SEL-TR-157

A Multiple $M/D_z/1/L$ Queueing Network Model of Crossbar-based Multiprocessors

B.A. Makrucki
T.N. Mudge

September 1981

This work was supported in part by
National Science Foundation grant MCS-8009315.

DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING
SYSTEMS ENGINEERING LABORATORY
THE UNIVERSITY OF MICHIGAN, ANN ARBOR 48109

SEL-TR-157

A Multiple $M/D_z/1/L$ Queueing Network Model of Crossbar-based Multiprocessors

B. A. Makrucki

T. N. Mudge

September 1981

This work was supported in part by

National Science Foundation Grant MCS-8009315

DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING

SYSTEMS ENGINEERING LABORATORY

THE UNIVERSITY OF MICHIGAN, ANN ARBOR 48109

ABSTRACT

This report describes a model and analysis of crossbar-based multiprocessor systems. A packet switched multiprocessor system model is developed and analyzed under certain system behavior assumptions. The model is based on an infinite customer multiple $M/D_g/1/L$ queueing network that represents the system configuration. Performance measures that provide insight into system behavior are derived.

TABLE OF CONTENTS

1. Introduction.

High performance multiprocessor computer architectures typically require high capacity connection channels between processing elements. A connection channel may be constructed in many ways, recently interconnection networks have been developed for this purpose. The selection of a particular interconnection network design is based on performance gains over other designs. An interconnection network's performance is typically classified according to three performance measures: network size; network connection capability; and network operation behavior. The first two of these three have been thoroughly studied [Ben65, Clo53, GoL73, Law75, Sie77, Sie80, WuF80], the third measure has also been studied but network behavior under several important operating conditions remains an open question.

This report describes a model and analysis of the most basic switching element of most interconnection networks of interest in high performance processing, the crossbar switch. The analysis is applied to a system whose interconnection network is a crossbar. Performance measures of the system are obtained for a particular processor/program behavior model and a class of memory behavior models.

Work done in the area has basically assumed that the system in some way or another operates synchronously. Typically processors and memories have been modeled where one or both are synchronous subsystems [SkA69, Str70, Bha75, BaS76, Hoo77, SeD79, Pat79, Rau79, MaM81]. In this report a queueing model is developed that models processor behavior as a Poisson process and memory behavior as a multiple deterministic server. This model maintains the deterministic model of memory behavior, but allows a queueing analysis to be done. For special cases the analysis described is seen to agree with some of the earlier analyses referenced above.

The system to be modeled and analyzed is an n source, m sink system. A *source* is a device which emits packets of constant length.[1] Packets emitted by a source are sent to 1 of m *sink* devices. Sources may be devices such as processors or memories, similarly sinks may be processors or memories. The model applies to processor-to-processor connected systems, processor-to-memory connected systems, etc. For simplicity, sources in this report are considered to be processors and sinks are considered to be memory modules. Figure 1 shows this configuration. The interconnection network is a crossbar. That is, the analysis will be applied to a packet switched crossbar-based processor-to-memory system.

There are several important measures to be found. They are measures of system performance that are functions of design choices and program demands on the memory system. The measures to be derived here are:

(1) *Processor utilization* (PU) - the fraction of time that the n processors are executing instructions. That is, PU is the fraction of time that the proces-

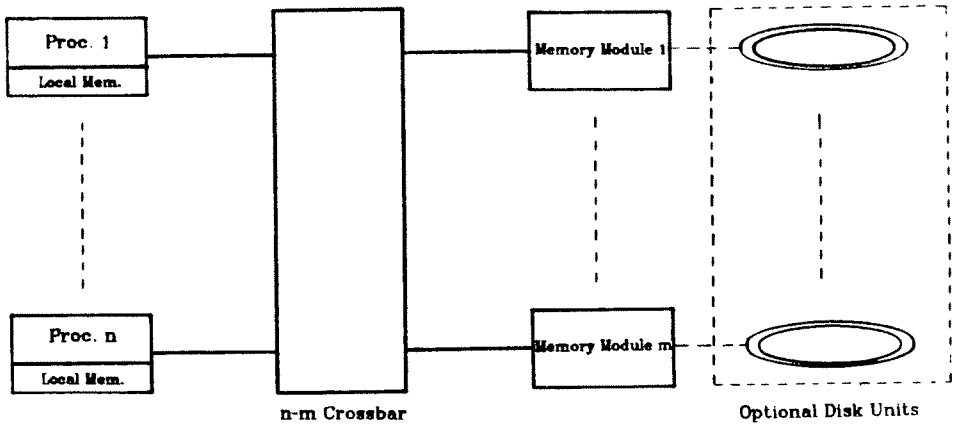


Figure 1. System to be modeled.

