^d/d!) \,/\, \left(\sum_{x=0}^{d} R^x/x!\right)$$

$$R = \lambda_d L_d / S$$

where λ_d is the mean arrival rate per second

L_d is the mean message length in bytes

S is the service rate in bytes per second (8E+3/second)

d is the data channels allocation (slots reserved for data circuits)

Figure 3 shows the blocking probabilities as a function of channel allocation. The analytical and simulation results match very closely and follow the indicated curves.

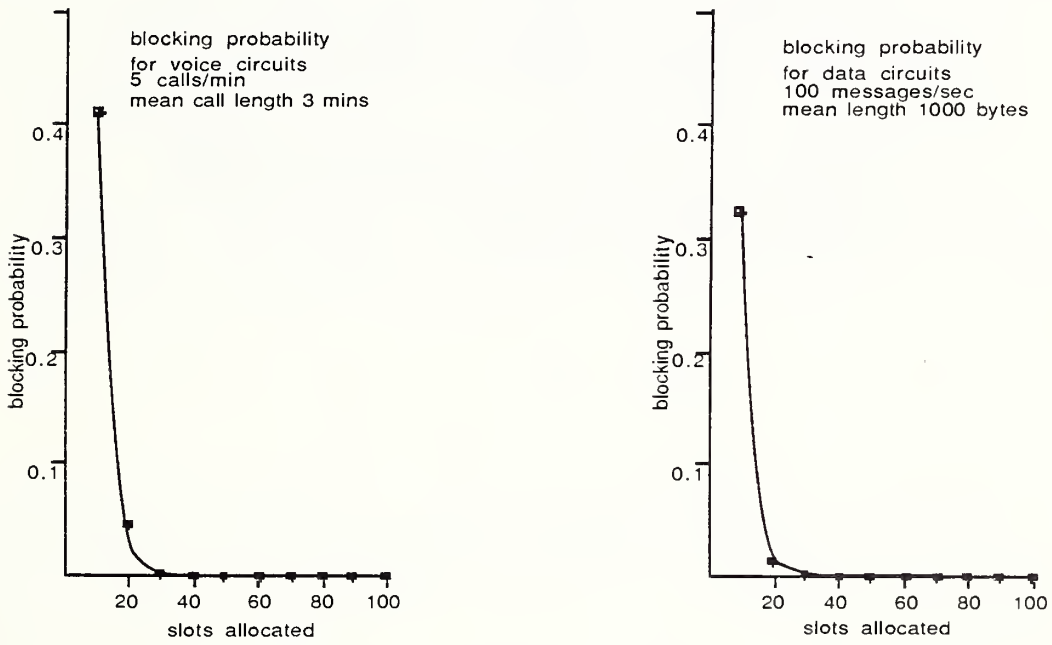


Figure 3 blocking probabilities for voice & data circuits

Conclusion

The message delays due to different service disciplines shows very little difference, specially for larger segment sizes. The mean message delay for the limited service discipline is the highest, followed by those for the gated and exhaustive service disciplines, as expected. Fairness between stations can be increased by using the limited service discipline, with price of a very small increase in the mean message delay. The comparison between the analytical and simulation results for the packet channel suggests that the analytical models can be used, under the given assumptions, to determine mean message delays in place of the more expensive simulation methods. Since the analytical results for the voice and data circuits match very closely with the simulation results, the former can be used effectively to determine the blocking probabilities under various load conditions.

REFERENCES

[1] B.K. Penny & A.A. Baghdadi, "Survey of computer communications loop networks:Parts 1 and 2", Computer Communications, vol. 2, pp 165-180, 244-261,1979.

[2] H. Takagi, "Analysis of polling systems", The MIT Press, 1986.

[3] R.L. Freeman, "Reference manual for Telecommunication Engineering", Wiley-Interscience, 1985.

[4] J.L. Hammond & J.P. O'Reilly, "Performance Analysis of Local Computer Networks, Addison Wesley, 1986.

[5] J.F. Hayes, "Modeling & Analysis of Computer Communications Networks", Plenum Press, 1984.

[6] W. Bux, "Local Area Subnetworks: A Performance Comparison", IEEE Trans. on Commun., Vol. COM-29, No. 10, Oct. 1981.

[7] G.S. Blair, "A Performance Study of the Cambridge Ring", Computer Networks, Vol 6 , 1982.

NBS-114A (REV. 2-8C)

U.S. DEPT. OF COMM. BIBLIOGRAPHIC DATA SHEET (See instructions)	1. PUBLICATION OR REPORT NO. NBSIR 87-3516	2. Performing Organ. Report No.	3. Publication Date FEBRUARY 1987

4. TITLE AND SUBTITLE

Workshop on Factory Communications
March 17-18, 1987

5. AUTHOR(S)

Robert Rosenthal, Editor

6. PERFORMING ORGANIZATION (If joint or other than NBS, see instructions)

NATIONAL BUREAU OF STANDARDS
DEPARTMENT OF COMMERCE
WASHINGTON, D.C. 20234

7. Contract/Grant No.

8. Type of Report & Period Covered

9. SPONSORING ORGANIZATION NAME AND COMPLETE ADDRESS (Street, City, State, ZIP)

10. SUPPLEMENTARY NOTES

☐ Document describes a computer program; SF-185, FIPS Software Summary, is attached.

11. ABSTRACT (A 200-word or less factual summary of most significant information. If document includes a significant bibliography or literature survey, mention it here)

Factory and laboratory automation requires computer communication networks capable of providing rich connectivity among diverse computer systems, sensors, actuators, robot controllers and other devices. Protocol specifications for networks capable of meeting manufacturing and technical office requirements are evolving nationally and internationally. These networks enable distributed computer applications within the factory; and, with these distributed computer applications, automated design and manufacturing is possible.

These workshop proceedings report recent efforts of government, industrial and academic researchers in factory communication networks. Four major research areas are addressed: the application of manufacturing automation protocols (MAP) in factory networks, the application of non-MAP protocols, design and simulation tools and analytic and simulation modeling.

12. KEY WORDS (Six to twelve entries; alphabetical order; capitalize only proper names; and separate key words by semicolons)

computer networks; factory automation; local area networks; MAP; modeling; simulation

13. AVAILABILITY

☒ Unlimited

☐ For Official Distribution. Do Not Release to NTIS

☐ Order From Superintendent of Documents, U.S. Government Printing Office, Washington, D.C. 20402.

☒ Order From National Technical Information Service (NTIS), Springfield, VA. 22161

14. NO. OF PRINTED PAGES

281

15. Price

$24.95

USCOMM-DC 6043-P80