ON BUFFER ALLOCATION IN TRANSPORT PROTOCOLS

Athanassios Zissopoulos

School of Computer Science

McGill University, Montréal

June 1987

A thesis submitted to the

Faculty of Graduate Studies and Research

in partial fulfillment of the requirements

for the degree of Master of Science.

ACKNOWLEDGMENTS

I would like to thank Prof. Carl Tropper, my thesis supervisor, for introducing me to the field (and for waiting while I picked daisies), for guiding me, and for being a friend.

I am indebted to my friends in heavy smokers' lab [EVEQ86]. Claude for the endless discussions and Lucy for uncomplainingly inhaling the pollution we generated.

I would also like to thank my parents for their support and encouragement. Lastly, I would like to thank my wife Susan, who sat with me through interminable revisions, and without whose patience this thesis might have been finished earlier.

ABSTRACT

We present, in this thesis a buffer allocation algorithm for use in a transport level protocol. The algorithm allocates buffers as a function of the estimated queue length for (user-defined) message categories of traffic, thereby ensuring fair behaviour on its part. Furthermore, higher priority is given to smaller message categories. Different categories of traffic might correspond to interactive and file transfer traffic. A deadlock avoidance mechanism is incorporated in the algorithm as well, in order to avoid reassembly deadlock.

RESUME

Nous présentons dans cette thèse un algorithme pour répartir les mémoires tampons attribuées au niveau transport d'un réseau à commutation de paquets. L'algorithme alloue les mémoires tampons en fonction des longueurs estimées des files d'attente des différentes catégories de trafic, assurant ainsi une répartition équitable. En outre, une priorité plus élevée est donnée aux catégories de trafic ayant les messages les plus courts. Ces catégories de trafic pourraient correspondre au trafic interactif ainsi qu'un trafic dû au transfert de fichiers. Un mécanisme évitant le verrou mortel dû aux réassemblages des paquets est incorporé dans l'algorithme.

Table of Contents

Chapter 1

Introduction

We present in this thesis a buffer allocation algorithm which can be used in a transport protocol such as the OSI's TP class 2,4 protocols or the ARPANET's TCP. The algorithm can be used in conjunction with the buffer reservation mechanism between a source-destination pair of hosts. The algorithm, if extended to account for different network categories and different communication media, can also be adopted for use at a gateway. Before we proceed, we give a brief introduction to the terms found throughout the thesis.

A computer network is a collection of independent computers (hosts) which are able to communicate with each other. The hosts are connected to a communication subnet which consists of switches (imps) connected to each other via some transmission media (telephone, cable, satellite, and fiber optic links). The rules that govern the communication between the computers are called protocols. Several benefits arising from this interconnection are:

- the ability to use remote computers, remote programs and databases
- an increase in reliability due to the availability of alternative resources [TANE81]
- electronic mail services

Typical services provided by a computer network are mail, news, and remote login. Future applications can include computer-aided education, teleconferencing and electronic funds transfer [TANE81]. There are two categories of networks, the local area and the wide area. In local area networks, which are usually owned by private organizations, the resources lie within a radius of a few kilometers and have high data rate

communication links (usually coaxial cable or fiber optics) of the order of a few Mbps. In wide area networks, the resources are located far apart (spread across a continent or around the world). The communication links are slower (of the order of 50 Kbps) and may be owned by public or private organizations Networks can also be interconnected through a gateway, thus forming supernetworks. The role of the gateway is the transfer of messages/packets from one network to another.

The first experimental (one node) network in North America was brought to life in 1969 [KLEI76] This arose from the ARPA community's realizing that it could take advantage of already existing but diversely located machines by interconnecting them. The interconnection was meant to increase the utilization and the availability of the resources in an economical fashion. The efforts ultimately resulted in the ARPANET packet switching network. During the two decades that have since elapsed, a clearer understanding of networking and a more systematic design process has emerged Significant efforts by international organizations such as the International Standards Organization (ISO) and the Comité Consultatif Internationale de Télégraphique et Téléphonique (CCITT) are leading towards a standardization of the protocols Typical examples of today's successfully operating networks are DATAPAC (Canada), TYMNET (USA), TRANSPAC (France), SNA (IBM), EIN (Europe), EUNET (Europe), and CSNET (USA) [QUAR86], [SCHW87].

One of the first efforts, with the ultimate goal being the international standardization of protocols, is the reference model for Open Systems Interconnection (OSI) suggested by the ISO [ZIMM80]. It consists of seven layers, all of which are derived from similar theoretical backgrounds. Each layer uses the services provided by the layer below and provides services to the layers above. The three lower layers (the Physical, Data link and Network) deal with the communication network, while the upper four

layers (the Transport, the Session, the Presentation and the Application) deal with the end users.

The Transport layer, which we deal with, guarantees reliable and efficient end-to-end communication. Efficiency can be achieved by multiplexing and segmenting, while reliability is achieved by reassembly and reordering, error detection and recovery techniques, and flow control. These techniques are implemented through a program called the *transport station.*

Flow control is a set of rules needed to regulate the traffic between two processes, in this way preventing buffer overflow at the receiver. It occurs in the lower layers as well as in the upper ones. One common technique of flow control is the *sliding window* mechanism. With this technique, the maximum number of unacknowledged messages between a pair of processes is restricted to the value of the window (a window of one is the *stop-and-wait* mechanism). The way it works is as follows. each acknowledgment carries back two numbers One is the sequence number of the next expected message and the other is the credit number (that is, the number of messages permitted to be sent until the next allocation). The sequence number is a unique number which is assigned to any data unit transferred between a source-destination pair It serves to identify the unit during the transmission phase, and helps reordering during the reassembly of a completely received message An interesting survey on flow control can be found in GERL80

TP2 and TP4, ISO transport protocol international standards, and the ARPANET's TCP use a sliding window of sequence numbers Credits are allocated by an exchange of ACK TPDUs or ACK segments. Initial allocation occurs during the connection establishment phase In TCP, the credits are allocated in bytes while in TP2 or TP4 they are designated in *transport protocol data units* (TPDUs) TPDUs can vary in size from 128 to 8192 octets (bytes) A detailed description of the TP and TCP credit alloca-

tion mechanisms can be found in SCHW87 or STAL87. For even more detail the specifications, ISO84a, NBS83a, and TCP83 may also be consulted.

The buffer allocation algorithm which we developed allocates buffers as a function of the message length and the estimated queue length at the destination transport station Space allocated for a message category is accessible to that category and to all the categories having smaller message lengths, ie , small messages have a higher priority than the larger messages Categories are determined by the length of the message and are user defined The objective is to provide a small delay for single packet traffic (interactive or mail) and at the same time ensure acceptable throughput levels for longer messages (file transfer, videotext).

The algorithm is comprised of two parts In the first part, the expected buffer occupancy for different categories of messages is estimated In the second part, each category receives its predicted space, while free space is made available to every other category Furthermore, a reassembly deadlock avoidance mechanism is incorporated within the algorithm for reasons explained below.

When the destination host buffers data and delivers them only upon reception of the complete message, a reassembly deadlock may arise. This event occurs when buffers are fully occupied by partially received messages. Since no more message segments can be sent, obviously none can be received. Hence, a deadlock situation, which necessitates the employment of our reassembly deadlock avoidance algorithm

In order to examine the performance of the algorithm, we have undertaken simulations comparing the proposed credit allocation to the one employed by the TP or TCP The simulations were carried out using the PAWS (Performance Analyst's Workbench System) simulation tool [PAWS84]

The remainder of the thesis is organized as follows Chapter 2 gives a description of the most popular transport protocols, the TP 0 to 4 class and the TCP Chapter 3 deals with the theoretical underpinnings for our approach to the problem as well as that of the previous work Chapter 4 gives a rather extensive discussion of the previous work done in the area. In Chapter 5, we discuss the rationale for our using the forecasting approach, describe the buffer allocation algorithm in more detail, and present the deadlock avoidance algorithm. Chapter 6 describes the simulation model and the simulation tools Finally, Chapter 7 contains the results, conclusions, and suggestions for future work.

Chapter 2

Transport Protocol

The transport layer is the layer that is responsible for transferring data between two user processes (the layers above) It is the first layer, starting from the bottom in the ISO hierarchy, that deals with end-to-end communication. This is in contrast to the lower layers, which are concerned with communication between (network) switches. The tasks of this layer include naming and addressing of user processes, connection establishment and termination, multiplexing and splitting of connections, segmenting and reassembly of messages, error recovery, and flow control The transport layer guarantees a quality of service to the users that is independent of the service provided by the underlying layers and network(s) The program which implements these services is called the **transport station** It runs on a host computer and is (or it could be) part of the host's operating system.

Services are requested by name from the user, hence it is the responsibility of the transport layer to map the name to its transport address. These addresses can be either hierarchical or flat [TANE81]. The hierarchical address is a set of disjoint fields numbered similarly to telephone numbers. Knowledge of the address determines the exact location A flat address is just a unique number with no relationship to the rest of the addresses.

Once the address is known, the connection establishment phase proceeds. Part of this phase is the negotiation of the quality of service (acceptance of the suggested quality of service by the sender or modification by the receiving user) between the user processes The quality of service deals with parameters such as sequence number space,

data unit size, throughput and delay requirements, etc. This phase's being successful leads to the data transfer phase.

If high throughput is the issue, the transport connection may be split into multiple network connections. On the other hand, the multiplexing of multiple transport connections onto a single network connection can reduce the cost of the connection. Segmenting may occur both if the transport data unit exceeds the network data unit or if the destination reduces the proposed data unit size. Segmenting at one end causes data reassembly (reordering of the data) at the other end, since data have to be delivered to the user as they were sent. Flow control prevents overflow of the destination's buffers by a fast source. Transmitted (unacknowledged) data can be buffered at the sender, at the destination, or both, depending upon the reliability of the underlying network(s). At the destination, buffer space is allocated, if it is available, upon request. The lack of buffer space naturally causes the sender to wait.

The transport protocol should be able to detect, and recover from, any detrimental situations such as duplicate data removal, reassembly deadlock, etc.

In the following, we describe the ISO TP 0-4 class transport protocol, with emphasis on TP4, and the ARPANET's TCP.

2.1 OSI Transport Protocol (TP)

TP is the ISO international standard transport-layer protocol [ISO84a], [NBS83a]. In an attempt to deal with different categories of network services, the protocol consists of five distinct classes. Going from class 0 to class 4, the protocol provides increasingly more complex functions. The choice is determined by the services offered by the underlying network connections.

There are three types of network connections [SCHW87]. These network connections are assumed to be less reliable as one moves from Type-A to Type-C. In a Type-A connection, there is no out of order arrival or loss of packets (ie. networks with virtual circuit services). The rate of both the errors and the signalled failures to the transport protocol is acceptable In Type-B, the error rate is acceptable but not the signalled rate of failures. In this case, the transport protocol should provide error recovery Finally, in Type-C, the error rate is also not acceptable and the transport protocol should support functions for error detection and recovery (ie. networks with datagram services) Classes 0 and 2 operate on Type-A network connections, classes 1 and 3 on Type-B and class 4 on Type-C. Class 0 provides set up connection, segmenting, and data transfer services. Class 1 provides simple error recovery services as well Class 2 provides multiplexing, class 3 provides error recovery and multiplexing, and finally class 4 provides error detection and recovery

2.1.1 TP4

As was mentioned above, TP4 was designed to work under pathological situations The basic unit of information exchange between the transport stations is referred to as the Transport Protocol Data Unit (**TPDU**).

A connection is established between two transport addresses, the transport service access point identifiers, by a Connection Request (CR) TPDU (maximum size for each TPDU header is 254 octets) In the case that both ends try to establish a connection, then two connections are set up. The CR TPDU, among its other duties, suggests the data TPDU size (possible range between 128 octets to 8192 octets), the sequence number space (7 or 31 bit field), the initial credit allocation (4 or 16 bit field) for traffic

flowing from the receiver to the sender, and the quality of service parameters The receiver responds either with a Connection Confirm (CC) TPDU accompanied by an initial credit allocation, in case it wishes to communicate, or with a Disconnect Request (DR) TPDU in case it doesn't Moreover, it is permitted to decrease (although not beyond a specified limit) any parameters (suggested by the sender), but is never allowed to increase them CR and CC can carry up to 32 octets of data, while a DR may carry up to 64 octets of data

Upon completion of the connection establishment phase, which uses a three-way handshake (CR, CC and ACK or DT), the protocol enters the data transfer phase by transmitting Data (DT) TPDUs. The DT TPDUs are acknowledged by Acknowledgment (ACK) TPDUs, the delay of which causes data to be retransmitted after the expiration of the appropriate timer The destination transport station can deliver the data to the transport user either piece by piece or after receipt of the complete message The choice is implementation dependent.

Flow control is carried out by a sliding window mechanism. More precisely, each ACK TPDU carries the sequence number of the next expected TPDU (lower edge window) plus the number of credits allocated (the number of DT TPDUs the sender is permitted to send until the next allocation).

Termination of the connection is carried out by a Disconnect Request and Disconnect Confirm (DC) TPDUs. Any data that were sent before the termination and were not delivered after the receipt of the DR and DC TPDUs, are discarded. The Graceful close Request (GR) TPDU helps these waiting data to be delivered (available only in the NBS FIPS version). It is acknowledged by an ACK TPDU

Connectionless data transmission is accomplished by using a single Unit Data (UD) TPDU.

Fast delivery of urgent data ($\leq$ 16 octets) can be done with the Expedited Data (ED) TPDU, which should in turn be acknowledged by an Expedited Acknowledgment (EA) TPDU. Only one ED TPDU can be outstanding at any time.

Finally, transmission error detection is accomplished by using the Fletcher checksumming procedure. Also, since each TPDU is always acknowledged by another TPDU (CR is acknowledged by a CC, CC by DT or ACK, DT by ACK, and ED by EA), further error detection and recovery is accomplished by the acknowledgments and the timers which are set during transmission.

2.2 Transmission Control Protocol (TCP)

TCP was designed to work over unreliable networks [TCP83]. It operates in tandem with the Internet Protocol (IP), whose purpose is to support the interconnection of networks. The TCP transport station accepts messages of arbitrary length, the **letters**, and splits them into **segments** ($\leq$ 65K), which are the basic data units. There are four types of segments: DATA, SYN, FIN and ACK. The minimum size of the header is 20 bytes.

The connection is established by the SYN (from both sides) and ACK segments, and a three-way handshake is employed. There is only one connection set up between each pair of source-destination addresses (the sockets as they called in TCP). This is an effect of the fact that the sending process can be in either active or passive mode. When in active mode, the sending process denotes a destination with which it would like to connect, whereas in passive mode it waits for connection establishment by another process. Data are carried on DATA segments and are acknowledged by ACK segments. ACK segments, in turn, can be piggybacked onto DATA segments. They may ack-

nowledge segments individually or in groups. Timers are set up upon the transmission of DATA segments, and their expiration causes DATA retransmission.

Flow control is carried out by employing a variable-size sliding window (16 bit field), but the protocol allocates bytes, in contrast to TP4 which allocates TPDUs (blocks of data). The protocol encourages the sender TCP to transmit periodically, even if the window is zero, in order to avoid the possibility of a deadlock in case ACK segments were lost. Since each byte of data has its own sequence number, the sequence number space is extremely large (32 bit field) This guarantees that old duplicates have vanished before the same sequence numbers are used again (ie "wrap-around") In addition, each segment has a maximum lifetime in the network, after which it is destroyed.

A connection is terminated by the issuance of an FIN segment accompanied by an ACK segment from the other side A Graceful close mechanism is also available, and requires FIN segments by both sides followed by an ACK segment (three-way handshake termination)

Data marked as URGENT are transmitted immediately by the TCP. There is no limit to the number of outstanding data of this kind A PUSH service can also be initiated by the sender process in order to receive fast response by the destination process. This service is meant to help the interactive users and to avoid the possibility of a deadlock.