[1] In applications such as computer network analysis, packets are messages that vary in length. Packets here are memory reference words of constant length.

sors are doing useful work.

(2) *Memory utilization* (MU) - the fraction of time that the memory modules are performing memory operations (reads and writes). MU is the fraction of time that memory modules are busy.

(3) *Average queue length* - the average number of packets present in one of the memory module queues (to be discussed later).

(4) *Packet delay time* - the amount of time it takes for a packet to reach its its destination after it is emitted. It is the amount of delay encountered from packet emission to completion of service at the destination memory module.

The report is organized as follows: Section 2 describes the multiprocessor system to be modeled. Section 2 also discusses multiprocessor system architectures that are appropriate for application of the model described here. One is a general purpose machine, the other is a numerically oriented machine where processors have memory interface queues that buffer addresses and data. Section 3 describes the analysis of a queueing model that represents the system. A solution for an $M/D_s/1/L$[2] queue is derived and used. A general solution for $M/G/1/L$ queues may be found in [Coo72, GrH74]. Section 4 uses the results from section 3 to derive expressions for the measures cited above. Other measures have been proposed [MaG81], but those developed here can often form a basis for others.[3] Section 5 discusses special case solutions for

[2] This notation will be explained later.

[3] For example, [MaG81] defines *processing power* P, as the average number of active processors, in the model here, P = n. They also define the average number of queued processor requests, N_q. In the model presented here, if packets in service are included, $N_q = \sum_{j=1}^{m} E[N_j]$. See section 4 for details of notation.

particular parameter choices. This provides insight into the way model parameters interact in determining system measures. Section 6 is the conclusion.

2. System Model and Assumptions.

The packet switched system in Figure 1 is to be modeled with certain simplifying assumptions:

(1) All processors behave independently. In particular this means interprocessor data dependencies are not modeled here.

(2) Each processor emits packets as a Poisson process with rate λ_i (i is generally a processor number, $1 \leq i \leq n$). Thus program behavior is modeled as follows: a program executing on processor i emits packets[4] at any time, the average time between packet emissions is $\frac{1}{\lambda_i}$. This is simply a continuous time extension of discrete time processor models [SkA69, Str70, Bha75, BaS76, Hoo77, SeD79, Pat79, Rau79, MaM81] where the packet emission process is a Bernoulli process. In discrete time processor models, it is assumed that processors emit packets during system cycles with probability p (which may be viewed as the fraction of instructions that reference global memory, such fractions are characterized by Gibson mix relative frequencies). The continuous time model is simply an extension of this idea, in fact it is the process that arises if the system cycle time goes to zero (with an appropriate adjustment in p) in the discrete time model.

(3) All packet emissions are independent of each other. This assumption means that programs/processors do not make global memory references and wait for responses. This assumption makes all processor Poisson processes independent of each other. It also means processors are never idle waiting for memory responses.

---

[4] Packets here are global memory reference words, global memory is that memory seen commonly by all processors, it is comprised of the memory modules on the right side of the crossbar of Figure 1.

(4) When a program makes a global memory reference (a processor emits a packet), the selection of a destination memory module is modeled using a routing matrix **P**, where

$$\mathbf{P} = (P_{i,j}) ,$$

$$P_{i,j} = \Pr\{ \text{packet emitted by proc. i is for mem. mod. j} \} .$$

Thus $P_{i,j}$ is a probability conditioned on the event that a packet was emitted. This is an observable measurement on the system. More specifically, $P_{i,j}$'s may be determined from examining program memory reference patterns [BaS76, Hoo77]. By assumption (3) each global memory reference is modeled as being independent of previous and successive global memory references.

