PERFORMANCE MODELS OF A NETWORK INTERFACE IN A PACS

David D'Lima

Center for Communications and Signal Processing
Department of Electrical and Computer Engineering
North Carolina State University

January 1989

CCSP TR-89/3

ABSTRACT

D'LIMA DAVID. Performance Models of a Network Interface Unit in a PACS. (Under the direction of Harry G. Perros)

Increasingly, data output from imaging equipment in hospitals is being generated in a form suitable for transport, from the radiology department to referring physicians as well as around the radiology department, over electronic communications facilities such as picture archival and communications systems. To some extent, there may be communications incompatibilities between products so that network interface units are needed to interconnect subsystems.

In this thesis, we present some queueing network performance models of a network interface unit that permits communications between the image database and display stations in a PACS. The ease of solution and accuracy afforded by these models vary widely. A preliminary model is a product-form queueing network but incorporates some unrealistic assumptions. Another is extremely detailed so that only a discrete-event simulation can be used to extract performance measures. Two other approximations were developed. They are based on some simplifying assumptions about the detailed model and are extremely easy to solve. Comparing the results with those obtained by simulation indicates that the second model is invariably more accurate than the first. In addition, a program was written to simulate an acquisition station communicating with the image database.

TABLE OF CONTENTS

CHAPTER 1

INTRODUCTION

1.1 Outline

During the past decade, digital technology has become the method of choice for the design of medical imaging equipment. Computerized tomography, ultrasound, digital subtraction angiography, storage phosphor computed radiography and magnetic resonance systems provide hitherto unavailable medical diagnostic capabilities to radiologists. Consequently, they can make better recommendations to referring physicians. Driven by the growing need for such services and the availability of mature technology for both mass digital storage devices, like the optical disk, as well as high speed digital communications systems, like local area networks, a number of institutions have initiated development efforts for picture archival and communications systems (PACS) for hospitals [7].

A PACS comprises a database of medical images, display stations (at which these images may be viewed), acquisition systems (attached to medical imaging modalities) and the broadband communications network that ties them together.

Traditionally, data output from imaging devices like X-Ray machines was generated in a form suitable for transport, from the radiology department to referring physicians as well as around the radiology department, by a human operator rather than over electronic communications facilities. Some of the newer digital imaging devices have been

designed with communications ports but the intent was usually to perform stand-alone functions. The communications approach, commands and data format from product to product are, not surprisingly, usually unrelated, so the age-old problem of communications incompatibility arises. Some protocol translation (of headers) and signal processing (of pixel data) may need to be performed by network interface units on the messages carried on the PACS in order that a hospital be able to purchase the expensive systems that comprise a digital radiology department without being tied to a particular vendor's family of products.

1.2 Problem Statement

There are several possible architectural approaches for designing a PACS. For any given system, an important consideration, in a multi-vendor environment, is the impact that communications incompatibilities between different vendors' products have on the performance of the PACS.

In this thesis, we examine one such PACS architecture. The system comprises several display workstations communicating with a centralized image database. Requests entered at the workstations are for images to be retrieved from the database. Images are archived onto the database from acquisition stations. These stations may be made by different vendors. However there is a network interface unit to resolve communications incompatibilities between the two types of systems. The performance measures of interest include the average end-to-end delay per image from database to workstations and design parameters for the network interface units. In this thesis we address the construction of models of these and related systems and their solution in order to extract the performance

measures of interest.

1.3 Organization of the Thesis

Chapter 2 first presents a detailed outline of the PACS system that was studied as well as design considerations for the network interface unit. Next, the basic theory that is used in formulating the performance models is dwelt on.

In Chapter 3, we describe the PACS environment in which the network interface unit is used. A queueing network model for this system is presented. In order that all the details be captured, this model can only be analyzed by simulation which takes very long to run.

In Chapter 4, some analytical performance models are presented. Based on some simplifying conditions, quick approximate solutions can be obtained.

In Chapter 5, results from the approximate analytical performance models are compared with those obtained by simulation with a view to demonstrating their accuracy and evaluating some design alternatives.

In Chapter 6, a simulation model of an acquisition station communicating with the image database is presented.

The listings of some of the programs developed are included in the Appendices.

CHAPTER 2

SYSTEM DESCRIPTION AND MODELING TECHNIQUES

2.1 System Description

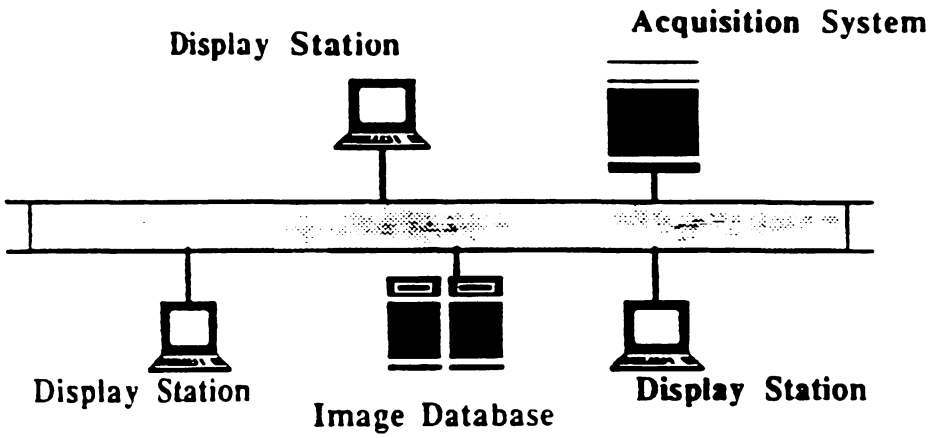
2.1.1 Description of the PACS

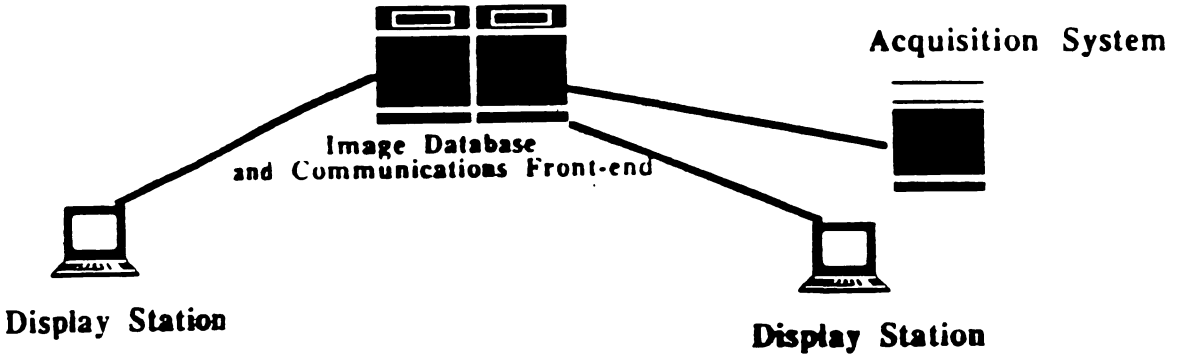
There are two prominent architectural approaches in vogue for the design of a PACS - a bus-based system and a star network. See Fig. 2.1. The system that is examined in this thesis is a star network and is based on an actual product - the AT&T CommView™ PACS [6].

This PACS comprises a centralized image database and a colocated centralized communications hub that is connected to display stations and acquisition systems over point-to-point optical fiber links. Images are acquired from various modalities and eventually archived in the centralized image database. Radiologists at diagnostic viewing stations request images from the image database and can manipulate these images to improve the quality of their diagnosis. Referring physicians request images and text diagnoses from the database. Server processes run in the nodes to handle requests, image and text transfers as necessary. Both kinds of requests are "single-threaded", i.e. a second request may not be entered till the current request has been completed.

Images are of varying sizes while the database uses a constant block size for fragments of images. In this implementation, the database block size is 256 kilobytes. This

Figure 2.1. Possible architectures for a PACS





corresponds to the smallest possible image which is of dimensions 512x512 pixels and 8 bits deep. A much larger image, say a 2048x2048 image, with each pixel having 12 bits for gray scale range, corresponds to 32 blocks of database storage. Though database

blocks cannot be individually addressed, except in the case when one block corresponds to a full image, they are the logical units in the communications between session layers of communicating devices. Compression algorithms, exploiting the redundant areas within an image, are applied so as to reduce both the network transmission time and the memory requirements for storing the image in the database. Bit-preserving compression allows for a ratio of upto 3:1 compression in pixel data. Clearly there will be small variations in the degree of compression that can be achieved when comparing images from the same modality, but relatively larger variations when comparing images acquired from different modalities. The compression (decompression) time is nearly invariant to the degree of compression desired (used).

There is a hierarchy associated with messages circulating in a PACS. At the lowest level, the messages are termed *images*. The set of all images acquired from the same patient at a certain time is called a *study*. Within a study, there may be a related group of images, called a *series* and within a series the component images must have come from multiple *acquisitions*. However, a single acquisition, which refers to the collection of physical data, may result in more than one image. The database search keys use the complete hierarchical description to locate individual images.

In this thesis, this terminology is changed somewhat. We consider that requests are for *exams* and that each exam is made up of a variable number of *image records*. The hierarchy is carried upward by employing a transaction called a *batch* which is a series of exams. Within an exam, all image records are homogenous. A large image is modeled by considering it as an exam with several image records.

The ACR-NEMA Standard [3] specifies commands, communications and data formats to follow in interconnecting digital imaging equipment from different vendors. A sample session, depicting the possible message flow for image retrieval, is illustrated in Fig. 2.2. As the standard only governs equipment connected on point-to-point links, network interface units, or, more loosely, protocol converters, will be needed to connect imaging equipment from one vendor to another vendor's PACS network.

2.1.2 Network Interface Units [2], [3]

Network interface units are expected to provide translation and relay, or a virtual circuit connection, from the imaging equipment at one end to the PACS on the other. The interface is expected to be transparent with regard to end-to-end message delivery and response. Fig. 2.3 illustrates an ISO Open Systems Interconnection representation of a network interface unit. The two half-gateways may be located in a single physical unit or

Figure 2.2. Sample message flow in a PACS

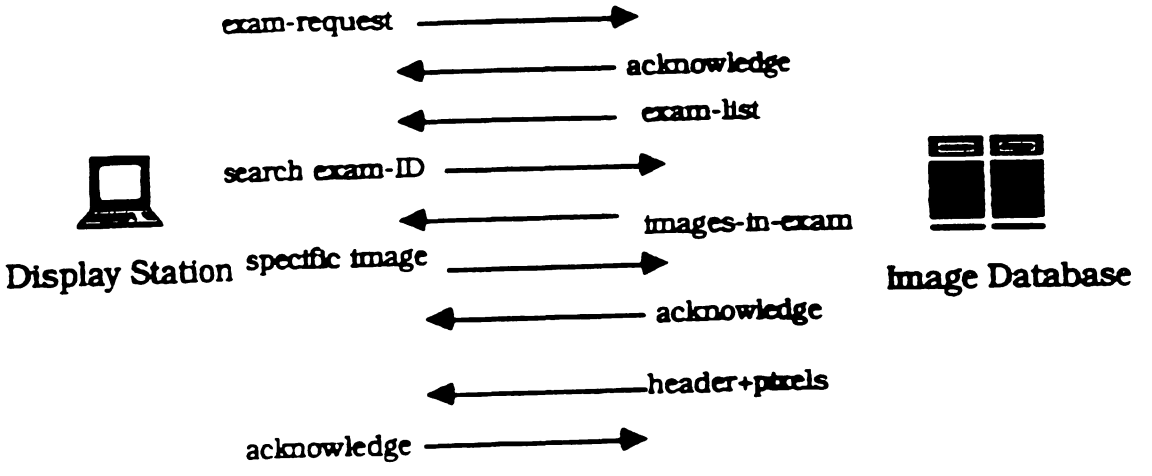
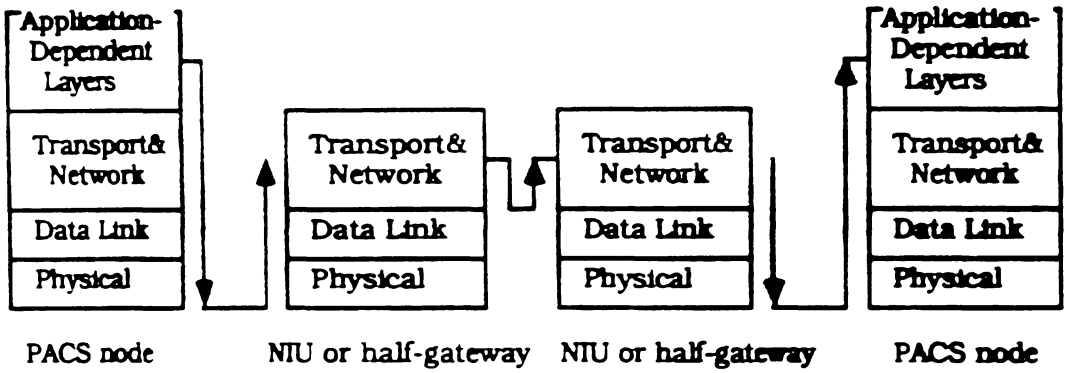


Figure 2.3. OSI Representation of a Network Interface Unit



in separate units at either end of the network.

There are a variety of possible complications that may be introduced when imaging equipment from one vendor has to be connected to a PACS supplied by another vendor. The communications formats, notably the packet sizes, may be different on either end of the interface so that packet fragmentation and reassembly will have to be performed. There may be some inconsistencies between database key organizations so that some parsing of the image header fields becomes essential. The ACR-NEMA standard specifies "low order byte first, low order 16 bits first" governing transfers across the interface whereas some computers store long words differently, i.e. the order of 16 bit words is reversed, so that word-swapping has to be carried out in such cases. Finally, the compression algorithms used on either side of the interface may be different, or images may be archived in compressed form but transmitted to viewing stations uncompressed so that decompression (or compression in the reverse scenario) becomes necessary.

2.2 Performance Models

Queueing networks are used to model a variety of applications characterized by contention that arises as a result of transactions having to share a set of resources at the queues (or service centers) in the network. If a transaction finds that all the servers at a queue are occupied, it is enqueued at that center and will be served according to some scheduling discipline once a server is freed. After completion of service, a transaction will proceed onward to another (or possibly even the same) queue. In an open queueing network transactions eventually depart, whereas in a closed queueing network they circulate continuously.

There is a class of queueing networks that has a "product-form" steady-state probability distribution [5]. Algorithms are available to extract the mean value of various performance measures, such as response and utilization, from this distribution with minimum computational effort.

A closed queueing network, with customers who may change classes, and service stations that must be of the following types: (i) first-come-first-served discipline with the same exponentially distributed service time for all classes, (ii) processor-sharing discipline, (iii) infinite-servers or server-per-job so that no queue develops or (iv) last-come-first-served discipline with pre-empt resume, can be shown to have a product-form steady-state probability distribution.

QNAP2 [4] is a queueing network analysis package that can be used to compute performance measures of such "product-form" queueing networks rapidly. It can also be used

to analyze certain classes of non-product-form queueing networks. Finally, QNAP2 permits the user to simulate a queueing network and also to analyze it using a numerical technique for Markov chains.

2.3 Objectives

In this thesis, we address the construction of models of display stations communicating with the centralized image database with a view to obtaining the average end-to-end delay per image from database to workstations, under varying traffic conditions and system parameters. Additionally, some design parameters, notably the buffer size, for the network interface unit can be obtained, as well as design alternatives that would reduce the delays across the interface.

CHAPTER 3

A QUEUEING NETWORK MODEL OF THE SYSTEM

3.1 Introduction

The specific configuration that will be examined is depicted in Fig. 3.1. There are four display workstations communicating with a centralized image database through a network interface unit. Point-to-point communications links connect the nodes. Images are stored compressed in the database but need to be transmitted uncompressed to the workstations.