Finally, checksumming procedures employ the one's complement algorithm, while error detection and recovery is incorporated into the protocol

We close this section by pointing out that certain portions of TCP have been adopted, although with few modifications, by TP4. These include the three-way handshake connection establishment and the flow control mechanism.

Chapter 3

Theoretical Underpinnings

In this chapter, we give a brief introduction to the mathematical theory involved both in the previous work relating to the problem of buffer allocation and flow control and in our own approach to the problem. The theories introduced are the Markovian Decision Processes and Time Series Analysis. A more complete analysis may be found in [HOWA60] and [HEYM84] for the Markovian Decision Processes and in [BOX76] for Time Series Analysis, from whence this description was drawn.

3.1 Markov Decision Process (MDP) and Policy-Iteration Method

A Discrete Time Markov Chain is a set of random variables $X_1, X_2, .., X_n$, which satisfies the following equality

$$P[X_n = j \mid X_1 = i_1, X_2 = i_2, .., X_{n-1} = i_{n-1}] = P[X_n = j \mid X_{n-1} = i_{n-1}]$$

that is, the probability that the process is moving to the next state is dependent only on the current state (or in other words the history of the process is summarized in the current state) Such a process is described completely by the set of **states** S and a **transition (stochastic) matrix** $P[p_{ij}]$ where $p_{ij} = P[X_n = j \mid X_{n-1} = i]$.

The probability of finding the system in state j after n transitions is defined by $\pi_j(n) = \sum_{i=1}^{|S|} \pi_i(n-1)p_{ij}$ or in vector form by $\pi(n) = \pi(0)P^n$. When this state occupancy vector is independent of the initial state for a large number of transitions n (an ergodic process), the vector is called a **limiting state probability** vector. This quantity

is of interest since it describes the process sufficiently The transient behavior of the system can be found by applying z-transform analysis. It can be shown that $\Pi(z) = \pi(0)[I - zP]^{-1}$ where $\Pi(z)$ is the z-transform of the $\pi(n)$ vector and I is the identity matrix. The inverse transform of the matrix $[I - zP]^{-1}$ consists of two parts; the steady state portion, S, and the transient behavior portion, $T(n)$. The (i,j) element of the matrix gives the probability that the system will occupy state j after n transitions given that state i was the initial state The transient part (which is a function of n) decreases to zero as n increases Each row of the steady state matrix, S, is the limiting state probability vector of the process.

A Markovian Decision Process (MDP) is the model obtained from a Markov chain by adding actions and rewards An **action** refers to the feasible paths (alternatives) taken in every state and is defined for each state An immediate **reward** (which is a random variable) refers to the positive or negative gain from taking a possible action while being in a particular state The next state and the immediate reward are independent of the history, given the current state and action (the Markovian assumption). The choice of an action can affect the future in terms of the information available, the feasibility of an action, and the gain of the rewards.

A quantity of interest, the **expected total reward** $u_i(n)$ after n transitions given that the system is currently in the state i, is given by

$$u_i(n) = q_i + \sum_{j=1}^{|S|} p_{ij}\, u_j(n-1) \quad i = 1,2,..,|S| \quad n = 1,2,...$$

that is, the sum of the expected immediate reward due to transition out of state i and the expected reward from state j with $n-1$ transitions. The above equation in vector form is $u(n) = q + Pu(n-1), \quad n = 1,2,3,...$

A z-transform analysis shows that $u_i(n) = ng_i + u_i$, $i = 1,2,3,..., |S|$, where g_i is the gain of the state i and u_i is the intercept of the asymptote $u_i(n)$ at $n = 0$. In the case that the Markovian Decision Process is ergodic, then g_i is the same for all i, and is called the gain (g) of the process.

If we introduce the notion of an alternative action, then by applying the **Principle of Optimality**, the formula for the expected total reward becomes

$$u_i(n) = \max_k \{ q_i^k + \sum_{j=1}^{|S|} p_{ij}^k u_j(n-1) \} \quad i = 1,2,.., |S| \quad n = 1,2,3,...$$

which states that having found which actions to employ up to transition $n-1$, the above equation is maximized by finding which action maximizes q_i^k. The action selected for each state is called the **decision** for this state and the set of decisions (vector) for all states is called the **policy**. The optimal policy is the policy that maximizes the expected total reward.

One of the (efficient) methods of finding the optimal policy is the **Policy-Iteration method**. This method consists of an iteration cycle which in turn consists of two parts a) the value-determination part and b) the policy-improvement part.

In the first part, the relative values u_i (asymptotic intercepts of $u_i(n)$) and the gain g for a given policy are calculated by solving a system of $|S|$ linear equations. This system is produced by substituting the set of equations $u_i(n) = ng + u_i$ into the set $u_i(n) = q_i + \sum_{j=1}^{|S|} p_{ij} u_j(n-1)$. The result is the set of $|S|$ equations

$$g + u_i = q_i + \sum_{j=1}^{|S|} p_{ij} u_j \quad i = 1,2,.., |S| \quad n = 1,2,...$$

with $|S| + 1$ unknowns. This system is solved by setting one of the u_i equal to zero. The values found are used in the next step of the iteration cycle.

In the policy-improvement part for each state, the action k that maximizes the quantity $q_i^k + \sum_{j=1}^{|S|} p_{ij}^k u_j$ is found. The action k becomes the new decision, q_i^k becomes q_i and p_{ij}^k becomes p_{ij}, and the iteration cycle is repeated until convergence of the algorithm is met Each step of the iteration produces a higher gain than the previous one. The cycle terminates when two successive policies are identical. It can be proven that at this point the optimal policy is found.

3.2 Time Series Analysis

Time series analysis is an analysis of dependent time-sequenced observations $z_1, z_2, ., z_t, ...$ (**time series**). The objective of the analysis is the derivation of a model which can give a) useful information about the system which generates the series, and b) optimal predicted future values of the series. This analysis, then, can be thought of as an extrapolation (projection) of past values into the future

The knowledge of past behavior can be used by many models One family of models which guarantees optimal forecasting (in terms of smaller square forecast error than any other single series) is the *ARIMA* model, which is an Integration of the Auto-Regressive (*AR*) and Moving Average (*MA*) model. We proceed to some definitions which may ease the description of the *ARIMA* models, as well as the procedures for adopting each model.

White noise a_t is a process which is transformed to the process z_t by a linear filter. It is assumed to be normally distributed.

The **deviation** from the estimated mean $\bar{z}$ is defined as $\tilde{z}_t = z_t - \bar{z}$

A backward shift operator is defined as $B^n z_t = z_{t-n}$

The first **difference** is defined as $w_t = z_t - z_{t-1} = (1-B)z_t$. The nth difference is

defined as $(1-B)^n z_t$. Differencing is employed when the mean of the series is not stationary.

The **autocorrelation function** is a function that measures the correlation between ordered pairs $(\tilde{z}_t, \tilde{z}_{t+l})$ drawn from a time series.

The **partial autocorrelation function** is a function that measures the correlation between ordered pairs $(\tilde{z}_t, \tilde{z}_{t+l})$ drawn from a time series, also taking into account the effects of the observations lying between these ordered pairs.

An **AR(p)** model of order p is

$$\tilde{z}_t = \phi_1 \tilde{z}_{t-1} + \phi_2 \tilde{z}_{t-2} + \cdots + \phi_p \tilde{z}_{t-p} + a_t$$

or

$$(1-\phi_1 B - \phi_2 B^2 - \cdots - \phi_p B^p)\, \tilde{z}_t = a_t$$

In this model, the present value of the process is defined as a linear aggregation of previous values and of white noise.

The **MA(q)** of order q model is defined as a linear aggregation of previous white noise values, a_t

$$\tilde{z}_t = a_t - \theta_1 a_{t-1} - \theta_2 a_{t-2} - \cdots - \theta_q a_{t-q} = (1-\theta_1 B - \theta_2 B^2 - \cdots - \theta_q B^q)\, a_t$$

Finally the **ARIMA(p,d,q)** model is defined as an integration (differencing) of the above models

$$(1-\phi_1 B - \phi_2 B^2 - \cdots - \phi_p B^p)(1-B)^d\, z_t = (1-\theta_1 B - \theta_2 B^2 - \cdots - \theta_q B^q)\, a_t$$

Special cases of the *ARIMA(p,d,q)* models are the *ARIMA(0,1,1)* or $z_t = z_{t-1} - \theta a_{t-1} + a_t$ which is called **Exponentially Weighted Moving Average (EWMA)**, the *ARIMA(1,0,0)* or $\tilde{z}_t = \phi_1 \tilde{z}_{t-1} + a_t$ which is a **Markov Process** and the *ARIMA(0,1,0)* or $z_t = z_{t-1} + a_t$ which is a **Random Walk**

An iterative procedure that leads to the selection of the appropriate model out of the *ARIMA(p,d,q)* family models has been developed by Box and Jenkins [BOX76]. The suggested sample size is comprised of approximately fifty observations. The procedure applies only to time series that have a mean, variance and an autocorrelation function invariant through time (stationary series) Nonstationary series can be transformed to stationary ones, often by differencing. The iterative procedure has three phases During the first phase, identification, the statistical relationships between the roughly estimated and the theoretical values of both the autocorrelation and the partial autocorrelation functions are checked, and the model with the higher correlation is chosen. In the estimation phase, a more efficient estimation of the coefficients of the model chosen during the previous phase is attempted If these estimates do not satisfy certain conditions for stationarity, invertibility, etc, the model is rejected Finally, during the last phase, diagnostic checking, the statistical adequacy (ie. analysis of the estimated random shocks) of the model is checked Failure to meet these tests causes rejection of the model and repetition of the procedure. Otherwise, the model is used for forecasting.

Having found the appropriate model and its coefficients, the future values of the series at lead time $l \geq 1$ can be forecasted given the knowledge of the series up to some origin time t The forecast of $\hat{z}_{t+l}$ is equal to $E(z_{t+l} \mid z_t, z_{t-1}, ..)$. The conditional expectations are calculated by replacing actual values for the past observations $\tilde{z}$ and random shocks a, and forecasting $\tilde{z}$ for future values. Future random shocks are approximated by zeroes.

Application of the above into an *EWMA* gives $\hat{z}_t(l) = (1-\theta)z_{t-1} + \theta\hat{z}_{t-1}(l)$. The above formula shows that small values of θ place more emphasis on the new observations than on the ones from the past. Large values of θ, on the other hand, give more emphasis to a weighted average of the past history. A fascinating thing about the

EWMA is that, while different weights can average past observations, there is no need for keeping all of them. The old forecasted value is sufficient.

Chapter 4

Previous Work

To date, there have been few efforts in the modelling of the buffer (credit) allocation in transport level protocols. We discuss three papers here which deal with the modelling of an end-to-end window flow control mechanism. This mechanism is quite similar to the one employed by TP4 and TCP.

The first paper which we discuss, KLEI80, develops a queueing model with a window flow control policy for end-to-end protocols. The queueing model employed does not provide for a closed form solution, so numerical techniques were adopted. The parameters of the system are invariant despite fluctuations in the system load. The picture below depicts the structure of the model.

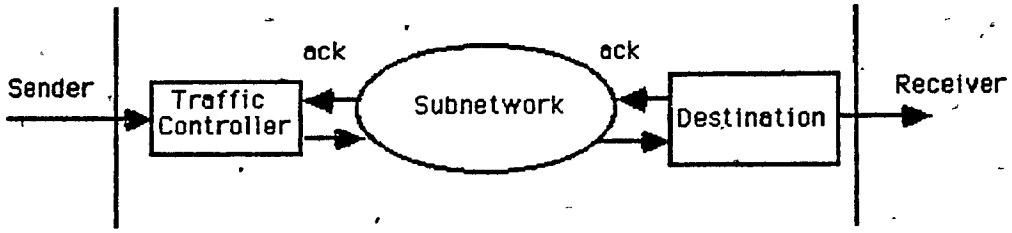


fig. 4.1

The task of the traffic controller is to keep the number of outstanding messages between the source and the destination node below a certain value (w). More specifically, before being accepted into the network, each message grabs a token which it keeps until the receipt of an acknowledgement In the case where an acknowledgment doesn't arrive before the traffic controller's specified timeout value τ, another copy of the message is sent to the network Only the last acknowledgment is accepted by the traffic

controller.

The destination node is modelled as a finite capacity queue. Lack of buffer space causes messages to be rejected without acknowledgment Duplicates are removed and while only the first copy of the message is passed to the receiver's host, acknowledgements are issued for all of them

The round trip subnetwork delay is represented by an Erlang distribution Only single packet messages are considered, and the model operates under a *heavy traffic assumption*, that is, each time the traffic controller is ready to accept a new message, the sender has one ready The model focuses on a single connection

The following assumptions were made.

a) exponential message length distribution with mean $1/\mu$

b) Poisson arrival rate with mean λ

c) independence assumption

d) the acknowledgments are i i.d drawn from Erlangian distribution of degree two

e) finite buffer space at the destination

f) negligible transmission errors

g) the arrival rate at the destination after the removal of the duplicates it is assumed to be Poisson

A brief discussion of the analysis is given below:

The acknowledgment delay distribution is given by $F_{t_a} = (1 - P_l)\,[1 - e^{-\Lambda t} - \Lambda\, te^{-\Lambda t}]$ where $\Lambda = \gamma_1 - \lambda_e$, (assumption d), λ_e is the effective network traffic rate, $\gamma_1 = \mu C_1$, and C_1 is the capacity of the channels. P_l is the probability that an acknowledgment is not received.

The retransmission probability P_r is given by

$$P_r = P_l + (1 - P_l)\, Prob\,[\hat{t}_a > \tau \mid no\ loss] = P_l + (1 - P_l)(1 + \Lambda\tau)e^{-\Lambda\tau}$$

The effective input to the network, then, is given by $\lambda_e = \frac{\lambda}{1 - P_r}$

The time a message occupies a buffer at the traffic controller is

$$T_{oc} = E[\hat{t}_a \mid \hat{t}_a \leq \tau] + \frac{P_r}{1 + P_r}\tau = \frac{2}{\Lambda} + \frac{P_l + (1 - P_l)e^{-\Lambda\tau}}{(1 - P_l)(1 - [1 + \Lambda\tau]e^{-\Lambda\tau})}$$

The maximum input rate is related to w and T_{oc} by $\lambda^* = \frac{w}{T_{oc}^*}$, where T_{oc}^* is the buffer occupancy time when the input rate is maximum λ_e now is

$$\lambda_e^* = \frac{w}{(1 - P_r)T_{oc}^*} = \frac{w\,\Lambda}{2(1 - P_l)(1 - [1 + \Lambda\tau]e^{-\Lambda\tau}) + [P_l + (1 - P_l)e^{-\Lambda\tau}]\Lambda\tau}$$

$$= G_1(\tau, w, \gamma_1, P_l, \lambda_e) \qquad (4.1)$$

The unknowns in the equation are λ_e and P_l At equilibrium, the input rate to the network is equal to the output rate of messages from it The arrival rate to the destination buffer is $\lambda_d = \frac{\lambda^*}{1 - P_l}$. The destination buffer is modelled as an M/M/1/B/ queue (B being the numbers of buffers at the destination), so

$$P_l = \frac{\gamma_2 - \lambda_d^*}{\gamma_2^{B+1} - \lambda_d^{*B+1}}\lambda_d^* B = G_2(\gamma_2, B, P_l, \lambda_e) \qquad (4.2)$$

where $\gamma_2 = \mu C_2$ and C_2 is the output channel capacity

The results are obtained by solving the equations 4.1 and 4.2 iteratively for different sets of network parameters, ie (τ, w, μ, C_1, C_2) These parameters are not adjusted dynamically and hence, we have a model of static flow control

Finally, the end-to-end delay (T_{ee}) is the sum of the average (T_{oc}^*) delay, and the destination buffer delay (T_{del}^*)

$$T^* = \frac{1}{\gamma_1 - \lambda_e^*} + \frac{P_l}{1 - P_l}\,\tau$$

and

$$T_{dst}^* = \frac{(1/\mu\, C_2)}{1 - \rho_d^*}\;\frac{1 - (1+B)\rho_d^{*B} + B\,\rho_d^{*B+1}}{1 - \rho_d^{*B}}$$

where $\rho_d^* = \lambda_d^*/\mu C_2$ is the utilization at the destination

The analysis demonstrates that.