Even though the queueing model of the system is packet switched, actual system operation resembles a paged memory system. In the system to be modeled, processors emit read or write request packets. In the case of a global memory read operation, a memory queue server (see memory modules shown in Figure 3), upon receiving a read request, sets up a connection to the requesting processor. The connection is provided by a data-path crossbar (not shown in Figure 1) which makes connections from memory queue servers to requesting processors. After a connection has been set up by a memory queue server (of which there are m, all of which operate independently), the desired memory page is transferred to the requesting processor's local memory. There may be many page sizes available to algorithms running on various processors. Notice that no synchronization is present in memory service behavior, i.e., processors and memories run asynchronously with respect to each other. It is assumed that processors are able to handle the return connection data transfer rate. Future work might include analysis of return queues at processors.

In the case of a global memory write operation, a memory queue server, upon receiving a write request, transfers a page of local memory (through the data-path crossbar) into its global memory module. Global memory write operations could alternatively be handled by transferring a word at a time to global memory modules using the request packets as data carriers. The former approach will be assumed for write operations.

The model may be applied to the analysis of MIMD (multiple instruction stream, multiple data stream) environments where processor packet emissions are independent and processors behave independently. These characteristics are satisfied in a general purpose environment where each processor is multiprogrammed among several users. In this case each processing element behaves like a multiprogrammed processor with memory. Global memory references may be treated as page faults, so that processors context switch to different jobs on each global memory reference. This treatment of user jobs ensures that successive global memory references are independent. This situation also exhibits independent processor behavior. Although there are many variations on general purpose machine operation, the model given here applies particularly well to the above mode of operation due to the independence of user jobs.

The analysis may also be applied to tightly coupled systems executing parallel algorithms in an MIMD mode of operation. Consider a system of multiple memory modules that store data, and multiple processors each organized as shown in Figure 2. An example of such a processor is the MAP-200 [CoS81]. Each processor is divided into two subsystems: an address processor and a data processor. These two subsystems are relatively loosely coupled so that they may run asynchronously with respect to each other. It has been found that such processors lend themselves to a wide variety of numerically intensive computations, but in particular to weakly vectorizable[5] numerical

computations [Smi81]. In fact, our motivation for imbedding the processor in a multiple processor architecture, and as a consequence developing the model in this report, was to study the effectiveness of such multiprocessors for solving sparse matrix problems -- a weakly vectorizable computation.

To see how the processor of Figure 2 operates in the context of the multiprocessor architecture of Figure 1, consider the following. Assume each pro-

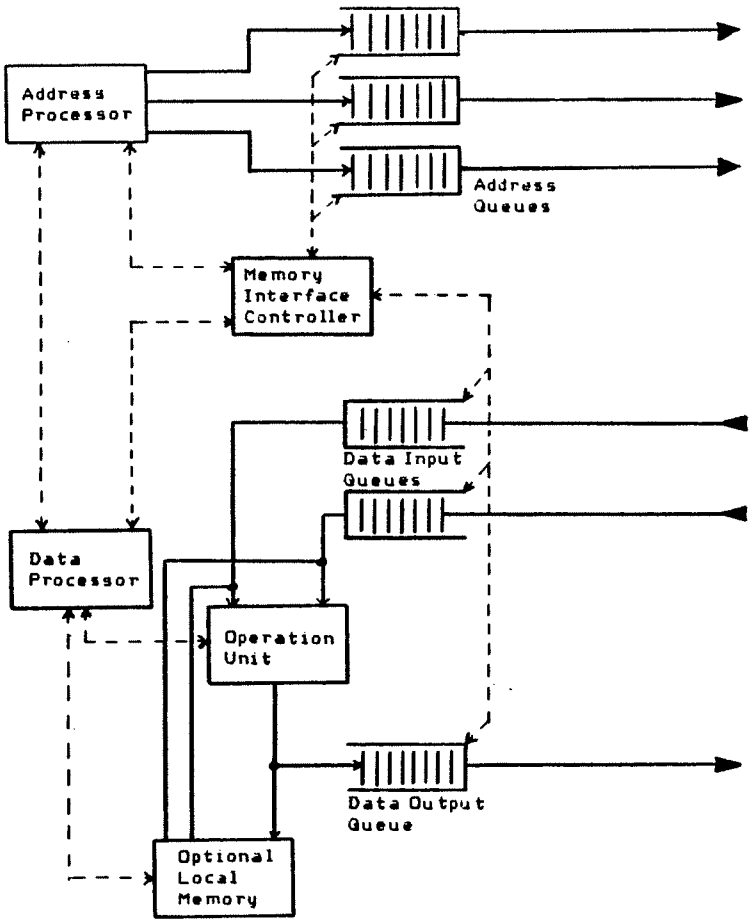


Figure 2. Details of the processor.