3.2 A Queueing Network Representation

We now present a queueing network model of the aforementioned system. It is illustrated

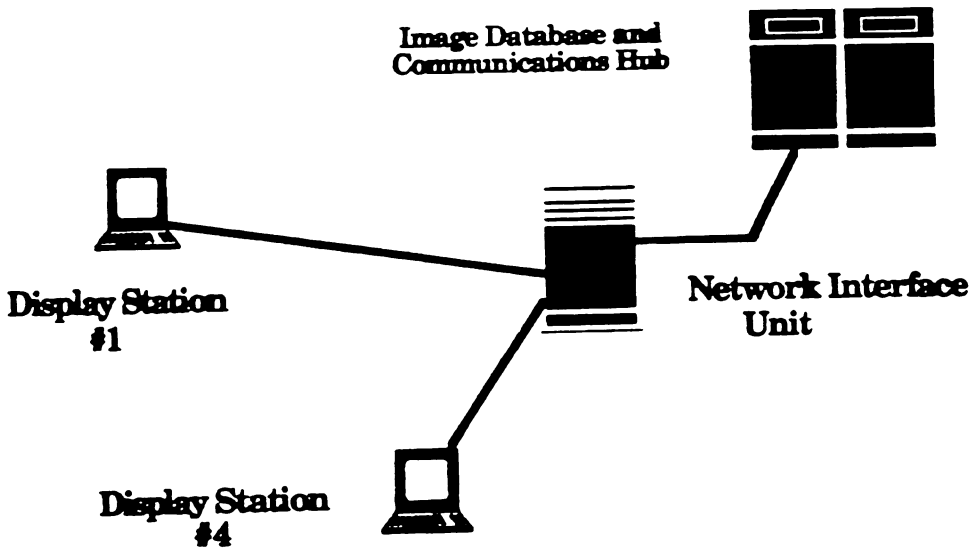


Figure 3.1. The PACS configuration under study

in Fig. 3.2.

Requests entered at the workstations are for exams, or a series of image records, to be retrieved from the database. The service experienced by requests for exams, proceeding

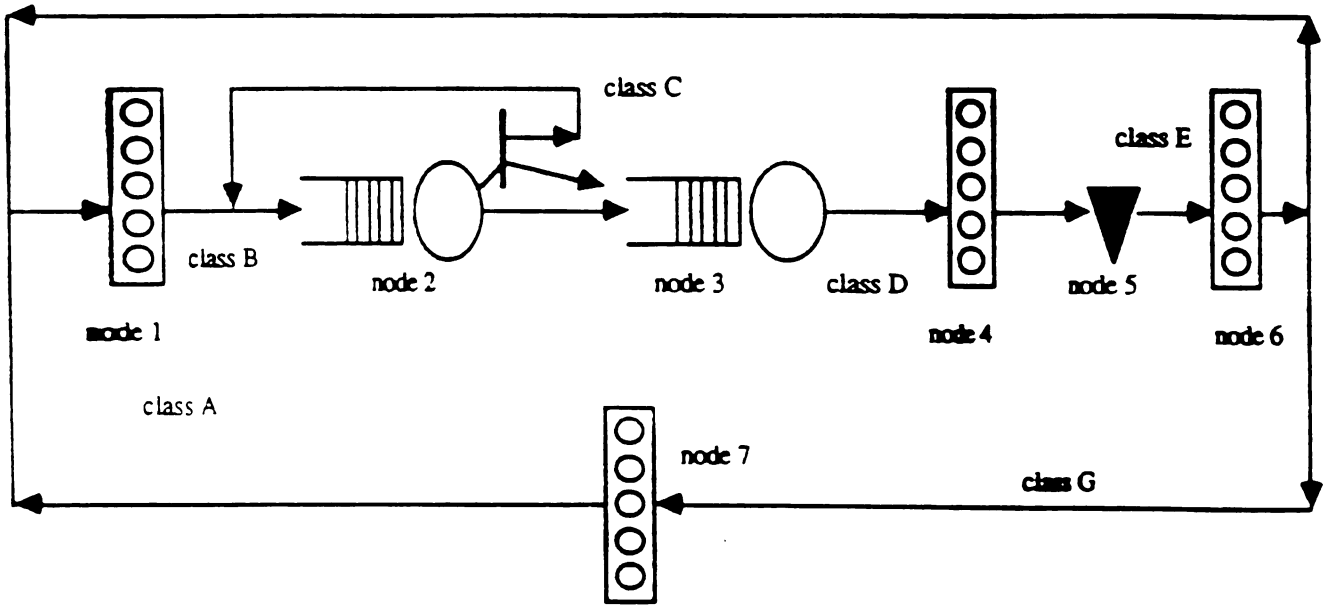


Figure 3.2. A queueing network model of the PACS configuration under study

from the workstations, across one fiber link, through the interface, to the database, was modeled as an infinite-server station. This station is labeled as node 1. As request packets are expected to have higher priority than image packets, there will be almost no queueing delay for exam requests so the only delay they encounter is the time to transmit an exam request to the database. That is to say, the priority of the server process that handles an exam request will be higher than those which handle image and text transfers so that requests would typically not have to wait for the appropriate server process. Additionally, it is assumed that this delay would not vary significantly with increased workstation traffic or even increased external load on the database. The latter effect can easily be captured nevertheless by calibrating these delays uniformly.

The transactions, labeled as class A, arriving at node 1 are exam requests and they proceed to the database manager, represented by node 2, which services these exam requests on a per-image-record basis. Indeed, an analogy can be made with the following queueing system: on service completion, a transaction (to wit, an *image record*) proceeds onward, but, in addition, may, with some probability, spawn a sibling which is fed back to the queue. Fresh transactions, labeled as class B, represent exam requests, while fedback and spawned transactions represent image records, labeled as class C. Upon service at node 2, these customers proceed onward besides which they may spawn a sibling which returns to the composite as class C with probability $p_{B-to-C} = [(size - 1)/size]$. That is, a geometric distribution is used to determine the probability that all the *record*s that make up an *exam* have been read off the database. As the database manager is also serving requests by the other nodes, both acquisition and display, that comprise the PACS, the mean composite DBMS retrieval time is allowed to vary from a lower value

representing an otherwise idle database to an upper limit representing a fully loaded PACS.

A composite server is used to model the delay in servicing image records at the interface. This is made up of the time spent in decompressing the pixel data and performing the required protocol conversion therein, till it is available in interface memory in a form suitable for processing by the other half-gateway. The scheduling discipline is first-come-first-served on a per-image record basis. This station is represented in the model as node 3. The transactions, labeled as class D, are image records.

Considering that there is a dedicated fiber link from the interface to each workstation, the service experienced by image records, propagating from the local half-gateway of the interface to the workstations, was also modeled as an infinite-server station. Additionally, it is assumed that the interface delay is much larger than the communications delay. This represents the protocol delay in sending an image record from interface memory to workstation memory. This is depicted as node 4 in the queueing network.

At the station labeled node 5, image records are reassembled to form an exam. Consequently, transactions labeled class D must be distinguished from those labeled class E, that represent exams.

The complete set of image records is now available for examination at the display station. The service for a workstation user, either a physician or a radiologist, who spends some time studying the retrieved exam before entering a new exam request, is modeled as an infinite server station similar to the "think" time for terminal users in conventional

multiprogramming computer system performance models. This station is represented by node 6 and is visited by every class E transaction.

It is assumed that there is single-threading of exam requests at each workstation so that once a request is entered, the workstation gets blocked to further requests till the current request has been serviced. This limits the population of customers of class *exam* to one per workstation-to-database loop. If there is adequate local memory at each workstation, there could conceivably be a chain of exam requests in service at each workstation.

Furthermore, the phenomenon of a workstation being completely idle (such as when the radiologist is on a relief break) can also be modeled by forcing the transactions to periodically visit a station which introduces the idle time delay. In the model, this is achieved by introducing a branching probability after service completion at node 6. Transactions, labeled class E, may either change class to type G and visit the idle time delay station, which is node 7, or change class to type A and return to node 1. Class E transactions change to class A transactions with probability $p_{E-to-A} = [(batch - 1)/batch]$ where *batch* is the average size of a batch of exams. That is, the geometric distribution of the number of exam requests in a batch is used to determine the branching probability that the workstation is going to be idle for awhile. Thus, type A transactions may be considered to be the *last* exam in a batch of exams while type G transactions represent any *other* exam in the batch.

3.3 Solution of this Model

This queueing network representation is observed to violate the assumptions of

"product-form" queueing networks with regard to the permitted service disciplines. As this queueing network no longer has a product-form solution, a discrete-event simulation was carried out instead. The simulation program was written in QNAP2 and a listing is in Appendix 4. Each simulation run took approximately 40 minutes on a VaxStation 2000 to generate 95% confidence intervals with the range of error less than 15% of the mean value.

CHAPTER 4

APPROXIMATE ANALYTICAL PERFORMANCE MODELS

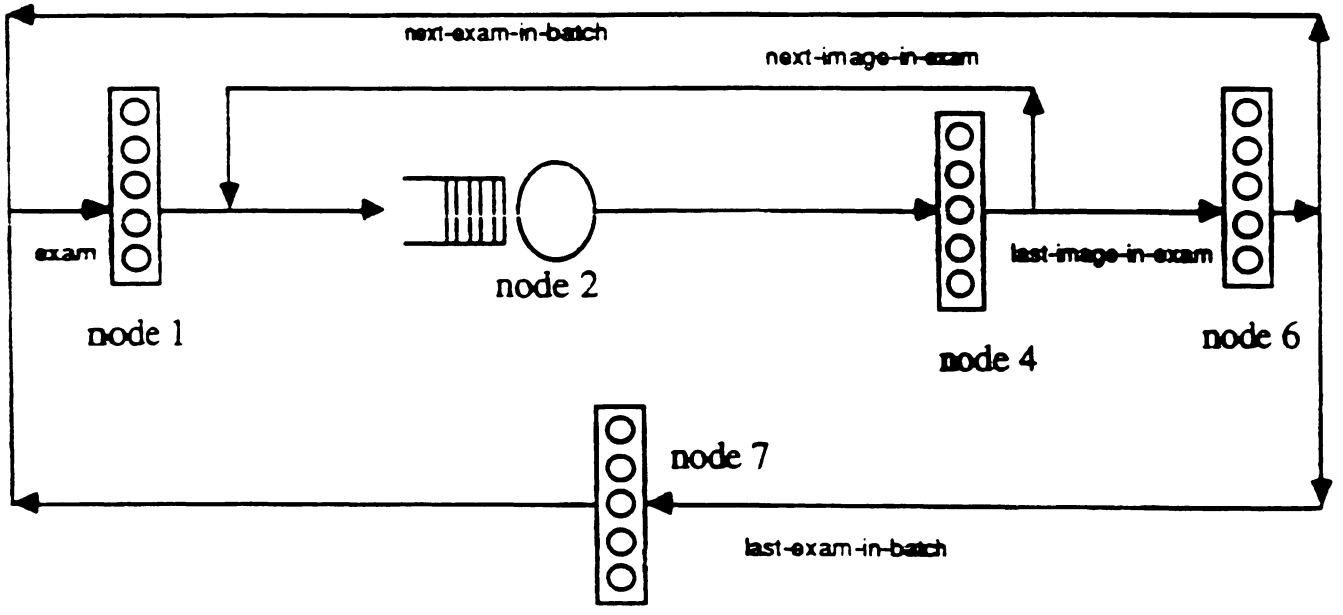
4.1 Introduction

In this chapter, we present three queueing network performance models: the first is of product form, the other two are not. Of the other two, the first model is solved numerically while the second is solved approximately by an iterative procedure. As opposed to the simulation model, neither captures all the characteristics of the actual system, however they take considerably less time to solve.

4.2 A Product-Form Queueing Network

If a composite server is used to model all the delays in servicing any image record at the database and in the interface, Fig. 4.1 results. Note that the station represented by node 2 now includes the time spent in locating the image record, transmitting the compressed image from the database to the local half-gateway of the interface, decompressing the pixel data and performing the required protocol conversion therein, till it is available in interface memory in a form suitable for processing by the other half-gateway. This queueing network representation is observed to satisfy all the assumptions of "product-form" queueing networks so that performance measures such as end-to-end delay of image records and utilization and mean queue length at the DBMS composite can be calculated very rapidly in spite of the variety of transactions circulating within. The individual stations may have completely different exam request patterns.

Figure 4.1. A product-form queueing network model

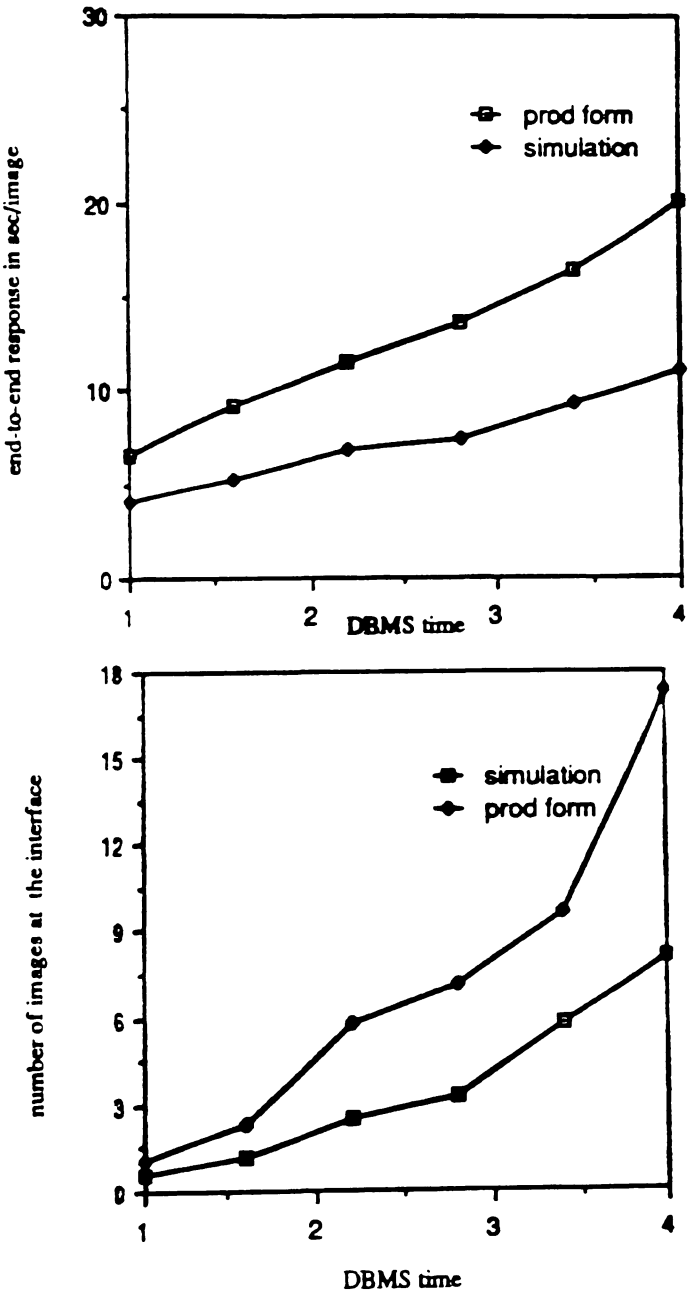


However this model has one important drawback. The design parameters of the network interface unit are not easy to obtain. One cannot claim the performance measures of the DBMS composite to be approximately those of the interface. This is clearly seen in Fig.

4.2, where the performance measures obtained from solving the product-form queueing network are compared with those from the simulation of the complete queueing network. The product-form queueing network consistently produces very pessimistic results, i.e. it

Figure 4.2. Comparison of results from the product-form queueing network and the complete network, studied by simulation

(Tx time = 1 img/sec, Exam size = 14 img, Think = 60, Idle = 120)



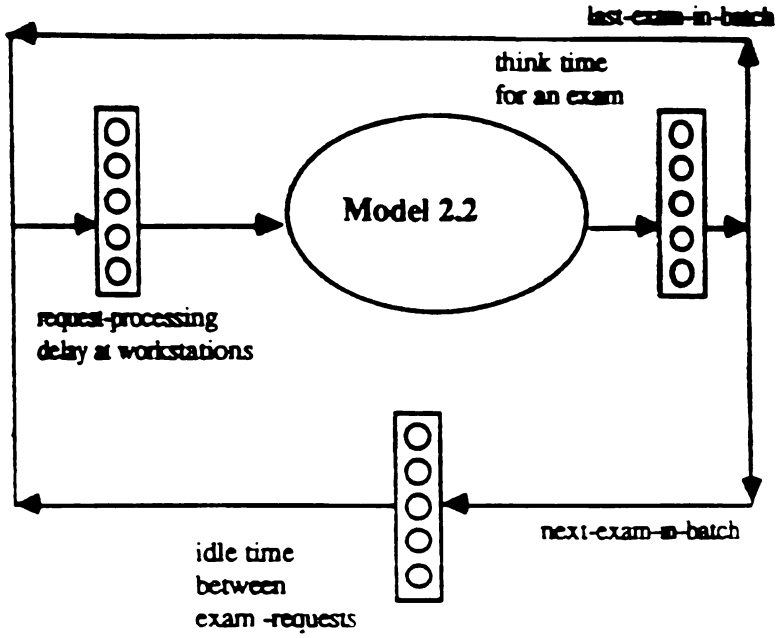
under-predicts system capacity. The error for the case illustrated is of the order of 160% for the mean queue length at the interface and 100% for the end-to-end response. The fundamental problem is that choosing a composite to represent a variety of delays is simplistic. Any reasonable implementation of a PACS would pipeline, to the best extent possible, all the delays that an image would encounter between the database and the workstations. A server process in the communications hub would first locate the exam header and then proceed to sequentially read off all the blocks that comprise the image. As a result some speedup is obtained through whatever degree of pipelining is afforded. However this mechanism is complicated and does not satisfy the "product form" queueing network assumptions.