1) for a fixed w, throughput increases with timeout τ until τ reaches a certain value, whereupon throughput begins to decrease

2) for fixed values of τ, throughput increases as w increases, but after a certain value of w it starts to decrease and,

3) for a fixed end-to-end delay, throughput increases with buffer capacity

Also, the analysis indicates that for a fixed value of w there is a value of τ that maximizes throughput

In the second paper we discuss, KERM80, the authors use the results of the previous paper on static flow control and develop an algorithm in which the window is computed dynamically As pointed out by the authors, the key idea behind this model is to prevent input to the network when the destination buffer is full This in turn prevents future retransmissions and rejections of messages, and as a consequence, unnecessary loading of the network The mathematical model developed is based on Markovian Decision theory. Since the exact solution grows exponentially, the authors develop a heuristic solution. This model also focuses on a single connection

The structure of the model is basically the same as the one presented in the previous paper. The difference is that at the destination there is a traffic director which, by looking at decision tables, can choose the optimal token limit and then can notify the

traffic controller to increase (or decrease) the token limit if the buffer occupancy at the destination decreases (or increases). This information is carried by control messages.

The destination buffer is modelled as an M/M/1/B queue. The input to the network is assumed to be Poisson with rate $\lambda \in \Lambda$, where $\Lambda = \{\lambda_1, \lambda_2, \ldots \lambda_K\}$ is the discrete set of arrival rates.

The message lengths are assumed to be exponentially distributed with mean $1/\mu$, as are the transmission rates out of the destination buffer ($\gamma_1 = \mu C_2$).

The model is formulated as a discrete time Markovian Decision Process [HOWA60]. Furthermore, only one arrival or departure can occur at each time slice δ, that is, each message arrives at the buffer with probability $\lambda\delta$ (Bernoulli trial). The average round trip delay is T_r and the discrete subdivision $\hat{T}_r = T_r/\delta$.

As was mentioned before, the traffic director decides on the token limit and then notifies the traffic controller. There is a random gap, however, between the time the decision is made and the time at which the new traffic arrives at the destination. This random gap makes the analysis difficult, so it was assumed that this random gap is equal to the round trip delay T_r. With this assumption, the decision that was made at time t becomes effective at the destination at time $t + T_r$. That is, the decision made at time t should be based on the buffer occupancy at time $t + T_r$ (T_r-*cycle-delay* MDP). This is reflected in the system state definition which is written as

$$X(\hat{t}) = \{n(\hat{t}), \hat{\lambda}(\hat{t},1), \hat{\lambda}(\hat{t},2), \ldots \hat{\lambda}(\hat{t},\hat{T}_r - 1)\}$$

or in short

$$i = \{n_i, \lambda_i^1, \lambda_i^2, \ldots \lambda_i^{\hat{T}_r - 1}\} \qquad (4.3)$$

where $0 \leq n(\hat{t}) \leq B$ is the buffer occupancy at time t ($\hat{t} = t/\delta$) and $\hat{\lambda}(\hat{t},k)$ is the input rate to the buffer at time $(t+k)$, $1 \leq k \leq \hat{T}_r$.

The cardinality of the set of states S is given by $|S| = (B+1)K^{(\hat{T}_r - 1)}$, where K is the set of different token limits.

The set of actions of the MDP is the set of different windows $W = \{w_1, w_2, \ldots, w_K\}$. As was proven in [KLEI80], for fixed w a timeout τ can be chosen to maximize the throughput of the network. The input rate set Λ can also be considered as the set of actions, since each (w, τ) determines the input rate on an individual basis.

The state transition probability for a policy f from a state i to a state j is $p_{ij}(f) \equiv Pr[X(\hat{t}+1) = j \mid X(\hat{t}) = i \text{ and policy } f \text{ is employed}]$

A policy is a decision rule which states that given the system in state i, the $\hat{\lambda}_k$ is used as the arrival rate to the buffer and this rate will become effective after $\hat{T}_r$ time units.

The reward function R is defined as $R = a\lambda - T$, $a \geq 0$, λ and T being the throughput and the delay of the system respectively.

The immediate expected reward reflects the gain due to the throughput and the loss due to the delay It also reflects the delay due to retransmission after the message is rejected The formula is given below:

$$\hat{R}_i(f) = \begin{cases} a\hat{\lambda}_i^1 - (n_i + \hat{\lambda}_i^1 \hat{T}_i^1) & n_i < B \\ a\hat{\lambda}_i^1 \hat{\gamma}_2 - (n_i + \hat{\lambda}_i^1 \hat{\gamma}_2 \hat{T}_i^1 + \hat{\lambda}_i^1 (1-\hat{\gamma}_2)\hat{\tau}_i^1) & n_i = B \end{cases}$$

The average reward per unit time is given by

$$g(f) = \lim_{n \to \infty} \left\{ \frac{1}{n+1} \sum_{t=0}^{n} [P(f)]^t R(f) \right\}$$

where the $[P(f)]^t$ is the t-step probability transition matrix under policy f. Since the action and the state space is finite and the reward function is bounded, there exists a

stationary policy under which the expected average reward per unit time is maximized.

This model can be solved by the policy iteration method [HOWA60]. But even for small values of parameters the state space grows rapidly An example given by the authors gives a very good idea of the rate of the space growth. They state that for $B = 10$, $K = 5$, $T_r = 0.2s$ and $\delta = 4$ms, then $\hat{T}_r$ becomes 50 and $|S| = 11X5^{49}$ Given that at each step the method requires the solution of $|S|$ linear equations, the solution of the problem becomes computationally infeasible A heuristic solution based on the following consideration was developed. the time for the decision to become effective can be reduced if the decision process is provided with the expected buffer occupancy at $\hat{T}_l$ time units later (look ahead time). With this, the new (reduced) state is defined as

$$X'(\hat{t}) = \{\bar{n}(\hat{t} + \hat{T}_l), \hat{\lambda}(\hat{t}, \hat{T}_l + 1), \hat{\lambda}(\hat{t}, \hat{T}_l + 2), .., \lambda(\hat{t}, T_r^* - 1)\}$$

An optimal policy can be found (if the new state space is not too large) for this process. By using this policy, a suboptimal policy for the original policy can be found.

The decision table for the (reduced) process can be set up after the optimal policy is found. For each state of the $(\hat{T}_r - \hat{T}_l)$-*cycle-delay* process, the optimal token limit is stored. The size of this table is $N_{tbl} = (B + 1)^{K^{(T_r - T_l - 1)}}$ By storing the current state, the traffic director can decide on the token limit by calculating the expected buffer occupancy and by setting up a vector similar to that of equation 4.3. Two factors should be considered while choosing the look ahead time, $\hat{T}_l$. the optimality of the decisions made, and the fact that the reduced decision process should result in a computationally feasible solution. Large values of $\hat{T}_l$ may result in far from optimal decisions (but computationally feasible solutions). On the other hand, small values of $\hat{T}_l$ result in close to optimal decisions. In this case, the problem becomes infeasible due not only to

the large state space but also to the multi-chain behavior of the system.

The results obtained are for values of $\hat{T}_l = \hat{T}_r - 1$ *(1-cycle delay)* process. In this case, the state of the system is obviously the buffer occupancy. These results show that the value a of the reward of the system can used to control the throughput. Also the largest value of the token limit indicates the number of buffers the source node needs. The largest and the smallest token limits are controlled by the value of a. Furthermore, the dynamic algorithm gives slightly better results when compared to the static one.

In the third and last paper we discuss, HARB82, we find a model of the Network Control Protocol (ARPANET's former transport protocol) which assumes a reliable subnetwork. Multiple connections are portrayed (as opposed to the single ones of the previous papers) and constant acknowledgment delays are assumed. A Markov decision model is developed to solve the model. The presence of the curse of dimensionality is strong here too, due to the exponential growth in the number of connections of the state space. The model's structure is depicted in figure 4.2.

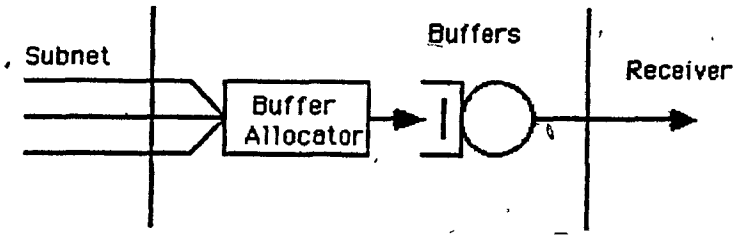


fig. 4.2

The destination node is modelled as an M/M/1/K queue, that is, a single server queue with finite buffer capacity, Poisson arrival rates of messages, and exponentially distributed (with mean $1/\mu$) service times.

The state space of the process is defined as:

$$S = \{<a_1^1,..,a_c^1;m_1^1,..,m_c^1;n^1>,..,<a_1^N,..,a_c^N;m_1^N,..,m_c^N;n^N>\}$$

where

a_k^i is the allocation for sender k while in state i

m_k^i is the number of messages received from k while in state i

n_i is the number of buffers occupied in state i

c is the number of connections

N is the number of states

the arrival rate of messages due to sender k in state i is given by

$$\lambda_k^i = \begin{cases} 0 & n^i = B \quad or \quad a_k^i = m_k^i \quad or \quad A_k^i = 0 \\ \lambda_k & otherwise \end{cases}$$

where B is the total number of buffers at the destination host. As we can see from this definition, the traffic from the senders is cut off when all the buffers are occupied. The authors claim that this assumption is justified whether the control messages flow on separate channels or whether they are prioritized.

The cardinality of the state space is given by

$$|S| = \left[\left(\frac{(B+1)(B+2)}{2} \right)^c - B^c \right] (B+1)$$

The action space $<A_1^i, \ldots, A_c^i>$ represents the new allocations for each connection in state i. Whenever $A_k^i = a_k^i$, then there are no control messages issued. In the case $A_k^i > a_k^i$, the destination has issued an ALL message. Finally, $A_k^i < a_k^i$ models the issuance of a GVB message by the destination. The RET messages are not modelled.

The immediate reward function is defined as

$$R^i = -\alpha R_d^i - \beta R_c^i + \gamma \sum_{i \neq j} R_i^{ij} T_{ij}$$

where

R_d^i is the reward per time unit due to delay in state i

R_c^i is the reward per time unit due to control message overhead while in state i

R_t^{ij} is the reward per time unit due to message throughput from state i to state j

T_{ij} is the transition rate from state i to state j

α,β,γ are normalizing constants

We can see from the above formula that a policy is penalized for the control message overhead and the delay, while it is rewarded for the throughput

In addition, the reward terms can be defined in the following manner:

$R_d^i = n^i$ per unit time

$R_c^i = r$ if for r links $a_k^i \neq A_k^i$

$$R_t^{ij} = \begin{cases} 1 & \text{if } n^j = n^i - 1 \\ 0 & \textit{otherwise} \end{cases}$$

$$T_{ij} = \begin{cases} \lambda_k & \text{if } n^{ij} = n^i + 1, \quad m_k^j = m_k^i + 1 \\ \mu & \text{if } n^j = n^i - 1 \\ 0 & \textit{otherwise} \end{cases}$$

As we mentioned before, the curse of information is present here too. An example given by the authors for 3 buffers and 2 connections, results in 364 states. Although the authors claim that the policy iteration method needs to be employed only once for each arrival rate, since in on-line applications buffer assignments are based on tables already set, the memory requirements for any reasonable set of arrival combinations will still grow quite high

The problem encountered in the previous models is not unique to these models. For any real application modelled by an MDP, the number of states is very large [HOWA78] (**curse of dimensionality**). **Acceleration** techniques [HEYM84] can be adopted in

order to speed up the computations. **Aggregation** techniques [HEYM84] have also been developed in order to reduce the state space by replacing the original process with an approximate one. The problem remains, however, since for any practical application, the resulting model is very large

Chapter 5

Buffer Allocation Algorithm

5.1 Observations

In the last chapter we presented the previous work done for the process-to-process protocols. The models defined, although they are valuable and reasonable (despite the simplifying assumptions they rely on), are computationally and memory-wise infeasible. This infeasibility results from both the size of the information available (combinatorial explosion) as well as the fact that special characteristics of each network are not taken into consideration

A closer look at various operational networks reveals that different kinds of services are offered for different applications (ie interactive, batch, terminal, or expedited data). Furthermore, measurements taken at different times on various networks lead to a conclusion that these categories possess some arithmetic values for the message (packet) size, connection time, etcetera, not considerably varying from the mean.

In the Merit network [AUPP83] for instance, there are four different types of access (traffic) offered to the users: host-to-host interactive, direct terminal, batch, and external (execute, print) access, each having its own distinctive properties. Terminal traffic accounts for 40% of the overall measured traffic. The number of bytes per packet per traffic type, the packets per connection per traffic type, and the connection time per traffic type do not vary significantly from the mean. In addition, the average packet size for all traffic types has remained almost constant throughout the ten years of the network's operation. Similar observations can be made in other networks.

As an example of similar observations, let us present measurements from the ARPANET.

Early measurements performed on the ARPANET [KLEIN76], show similar results for the message size (243 bit, 1.2 packets/message), and the round trip delay (90 msec). Moreover, one third of the sites generates 80% of the total traffic, while the most popular destinations receive 44% of the traffic, which is most heavy during weekday working hours.

In recent measurements of the same network, [COHN83], the performance of the NCP and the TCP is compared at two different measuring periods, 1982 and 1983. Among other observations are: a 58% increase in the average message size, a 36% increase in the average utilization, and a 23% increase in the round trip delay. Although there are significant differences between these measurements, the ARPANET community agrees that the changes are mainly due to the conversion of the transport protocol from NCP to the TCP, and to a lesser extent to the changes in the traffic scenario.

The above observations lead us to conclude that in each network there are distinctive classes of traffic and traffic patterns, as well as specific characteristics such as message size and connect time for each type of traffic, that do not change drastically over time. Taking these factors into consideration, a forecasting (prediction) approach to the problem seems reasonable.

5.2 The Algorithm

The major purpose of the buffer allocation algorithm is to assign buffer space to a connection as a function of the estimated queue length for different categories of traffic. The traffic categories are determined by the length of the messages sent along a

connection. An example of such a category is interactive traffic, which would be several characters long. File transfers would be of varying lengths, depending upon the application programs run at the hosts Furthermore, buffer space assigned to a category is accessible to any category of smaller message length (that is to say, smaller messages have higher priority). It is assumed that statistics on the frequency of different length messages are collected by the network. It is also assumed that the transport level protocol in each host (gateway) keeps statistics on the queue length for each category of network traffic.

In the context of OSI-style networks, each connection can then identify itself as belonging to a particular category, eg., interactive or file transfer For a datagram style gateway, it would be necessary for the gateway to determine the packet length of each packet from information contained in its header, or by directly determining its length.

The algorithm may be divided into two parts:

- Prediction (via exponential smoothing)
- Allocation (and deadlock avoidance)

In the prediction portion of the algorithm, the expected queue length for the different categories of traffic is forecasted.

During a measurement period $(t-1)$ of fixed length, the queue lengths $z_{(t-1),i}$ on the part of each category i of traffic are accumulated. At the end of the measurement period, the numbers accumulated for each of the categories are inserted into the exponential weighted moving average formula (EWMA) which we employ to predict the future requests for each category. The EMWA formula is given by

$$\hat{Z}_{t,i} = \alpha * z_{(t-1),i} + (1-\alpha) * \hat{Z}_{(t-1),i}$$

where

$\hat{Z}_{(t-1),i}$ is the previous estimate of the queue length for category i up to period $(t-1)$,

$z_{(t-1),i}$ is the measured queue length during the $(t-1)$ period

$\hat{Z}_{t,i}$ is the predicted queue length for period t

α is a parameter chosen between 0 and 1. Chosing α closer to 0 emphasizes the past, while chosing α closer to 1 emphasizes more recently made measurements. The value of the α can be estimated experimentally, that is, by comparing forecasted and observed values.