[5] Weakly vectorizable refers to computations in which it is difficult to organize the data into vectors all having the same length. Instead, data appears in a variety of (usually short) vector lengths and as simple scalars.

cessor has its own private program memory initially loaded by some form of system manager. Furthermore, assume each processor to be a three address machine. During instruction execution, the address processor enters three global memory data addresses into the three address queues. These queues are emptied into the global memory system by the memory interface controller. Thus these three address queues form the packet sources for each processor. When addresses are emitted they are routed to their appropriate memory module queues (Figure 3, see the next section). Data read from global memory is loaded into processor data input queues. Data to be written into global memory is stored in the data output queue until the required global memory connection is ready.

In this application, memory pages range from single memory words used in scalar operations, to vectors of memory words used in vector operations. As will be seen, multiple page size selection may be used to represent multiple vector sizes used during computations.

It is assumed here that interprocessor data dependencies and problem partitioning are handled by the compiler, consequently, run-time or dynamic parallelism is not achievable.

3. Analysis of the Model.

Under the assumptions of section 2, the model will now be analyzed to find the four performance measures listed in section 1. Steady state behavior is assumed.

The system may be redrawn as in Figure 3. Here processors are replaced by Poisson sources (which may be global memory address queues as described above) and memory modules are replaced by queues and servers. The crossbar is drawn as a bipartite graph to display the decomposition and superposition of the processor Poisson processes.

From Figure 3 and the fact that this is an infinite customer model, it may be seen that each queue behaves independently of others. A single queue solution will suffice to complete the queueing network analysis. All queue servers operate on a first come, first served (FCFS) basis for those packets in the queue. No other service disciplines are considered.

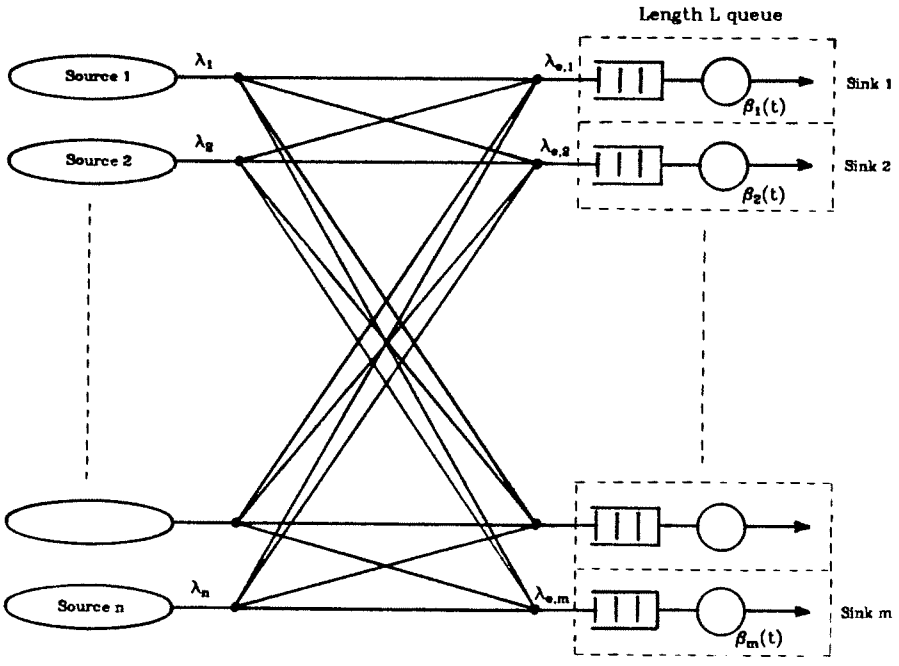


Figure 3. Queueing network model of the system.