4.3 An Approximate Equivalent

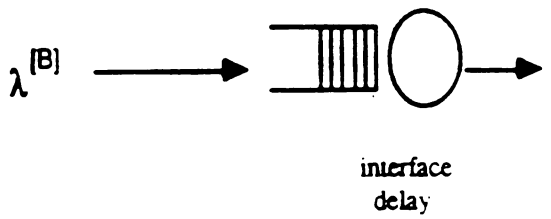
Consider the queueing network depicted in Fig. 3.2. As explained earlier, the reason why it did not lead to a product-form solution was because of the complicated mechanism by which image records were read off the database. Conditioning on the assumption that the actual time spent by individual images in the course of undergoing a database retrieval is small, compared to that spent in the interface, the natural extension is Fig. 4.3. The model is practically identical to that in Fig. 3.2 except for the station that represents the retrieval delay from the DBMS. Note that this model features two related subnetworks. Model 2.1 includes all the delays that exams encounter in circulating around the network. The only queueing delays are encountered in model 2.2. Model 2.2 only examines the delay that exams, each comprising a series of images, encounter in the interface.

In model 2.2, the network interface unit experiences bulk arrivals of image records

Figure 4.3. An approximate equivalent, studied numerically



Sub-model 2.1: Product-form queueing network to estimate arrival rate of exams to sub-model 2.2



Sub-model 2.2: State-dependent bulk arrivals to the interface

because the retrieval delay incurred by each image is neglected. The size B of the bulk is the mean number of images in an exam. All workstations are assumed to be homogenous with regard to their exam request patterns as well as think and idle time distributions.

However these arrivals are throttled because requests for exams from individual workstations are single-threaded which guarantees that there can never be more than four exams in service at the interface. This model is therefore a system with state-dependent bulk arrivals to an exponential server. The transactions are exam requests but service is by phases, i.e. image records. The parameter of the arrival process, $\lambda(n)$, with $n = 0,1,...,4$, is the rate at which exam requests arrive at the interface, given that there are $(4-n)$ exam requests already pending.

Model 2.1 is used to determine $\lambda(n)$. Transactions circulating in this closed network are exam requests. Conditioned on the fact that there are $(4-n)$ exam requests already pending, (i.e. awaiting service in model 2.2), application of Norton's Theorem [5] (short out the sub-network and measure the throughput of transactions in the aggregate) immediately gives the throughput of transactions. In the general case, the throughput of transactions in the aggregate has to be obtained by studying it as a closed queueing network with n transactions, but for the case of model 2.1, in which all stations are of the "infinite-server" type, the throughput can be written by inspection as:

$$\lambda(n) = \frac{n}{T_{req} + p_{next} T_{idle} + T_{link} + T_{think}} \qquad n = 0,1,...,4$$

because all stations are modeled as "infinite-server" queues with mean service times of the request delay, idle interval, transmission time (scaled up for an exam) and think time set at T_{req}, T_{idle}, T_{link}, T_{think} respectively and p_{next} is the probability that any exam is the last in the batch.

With this expression for $\lambda(n)$ available, we return to model 2.2. The state descriptor used for this station is (i,j) where:

i is the number of exam requests in the interface and

j is the number of pending image records of the exam request currently being served.

An exam is considered to complete service only when all its pending component image records have been served. Exam sizes are between 2 and 32 images; in this case we only allow exam sizes of 2, 8 or 32 images. Clearly there are 129 possible states in which the system can be found.

We note that because of the assumption that the network interface sees bulk arrivals of exam requests, there will be no interleaving of image records belonging to different exams (as should really have been the case) in the interface. The implication of this behavior is that the system basically serves exam requests to completion on a "first-come-first-served" basis rather than on a "per image record" basis.

The underlying continuous time Markov chain of state transitions for model 2.2 is illustrated in Fig. 4.4.

The steady-state equations from which the state probabilities can be determined are:

$$\mu p(1,1) = \lambda(4)p(0)$$

$$(\lambda(3)+\mu)p(1,j) = \lambda(4)p_j p(0) + \mu p_j p(2,1) + \mu p(1,j+1)$$

$$(\lambda(2)+\mu)p(2,j) = \lambda(3)p(1,j) + \mu p_j p(3,1) + \mu p(2,j+1)$$

$$(\lambda(1)+\mu)p(3,j) = \lambda(2)p(2,j) + \mu p_j p(4,1) + \mu p(3,j+1)$$

$$\mu p(4,j) = \lambda(1)p(3,j) + \mu p(4,j+1)$$

where p_j is the probability that the exam contains j images and $j = 0,1,...,32$.

The rate matrix Q is depicted in Fig. 4.5 and the state vector Π is of the form

Figure 4.4. Markov chain of state transitions for model 2.2

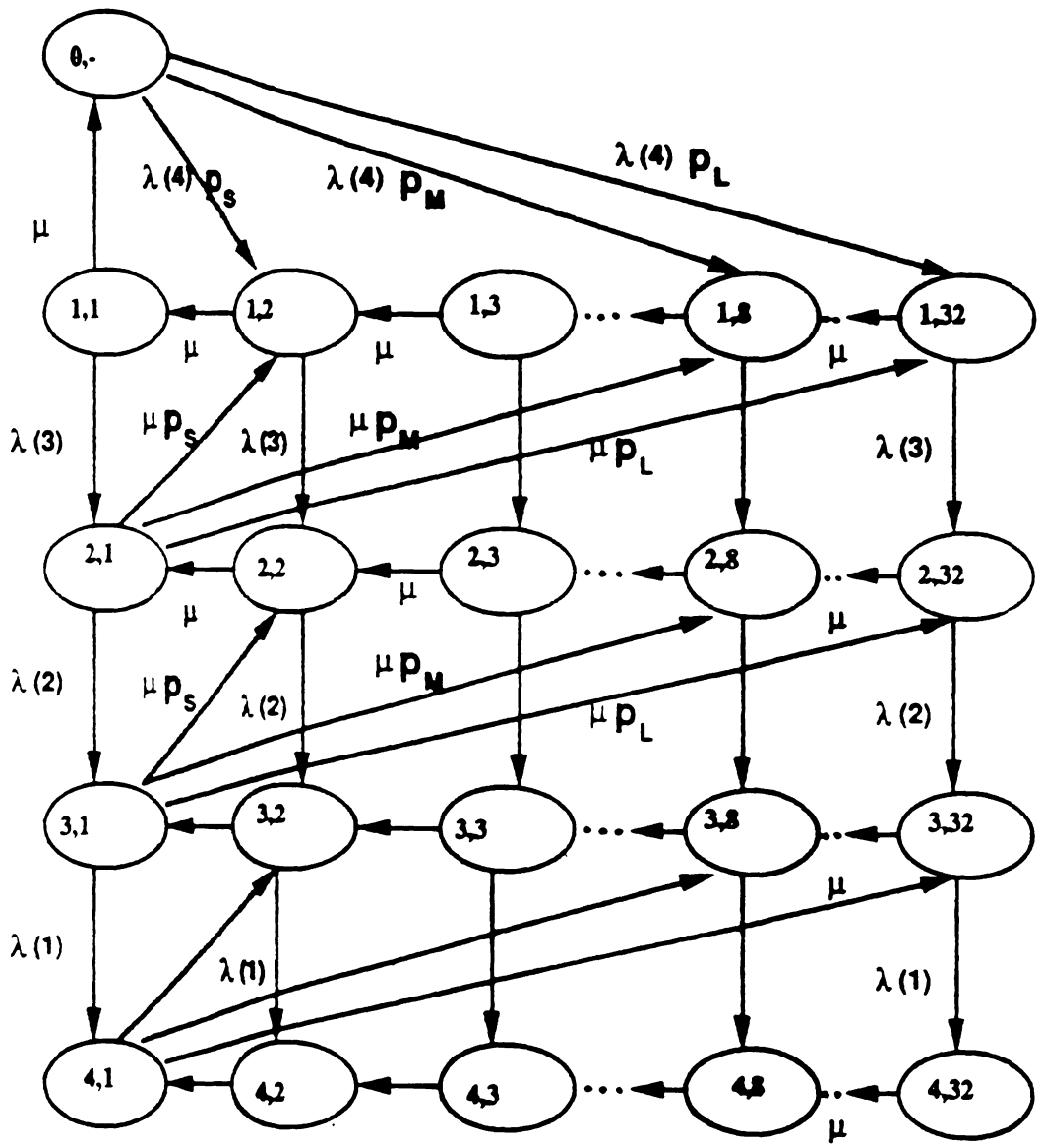


Figure 4.5. Structure of the rate matrix for model 2.2

$$\begin{bmatrix}
-\lambda(4) & \mu & 0 & 0 & - & 0 \\
\lambda(4)p_1 & -(\lambda(3)+\mu) & \mu & 0 & - & \mu p_1 \\
.. & 0 & -(\lambda(3)+\mu) & \mu & - & \mu p_2 \\
\lambda(4)p_{32} & 0 & 0 & -(\lambda(3)+\mu) & - & .. \\
0 & \lambda(3) & 0 & 0 & - & .. \\
.. & 0 & \lambda(3) & 0 & - & .. \\
.. & .. & 0 & \lambda(3) & - & .. \\
.. & .. & .. & 0 & & .. \\
.. & .. & .. & .. & - & .. \\
.. & .. & .. & .. & - & .. \\
0 & 0 & 0 & 0 & - & ..
\end{bmatrix}$$

$$[p(0)\ p(1,1)\ \cdots\ p(1,32)\ \cdots\ p(4,1)\ \cdots\ p(4,32)]^T$$

Using $Q^T \Pi = 0$, which yields 128 independent equations, with the additional constraint that all probabilities should sum to unity,

$$\sum_{i=1}^{4}\sum_{j=1}^{32} p(i,j) = 1$$

this system can be solved for the state probabilities. As the system is small, with only 129 possible states, a direct Gaussian elimination technique was used.

From the state probabilities, the following performance measures can be found:

utilization of the network interface unit as $1 - p(0,-)$,

throughput of exam requests as $\sum_{i=1}^{4}\sum_{j=1}^{32} \lambda(i)p(i,j)$,

buffer requirements at the network interface units as $\sum_{j=1}^{32}\sum_{i=1}^{4} jp(i,j)$

The entire solution technique was written as a C program with parameters being T_{req}, T_{idle}, T_{link}, $T_{interface}$ and distributions of exam size and batch size. The listing of the program is in Appendix 1. This was run for a range of parametric variations.

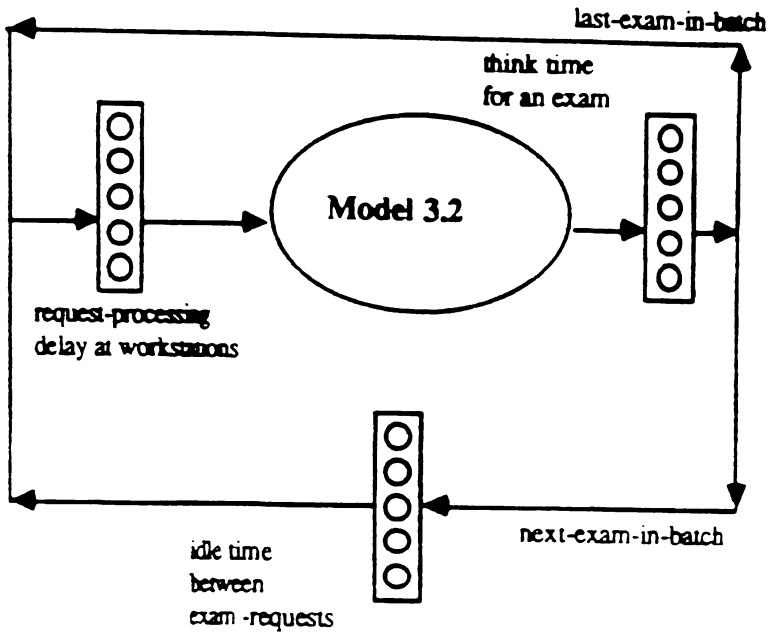
4.3 An Improved Approximation

In this model, depicted in Fig. 4.6, we discard the assumption that the database retrieval time is negligible compared to the time an image spends in the interface. As a result, the queueing network model closely resembles the one in Fig. 3.2 that was studied by simulation. Again, this model features two related subnetworks - model 3.1 includes all the delays that exams encounter in circulating around the network while model 3.2 only examines the delay that images encounter in database retrieval and in the interface. In model 3.1 it is assumed that there are no queueing delays. As before, all workstations are assumed to be homogenous with regard to their exam request patterns as well as think and idle time distributions.

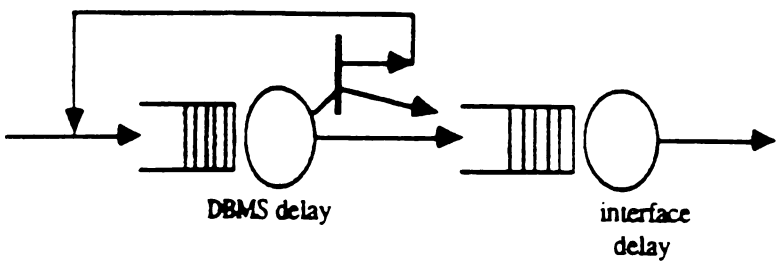
Consider model 3.2 in Fig. 4.6. The goal is to find the departure process of image records from the first queue (which represents the database) so that submodel 3.2 could be studied as a $G/M/1$ queue with state-dependent parameters that can be found from its complement, model 3.1, similar to the approach followed for the previous approximation.

To review the service mechanism of the database retrieval: on service completion, a transaction proceeds onward, but, in addition, may, with some probability, spawn a sibling

Figure 4.6. An improved approximation, solved by an iterative technique



Sub-model 3.1: Product-form queueing network to estimate arrival rate of exams to sub-model 3.2



Sub-model 3.2: G/M/1 system

which is fed back to the queue. Fresh transactions represent exam requests, while spawned and fed-back transactions represent image records. Again, the geometric distribution of the number of records in an exam is used to determine the spawning probabil-

ity. Each exam is made up of B fragments so that the probability p that any fragment spawns a sibling fragment is $p = (B-1)/B$.

Assume that the arrival process of fresh exams to the database in model 3.2 is a Poisson process with parameter λ. This is the first type of transaction in the database station. Let p be the probability that a sibling is spawned. Consider q_{n+1}, the number of transactions (now images) left behind by the departure of the $(n+1)$st transaction. This quantity depends on V_{n+1}, the number of arrivals during the departure process of the $(n+1)$th job as follows:

$$q_{n+1} = \begin{cases} q_n + V_{n+1} & \text{with probability } p \text{ if } q_n > 0 \\ q_n - 1 + V_{n+1} & \text{with probability } (1-p) \text{ if } q_n > 0 \\ 1 + V_{n+1} & \text{with probability } p \text{ if } q_n = 0 \\ V_{n+1} & \text{with probability } (1-p) \text{ if } q_n = 0 \end{cases}$$

In other words, the number of transactions left behind following the departure of the $(n+1)$st transaction depends on both, the state of the queue on departure, as well as the number of fresh arrivals during the departure process.