A brief discussion on time series analysis is given in chapter 3. The interested reader, however, will find an extensive and comprehensive discussion of the topic in BOX76.

In the allocation portion of the algorithm, the maximum buffer space allocated to each category $b_{t,i}$ is determined. The allocation for each category i is then the floor of the predicted queue length for each category. That is,

$$b_{t,i} = \lfloor \hat{Z}_{t,i} \rfloor$$

In the case that there is buffer space left over, this space is then made accessible to all connections on a FCFS basis.

Should there be a time period during which certain traffic is sporadic while other traffic is very heavy, the queues for the sporadically arriving traffic will not build up, while those for the heavy traffic will monopolize the allocatable buffer space In order to prevent starvation of the sporadic traffic, a minimum allocation is granted to each category. This minimum allocation can be network and/or implementation dependent. A minimum allocation equal to the sporadic traffic's average message length gives best results. A more detailed discussion of this policy can be found in chapter 7, results.

In the event that the receiving transport station buffers data, a deadlock avoidance algorithm is combined with the allocation portion of the algorithm so that the very real

possibility of reassembly deadlock is avoided The destination host buffers the incoming segments of data until the last segment arrives and then hands off the entire message to the receiving process Since buffer space may be allocated to partially received messages (depending upon their length), it is possible that the entire complement of space allocated by the host to the transport protocol is occupied by partially received messages Since none of these messages may be delivered to the receiving processes, we have the classic reassembly deadlock !

To avoid this deadlock, we employ the following algorithm

```
procedure avoideadlock,
 begin
      if new_request <= buffers_not_occup
          bf_allocated  = new_request
      else if new_request <= buffers
          bf_allocated := (buffers_not_occup − bf_needed_i )
      else refuse_allocation
 end,
```

where

new_request is the total number of buffers needed by a process, demanded during the connection establishment

buffers_not_occup is the number of buffers which are not occupied at the time the new request is made

buffers is the total buffer space at the receiver

bf_needed_i is the number of buffers needed, in order to be completed, by the i-th request which was partially satisfied (requests are labelled in increasing order)

bf_allocated is the current number of buffers allocated to a new request

We will prove the following *proposition:*

A request, once accepted, has an assured departure.

Proof

For an incoming request $i+1$, two situations may occur

either $bf\_needed_{i+1} = 0$ in which case the request is satisfied by the first condition of the algorithm and eventually will depart, or the $new\_request > buffers\_not\_occup$. In this case we claim that $bf\_needed_{i+1} > bf\_needed_i$ The proof for this is

$$\begin{aligned} bf\_needed_{i+1} &= new\_request - bf\_allocated \\ &= new\_request - (buffers\_not\_occup - bf\_needed_i) \\ &= (new\_request - buffers\_not\_occup) + bf\_needed_i) \end{aligned}$$

Since $(new\_request - buffers\_not\_occup) > 0$, $bf\_needed_{i+1} > bf\_needed_i$

At the arrival of the $(i+1)$-th (partially satisfied) job, the algorithm guarantees that the i-th job will depart $(bf\_allocated = buffers\_not\_occup - bf\_needed_i)$ Since $bf\_needed_i > bf\_needed_j$ for $i > j$, any incomplete job is also guaranteed to depart (as long as the buffers are allocated to it in such a way that the needs of a waiting job are completely satisfied before the allocation proceeds to the next job) ■

In the event that a partially satisfied job already exists and an allocation is not granted to a newly arrived job until the proceeding one is finished, then the algorithm operates on a fcfs basis

As we can see, the algorithm needs knowledge of either the size of the message or a segment of the message at connection establishment In TP, this information can be obtained from knowledge of the TPDU size and the suggested sequence space Otherwise, an additional field in the variable part can be used In TCP/IP, this size can be obtained directly from the IP header

Chapter 6

Simulations

6.1 Network Model

Figure 6.1 below portrays the network structure of the model employed to evaluate the allocation algorithm.

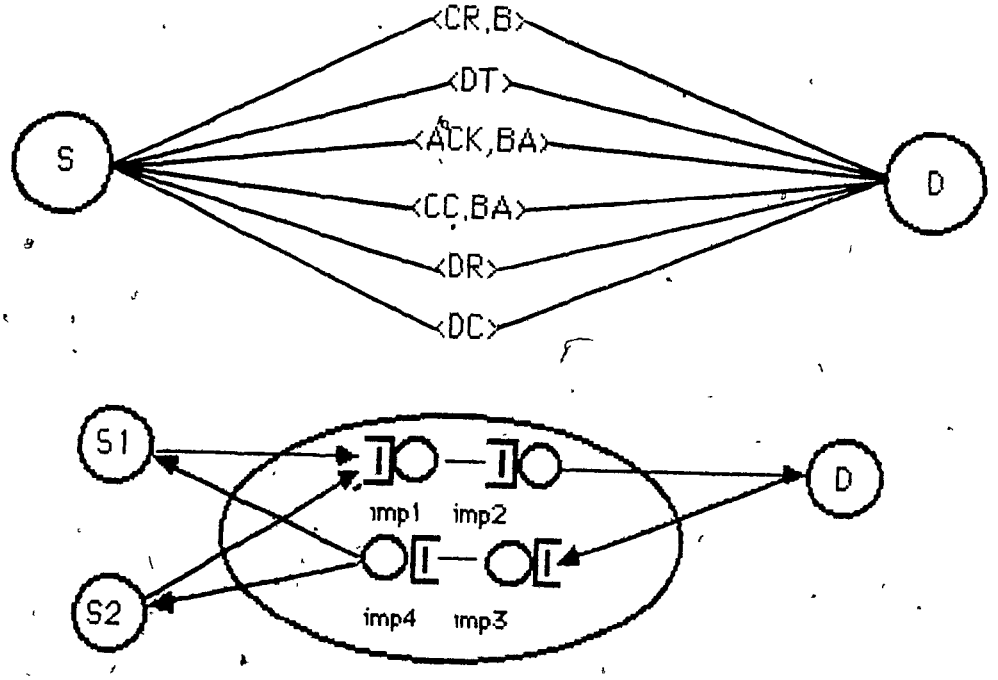


fig 6.1 TP Model

As can be seen from the figure, we represent two transport level connections (C_1, C_2) originating at sending stations S_1 and S_2 and terminating at receiving station D. The messages on both connections travel through the network along the top route through switches (imps) *1* and *2* where they are passed to the receiving transport station. Acknowledgements return along the bottom path via switches *3* and *4*.

Message lengths on both connections are chosen from exponential distributions. The mean message length on C_1 is 512 bits, while on C_2 60% of the messages are 1024 bits, 20% are 2048 bits and 20% are 4096 bits. Acknowledgements are 80 bits long. The buffer size at the receiving station is 512 bits, while there is a total of 16 buffers.

We assume that the link data rates are 9.6 Kbps and that the lines are free of errors. The data rates from the destination host to the end user are assumed to be 2.4 Kbps.

Access to the network is controlled via a fixed ($= 8$) window mechanism. Within the subnet we employ a fixed timeout retransmission protocol between switches in the event of buffer overflow at the destination switch.

Figure 6.1 also depicts the part of OSI class 4 transport protocol (TP4) which we simulate. As can be seen, connection establishment is accomplished by exchange of a control packet <CR,B> (connect_request, buffer_space_needed). The other side replies with a <CC,BA> (connect_confirm, buffer_space_allocated) in the event that it agrees to open the connection, or with <DR> (disconnect_request) if it doesn't. In the latter case, the sender tries to open the connection later by employing the same sequence of control packets.

Once the connection is established, the sender sends the data packets - DT -. The receiving station then replies with <ACK,BA> (acknowledgement, buffer_allocated).

In order to terminate the connection, the sender employs a - DR - and the receiver replies with <DC> (disconnect_confirm).

In order to evaluate the algorithm, we compare its performance to FCFS under varying traffic loads and for different values of α. The parameter α controls the emphasis on the most recent measurement period -- as α increases, the significance of the most recent period increases.

6.2 Simulation Model

As indicated in the introductory chapter, we employed the Performance Analyst's Workbench System (PAWS) as our simulating tool in evaluating different buffer allocation algorithms [PAWS].

PAWS is a simulation language which supports high level primitives such as polling or last come first served queueing policies, best or first fit memory management disciplines, or hyperexponential, erlangian distributions, thus releasing the user from the details of coding, and permitting concentration on the model itself The language also encourages and supports the pictorial representation of the model, or Information Processing Graphs (IPGs) as they are called in PAWS, as PAWS translates and evaluates these IPGs directly The IPGs consist of nodes and edges. Nodes represent the place in which information is processed, while edges signify the information flow from one node to the other. The basic unit of information is the transaction. A transaction is data that can be created, processed, moved from one node to another, or destroyed.

Figure 6 2 portrays the IPG of our model, and consists of three parts:

a) the source host

b) the communication subnet

c) the destination host

Our description is divided accordingly.

a) The Source Host

Transactions representing messages are created at node GEN according to a Poisson arrival Process Node CTL0 serves the purpose of controlling the input, in order to avoid saturation of the system. At node MSGPARM, a message length is assigned to

each transaction. After this node, a connection establishment with the destination node is attempted by creating a control packet at node MSGSPLIT. At node PPARM each packet is assigned its own parameters. The original transaction waits at node MSGTO until the reply of the other side returns. The nature of the reply causes the original message to be split into more packets (nodes PFORMAT and MSGSPLIT) or to wait for buffer space allocation At node MORETPDU, the message is split into smaller units, which are transmitted separately, in case the destination can not store (reassemble) the complete message

Each packet leaving the MSGSPLIT node goes through the node ALLW1 or ALLW2 These nodes are used to simulate a fixed window flow control policy at the network level. Each packet should grasp a credit, before its admittance to the network, which is returned back through node RELW upon the receipt of the acknowledgment. Next in the row is the node BACKUP There, a copy of each packet is kept for retransmission in case the acknowledgment doesn't arrive prior to the expiration of the timer The node TIMER is used for this purpose.

Finally, the SENDER nodes represent the communication links from the sender processes to the network

In addition, throughout the diagram the COLLECT nodes, in conjunction with their corresponding BH nodes, represent the exit of a transaction from the system.

b) The Communication Subnet

The IMP nodes represent the links from one switch to another in the network A finite buffer space is associated with each link, hence packets are retransmitted in the event that no acknowledgment is received after a time-out period. This is handled by the

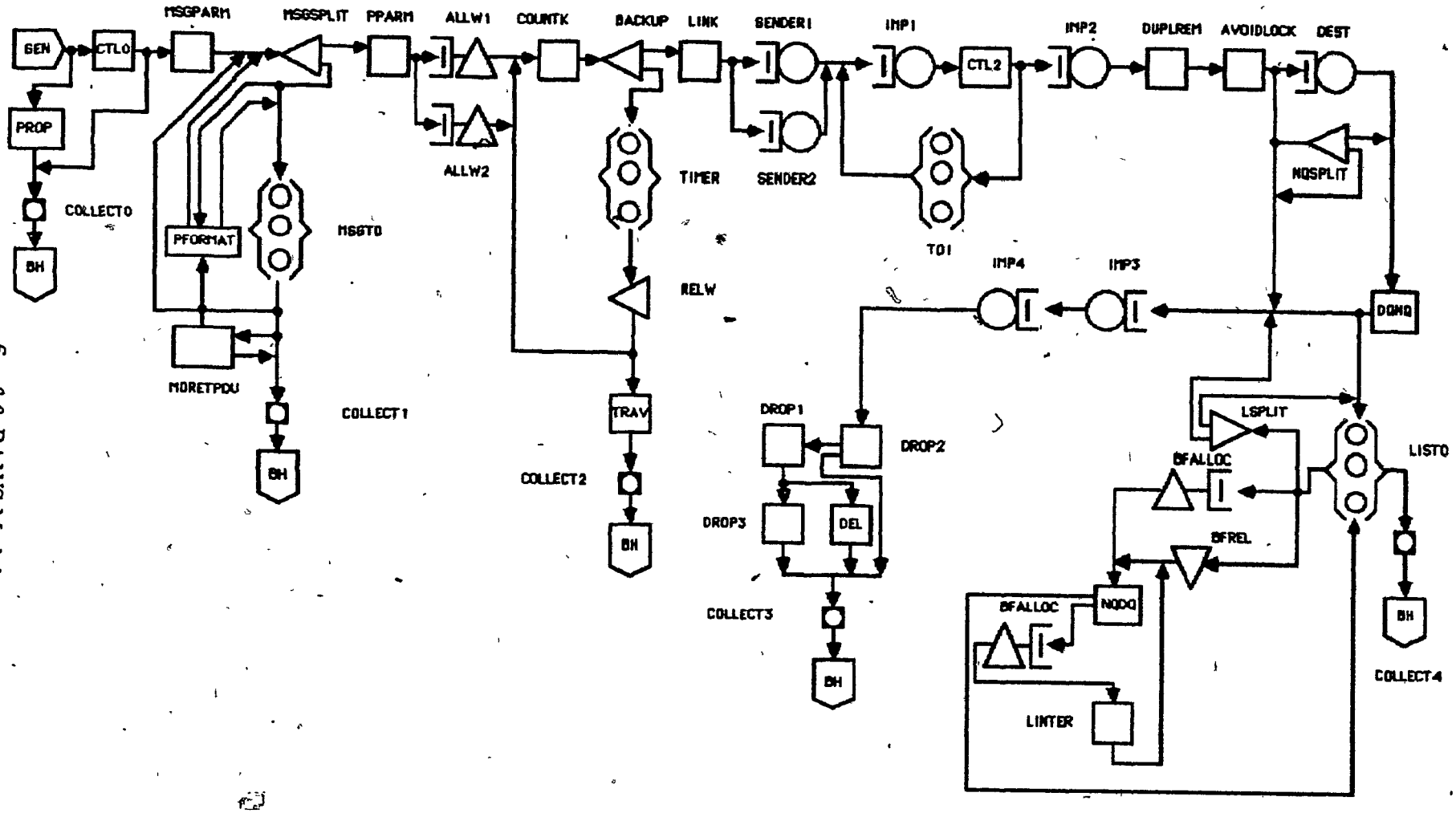


fig. 6.2 PAWS Model

CTL2 and TO1 nodes IMP nodes 1 and 2 represent one direction of data flow, while IMP's 3 and 4 represent the reverse direction of flow. The DROP and DEL nodes serve the purpose of notifying the sender of the successful delivery of a packet or a message. They accomplish this by delivering (to the sender) the control packets.

c) The Destination Host

This is where the buffer allocation algorithm resides. The DUPLREM node serves the purpose of removing the duplicate packets, while node AVOIDLOCK keeps track of the incoming requests for each message category We simulate two categories of messages, long and short, single TPDU's and multiple TPDU'S The exponential smoothing and split of the buffer space is done at node PROP The deadlock avoidance algorithm resides at node AVOIDLOCK, while buffers are allocated and released through nodes BFALOC and BFREL respectively. Nodes LISTO, DQNQ, and NQDQ keep track of the current buffer space allocated and the relevant details for each packet/message Acknowledgment, control and allocation packets are returned by the NQSPLIT, LSPLIT, and LINTER nodes Finally, completely received messages are delivered to the end user through the node DEST.

Chapter 7

Results, Conclusion, Suggestions

7.1 Results

As previously indicated, the buffer allocation algorithm was compared (when appropriate) to the one adopted by TCP and TP (which is called here fcfs) The simulation package employed in order to carry out the simulation was PAWS, and was run on a VAX 11/780 The average simulation length for each run was three CPU hours. The results presented constitute a small but representative subset of the runs. Averaged batch values are presented

Our results, depicted in the graphs at the end of the chapter, are for

a) different interarrival rates (graphs 7.1a to graphs 7.1j),

b) varied minimum allocations for the buffer space, a factor which becomes relevant under extreme conditions, (graphs 7.2a to 7.2b)

c) different values of α (graphs 7.3a to graphs 7.3h).