3.1. Single Queue Solution.

A solution for a single M/G/1/L queue is required for each of the m memory module queueing stations. It is highly desirable to make the queues finite length because implementations are finite in size. The analysis with finite length queues shows performance verses queue size, thus providing insight into system behavior as a function of queue sizes.

From decomposition and superposition of Poisson processes, it may be seen that the jth memory module queueing station sees a Poisson process with rate $\lambda_{e,j}$ at its input, further

$$\lambda_{e,j} = \sum_{i=1}^{n} P_{i,j}\lambda_i .$$

A column vector $\boldsymbol{\lambda}_e$ may be defined from this and $\boldsymbol{\lambda}$, the processor emission rate column vector:

$$\boldsymbol{\lambda}_e = \mathbf{P}^T\boldsymbol{\lambda}. \tag{1}$$

A solution for the jth M/G/1/L memory queueing station may be found by using the imbedded Markov chain that is defined by examining the queue size after departure instants [Cin75, Coo72, GrH74, Kle75]. Define the imbedded Markov chain transition matrix **T** for a queueing station with a queue of length L (not including the server, that is, L is length of the queue proper). The random variable on which the Markov chain is defined is the number of packets left in the queueing station immediately after a departure.

$$
\mathbf{T} = \begin{array}{c} \\ 0 \\ 1 \\ 2 \\ 3 \\ 4 \\ . \\ . \\ . \\ L-3 \\ L-2 \\ L-1 \\ L \end{array}
\begin{array}{c}
\begin{array}{cccccccccccc} 0 & 1 & 2 & 3 & . & . & . & L-3 & L-2 & L-1 & L \end{array} \\
\left[\begin{array}{cccccccccccc}
q_0 & q_1 & q_2 & q_3 & . & . & . & q_{L-3} & q_{L-2} & q_{L-1} & c_1 \\
q_0 & q_1 & q_2 & q_3 & . & . & . & q_{L-3} & q_{L-2} & q_{L-1} & c_1 \\
0 & q_0 & q_1 & q_2 & . & . & . & q_{L-4} & q_{L-3} & q_{L-2} & c_2 \\
0 & 0 & q_0 & q_1 & . & . & . & q_{L-5} & q_{L-4} & q_{L-3} & c_3 \\
0 & 0 & 0 & q_0 & . & . & . & q_{L-6} & q_{L-5} & q_{L-4} & c_4 \\
. & . & . & . & . & . & . & . & . & . & . \\
. & . & . & . & . & . & . & . & . & . & . \\
. & . & . & . & . & . & . & . & . & . & . \\
0 & 0 & 0 & 0 & . & . & . & q_1 & q_2 & q_3 & c_{L-3} \\
0 & 0 & 0 & 0 & . & . & . & q_0 & q_1 & q_2 & c_{L-2} \\
0 & 0 & 0 & 0 & . & . & . & 0 & q_0 & q_1 & c_{L-1} \\
0 & 0 & 0 & 0 & . & . & . & 0 & 0 & q_0 & c_L
\end{array}\right]
\end{array}
$$

Where the c_w, $1 \le w \le L$, are chosen to make the rows sum to unity. A detailed discussion of the structure of the matrix **T** may be found in [Cin75]. Put briefly: for rows 1 through L, only steps down one (if no arrivals occur between two departures) or steps up of magnitude 0 through L − w for row w may take place. If, say, k arrivals occur between two departures then the queue size is w + k − 1 in the next state provided $w + k - 1 \le L$. If $w + k - 1 > L$ then the next state becomes L because packets are rejected when the queue is full.

$q_{k,j}$, q_k for queue j as seen in the **T** matrix for queue j, is given by:

$$
\begin{aligned}
q_{k,j} &= \Pr\{ \text{ k packets arrive during a service } \} \\
&= \int_0^{\infty} \frac{e^{-\lambda_{e,j}t}(\lambda_{e,j}t)^k}{k!} d\beta_j(t)
\end{aligned}
\qquad (2)
$$

Where,

$$\beta_j(t) = \Pr\{ \text{ time of service completion for server } j \leq t \} .$$

That is, $\beta_j(t)$ is the probability distribution for service time in queue j. See [Cin75, Kle75] for more details about the derivation of q_k's.