Consider the random variable q_n. Its z-transform is

$$Q(z) = \sum_{k=0}^{\infty} Pr[q_n = k] z^k = E[z^{q_n}]$$

We proceed to find $E[z^{q_{n+1}}]$, the z-transform of the probability of finding q_{n+1} transactions in the system immediately following the departure of a transaction. This can be written as:

$$E[z^{q_{n+1}}] = E[z^{q_{n+1}} | q_n = 0]q_0 + \sum_{k=1}^{\infty} E[z^{q_{n+1}} | q_n = k]q_k$$

where

$$E[z^{q_{n+1}} | q_n = 0] = pE[z^{1+V_{n+1}}] + (1-p)E[z^{V_{n+1}}]$$

and

$$E[z^{q_{n+1}} | q_n = k] = pE[z^{k+V_{n+1}}] + (1-p)E[z^{k-1+V_{n+1}}]$$

Denoting $E[z^{V_{n+1}}]$ and $E[z^{q_{n+1}}]$ by $V(z)$ and $Q(z)$ respectively,

$$Q(z) = (pz + 1 - p)V(z)q_0 + \sum_{k=1}^{\infty} q_k z^{k-1}(pz + 1 - p)V(z)$$

$$Q(z) = (pz + 1 - p)V(z)[q_0 + (\frac{1}{z})(Q(z) - q_0)]$$

$$Q(z) = \frac{(pz + 1 - p)V(z)q_0(z - 1)}{z - V(z)(pz + 1 - p)}$$

q_0 can be obtained by using L'Hôspital's rule and the following:

$$Q(1) = 1 \; ; \; V(1) = 1 \; ; \; V^{(1)}(1) = \rho$$

the third equation representing the average number of fresh exam arrivals during the service interval of an image. Finally,

$$q_0 = (1 - \rho - p)$$

from which we obtain a useful condition relating the utilization of the database station with the upper limit on the probability of a fresh arrival spawning a transaction as

$$p \leq (1 - \rho)$$

Finally, x_1, the mean queue length in the first station is found by applying L'Hôspital's rule twice to the expression for $Q^{(1)}(z)$ with $V(z)$ set to $B^*[\lambda(1-z)]$ where $B^*[s]$ is the Laplace Transform of the (exponential) service time distribution with mean $\frac{1}{\mu}$:

$$x_1 = \frac{q_0\mu(\lambda - p\lambda + \mu p - \mu p^2)}{(\mu - \lambda - \mu p)^2}$$

and R_1, the mean response of the first station is obtained from Little's Law:

$$R_1 = \frac{x_1}{B\lambda}$$

where B is the average number of images in an exam. Note that by introducing the average exam size B, we have converted the mean response time per *image* at the first station to an approximate response time per *exam*.

Conditioned on the fact that arrivals to the queue follow an exponential distribution, the Laplace Transform of the departure process can be written as

$$D^*(s) = q_0 \frac{\lambda}{(\lambda+s)} \frac{\mu}{(\mu+s)} + (1 - q_0) \frac{\mu}{(\mu+s)}$$

Thus the departure process of the first queue, which is, in turn, the arrival process to the second queue, is known and model 3.2 can now be studied as a $G/M/1$ system.

For a $G/M/1$ system, if $A^*(s)$ is the Laplace Transform of the arrival process and μ is the mean service time, the distribution of r_k, the number of customers at arrival instants is, from [8],

$$r_k = (1 - \sigma)\sigma^k \qquad k = 0,1,...$$

where σ is the unique root of

$$\sigma = A^*(\mu - \mu\sigma) \qquad 0 < \sigma < 1$$

The mean queue length, N and response, R, are

$$N = \frac{\sigma}{1 - \sigma} \quad ; \quad R = \frac{1}{\mu(1 - \sigma)}$$

In our case, we have to find σ from:

$$\sigma = A^*(\mu_2 - \mu_2\sigma)$$

and

$$A^*(s) = q_0 \frac{\lambda}{(\lambda+s)} \frac{\mu_1}{(\mu_1+s)} + (1-q_0)\frac{\mu_1}{(\mu_1+s)}$$

where μ_1 and μ_2 are the mean service rates of the database and interface stations.

After some algebraic manipulations and factoring out $(\sigma - 1)$ as a root, the required solution is found from:

$$A_2\sigma^2 + A_1\sigma + A_0 = 0 \qquad 0 < \sigma < 1$$

and the coefficients of this quadratic equation in σ are:

$$A_2 = \mu_2^2$$

$$A_1 = -\mu_2(\mu_2 + \mu_1 + \lambda)$$

$$A_0 = \mu_1(\mu_2 - q_0\mu_2 + \lambda)$$

from which x_2, the mean queue length, and R_2, the mean response, at the second station are calculated as

$$x_2 = \frac{\sigma}{1-\sigma} \quad ; \quad R_2 = \frac{1}{\mu_2(1-\sigma)}$$

The only unknown quantity that remains is λ, the arrival rate of fresh exams to model 3.2. Unlike for the case of the previous model, the transactions circulating in the two sub-networks are not the same. Transactions in model 3.1 are exams while transactions in model 3.2 are images. Hence the technique adopted in the previous approximation can no longer be used directly to find λ. An arbitrary value of λ will have to be used as a starting point. Next, an approximation will be used to find the mean response time of an exam in model 3.2. The sojourn time of a single image in model 3.2 is represented by

$(R_1 + R_2)$. With the approximation that the mean time that an exam, whose mean size is B images, will spend will be B times as long as that of a single image, the sojourn time, T_{soj}, spent by an exam in model 3.2 can be used to find the throughput of exams in model 3.1 as before:

$$\lambda = \frac{4}{T_{req} + T_{soj} + p_{next} T_{idle} + T_{link} + T_{think}}$$

because all stations are modeled as "infinite-server" queues with mean service times of the request delay, idle interval, transmission time (scaled for an exam) and think time set at T_{req}, T_{idle}, T_{link}, T_{think} respectively and p_{next} is the probability that any exam is the last in the batch.

An iterative scheme, outlined below, suggests itself:

Step 1: Assume an arrival rate, $\lambda^{(0)}$ of exams to model 3.2

Step 2: Solve the $G/M/1$ system and find the sojourn time of an image in this model as the sum of the mean response times at the two stations

Step 3: Extend this result to estimate the time that an exam would spend in this model by multiplying it by the mean exam size

Step 4: Use this delay as the service time for the equivalent (which is an "infinite-server" station) and find the throughput of exams in the product form queueing network that is model 3.1

Step 5: Use this value as an updated arrival rate, $\lambda^{(1)}$, for Step 1 and repeat this algorithm till a convergence criterion is reached

The performance measures that can be found are the occupancy of the interface,

throughput of exams and buffer requirement at the interface. The entire solution technique was written as a C program with parameters being T_{req}, T_{idle}, T_{link}, $T_{interface}$, T_{dbms} and distributions of exam size and batch size. The listing of the program is in Appendix 2. This was run for a range of parametric variations. The convergence criterion was set to reaching a specified tolerance between two successive values of the arrival rate of fresh exams.

CHAPTER 5

RESULTS AND DISCUSSION

5.1 Parameter Variations

The request processing delay includes all the delays from the point an exam request is keyed in at the display station upto the point when the first image is about to be read off the database. It is assumed that the request message, and its acknowledgement, are smaller than and have higher priority than the actual images. As a consequence, these transactions experience only the service times of the intermediate stations and no queueing delays. This parameter is set to follow an exponential distribution with the mean allowed to vary from 300 millisec to 900 millisec. This delay is associated with each exam. For radiologists using a worklist, this delay will still be incurred by every exam in the worklist.

The communications link time represents the delay in moving images from the network interface unit to the workstations. As each workstation has its own link to the interface, and if, as is very likely, there are four different communications server processes in the interface, again it is clear that there will be practically no queueing delays on this link. Implicitly it has been assumed that the transmission delay is smaller than the interface delay. Images are transmitted uncompressed on this link. The mean of this exponentially distributed service time is allowed to vary between 1.0 and 2.0 sec per image. Being a composite, any variation in the link level communications protocol does not directly affect this parameter as the value can be calibrated depending on window sizes at

different layers and other protocol details. Note that it represents the delay incurred in transactions moving across the link, hence software overhead would eat away at the large bandwidth of the chosen communications medium.

The size of the transaction at the session layer is 256 KB, abbreviated as SPF (for standard picture frame). The compression ratio is kept fixed at 2:1. An entire image must be reassembled in interface memory from its component frames (on the communications link).

The think time and idle time distributions were also assumed to be exponential with means varying from 60 sec to 240 sec per exam in each case. It is assumed that there is only one server process available at each workstation that retrieves images; moreover it gets blocked even when the user is in the "think" phase.

The database retrieval time is again a composite. It includes all the delays incurred by an image from the time the exam request arrived at the database to the point when the image is available in interface memory. We assume that it is not possible to pipeline the two components of this delay, i.e. the disk read from the database and the transmission delay to the interface. These can be separated out into two different stations, an upstream database retrieval and a downstream transmission delay, should pipelining in fact be possible. Even in the latter case, if there are multiple interfaces connected to the central database, the pipelining effect of images between the database and the respective interface station would be effectively very small. Additionally, we assume that the time to locate the exam is fixed, i.e. there is no distinction between memory levels (such as on-line magnetic storage and secondary storage on an optical disk archive). The range of values

chosen was 1.0 to 4.0 sec, exponentially distributed, per image.

The interface delay will clearly vary with the functions that it needs to perform. In general, however, this interface can be anticipated to be a system bottleneck primarily because it will invariably have several display stations connected. For the configuration we studied, this was set to be an exponentially distributed parameter with mean between 0.75 and 2.0 sec per image.

Finally, the exam size distribution varied over mixes of small (2 images/exam), medium (8 images/exam) and large (32 images/exam). exams while the batch size distribution varied from 1 to 20 exams per batch. Detailed studies of patterns of radiographic usage, such as [1], list the specific distributions for certain modalities.

In the results presented over the following pages, the following key is used on the abscissa:

A: DBMS retrieval 4.0 sec, Interface delay 2.0 sec

B: DBMS retrieval 2.0 sec, Interface delay 2.0 sec

C: DBMS retrieval 2.0 sec, Interface delay 1.0 sec

D: DBMS retrieval 1.0 sec, Interface delay 1.0 sec

E: DBMS retrieval 1.0 sec, Interface delay 0.75 sec

5.2 Performance Criteria

For referring physicians and radiologists at display stations, the primary consideration is the end-to-end response on each image. There would be local memory at the display

stations for storing previously retrieved images or performing any signal processing thereon.

For technicians at acquisition stations and image archival and database staff, throughput of images becomes more important. In each case, images are always archived from disk, never directly from the modality.

At the interface, the over-riding concern is to identify the bottleneck resource and design the system around this. This is manifested in, for instance, specifying buffer requirements and response per image at the interface.

5.3 Results and Discussion

In Figs. 5.1 through 5.9, selected results of the modeling exercise are presented. Though results were obtained over a wide parametric variation, only variations in traffic mix are considered for presentation purposes.

The end-to-end delay is computed by adding up the mean response time per image at the stations that represent the request processing delay, the database, the interface and the second communications link.

Over controlled variation in the database retrieval and interface delays, running along the abscissa, a family of curves is presented for a particular exam size distribution (in images) and batch size (in exams). The three curves, labeled **M1**, **M2**, **M3**, in each graph represent results obtained from simulation of the queueing network of Fig. 3.2, the approximation illustrated in Fig. 4.3 and the approximation illustrated in Fig. 4.6 respec-

tively.

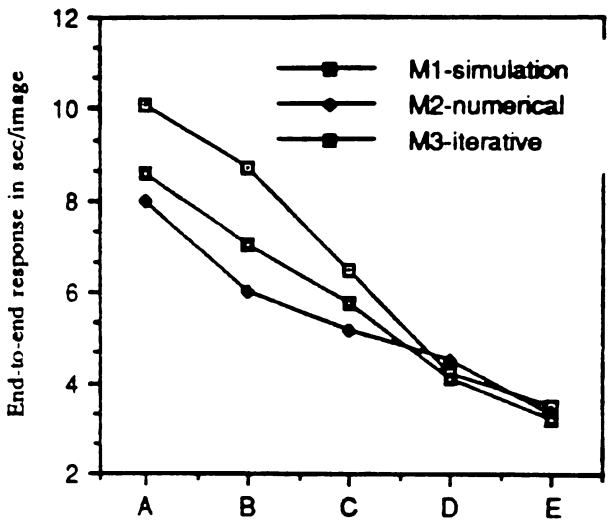
The delay and queue length at the interface decrease, as expected, with lower database retrieval and interface delays. At the same operating point, if either the batch size in exams or the exam size in images is increased (from 1 to 10 exams per batch and 5 to 14 images per exam respectively), the delay and queue length increase, as expected. The "think" and "idle" time delays would clearly have had an effect on the slope of the curves - e.g. being comparable to the delays of the other stations would have resulted in sharper variations; but they were held constant throughout. The transmission time and compression ratio were also unchanged.

These characteristics are observed irrespective of the model chosen for examination. However, these graphs also reveal that both the approximations almost invariably result in optimistic predictions, i.e. they over-estimate system capacity in some way. Moreover the first approximation is invariably inferior to the second.

In the first approximation, the optimistic trend is easy to explain. The database retrieval delay was ignored in computing the queue length at the interface, though it was patched on as a component in the calculation of end-to-end delay. In the second approximation, as the database retrieval is considered when computing the performance measures of the interface, the second model can be expected to be more accurate than the first approximation. Less easy to explain is why the second model also results in conservative estimates. We surmise that the pessimistic approximation resulting from assuming that the arrival process of fresh exams is Poisson is counterweighed by the optimistic assumption that the delay encountered by an image in the interface is scaled down linearly from that

encountered by an exam.

Figure. 5.1 (Tx time=1 image/sec, Exam sizes either 2,8 or 32 images, Batch size =1 exam,Think time=60 sec, Idle Time=120 sec)



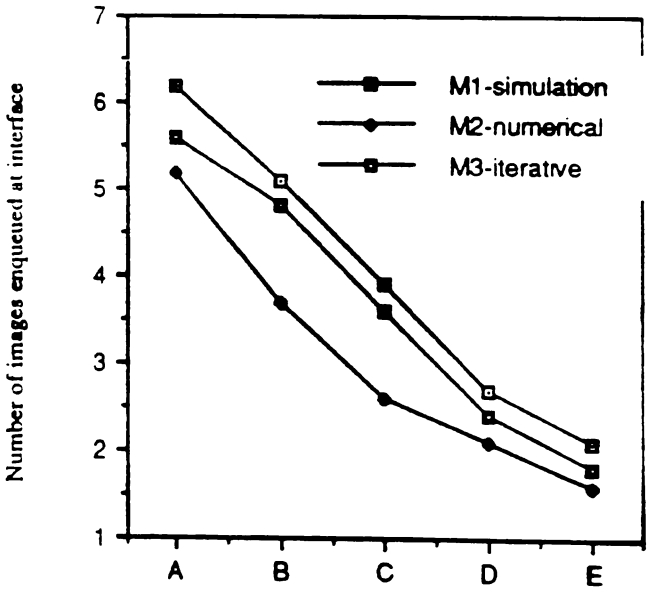
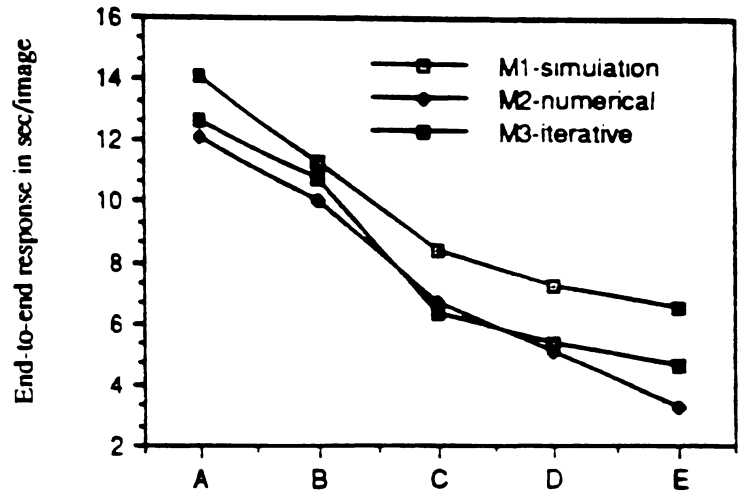