The performance values of interest are the end-to-end delay and throughput. The graphs 7 1 and 7.2 depict four curves, one for each category of traffic under each algorithm, while the graphs 7.3 depict two curves, one for each category. The letter E in the graphs stands for exponential, the F for fcfs, the S for short, and the L for long.

a) different interarrival rates

For graphs 7.1 the average message length for long messages is 4.6 packets/message

while the short messages have 1 packet/message. Each packet is 512 bits. The exponential smoothing parameter α has the value of 0.6, while the measurement period during which the statistics are accumulated is 40 seconds. The interarrival rates for short messages (λ_1 seconds) and for long messages (λ_2 seconds) take values from the set $\Lambda = \{0.1, 0.5, 1.0, 1.5, 2.0\}$ This set was chosen in order that all possible input rates, from high utilization ($\sim$ 90) to low utilization ($\sim$ 30) may be covered The receiving station has 16 buffers (512 bits each).

1) End-to-End Delay (graphs 7.1a to graphs 7.1e)

Starting from graph 7.1a where $\lambda_1 = 0.1$, we notice that when the utilization is heavy ($\sim$ 96%) there is a small gain ($\sim$ 8%) for short messages which diminishes to fcfs at low ($\sim$ 62%) utilization On the other hand, long messages pay the corresponding price up to $\sim$ 15%

In the next graph (7.1b) where $\lambda_1 = 0.5$, we observe that for $\lambda_2 = 0.1$ and $\lambda_2 = 0.5$, there is a small gain for short messages while longer messages again register a corresponding loss. At the point where $\lambda_2 = 1.0$, the gain is becoming noticeable ($\sim$ 21%) and the loss for the longer messages is about 6% At lower utilizations ($\sim$ 56%) there is still a gain, however, since the values are small and the variances are large this difference is not significant.

In the graphs (7.1c, 7.1d, 7.1e) where the short messages arrive at a lower rate (λ_1 is 1.0, 1.5, and 2.0 respectively), we notice that there are points where the gain becomes much more visible while long messages seem to gain as well

The phenomena are explained thus:

When both short and long messages arrive at a very high rate, there is an opportunity for short messages to avail themselves of their required buffer space even under

fcfs. This is why the gain under the algorithm is not impressive.

There are certain combinations of interarrival rates (especially when short messages arrive at a relatively lower rate as compared with longer messages) where under the fcfs policy there is a distinct possibility for shorter messages not to receive buffer space due to their being overwhelmed by the longer messages' demands On such occasions the algorithm shows its value, in that at all times it guarantees buffer allocation to the shorter messages (in varying amounts according to the predicted buffer occupancy) while at the same time, not preventing them from occupying empty buffer space in the section set aside for longer messages.

The gain noticed for the longer messages results because better buffer management is accomplished by the reservation algorithm.

2) Throughput (graphs 7.1f to graphs 7.1j)

We notice an insignificant difference between the reservation algorithm and the fcfs (a very small gain for short messages accompanied by a very small loss for longer messages). This is indicative of the algorithm's fairness. The small loss for the longer messages is the result of the gain, in both the delay and throughput, for the short messages.

b) varied minimum allocations

Graphs 7.2a to 7.2b deal with the (extreme) case where one traffic stream is sporadic and the other is frequent. In this case, the prediction indicates that all buffer space is taken by the heavier traffic stream, while the other stream experiences starvation. The algorithm was tested for different values of minimum allocation; one buffer, two, and three (shown in the graph labelled mintpdu) We assume that $\lambda_1 = 0.005$, $\lambda_2 = 5.0$, $\alpha = 0.6$, and the total number of buffers is equal to 4. Finally, the average

message length for sporadic traffic is equal to 3 packets/message, while short messages have 1 packet/message.

We observe in the graphs that with the value of minimum allocation being equal to one, the sporadic messages suffer a big delay and a corresponding degradation in throughput, since the small allocation necessitates their being split into multiple one-packet messages.

When the minimum allocation is equal to two, the loss for sporadic messages in delay and throughput is almost halved Contrary to what one might initially expect, the short messages exhibit a general, albeit small, improvement under this type of allocation. This becomes obvious when one realizes that the longer messages are now split into fewer packets, which translates into fewer control packets and their spending less time occupying the available buffers.

With the minimum allocation set at three, the results show neither significant improvement nor deterioration over the fcfs. In general, large average message lengths for sporadic traffic and small values for minimum allocation create problems all around. A value of minimum allocation close to average message length seems to present the most desirable situation.

c) different values of α

This section focuses on the effect upon the results when different values of the exponential smoothing factor α are introduced. Specifically, the values used are: 0.1, 0.3, 0.6, 0.9. The average message length for each category, as well as the number of buffers at the receiving station, is the same as in graphs 7 1.

For graph, 7 3a, where $\lambda_1 = 0.5$ and $\lambda_2 = 0.1$, we notice that under high utilization

($\sim$ 95%), the higher values of α give better results for delay. The same holds for the throughputs portrayed in graph 7.3e. The results for both are explained by the fact that under higher values of α, the algorithm reacts faster and naturally gives better predictions. In graphs 7.3b and 7.3f (delay and throughput respectively), where $\lambda_1 = 0.5$ and $\lambda_2 = 0.5$, the pattern mentioned above is repeated.

Turning our attention to graph 7.3c, where $\lambda_1 = 0.5$ and $\lambda_2 = 1\ 5$, we see that under lower utilizations ($\sim$ 75%), smaller values of α, ie. $\alpha = 0\ 3$, give better results for delay. The same is true for the throughput of the long messages (graph 7 3g), while the results for throughput of short messages do not differ significantly Furthermore, simulation runs attempting different values of α for each category of messages (not shown in the graphs) showed that the value of α for short messages is important while the corresponding α value for the long messages is irrelevant. This is explained if we note that 1) the higher the value of α for short (frequently arriving) messages results in a larger allocation, and that 2) short messages can acquire and use the buffer space initially allocated for the large messages, but not vice versa. Hence, a high value for α predicts a larger allocation which is taken from the longer messages' space. Consequently, small values of α give better results. This points out the importance of selecting the values of α based upon arrival rates and buffer utilization.

Finally, from graphs 7 3d and 7.3h, where $\lambda_1 = 1.0$ and $\lambda_2 = 2.0$, we notice that for low utilization ($\sim$ 38%) it does not appear necessary to optimize the value of α since the variances in all cases are quite large and the delays are small.

We conclude from these graphs that:

- at low to moderate utilizations the choice of α does not seem to significantly affect the results
- at high utilizations higher values of α seem to give better performance but different values of α should be assigned to the categories depending upon their arrival rates.

Simulation runs for connectionless transmission of short messages showed no significant difference over the connection-oriented transmission. This is explained if we take into consideration that the gain resulting from the elimination of the connection establishment phase can be diminished due to the extra queuing delay of the complete messages.

7.2 Summary and Conclusion

We have presented in this thesis a buffer allocation algorithm for use in a transport protocol such as ISO's TP class 2,4 or TCP. The objective for the algorithm was that of obtaining a fast response to short messages while maintaining an acceptable throughput for longer messages.

The algorithm allocates buffer space as a function of the estimated queue length for each category of traffic (categories are user defined) thereby ensuring fairness on its part. Furthermore, categories of smaller message length can acquire and use buffer space, if available, initially allocated to categories of longer message length, but not the other way around. This way, it is guaranteed that for short messages the algorithm operates at least as well as the one using fcfs discipline. The prediction was done via exponential smoothing. A deadlock avoidance algorithm was also employed to avoid the possibility of reassembly deadlock.

The algorithm was compared by simulation to a fcfs discipline. Different traffic scenarios, exponential smoothing factors (α), and buffer spaces were examined in the course of the simulations.

- Under heavy traffic conditions, and especially when short messages arrive at a relatively lower rate as compared with longer messages, the algorithm resulted in a gain in

the end-to-end delay, and throughput for shorter messages as compared to fcfs. This was the result of reservations being made for the shorter messages, thus avoiding dominance of the buffer space by longer messages On the other hand, there is a degradation in the performance for the longer messages which pay the price for the gain of the shorter messages. Under low utilization the algorithm fluctuates around the levels achieved by the fcfs strategy.

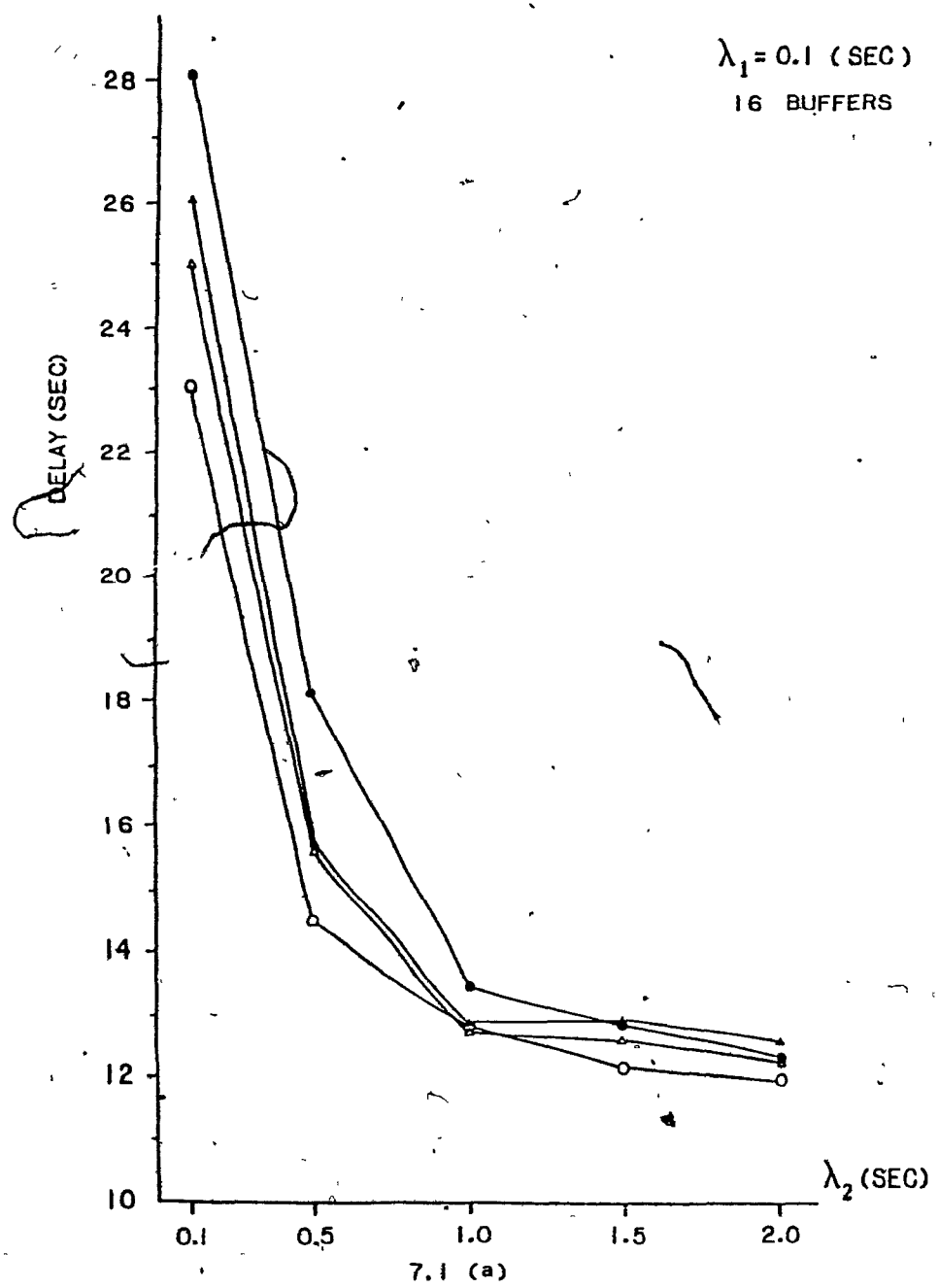
- In the case of sporadic (unpredictable) traffic, the algorithm degenerates to fcfs when the values of minimum allocation are close to average message length for this traffic.

- The choice of α should be made according to utilization For low utilization, the choice of α is largely irrelevant. At high utilizations, larger values of α seem to give better results Careful consideration should therefore be given to chosing α for each category as a function of the arrival rate of messages in each category as well

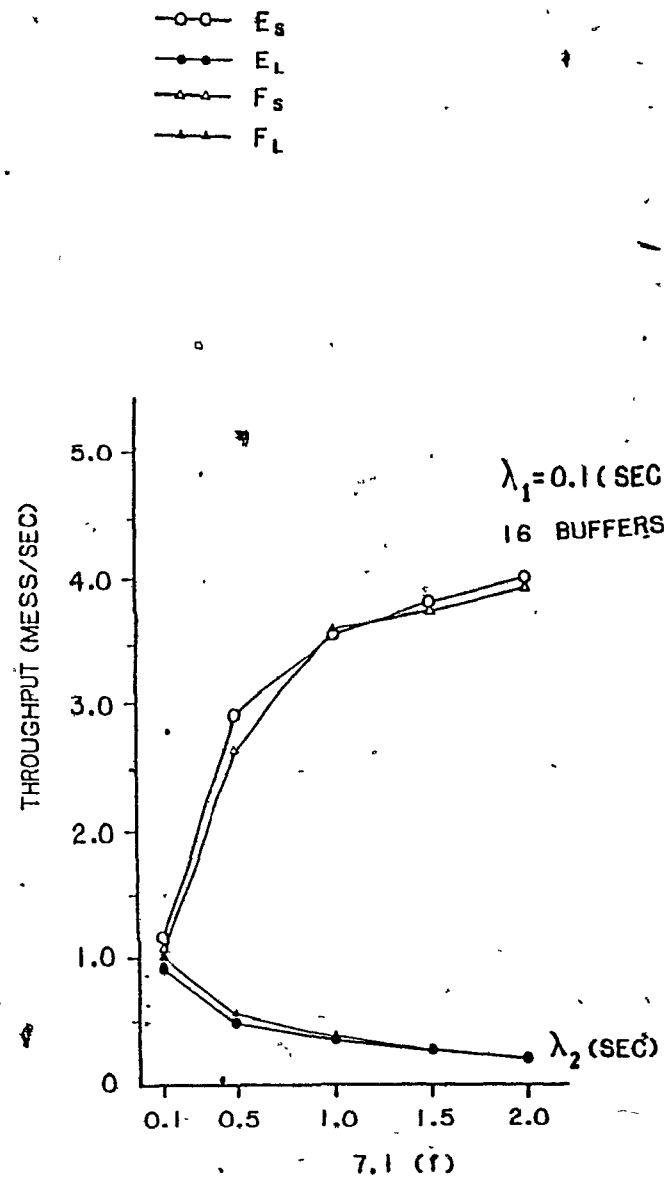
7.3 Suggestions for Future Work

The EWMA formula, although it seems a reasonable approximation, is by no means the optimal one for all possible traffic scenarios It is up to the network monitor to collect the data and find the appropriate ARIMA model (or models) and its associated parameters (ie α) that best fits the data