As mentioned previously, processors may make requests for various sized page transfers. The selection of the desired page size will be modeled with probabilities as follows:

Let,

$$\alpha_w = \Pr\{ \text{ page size w is requested } \} .$$

Where page size w is chosen from a set of available page sizes. Another requirement on α_w's is:

$$\sum_{w=1}^{z} \alpha_w = 1.$$

There are z page sizes available $\{ S_1, S_2, \ldots, S_z \}$. Let the actual page sizes be measured in units of time required for transfer, i.e., there is a mapping: $\{ S_1, \ldots, S_z \} \rightarrow \{ t_1, \ldots, t_z \}$ that depends heavily on memory module speed. If the memory parts chosen for the memory module design are slow, then t_i's may be large. Alternatively if memory parts are fast, then t_i's may be small. Then,

$$\beta_j(t) = \sum_{w=1}^{z} \alpha_w u(t - t_w) ,$$

where u(t) is the unit step function.

If disk access modeling is desired the $u(t - t_z)$ could be changed to represent an Erlang server with average service time t_z being the average disk

access time. This model is useful for analysis of systems where global memory module storage is page allocated in a similar manner to systems where virtual memory is implemented. An Erlang server seems appropriate if several sequential Poisson service times are involved in a disk access operation. See [FuB75] for an analysis of drum type storage devices. Here though the simple multiple deterministic server (a D_z server) will be used for notational simplicity.

The probability density of the service time random variable becomes:

$$\frac{d\beta_j(t)}{dt} = \sum_{w=1}^{z} \alpha_w \delta(t-t_w) .$$

Where $\delta(t)$ is the Dirac delta function.

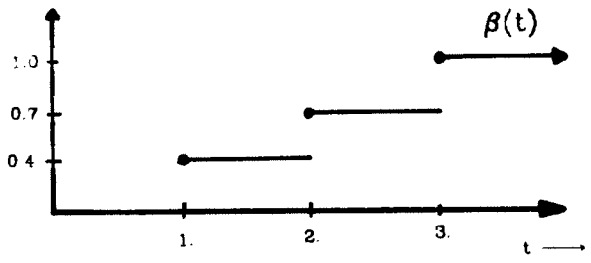
So equation (2) becomes:

$$q_{k,j} = \int_0^\infty \frac{e^{-\lambda_{e,j}t}(\lambda_{e,j}t)^k}{k!} \frac{d\beta_j(t)}{dt} dt .$$

This leads to an evaluation of q_k for queue j.

$$q_{k,j} = \sum_{w=1}^{z} \frac{\alpha_w(\lambda_{e,j}t_w)^k}{k!} e^{-\lambda_{e,j}t_w} . \qquad (3)$$

For simplicity $\beta_j(t) = \beta(t)$ has been assumed to be the same for all queues and all processor service requests. A $\beta(t)$ might look like:



$\beta(t)$ may be defined to be a D_z server, it is a z stage type server. Figure 4 shows a graphic interpretation of this type of server, only one packet may be

in the service facility at any time.

Define a probability distribution for queue j:

$$\pi_{k,j} = \Pr\{\text{ k packets are in queue j after a departure }\} .$$

Then define a row vector of these probabilities:

$$\boldsymbol{\pi}_j = \left(\pi_{0,j} , \pi_{1,j} , \pi_{2,j} , \ldots , \pi_{L,j} \right) .$$

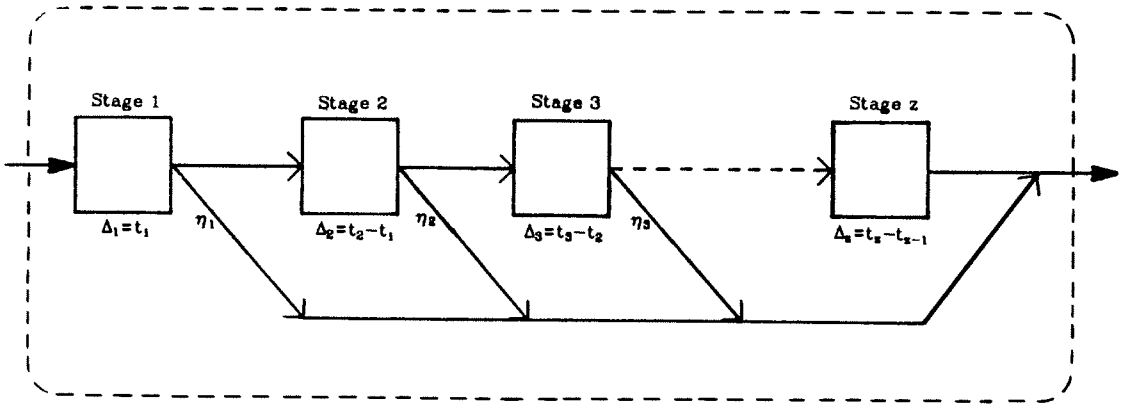
In the steady state, $\boldsymbol{\pi}_j$ is given by the unique solution to:

$$\boldsymbol{\pi}_j = \boldsymbol{\pi}_j \mathbf{T}_j , \tag{4}$$

$$\boldsymbol{\pi}_j (1\ 1 \ldots 1\ 1)^T = 1 . \tag{5}$$

Equations (4) and (5) may be re-written as:

$$\mathbf{T}_j^T \boldsymbol{\pi}_j^T = \boldsymbol{\pi}_j^T ,$$



$$\eta_1 = \alpha_1 , \quad \eta_k = \alpha_k / \prod_{w=1}^{k-1} (1 - \eta_w) .$$

Figure 4. A D_z service facility.

$$(1\ 1 \ldots 1\ 1)\pi_j^T = 1 .$$

Which leads to,

$$(\mathbf{T}_j^T - \mathbf{I})\pi_j^T = 0, \tag{6}$$

$$(1\ 1 \ldots 1\ 1)\pi_j^T = 1 .$$

This may be re-written as 1 equation because one of the rows in (6) is a linear combination of other rows, i.e., it is not required. Replace the bottom row of (6) with (1 1 . . . 1 1). So finally the set of equations to be solved is:

$$\mathbf{A}_j\pi_j^T = (0\ 0 \ldots 0\ 1)^T . \tag{7}$$

And for the jth queue $\mathbf{A}_j = \mathbf{A}$ is:

$$\mathbf{A} = \begin{array}{c} \\ 0 \\ 1 \\ 2 \\ 3 \\ . \\ . \\ . \\ L-3 \\ L-2 \\ L-1 \\ L \end{array} \begin{array}{c} \begin{array}{cccccccccc} 0 & 1 & 2 & 3 & . & . & . & L-3 & L-2 & L-1 & L \end{array} \\ \begin{bmatrix} q_0-1 & q_0 & 0 & 0 & . & . & . & 0 & 0 & 0 & 0 \\ q_1 & q_1-1 & q_0 & 0 & . & . & . & 0 & 0 & 0 & 0 \\ q_2 & q_2 & q_1-1 & q_0 & . & . & . & 0 & 0 & 0 & 0 \\ q_3 & q_3 & q_2 & q_1-1 & . & . & . & 0 & 0 & 0 & 0 \\ . & . & . & . & . & . & . & . & . & . & . \\ . & . & . & . & . & . & . & . & . & . & . \\ . & . & . & . & . & . & . & . & . & . & . \\ q_{L-3} & q_{L-3} & q_{L-4} & q_{L-5} & . & . & . & q_1-1 & q_0 & 0 & 0 \\ q_{L-2} & q_{L-2} & q_{L-3} & q_{L-4} & . & . & . & q_2 & q_1-1 & q_0 & 0 \\ q_{L-1} & q_{L-1} & q_{L-2} & q_{L-3} & . & . & . & q_3 & q_2 & q_1-1 & q_0 \\ 1 & 1 & 1 & 1 & . & . & . & 1 & 1 & 1 & 1 \end{bmatrix} \end{array}$$

Therefore, π_j^T = column L of $\mathbf{A}_j^{-1}$.

For demonstrative purposes, consider an example system:

$$L = 3,$$

$n = m$,

$\lambda_i = \lambda = 1$,

$P_{i,j} = \frac{1}{m}$, for all i, j ,

$\beta(t)$ is the example shown above.

Note that all queues behave identically due to symmetry in **P** and λ .