Figure 5.2 (Tx time=1 image/sec, Exam sizes either of 2,8 or 32 images, Batch size of 5 exams, Think Time=60 sec, Idle Time=120 sec)



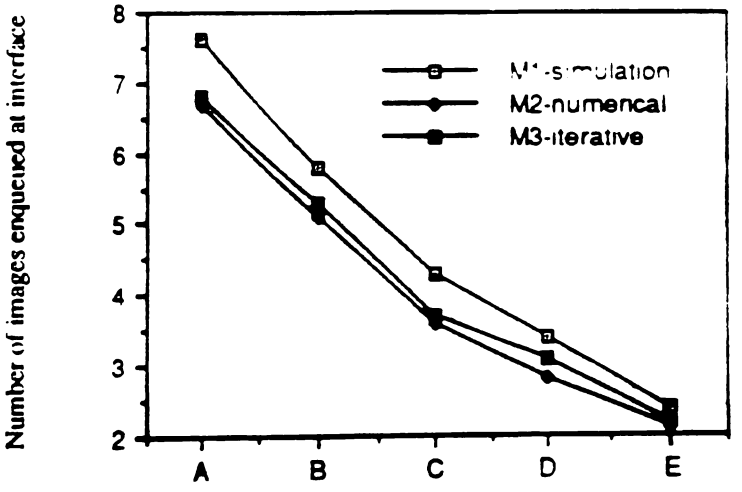
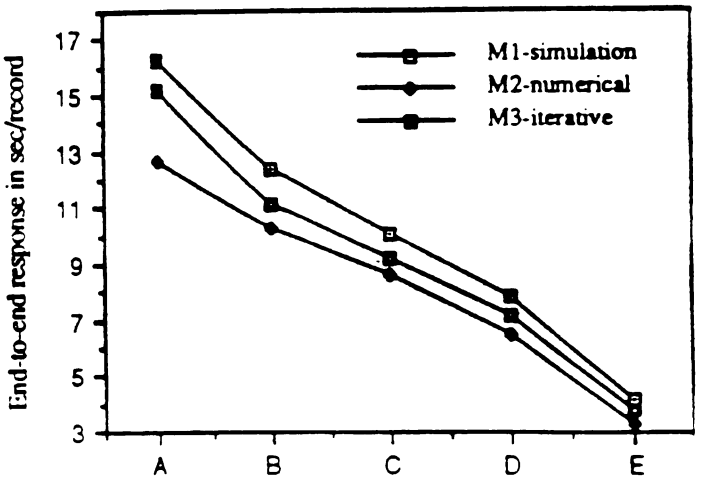


Fig. 5.3 (Tx time=1 record/sec, Exam sizes either of 2, 8 or 32 records, Batch size=10 exams, Think time=60 sec, Idle Time=120 sec)



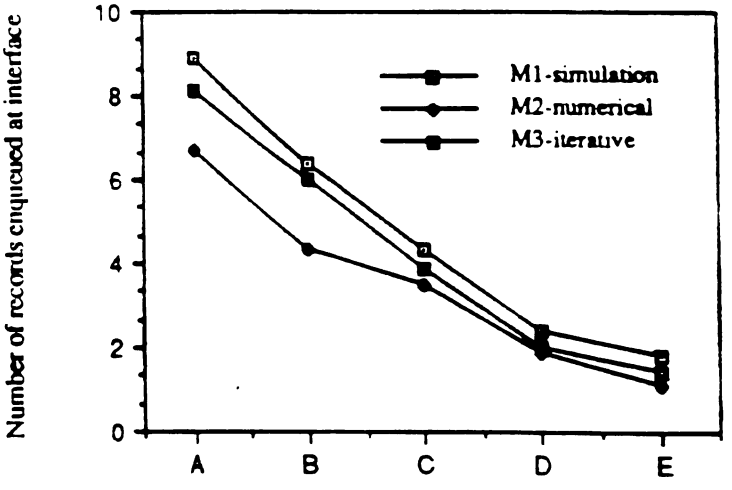
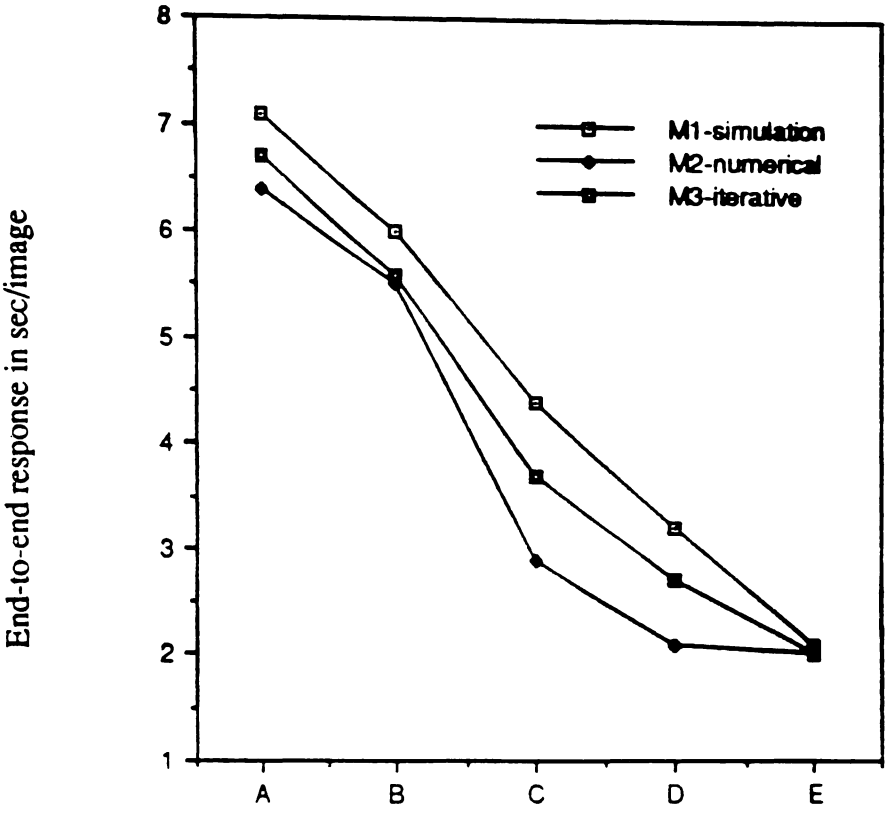


Figure 5.4 (Tx time = 1 img/sec, Exam = 2 or 8 images, Think = 60 s, Idle = 120 s, Batch = 1 exam)



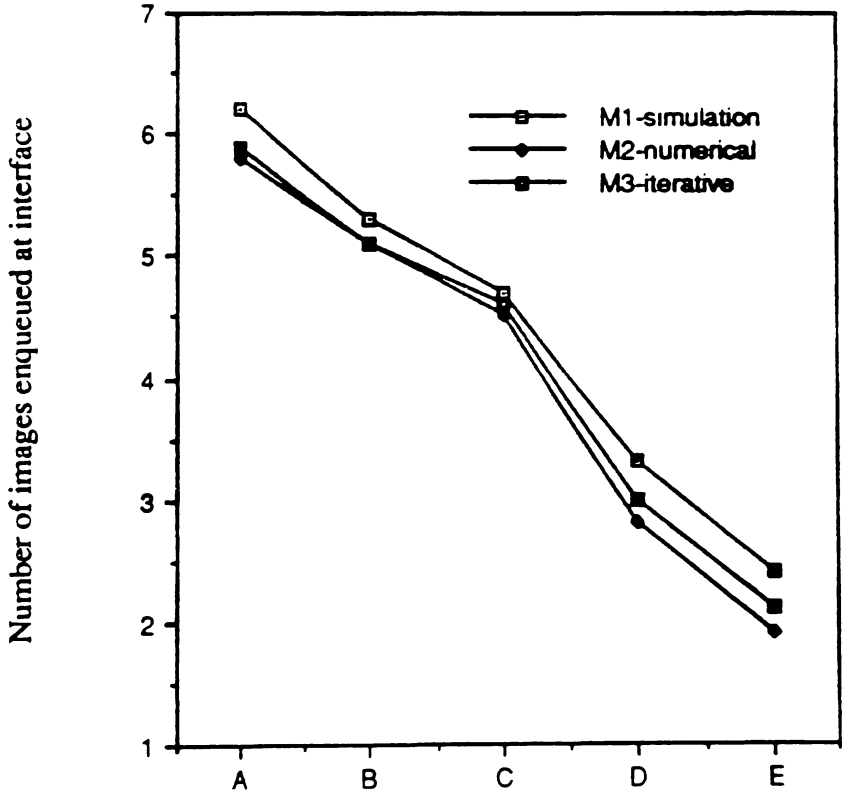
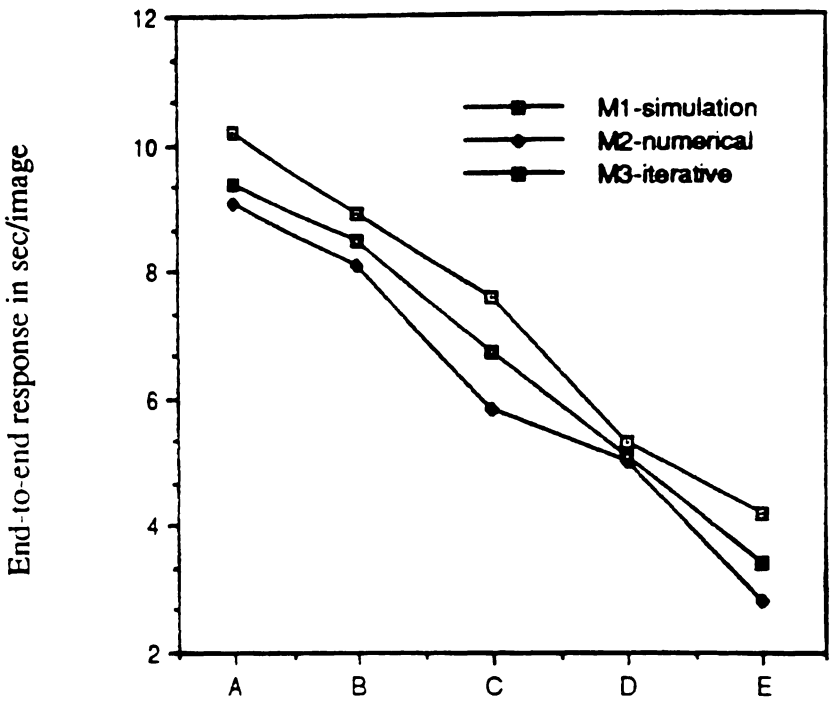


Figure 5.5 (Tx time = 1 img/sec, Exam = 2 or 8 images, Think = 60 s, Idle = 120 s, Batch = 5 exams)



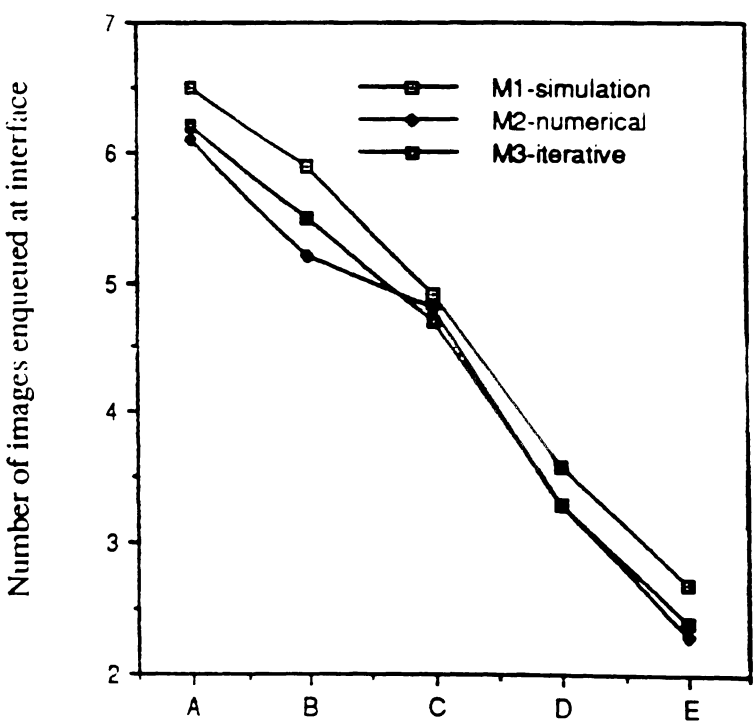
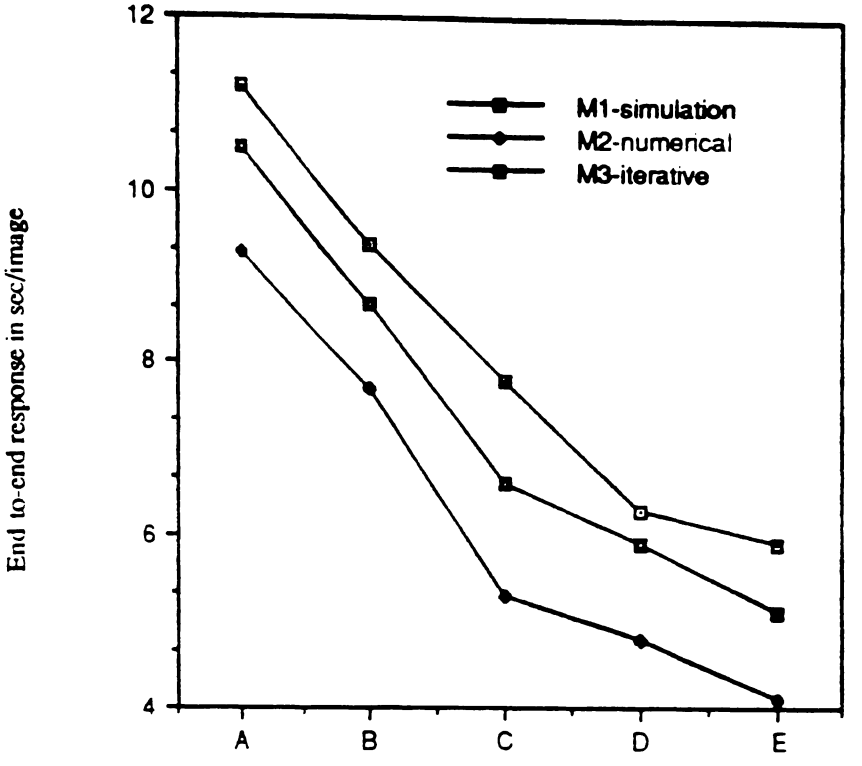


Figure 5.6 (Tx time = 1 img/sec, Exam = 2 or 8 images, Think = 60 s, Idle = 120 s, Batch = 10 exams)



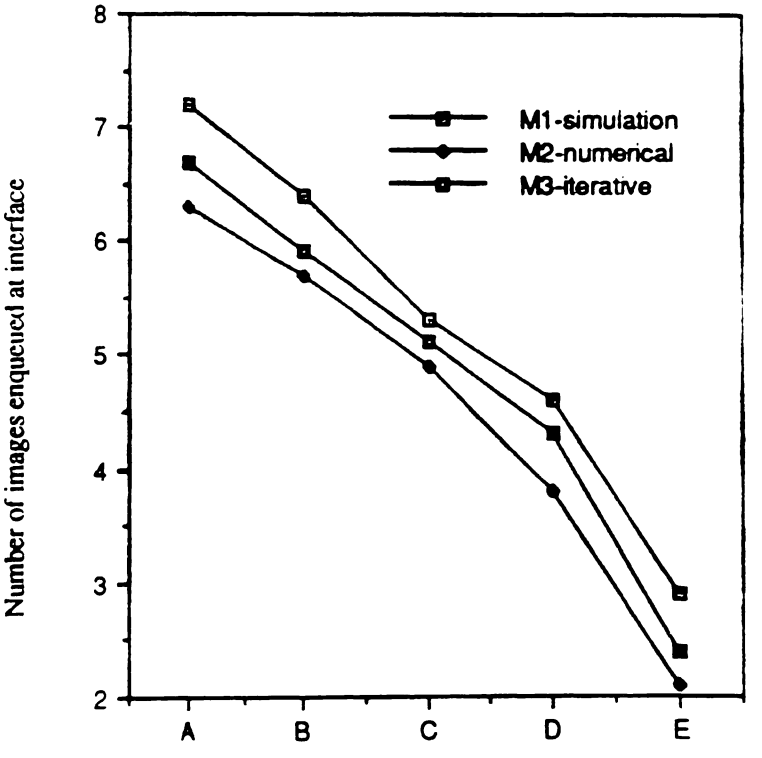


Figure 5.7 (Tx time = 1 image/sec, Exams 8 images, Batch = 1 exam, Think time = 60 s, Idle time = 120 s)