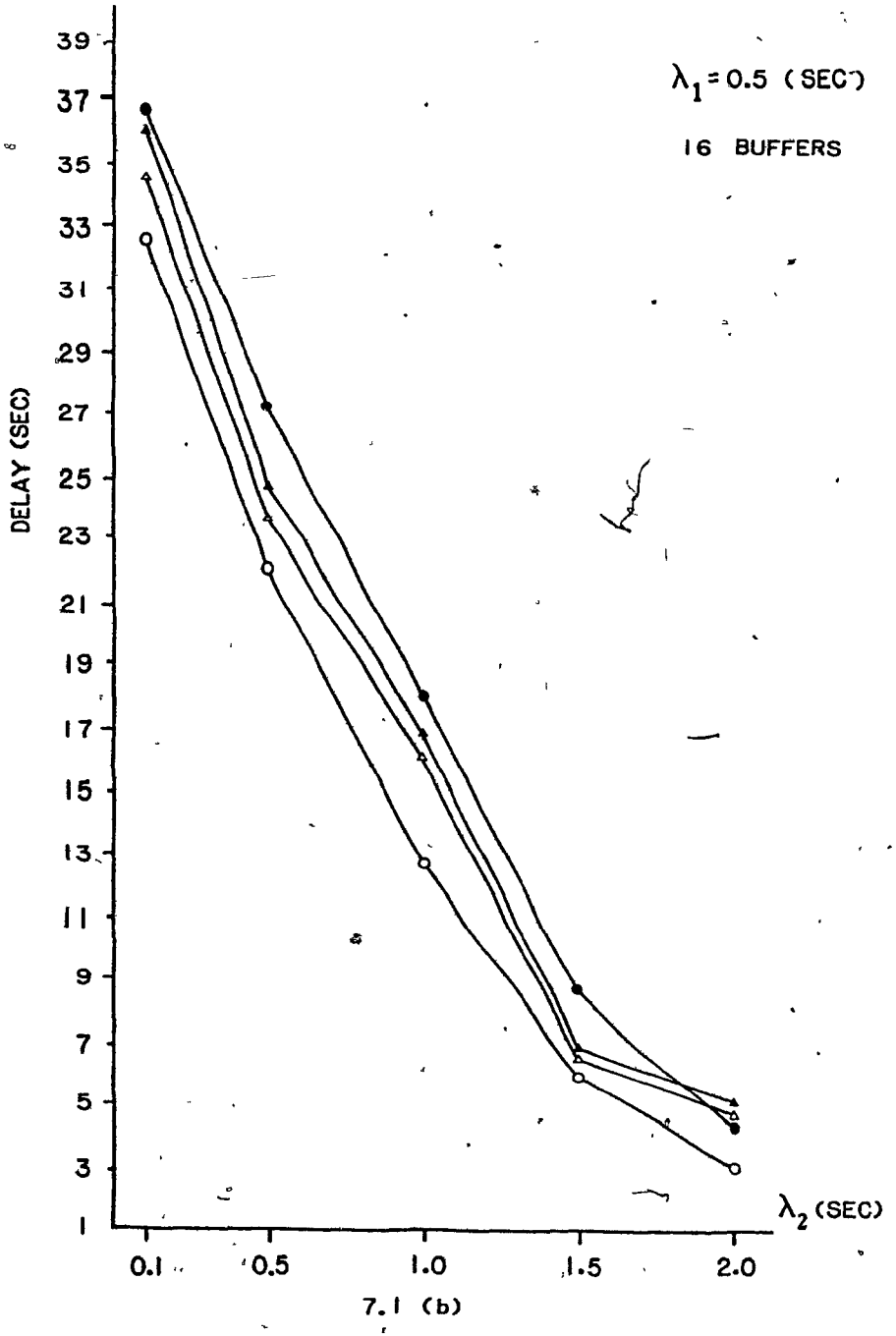
An extension of the algorithm can be used in a gateway between different networks In this case, the algorithm should account for different transmission media as well as for different categories of networks


λ_1 = 0.1 (SEC)
16 BUFFERS
DELAY (SEC)
28
26
24
22
20
18
16
14
12
10
0.1
0.5
1.0
1.5
2.0
λ_2 (SEC)

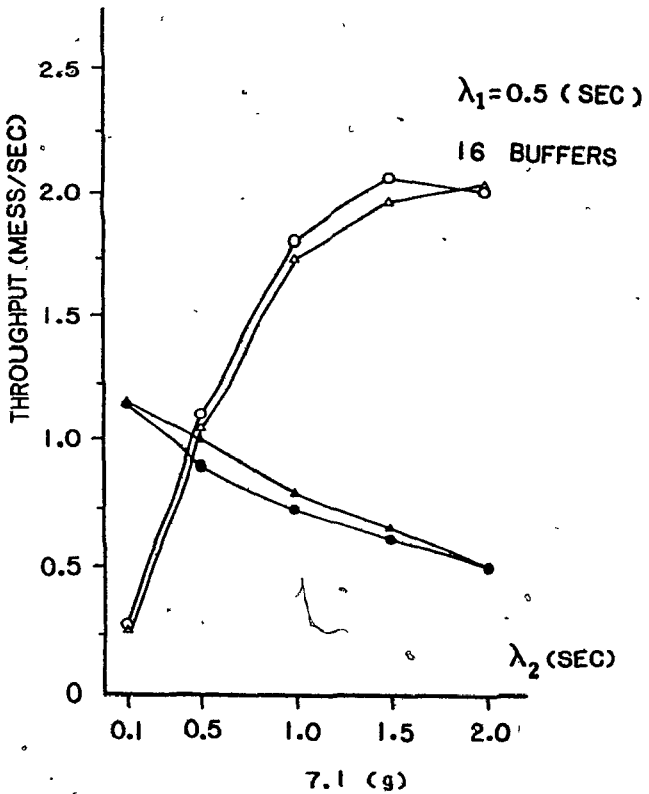
7.1 (a)


E_S
E_L
F_S
F_L
λ_1=0.1 (SEC)
16 BUFFERS
THROUGHPUT (MESS/SEC)
5.0
4.0
3.0
2.0
1.0
0
0.1
0.5
1.0
1.5
2.0
λ_2 (SEC)

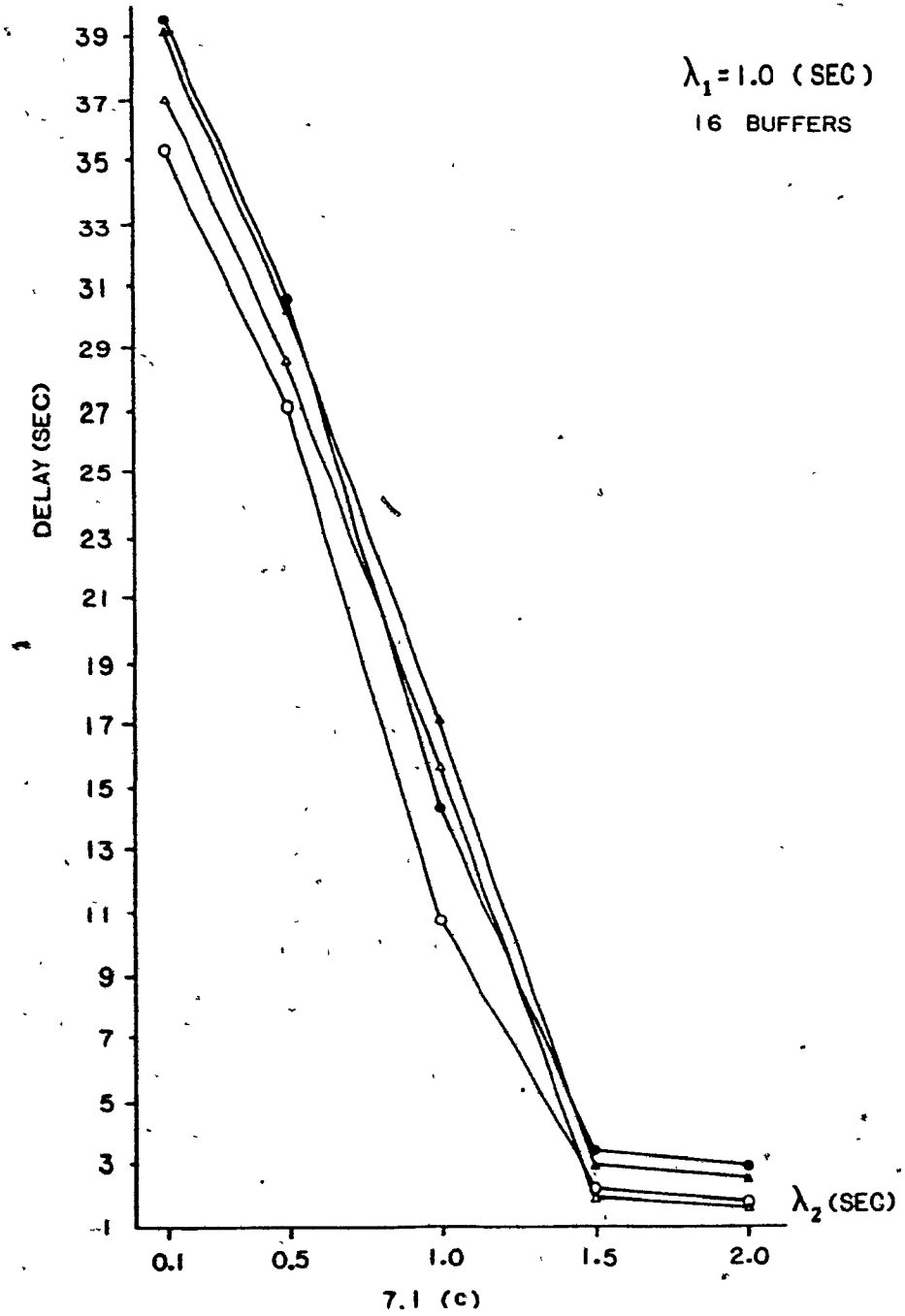
7.1 (f)


λ_1 = 0.5 (SEC)
16 BUFFERS
E_S
E_L
F_S
F_L
DELAY (SEC)
39
37
35
33
31
29
27
25
23
21
19
17
15
13
11
9
7
5
3
1
0.1
0.5
1.0
1.5
2.0
λ_2 (SEC)

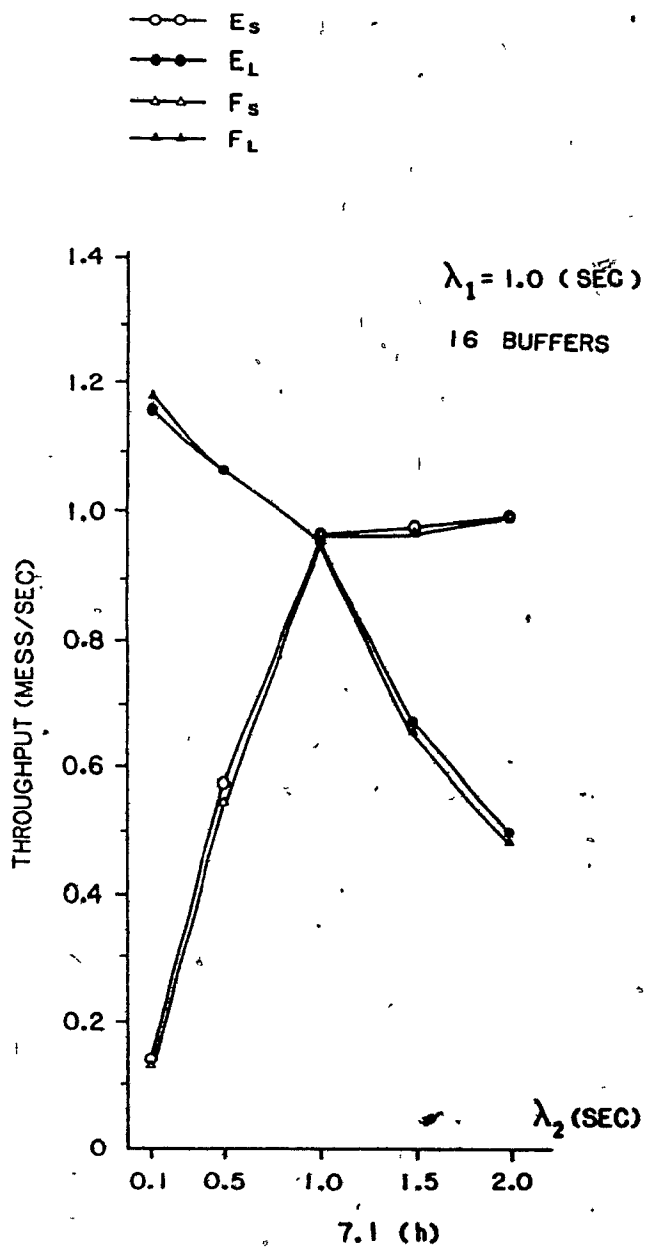
7.1 (b)


λ_1 = 0.5 (SEC)
16 BUFFERS
THROUGHPUT (MESS/SEC)
2.5
2.0
1.5
1.0
0.5
0
0.1
0.5
1.0
1.5
2.0
λ_2 (SEC)

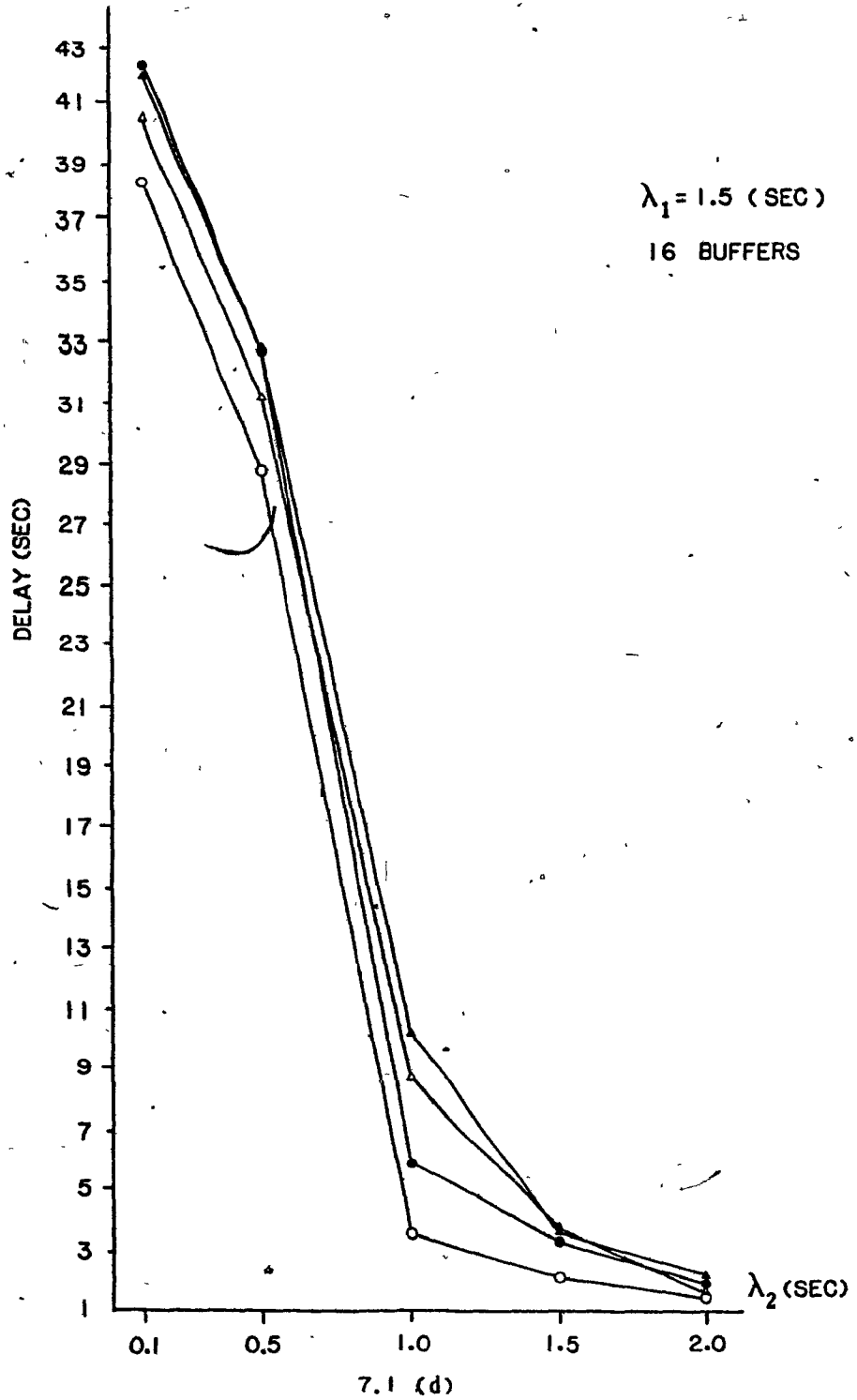
7.1 (g)