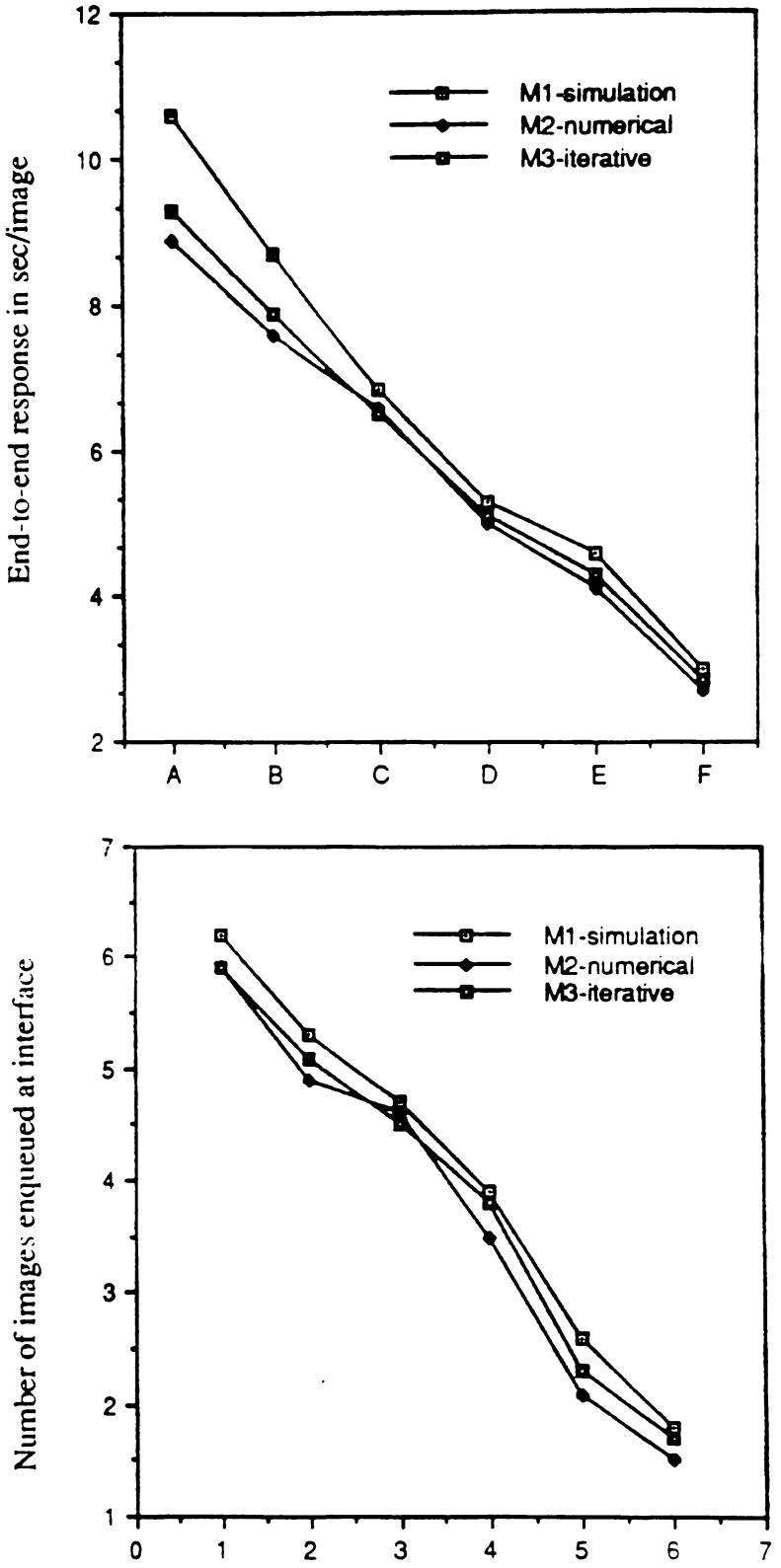


Figure 5.8 (Tx time = 1 image/sec, Exams 8 images, Batch = 5 exams, Think time = 60 s, Idle time = 120 s)

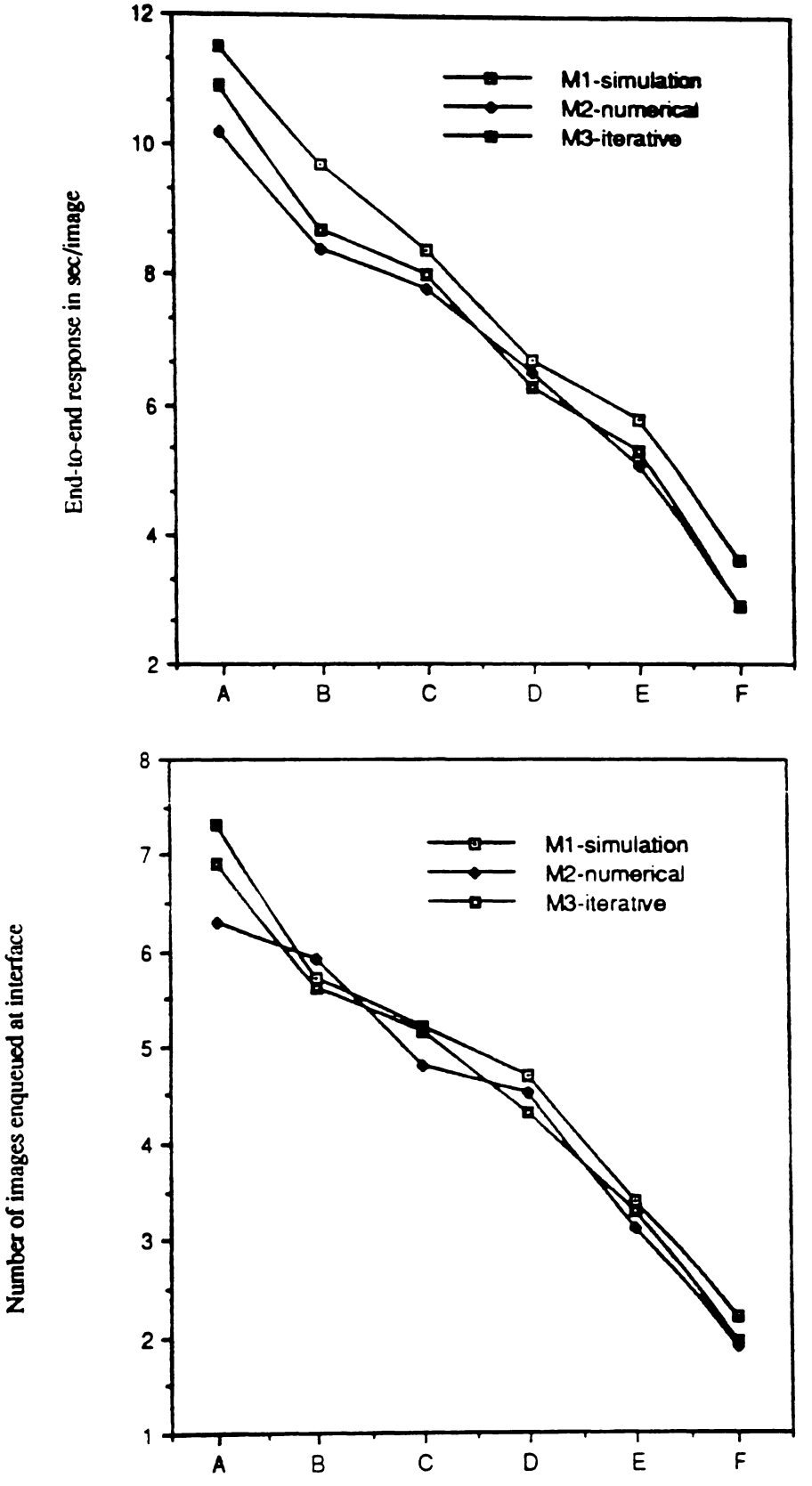
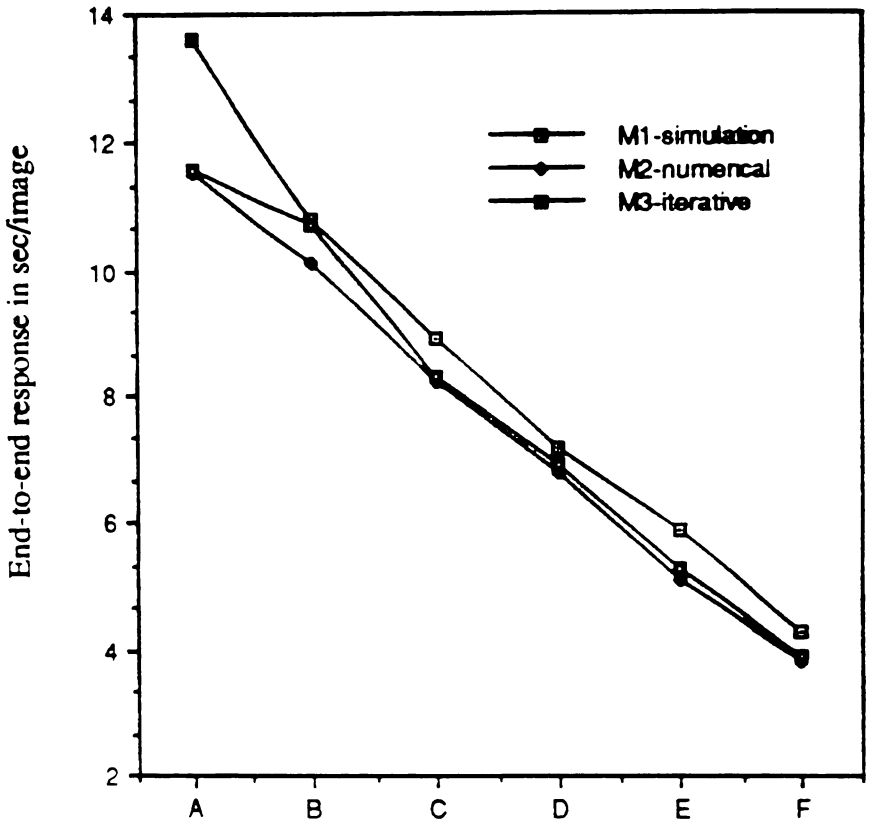
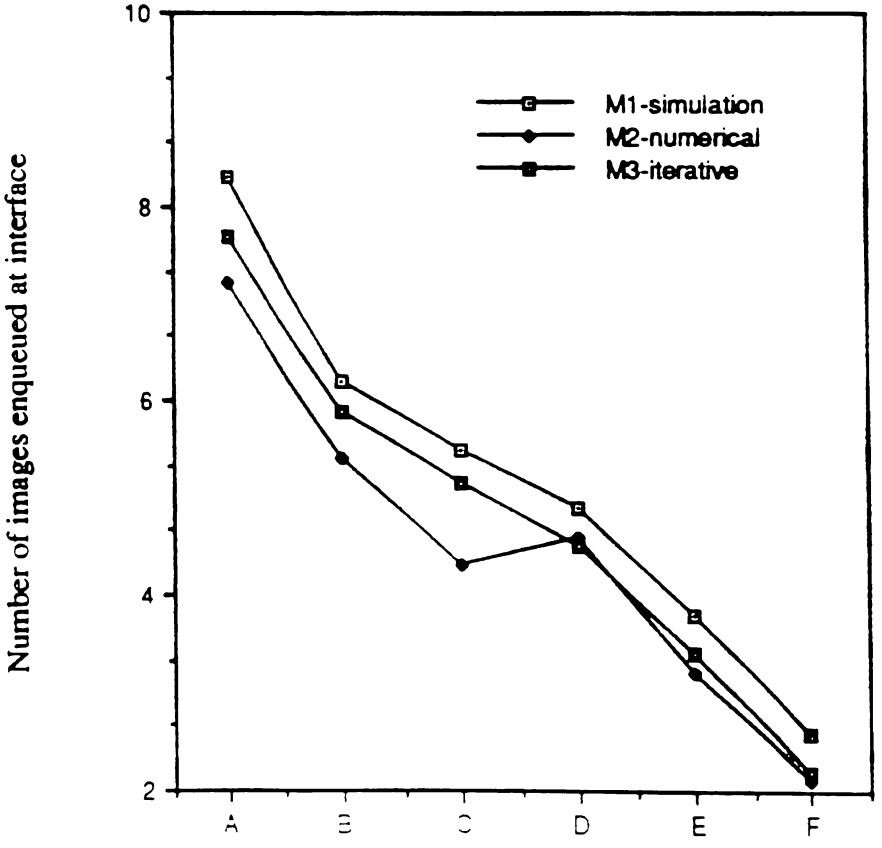


Figure 5.9 (Tx time = 1 image/sec, Exams 8 images, Batch = 10 exams, Think time = 60 s, Idle time = 120 s)





CHAPTER 6

SIMULATION MODEL OF AN ACQUISITION STATION

6.1 Introduction

The configuration that will be examined in this chapter is illustrated in Fig. 6.1. There is an acquisition station that is connected to a single imaging modality, such as a film digitizer, that archives images onto the centralized database through a gateway that resolves any communications incompatibilities that arise between devices. Images are acquired uncompressed but need to be stored in compressed form in the database.

6.2 System Description

We now present a queueing network representation of the above system in Fig. 6.2.

Figure 6.1. The configuration under study

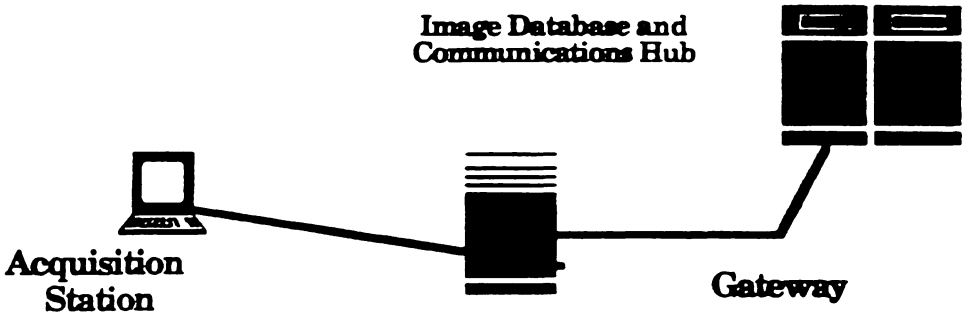
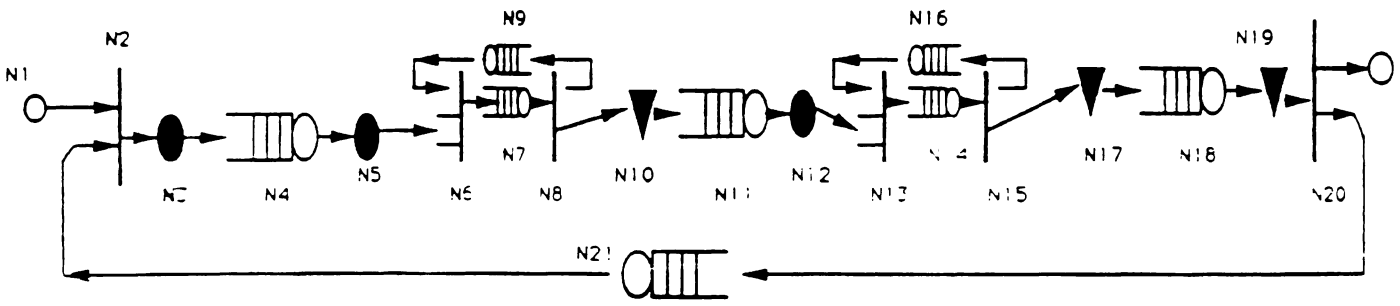


Figure 6.2. A queueing network representation of the system



Exams, such as radiographic films that are to be digitized, are fed into the acquisition station, *N1*, at periodic intervals. Processing of exams does not start immediately. The archival of new exams is flow-controlled across the interface at synchronization station

N2. Once an exam is scanned, at *N3*, a set of images results. Each of these images undergoes some protocol processing delay, represented by station *N4*. Individual images are split into frames at *N5* before being put out onto the communications link *N7* which is flow controlled at *N6*. Frames are reassembled at the half-gateway, *N10*, of the interface to form images which experience some signal and/or protocol processing delay at the interface, *N11*. Images must now be fragmented, at *N12*, before being sent out to the database over communications link *N14* which is flow-controlled at *N13*. Images are reassembled from frames at *N17* and exams are reassembled from images at the database *N19* after some communications protocol processing at *N18*. *N8*, *N15* and *N20* are stations that result in permits for subsequent frames (or exams) to be returned to the originating station in each case because of successful transmission of frames (or exams) across the communications link (or interface). Finally, *N9*, *N16* and *N21* represent the delays that the acknowledgements encounter.