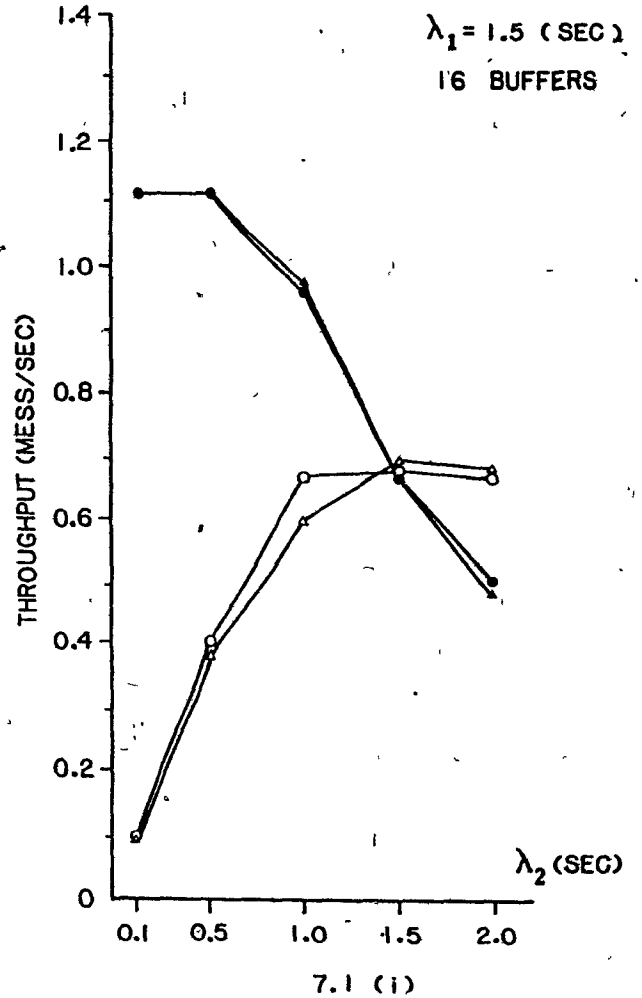

λ_1 = 1.0 (SEC)
16 BUFFERS
DELAY (SEC)
39
37
35
33
31
29
27
25
23
21
19
17
15
13
11
9
7
5
3
1
0.1
0.5
1.0
1.5
2.0
λ_2 (SEC)

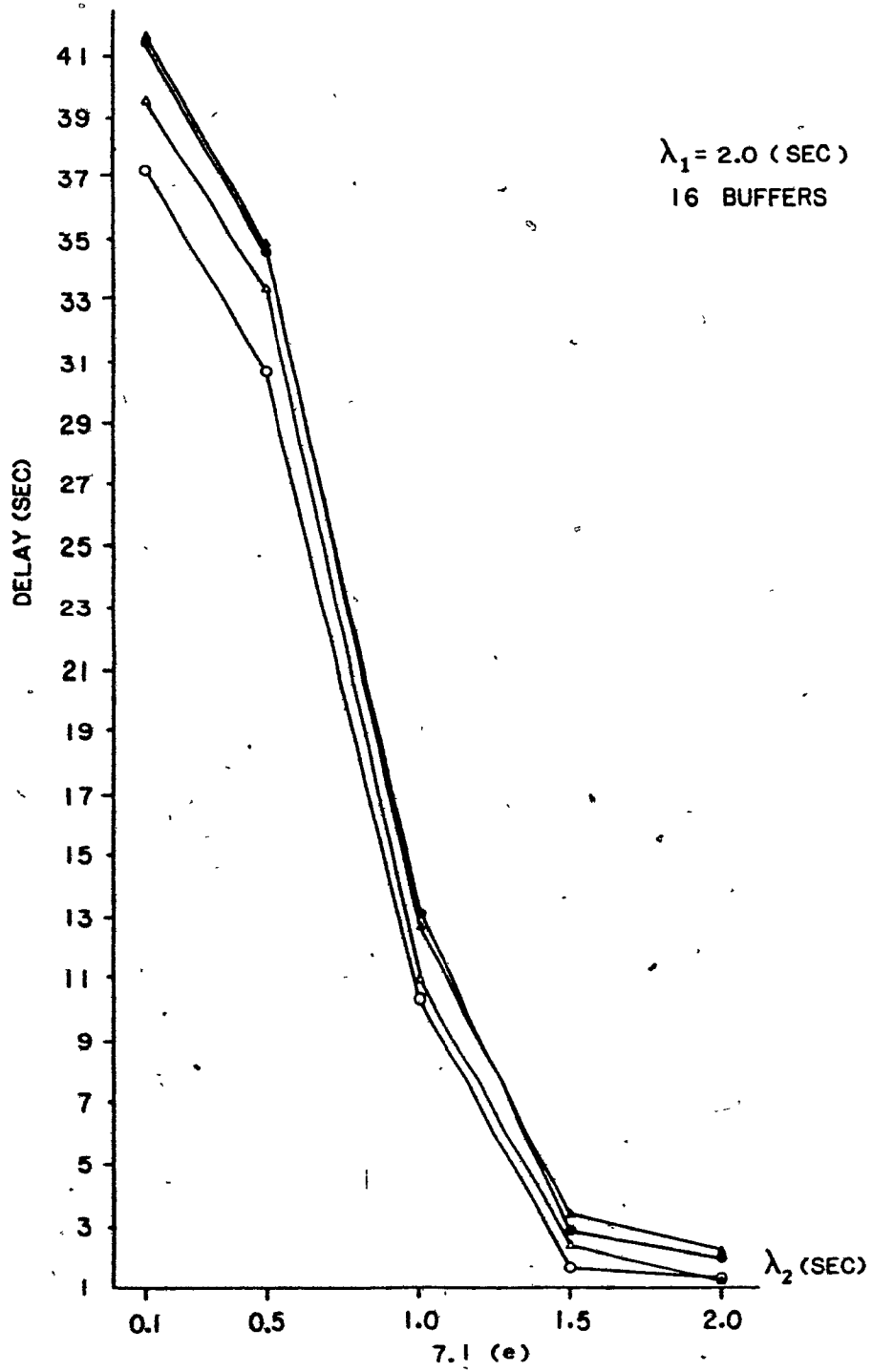
7.1 (c)

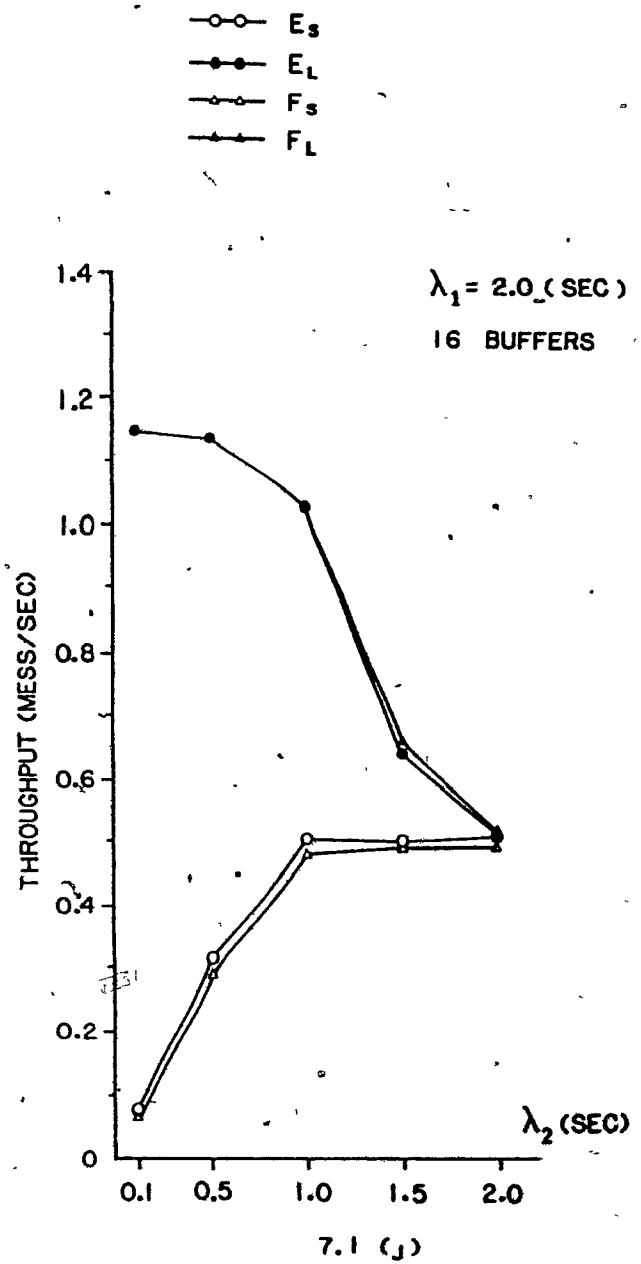

E_S
E_L
F_S
F_L
λ_1 = 1.0 (SEC)
16 BUFFERS
THROUGHPUT (MESS/SEC)
1.4
1.2
1.0
0.8
0.6
0.4
0.2
0
0.1
0.5
1.0
1.5
2.0
λ_2 (SEC)

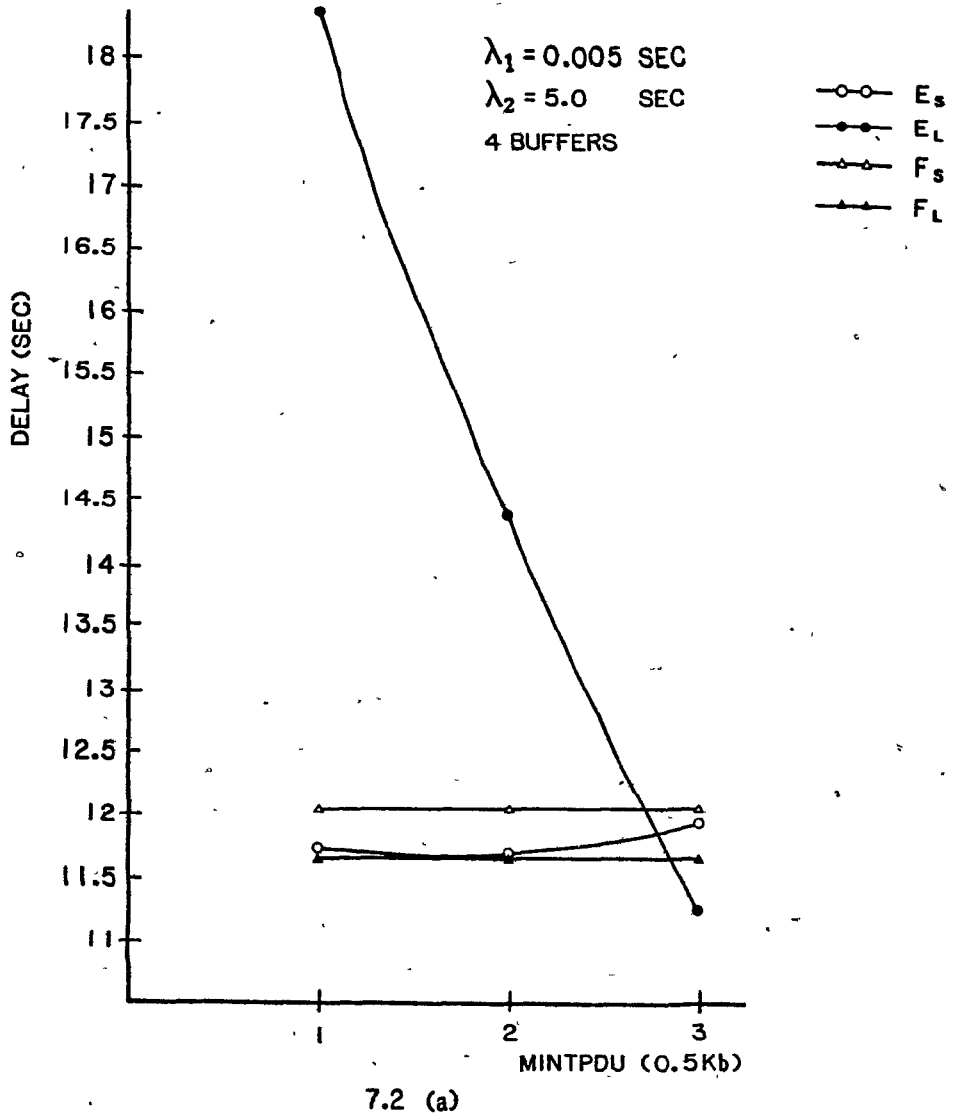
7.1 (h)


DELAY (SEC)
43
41
39
37
35
33
31
29
27
25
23
21
19
17
15
13
11
9
7
5
3
1
λ_1 = 1.5 (SEC)
16 BUFFERS
E_S
E_L
F_S
F_L
λ_2 (SEC)
0.1
0.5
1.0
1.5
2.0
7.1 (d)

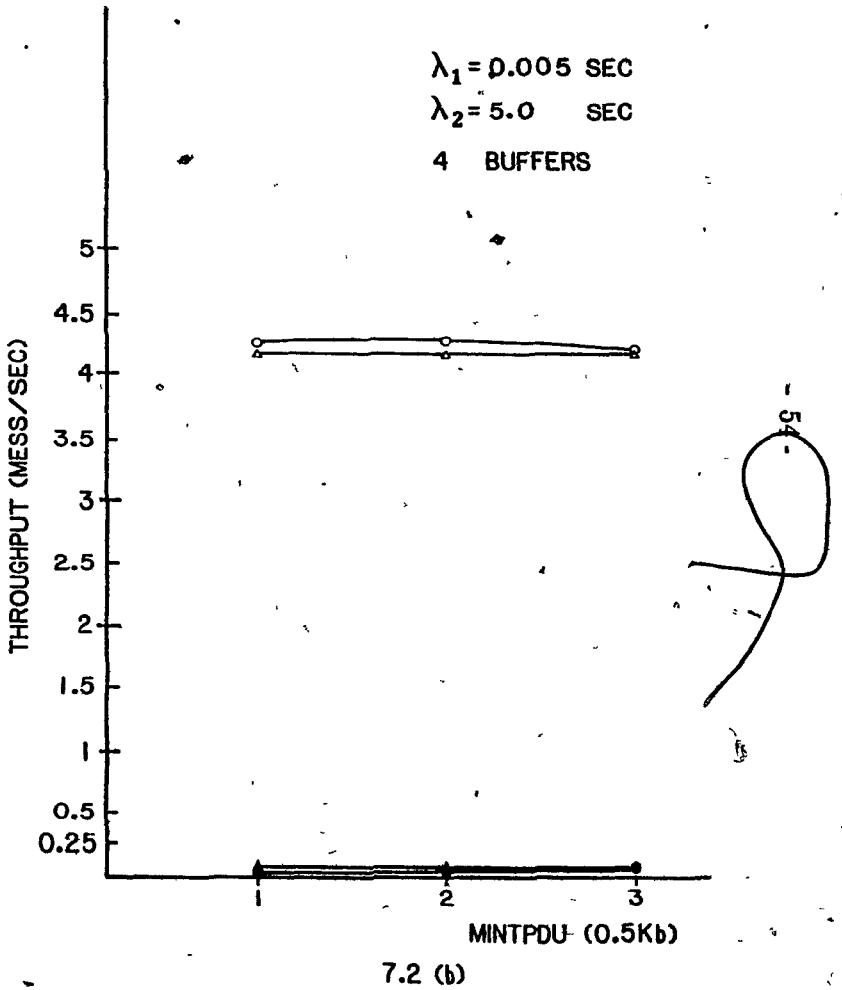

THROUGHPUT (MESS/SEC)
1.4
1.2
1.0
0.8
0.6
0.4
0.2
0
λ_1 = 1.5 (SEC)
16 BUFFERS
λ_2 (SEC)
0.1
0.5
1.0
1.5
2.0
7.1 (i)