The level of detail that may be captured in this model is considerably more than that possible in models examined in earlier chapters. The effect of changing a communications system parameter such as frame size or window size can be studied. The drawback is that the system needs to be studied by simulation as the fragmentation and merge mechanisms are difficult to study analytically.

6.3 The Simulation Model

The discrete-event simulation model was written in C and run on a 3B2-300 personal computer. A listing of the program is in Appendix 3.

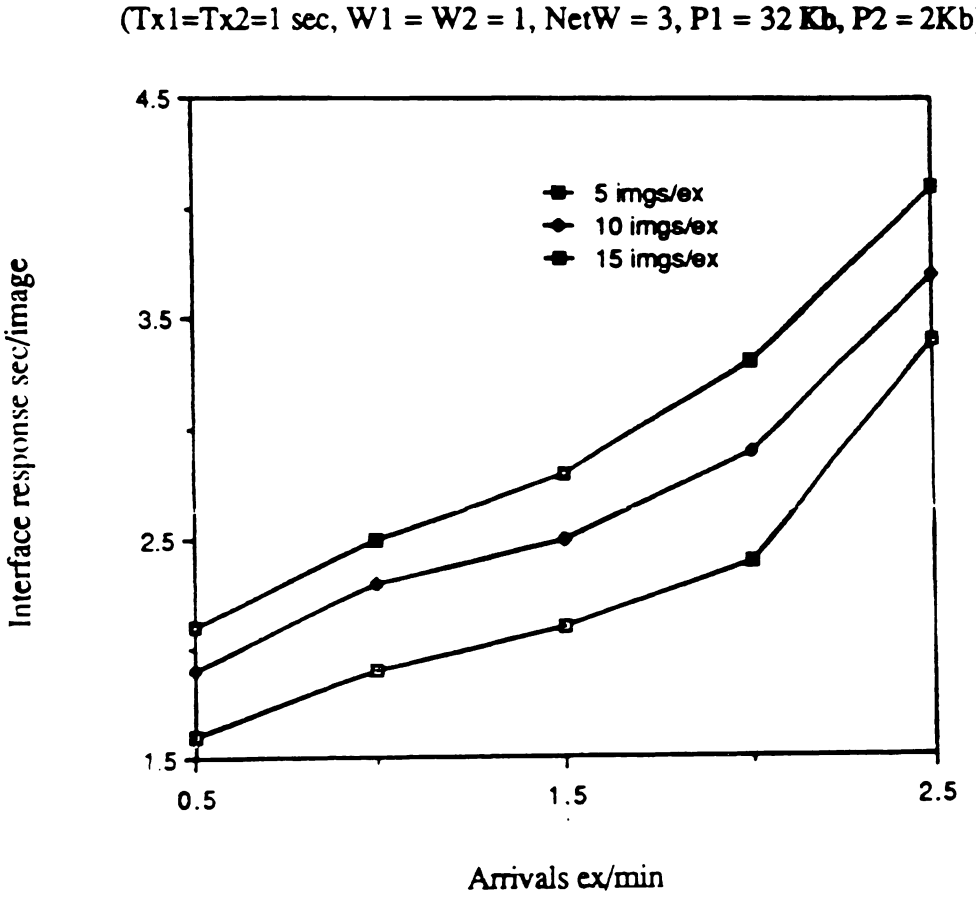
There are five different kinds of nodes in the network. They are denoted as *split*, *merge*, *sync*, *fork* and *delay* stations. The *split* stations are *N3, N5* and *N12*; the *merge* stations are *N10, N17* and *N19*; the *sync* stations are *N2, N6* and *N13* while the *fork* stations are *N9, N15* and *N20*. Two global data structures are maintained: one is the event list, ordered according to the next chronological event that will be performed, while the second is an array of queues that is used to keep track of performance measures at the different nodes. There are as many different kinds of events as there are nodes; execution proceeds according to the type of the next event that is popped off the event list.

The sustained arrival rate of exams over the length of time simulated was a parameter. The principal inputs to the model are the protocol processing delays at the two nodes, the number of images per exam, the window sizes for flow control over the two communications links and the interface delay.

The transmission delays on the links and the compression ratio were fixed.

Some graphical results are presented in Fig. 6.3 to show the variation of the delay at the interface with different exam sizes and arrival rates. In this case the network-wide window flow control is set to three transactions and the frame sizes on the links are set to 32 KBytes and 2 KBytes.

Figure 6.3 Variation of delay at the interface with different exam sizes and arrival rates.



CHAPTER 7

CONCLUSIONS

In this thesis a set of queueing network models were constructed with the intention of obtaining performance measures of a PACS that included component systems from different vendors communicating through network interface units. The specific configurations examined were display stations requesting images from and an acquisition station archiving images onto a centralized image database. The performance measures were the mean queue length and response at the interface; the end-to-end delay encountered when users at display stations requested images from the database in the first case and the throughput that could be sustained by the interface in the second configuration.

For the case of the acquisition station a simulation program was developed. Four different models were examined for the display station configuration. The ease of solution and accuracy afforded by these models vary widely. A preliminary model is a product-form queueing network but incorporates some unrealistic assumptions, rendering it unsatisfactory in spite of its ease of solution even when workstations have different image traffic workloads. Results are not illustrated in any detail except to draw a comparison between this model and the complete queueing network that had to be simulated in order to extract performance measures. Two other approximations were developed. They are based on some simplifying assumptions about the detailed model and are extremely easy to solve. Analysis shows that both approximations over-estimate system capacity. The second approximation is invariably superior to the first.

REFERENCES

[1] D'Silva, V., "Design of a Picture Archival and Communications System Based on Collected Data", *Masters' Thesis,* Dept. Of Elect. & Computer Engg., North Carolina State University, 1986

[2] Alzner, E. Arink, G. et al, "A Standard Product Interface for Digital Medical Imaging Equipment", *Proceedings of SPIE,* 1984

[3] ACR-NEMA, "Digital Imaging and Communications", *Standards Publication 300,* 1985

[4] Veran, M. and Potier, D., "A Portable Environment for Queueing Systems Modeling", *INRIA Report 314,* 1984

[5] Gelenbe, E., and Mitrani, I., "Analysis and Synthesis of Computer Systems", *Academic Press,* 1980.

[6] Hegde, S. S., Gale, A. O. and Giunta, J. A. "AT&T PACS Architecture", *SPIE Proceedings Medicine XIV and PACS IV,* 1986

[7] Cox Jr., J. R., Blaine, G. J. et al, "Some Design Considerations for Picture Archiving and Communications Systems", *IEEE Computer,* August 1983.

[8] Kleinrock, L. "Queueing Systems - Volume I: Theory", *John Wiley,* 1975.

APPENDICES

1. Listing of the program to solve model 2

2. Listing of the program to solve model 3

3. Listing of the program to simulate the acquisition station

4. Listing of the QNAP2 program to simulate the queueing network

APPENDIX 1

Listing of the program to solve model 2

```
#include <stdio.h>
#include <math.h>

float coeff[129][129], col[129];
float qmat[129][129], col[129];
float res[129];
float pbig, pmed, psmall;
float lambda, mu;
float link, think,idle,req,prob;
FILE *fpout;

main(argc, argv)
int argc;
char *argv[];

{
        FILE *fpin, *fopen();
        extern FILE *fpout;
        extern float coeff[129][129];
        int i, batch;
        extern float lambda, mu;
        extern float pbig, pmed, psmall;
        extern float link, think,idle,req,prob;


        fpin = fopen(argv[1], "r");
        fpout = fopen(argv[2], "w");
        fscanf(fpin, "%f %f %f %d", &think, &link, &idle, &req, &batch );
        prob = (batch-1) /batch;
        lambda = eval(link, think,idle,req,prob);
        fscanf(fpin, "%f %f %f", &mu. &pbig, &pmed );
        psmall = 1.0 - pbig - pmed;
        fclose(fpin);

        popmat(lambda, mu, pbig, pmed, psmall, coeff);
        transmat();

        gedrive();
}


void popmat(lambda, mu, pbig, pmed, psmall, coeff)
float lambda, mu, pbig, pmed, psmall, coeff[129][129];

{
        int i, j;
        float sum;
        int n = 129;

        for (i=0; i<n; i++) {
                for (j=0; j<n; j++) {
                        coeff[i][j] = 0.0;
                }
        }
        coeff[0][2] = 4 * lambda * psmall; /* four */
        coeff[0][8] = 4 * lambda * pmed;
        coeff[0][32] = 4 * lambda * pbig;

        for (i=1; i<=3; i++) {
                coeff[((32*i) + 1)][2 + 32*(i-1)] = mu * psmall;
                coeff[((32*i) + 1)][8 + 32*(i-1)] = mu * pmed;
                coeff[((32*i) + 1)][32 + 32*(i-1)] = mu * pbig;
        }

        for (i=0; i<=2; i++) {
                for (j=1; j<=32; j++) {
                        coeff[((32*i) - j)][((32*(i+1)) - j)] = (3 - i) * lambda;
```

```
                        /* 3 - i */
                }
        }

        for (i=1; i<n; i++) {
                coeff[i][(i-1)] = mu;
        }

        /* diagonal elements */

        for (i=0; i<129; i++) {
                sum = 0.0;
                for (j=0; j<129; j++) {
                        sum += coeff[i][j];
                }
                coeff[i][i] = -1.0 * sum;
        }


        for (i=0; i<129; i++) {
                coeff[i][128] = 1.0;
        }

}

void gecrive()

{
        extern float coeff[129][129];
        extern float qmat[129][129];
        extern float col[129];
        extern float res[129];
        extern float lambda, mu;
        extern float pbig, pmed, psmall;
        float avlambda, avgfrags, p1, p2, p3;
        int dim, i, j;
        FILE *fpin, *fopen();
        extern FILE *fpout;
        FILE *fpdia;

        dim = 129;

        for (i=0; i<128; i++) {
                col[i] = 0.0;
        }
        col[128] = 1.0;

        gauelim(dim);

        fpdia = fopen ("tmp", "w");

        fprintf(fpdia, "\n\nroots of the system:\n");
        for (i=0; i<dim; i++) {
                fprintf(fpdia, "res[%d] = %g\n", i, res[i]);
        }

        fclose(fpdia);

        fprintf(fpout, "Tdecomp = %g \n", (1 /mu));
        fprintf(fpout, "pbig = %g  psmall = %g \n", pbig, psmall);
        fprintf(fpout, "prob. of empty system = %g\n", res[0]);
        avlambda = res[0] * 4 * lambda;
        p1 = 0;
        for (i=1; i<=32; i++) {
                p1 += res[i];
        }
```

```
        avlambda +=
             fprintf(fpout, "prob. of only one exam = %g\n", p1);
        p2 = p1;
        for (i=33; i<=64; i++) {
                p2 += res[i];
        }
        fprintf(fpout, "prob. of at least two exams = %g\n", p2);
        p3 = p2;
        for (i=65; i<=96; i++) {
                p3 += res[i];
        }
        fprintf(fpout, "prob. of at least three exams = %g\n", p3);
        avgfrags = 0;
        for (i=1; i<=32; i++) {
                avgfrags += ( i * ( res[i] + res[(33+i)] + res[(65+i)] +
                     res[(97+i)] ) );
        }
        fprintf(fpout, "mean number of fragments = %g\n", avgfrags);
        fclose(fpout);
}

void transmat ()
{
        extern float coeff[129][129];
        extern float qmat[129][129];
        int     i,j;
        for (i=0; i<129; ++i) {
                for (j=0; j<129; ++j) {
                        qmat[j][i] = coeff[i][j];
                }
        }
}


void gauelim(n)
int n;


{
        int i, j, k, l, place, temp, abb1, abb2, status;
        extern float qmat[129][129], col[129];
        extern FILE *fpout;
        FILE *fptmp;
        float pivot, sum;
        extern float res[129];

#define error 0
#define continue 1


        fptmp = fopen("tmp", "w");
        status = continue;

        {
                for (i=1; i<n; i++) {
                        for (j=0; j<i; j++) {
                                pivot=qmat[i][j] /qmat[j][j];
                                for (k=j; k<n; k++) {
                                        qmat[i][k] = qmat[i][k] - (qmat[j][k] * pivot);

                                }

                        }
                }

        }
        if (qmat[(n-1)][(n-1)] == 0.0){
```

```
                status = error;
                printf("pivot = 0\n");
        }
        else {
                res[(n-1)] = col[(n-1)] / qmat[n-1][n-1];
        }
        for (i=(n-2); i>=0; i--) {
                sum = 0.0;
                for (j=(i+1); j<n; j++) {
                        sum += qmat[i][j] * res[j];
                }
                res[i] = (res[i] - sum) / qmat[i][i];
        }

}

float eval(tag1,tag2,tag3,tag4)
float tag1,tag2,tag3,tag4;

{
        float value;

        value = 1.0 /(tag1+tag3+tag2 *tag4);
        return(value);
}
```

APPENDIX 2

Listing of the program to solve model 3

```
#include <stdio.h>
#include <math.h>


main(argc, argv)
int argc;
char *argv[];

{
        FILE *fpin, *fpout, *fopen();
        float lambda, servt1, servt2;
        float rho, qzero, num, den, A0, A1, A2, det, test1, test2;
        float resp;
        float tx;
        float test, root;
        float maxerr;
        float psiz;
        float mu1, mu2;
        float link, siz, think,idle,req,prob;
        float eval();
        float guess, error, delta, delay;
        float pbig, pmed, psmall;
        float N1, N2, R1, R2;
        int batch;
        int niter, maxiter;


        fpin = fopen(argv[1], "r");
        fpout = fopen(argv[2], "w");
        fscanf(fpin, "%f %f %f %f %d", &think, &idle, &tx, &req, &batch);
        fscanf(fpin, "%f %f %f", &servt1, &servt2, &pmed, &psmall );
        fscanf(fpin,"%f",&error);
        fclose(fpin);

        psmall = 1.0 - pbig - pmed;
        prob = (batch-1) /batch;
        siz = (psmall*2 + pmed*8 + pbig*32);
        psiz = (siz - 1) /siz;

        link = tx * siz;

        mu1 = 1 /servt1;
        mu2 = 1 /servt2;

        guess = (think+prob*idle+req+link) /2;


        delta = 100.0;
        maxerr = 0.1;
        maxiter = 100;
        niter = 0;

        while ( (abs(delta) > maxerr) && (niter < maxiter) ) {

                lambda = eval(guess,think,idle,req,prob,link);
                rho = lambda /mu1;

                qzero = 1 - psiz - rho;

                num = lambda - psiz*lambda + mu1*psiz - mu1*psiz*psiz;
                den = (mu1 - lambda - mu1*psiz)*(mu1 - lambda - mu1*psiz);
                N1 = mu1 * qzero * num / den;
                R1 = N1 / (lambda * siz);


                A0 = mu1*(mu2 - qzero*mu2 + lambda);
                A1 = 1 * mu2 *(mu2 - mu1 - lambda);
```

```
                A2 = mu2 * mu2;

                det = sqrt((A1*A1) - (4.0 * A2 * A0));

                test1 =  A1 + det;
....            test1 = test1 /(2.0*A2);
                test2 = A1 - det;
                test2 = test2 /(2.0*A2);


                if (test1 > 0.0) {
                        if (test1 < 1.0) root = test1;
                }

                if (test2 > 0.0) {
                        if (test2 < 1.0) root = test2;
                }

                N2 = root /(1.0 - root);

                R2 = 1 /(mu2 * (1 - root));

                delay = batch *(R1+R2);

                delta = guess - delay;
                niter += 1;
                guess = delay;

        }
        resp = R1 + R2 + (tx) + (req /siz);
        fprintf(fpout,"%d iterations    error = %f \n",niter, delta);
        fprintf(fpout,"lambda = %f exams /sec\n", lambda);
        fprintf(fpout,"req = %f sec    think = %f  sec    idle = %f sec\n\
dbms = %f sec  interface = %f sec    avgsize = %f spfs\n\
txm = %f sec  batch = %d exams\n",
        req,think,idle,servt1,servt2,siz,tx,batch);
        fprintf(fpout,"N1 = %f    N2 = %f \n", N1, N2);
        fprintf(fpout,"R1 = %f    R2 = %f \n", R1, R2);
        fprintf(fpout,"resp = %f\n", resp);
}

float eval(tag1,tag2,tag3,tag4,tag5,tag6)
float tag1,tag2,tag3,tag4,tag5,tag6;

{
        float value;

        value = 4.0 /(tag1+tag2+tag6+tag4+(1.0-tag5)*tag3);
        return(value);
}
```

APPENDIX 3

Listing of the simulation program of the acquisition station

```
#include <stdio.h>

struct node {
float value;
int queuenum;
int evttype;
int class;
struct node *next;
struct node *previous;
};
typedef struct node NODE, *LINK;
LINK evtlist, firstevt;


#define MAXNODES 30

struct qnode {
float info;
struct qnode *next;
struct qnode *previous;
};

typedef struct qnode QNODE, *QLINK;

QLINK queue[MAXNODES];

FILE *fpin, *fpout, *fopen;

float tx1d, tx2d, app1d, app2d;
int ws1, ws2, nws, size;
float arriv, interf, compr, simtime;


main(argc, argv)
int argc;
char *argv[];

{

fpin = fopen(argv[1],"r");
fpout = fopen(argv[2],"w");

fscanf(fpin,"%f %f %f %f", &tx1d,&tx2d,&app1d,&app2d);
fscanf(fpin,"%d %d %d %d", &ws1,&ws2,&nws,&size);
fscanf(fpin,"%f %f %f %f %f", &arriv,&interf,&compr,&simtime);


initsyst();

item = poisson(arriv);

insertevent(&evtlist, item, 1, 1, 0);

while (clock < simtime) {

while (evtlist != NULL) {

listitem = removehead(&evtlist);

switch (listitem->evttype) {

case 1: {
enqueue(1);
if (qcount(1) == 0) insertevent(&evtlist,2,2,0);
item = poisson(arriv);
insertevent(&evtlist, item, 1, 1, 0);
}
```

```
    break;

    case 2: {
    dequeue(1);
    if (qcount(1) > 0) insertevent(&evtlist,2,2,0);
    enqueue(2);
    if (qcount(2) == 0) sync(2,&nws);
    }
    break;

    case 3: {
    dequeue(2);
    if (qcount(2) > 0) insertevent(&evtlist,3,3,0);
    enqueue(3);
    if (qcount(3) == 0) split(3,b1);
    }
    break;

    case 4: {
    dequeue(3);
    if (qcount(3) > 0) insertevent(&evtlist,4,4,0);
    enqueue(4);
    if (qcount(4) == 0) insertevent(&evtlist,4,4,0);

    }
    break;

    case 5: {
    dequeue(4);
    if (qcount(4) > 0) insertevent(&evtlist,5,5,0);
    enqueue(5);
    if (qcount(5) == 0) split(5,b2);
    }
    break;

    case 6: {
    dequeue(5);
    if (qcount(5) > 0) insertevent(&evtlist,6,6,0);
    enqueue(6);
    if (qcount(6) == 0) sync(6,&ws1);
    }
    break;

    case 7: {
    dequeue(6);
    if (qcount(6) > 0) insertevent(&evtlist,7,7,0);
    enqueue(7);
    if (qcount(7) == 0) insertevent(&evtlist,7,7,0);
    }
    break;

    case 8: {
    }
    break;

    case 9: {
    dequeue(8);
    if (qcount(8) > 0) insertevent(&evtlist,9,9,0);
    enqueue(9);
    if (qcount(9) == 0) insertevent(&evtlist,9,9,0);
    }
    break;

    case 10: {
    dequeue(9);
    if (qcount(9) > 0) insertevent(&evtlist,10,10,0);
    enqueue(10);
```

```
if (qcount(10) == 0) merge(10,b1);
}
break;

case 11: {
dequeue(10);
if (qcount(10) > 0) insertevent(&evtlist,11,11,0);
enqueue(11);
if (qcount(11) == 0) insertevent(&evtlist,11,11,0);
}
break;

case 12: {
dequeue(11);
if (qcount(11) > 0) insertevent(&evtlist,12,12,0);
enqueue(12);
if (qcount(12) == 0) split(12,b3);
}
break;

case 13: {
dequeue(12);
if (qcount(12) > 0) insertevent(&evtlist,13,13,0);
enqueue(13);
if (qcount(13) == 0) sync(13,&ws2);
}
break;

case 14: {
dequeue(13);
if (qcount(13) > 0) insertevent(&evtlist,14,14,0);
enqueue(14);
if (qcount(14) == 0) insertevent(&evtlist,14,14,0);
}
break;

case 15: {
dequeue(14);
if (qcount(14) > 0) insertevent(&evtlist,15,15,0);
enqueue(15);
if (qcount(15) == 0) fork(15,&ws2);
}
break;

case 16: {
dequeue(3);
if (qcount(3) > 0) insertevent(&evtlist,4,4,0);
enqueue(4);
if (qcount(4) == 0) insertevent(&evtlist,4,4,0);
}
break;

case 17: {
dequeue(16);
if (qcount(16) > 0) insertevent(&evtlist,17,17,0);
enqueue(17);
if (qcount(17) == 0) merge(17,b2);
}
break;

case 18: {
dequeue(17);
if (qcount(17) > 0) insertevent(&evtlist,18,18,0);
enqueue(18);
if (qcount(18) == 0) insertevent(&evtlist,18,18,0);
}
break;
```

```
case 19: {
dequeue(18);
if (qcount(18) > 0) insertevent(&evtlist,19,19,0);
enqueue(19);
if (qcount(19) == 0) merge(19,b3);
}
break;

case 20: {
dequeue(19);
if (qcount(19) > 0) insertevent(&evtlist,20,20,0);
enqueue(20);
if (qcount(20) == 0) fork(20,&nws);
}
break;

case 21: {
dequeue(20);
if (qcount(20) > 0) insertevent(&evtlist,21,21,0);
enqueue(21);
if (qcount(21) == 0) insertevent(&evtlist,21,21,0);
}
break;

}

}

}



getstats();


void insertevent(list, arg1, arg2, arg3, arg4)

LINK *list;
float arg1;
int arg2;
int arg3;
int arg4;

{
        float head, tail;
        LINK temp, prev, curr, local;

        if (*list == NULL) {
                local = (LINK)malloc(sizeof(NODE));
                local->value = arg1;
                local->queuenum = arg2;
                local->evttype = arg3;
                local->class = arg4;
                local->next = NULL;
                local->previous = local;
                *list = local;
                local = *list;
        }

        else {
                head = (*list)->value;
                tail = ((*list)->previous)->value;
                if (arg1 <= head) {
```

```
                        temp = *list;
                        local = (LINK)malloc(sizeof(NODE));
                        local->value = arg1;
                        local->queuenum = arg2;
                        local->evttype = arg3;
                        local->class = arg4;
                        local->next = temp;
                        local->previous = temp->previous;
                        (*list) = local;
                        local = *list;
                }

                else {
                        if (arg1 >= tail) {
                                local = (LINK)malloc(sizeof(NODE));
                                local->value = arg1;
                                local->queuenum = arg2;
                                local->evttype = arg3;
                                local->class = arg4;
                                local->next = NULL;
                                local->previous = (*list)->previous;
                                temp = (*list)->previous;
                                temp->next = local;
                                (*list)->previous = local;
                                local = *list;
                        }

                        else {
                                curr = (*list);
                                while ((arg1 >= curr->value) && (curr != NULL)) {
                                        temp = curr;
                                        curr = curr->next;
                                }
                                local = (LINK)malloc(sizeof(NODE));
                                local->value = arg1;
                                local->queuenum = arg2;
                                local->evttype = arg3;
                                local->class = arg4;
                                local->next = curr;
                                local->previous = curr->previous;
                                curr->previous = local;
                                temp->next = local;
                                local = *list;
                        }

                }
        }
}
}

;


LINK removehead(list)
LINK *list;

{
LINK curr, temp;

if (*list != NULL) {
curr = (LINK)malloc(sizeof(NODE));
curr->value = (*list)->value;
curr->queuenum = (*list)->queuenum;
curr->evttype = (*list)->evttype;
curr->class = (*list)->class;
temp = (*list)->next;
if ((*list)->next != NULL) {
```

```
            temp->previous = (*list)->previous;
}
free((*list));
*list = temp;
}
return(curr);
}

#define abs(x)   (( (x) > 0 ) ? (x) : -(x))

void enqueue(q, arg1)
QLINK *q;
float arg1;

{
            QLINK temp, prev, curr, local;

            if (*q == NULL) {
                        local = (QLINK)malloc(sizeof(QNODE));
                        local->info = arg1;
                        local->next = NULL;
                        local->previous = local;
                        *q = local;
            }

            else {
                        local = (QLINK)malloc(sizeof(QNODE));
                        local->info = arg1;
                        local->next = NULL;
                        local->previous = (*q)->previous;
                        temp = (*q)->previous;
                        temp->next = local;
                        (*q)->previous = local;
            }
}

QLINK dequeue(q)
QLINK *q;

{
            QLINK curr, local, temp;
            int qcount();

            if (*q != NULL) {
                        curr = (QLINK)malloc(sizeof(QNODE));
                        temp = (QLINK)malloc(sizeof(QNODE));
                        curr->info = (*q)->info;
                        temp = (*q)->next;

                        if (temp != NULL) {
                                    temp->previous = (*q)->previous;
                                    temp->next = ((*q)->next)->next;
                                    free((*q));
                                    *q = temp;
                                    local = *q;
                                    while (local != NULL) {
                                                local = local->next;
                                    }
                        }
                        else {
                                    free((*q));
                                    *q = temp;
                        }
                        return(curr);
            }
}
```

```
int qcount(q)
QLINK *q;

{
        int index = 1;
        float head;

        if (*q == NULL) return(0);
        else {
                if ((*q)->next == NULL) {
                        return(1);
                }
                else {
                        head = (*q)->info;
                        (*q) = (*q)->previous;
                        while ((*q)->info != head) {
                                (*q) = (*q)->previous;
                                index++;
                        }
                        return(index);
                }
        }
}

void split(index,size)
int index;
int size;

{
for (k=0; k<size;k++) enqueue(index);
if (qcount(index) == 0)
insertevent(&evtlist,x[index],(index+1),0);
}


void merge(index,size)
int index;
int size;

{
if (qcount(index) == size)
insertevent(&evtlist,x[index],(index+1),0);
}


void sync(index,tokens)
int index;
int *tokens;

{
if (qcount(index) > 0)
insertevent(&evtlist,x[index],(index+1),0);
if (tokens > 0) tokens--;
}


void fork(index,tokens)
int index;
int *tokens;

{
if (qcount(index) > 0)
insertevent(&evtlist,x[index],(index+1),0);
if (tokens > 0) tokens++;
}

#include <math.h>
```

```
float clock;
float response;
float marktime;

float uniform(upbnd, lobnd)
float upbnd, lobnd;
{
        float rand1();
        float value;

        value = lobnd + (upbnd - lobnd) *rand1();
        return(value);
}

float poisson(lambda)
float lambda;
{
        float value;
        float rand1();
        value = (-1.0 /lambda) * ((log(rand1())));
        return(value);
}

#include <time.h>
float rand1()
{
        float value;

        value = (float) rand() / (32768.0 * 2 *32768.0);
        if (value == 0) {
                value = (float) rand() / (32768.0 * 2 *32768.0);
        }
        return(value);
}

#define LAST 10
float normal(mean, stddev)
float mean;
float stddev;
{
        float sum, value, rand1();
        int i;

        sum = 0.0;
        for (i=1; i<LAST; i++) {
                sum += rand1();
        }
        value = mean +stddev *sqrt(12.0 /LAST) *(sum - (LAST /2));
        return(value);
}

int round(arg)
float arg;
{
        if ((arg - floor(arg)) <= 0.5) return(floor(arg));
        else return((1+floor(arg)));
}
```

APPENDIX 4

Listing of the QNAP2 program that simulates the queueing network

```
/CONTROL/ OPTION=NSOURCE;
/DECLARE/ QUEUE  ws, dbms, decomp, link2, think, idle;
        REAL Treq = 0.3;            &
        REAL Tdbms = 0.3;           &
        REAL Tlink2 = 2.0;    &
        REAL Tovhd = 3.0; &
        REAL Tthink = 30.0;   &
        REAL Tidle = 30.0;    &
        REAL Tdecomp;       &
        REAL PBIG = 0.333;    &
        REAL PMED = 0.333;    &
        REAL PNEXT;         & prob of next exam being the last
        INTEGER BSIZE = 32;   & big exam size
        INTEGER MSIZE = 8;    & medium exam size
        INTEGER SSIZE = 2;    & small exam size
        INTEGER BARR, MARR, SARR, BDEP, MDEP, SDEP;
        INTEGER BATCH:              & number of exams in a batch
        INTEGER LIMIT = 4;
        CLASS ex, big, med, small;

$MACRO workstn (N, P1, P2, exam, big, med, small)
/STATION/ NAME = N;
INIT(exam) = LIMIT;
TYPE = INFINITE;
SERVICE = EXP(Treq);
TRANSIT = dbms, big, P1, dbms, med, P2, dbms, small;
$END

$MACRO collect (CUST, DEP, SIZE, DEST, TYPE)
SERVICE(CUST) = BEGIN
EXP(Tthink);
IF DEP=SIZE
THEN BEGIN
DEP:= 0;
IF DRAW(PNEXT) THEN TRANSIT(DEST, TYPE) ELSE TRANSIT(idle, TYPE);
END
ELSE BEGIN
DEP:= DEP + 1;
TRANSIT(OUT);
END;
END;
$END

$MACRO spawn (ARR, SIZE, TYPE)
```

```
SERVICE(TYPE) = BEGIN
EXP(Tdbms);
IF ARR = 0 THEN EXP(Tovhd);
IF ARR<>SIZE
THEN BEGIN
TRANSIT(NEW(CUSTOMER), dbms, TYPE);
ARR:= ARR + 1;
TRANSIT(decomp);
END
ELSE BEGIN
ARR:= 0;
TRANSIT(decomp);
END;
END;
$END

$workstn (ws, PBIG, PMED, ex, big, med, small)

/STATION/ NAME = dbms;
$spawn (BARR, BSIZE, big)
$spawn (MARR, MSIZE, med)
$spawn (SARR, SSIZE, small)

/STATION/ NAME = decomp;
SERVICE = EXP(Tdecomp);
TRANSIT = link2;

/STATION/ NAME = link2;
TYPE = INFINITE;
SERVICE = EXP(Tlink2);
TRANSIT = think;

/STATION/ NAME = think;
TYPE = INFINITE;
$collect (big, BDEP, BSIZE, ws, ex)
$collect (med, MDEP, MSIZE, ws, ex)
$collect (small, SDEP, SSIZE, ws, ex)

/STATION/ NAME = idle;
TYPE = INFINITE;
SERVICE = EXP(Tidle);
TRANSIT(ex) = ws;

/CONTROL/ TMAX = 50000;
```

```
/CONTROL/ CLASS=dbms, decomp;
ACCURACY= dbms, decomp;

/EXEC/ BEGIN
SARR := 0; MARR := 0; BARR := 0; SDEP := 0; MDEP := 0; BDEP := 0;
PNEXT := (BATCH-1)/BATCH;
END;

/EXEC/ BEGIN
Tdecomp := 0.75;
BATCH := 5;
SIMUL;
PRINT("mean number of fragments is ", MCUSTNB(decomp));
PRINT("thruput of exams is ", MTHRUPUT(ws));
END;
/END/
```