λ_1 = 2.0 (SEC)
16 BUFFERS
DELAY (SEC)
41
39
37
35
33
31
29
27
25
23
21
19
17
15
13
11
9
7
5
3
1
0.1
0.5
1.0
1.5
2.0
λ_2 (SEC)
7.1 (e)

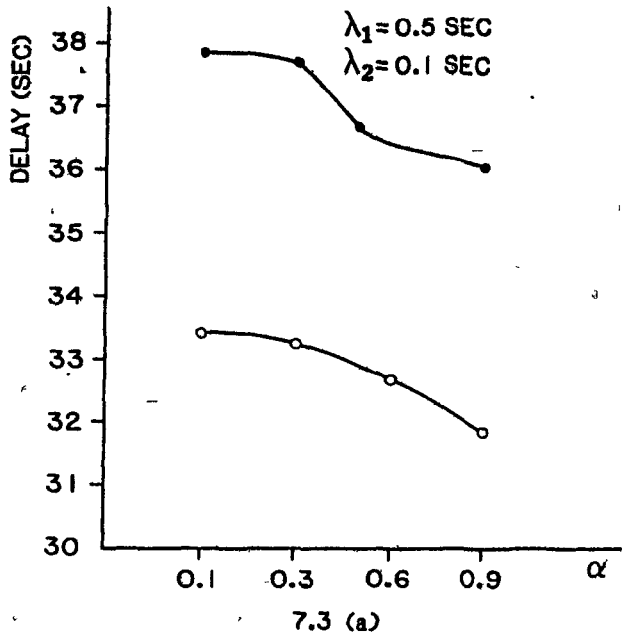

E_S
E_L
F_S
F_L
λ_1 = 2.0 (SEC)
16 BUFFERS
THROUGHPUT (MESS/SEC)
1.4
1.2
1.0
0.8
0.6
0.4
0.2
0
0.1
0.5
1.0
1.5
2.0
λ_2 (SEC)
7.1 (f)


λ_1 = 0.005 SEC
λ_2 = 5.0 SEC
4 BUFFERS
E_S
E_L
F_S
F_L
DELAY (SEC)
18
17.5
17
16.5
16
15.5
15
14.5
14
13.5
13
12.5
12
11.5
11
1
2
3
MINTPDU (0.5Kb)

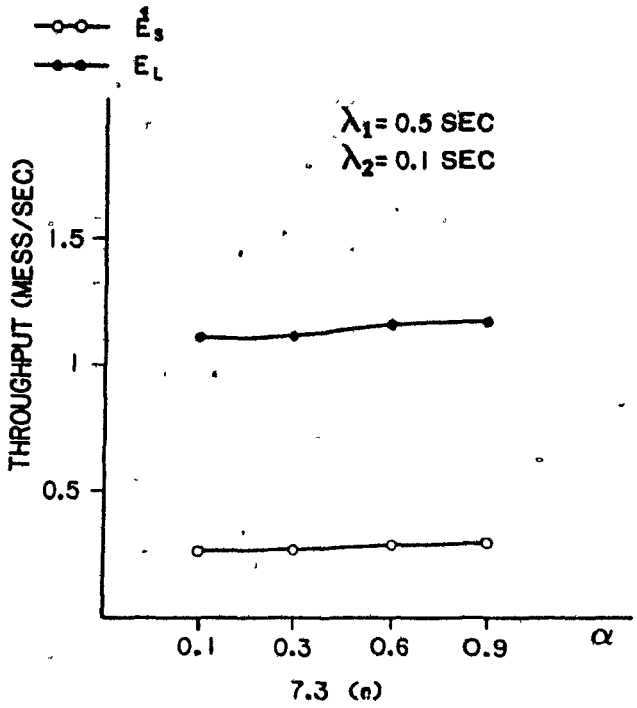
7.2 (a)


λ_1 = 0.005 SEC
λ_2 = 5.0 SEC
4 BUFFERS
THROUGHPUT (MESS/SEC)
5
4.5
4
3.5
3
2.5
2
1.5
1
0.5
0.25
1
2
3
MINTPDU (0.5Kb)

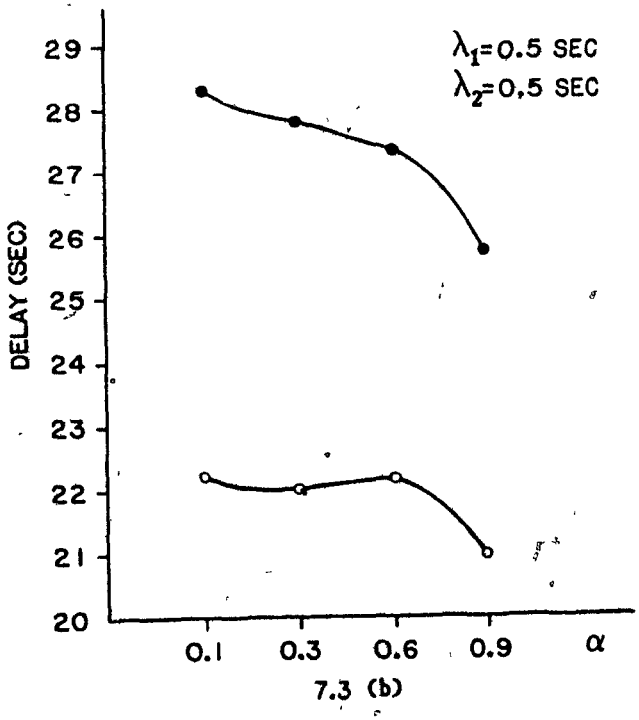
7.2 (b)



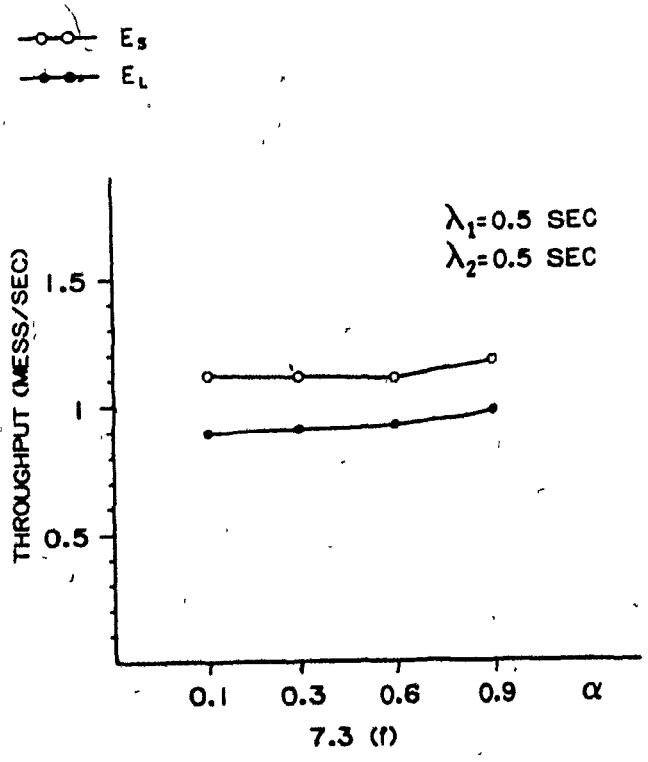
7.3 (a)



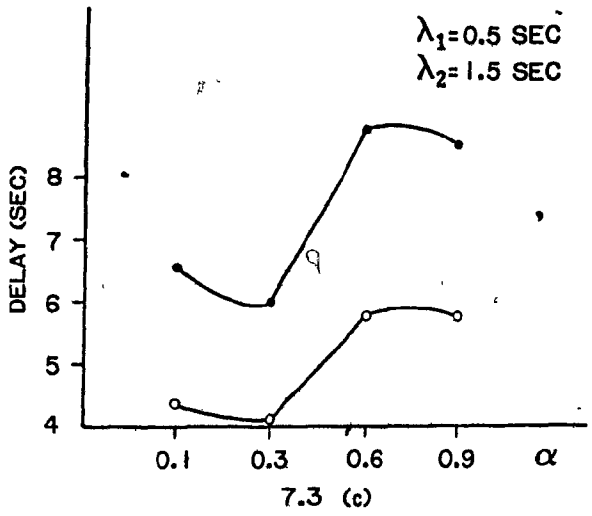
7.3 (e)



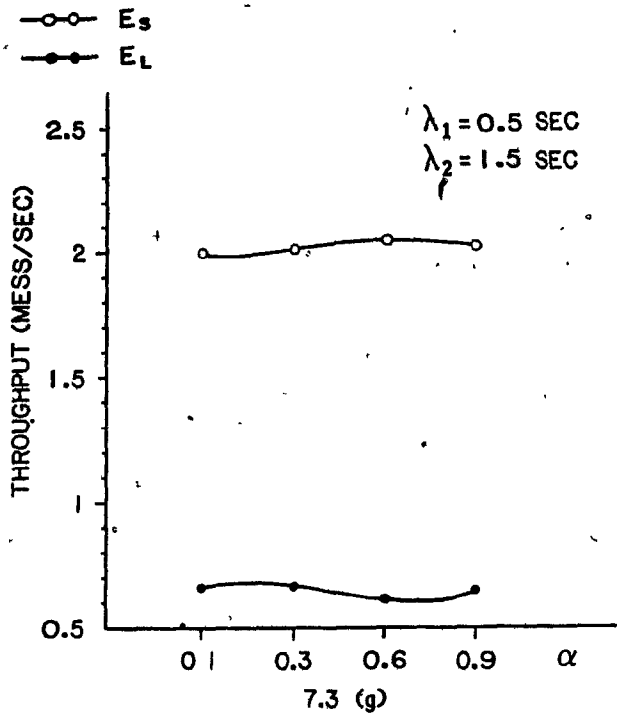
7.3 (b)



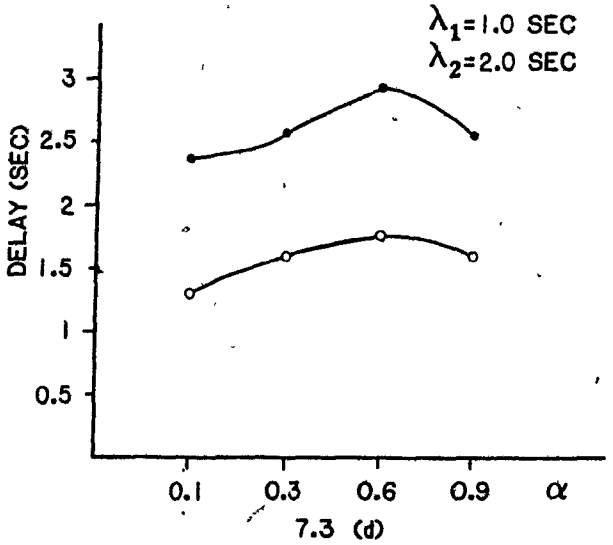
7.3 (f)



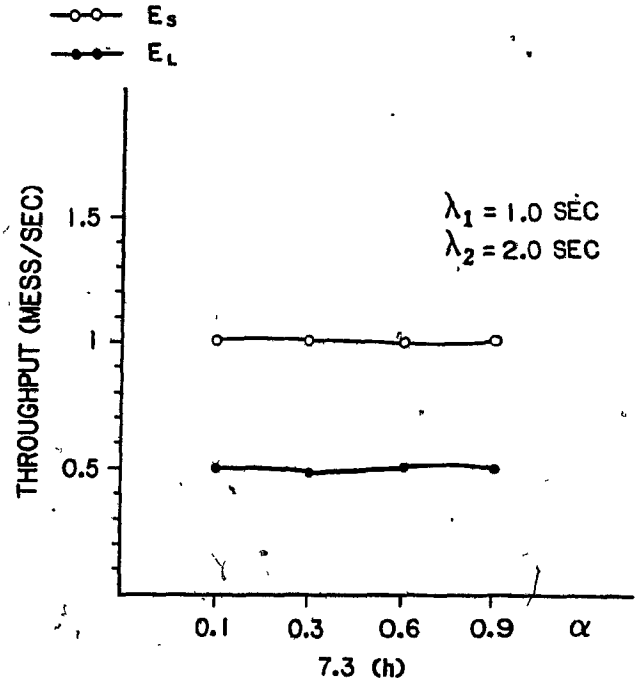
7.3 (c)



7.3 (g)



7.3 (d)



7.3 (h)

Bibliography

[AUPP83] *AUPPERLE, E. M.*, "Merit's Evolution - Statistically Speaking", IEEE Trans. on Comp., vol.c-32, no.10, pp. 881-902, Oct. 1983.

[BOX76] *BOX, G. E. P. and JENKINS, G. M*, Time Series Analysis, Forecasting and control, revised edition, Holden Day, Oakland, California, 1976.

[COHN83] *COHN, S. N.*, Measurement and Analysis of ARPANET Traffic May 1982 and March 1983, Bolt Beranek and Newman Inc., May 1983

[EVEQ86] *EVEQUOZ, C*, "On the Choice of Packet Size in Computer Communication Networks", M Sc, School of Computer Science, McGill University, Montreal, 1986.

[GERL80] *GERLA, M, KLEINROCK. L.*, "Flow Control: A Comparative Survey " IEEE Trans on Comm, vol COM-28, no 4, pp 553-574, Apr 1980.

[HARB82] *HARBITTER, A., TRIPATHI, S*, "A Model of Transport Level Flow Control", Procee. 1982 SIGMETRICS. Symp. Aug 1982.

[HEYM84] *HEYMAN, D P, SOBEL M J*, Stochastic Models in Operations Research, Volume II Stochastic Optimization McGraw-Hill 1984

[HOWA60] *HOWARD, R A*, Dynamic Programming and Markov Processes, M.I.T Press, Massachusetts Institute of Technology, Cambridge, Massachusetts, 1960

[HOWA78] *HOWARD, R A*, "Comments on the Origin and Application of Markov Decision Processes", Dynamic Programming And Its Applications, Editor M Puterman, Academic Press Inc., 1979

[KERM80] *KERMANI, P., KLEINROCK. L*, "Dynamic Flow Control in Store-and-Forward Computer Networks", IEEE Trans on Comm., vol.COM-28,

no.2, pp. 263-271, Feb. 1980.

[KLEI76] *KLEINROCK, L.*, Queueing Systems Volume II Computer Applications, Wiley-Interscience, New York, 1976

[KLEI80] *KLEINROCK. L., KERMANI, P.*, "Static Flow Control in Store-and-Forward Computer Networks", IEEE Trans on Comm., vol.COM-28, no.2, pp 271-279, Feb 1980

[ISO84a] *ISO, International Standard 8072,Information Processing Systems -Open Systems Interconnection- Transport Service Definition*, International Organization for Standardization,Geneva, 1984.

[NBS83a] *Specification of a Transoport Protocol for Computer Communications, Volume 1- Overview and Services,* Inst for Computer Sciences and Technology, National Bureau of Standards, Gaithersburg, Md., Jan. 1983

[PAWS84] *Performance Analyst's Workbench System* User's Manual, Information Research Associates, Austin Texas, Nov. 1984

[QUAR86] *QUARTERMAN, J S., HOSKINS, J C*, "Notable Computer Networks", Comm ACM, vol. 29, no 10, pp 932-971, Oct. 1986

[SCHW87] *SCHWARTZ, M*, Telecommunications Networks: Protocols, Modeling and Analysis, Addison Wesley, New York, 1987.

[STAL87] *STALLINGS, W.*, Data and Computer Communications, Macmillan Publishing Company, New York, 1985.

[TANE81] *TANENBAUM, A. S.*, Computer Networks, Prentice Hall, New Jersey, 1981

[TCP83] *Transmission Control Protocol, Military Standard,* directly MIL-STD-1778, U.S. Department of Defense, May 20, 1983.

[ZIMM80] *ZIMMERMANN, H.*, "OSI Reference Model - The ISO Model of Architecture for Open Systems Interconnection", IEEE Trans. on Comm., vol. COM-28, no.4, pp. 425-432, Apr. 1980.