Evaluating the matrix **A** leads to:

$$\mathbf{A} = \begin{bmatrix} -.79731 & .20269 & 0 & 0 \\ .27316 & -.72684 & .20269 & 0 \\ .22199 & .22199 & -.72684 & .20269 \\ 1 & 1 & 1 & 1 \end{bmatrix}.$$

And solving for π gives:

$$\pi = \begin{pmatrix} .017229 & .067775 & .219821 & .695175 \end{pmatrix}.$$

Consider now the case where $L = \infty$. Here the solution is found by solving the recurrence relation $\pi = \pi \mathbf{T}$ for queue j. From [Cin75] the solution for the jth queue follows:

$$\pi_{0,j} = 1 - \lambda_{e,j} t_s > 0$$

if a bounded solution for the average number of packets in queue j is to exist. t_s is the average time of service:

$$t_s = \sum_{w=1}^{z} \alpha_w t_w .$$

That is, $\lambda_{e,j} < \frac{1}{t_s}$ for a bounded solution to exist. If this is satisfied, then the rest of the $\pi_{k,j}$'s may be found from:

$$\pi_{1,j} = (1 - \lambda_{e,j} t_s) \left(\frac{r_{0,j}}{q_{0,j}} \right),$$

and for $\gamma = 1, 2, 3, \cdots$,

$$\pi_{\gamma+1,j} = (1 - \lambda_{e,j}t_s)\sum_{k=1}^{\gamma}\left(\frac{1}{q_{0,j}}\right)^{k+1}\sum_{v \in S_{\gamma,k}}(r_{v_1,j}\, r_{v_2,j} \cdots r_{v_k,j})$$

where,

$$r_{k,j} = 1 - \sum_{w=0}^{k} q_{k,j}\ , \qquad k = 0, 1, \cdots$$

$$S_{\gamma,k} = \left\{ v = (v_1, \ldots, v_k) : \sum_{w=1}^{k} v_w = \gamma\ ,\ v_w \geq 1 \right\}.$$

Note that in the $M/D_s/1/\infty$ case $\lambda_{e,j} < \frac{1}{t_s}$ is required if a bounded solution (for the average number of packets in the queue and the average time spent in the queue) is to exist. For the $M/D_s/1/L$ queue, any range of $\lambda_{e,j}$ is allowed and a bounded solution will exist.

Having obtained π_j for finite length queues, it is necessary to extend π_j to a vector of probabilities of queue occupancy at arrival (or random) times. The argument follows that of [Coo72]. For $k = 0, 1, 2, \ldots, L+1$ define

$$\pi^*_{k,j} = \Pr\{\text{ queue j contains k packets at an arrival }\}$$

$$\pi^*_{k,j} = \left[\frac{1}{\pi_{0,j} + \lambda_{e,j}t_s}\right]\pi_{k,j} \qquad k = 0, 1, \cdots, L \tag{8.1}$$

$$\pi^*_{L+1,j} = \frac{\pi_{0,j} + \lambda_{e,j}t_s - 1}{\pi_{0,j} + \lambda_{e,j}t_s} \tag{8.2}$$

The extended vector π^*_j is defined as

$$\pi^*_j = \left(\pi^*_{0,j},\ \pi^*_{1,j},\ \cdots,\ \pi^*_{L+1,j}\right).$$

Note that for infinite length queues $\pi^*_{k,j} = \pi_{k,j}$ for all j and k. From now on, π^*_j's

are the probability distributions used to find system expected values.

Continuing with the example,

$$\pi^* = \begin{pmatrix} .008986 & .035350 & .114656 & .362594 & .478414 \end{pmatrix}.$$

4. Measure Derivation.

Having obtained the π^*_j's from section 3.1, it is a simple matter to derive the four measures cited in section 1.

Processor utilization may be defined as the average over all processors of each processor's utilization:

$$PU = \frac{1}{n} \sum_{i=1}^{n} PU_i.$$

Note that since processors running programs are modeled as Poisson processes which are independent of memory system behavior, $PU_i = 1$. So

$$PU = 1.$$

That is, all processors are always busy.

Memory utilization is defined similarly:

$$MU = \frac{1}{m} \sum_{j=1}^{m} MU_j.$$

Because the probabilities π^*_j are stationary (the imbedded Markov chain is time homogeneous) MU_j is noted to be the fraction of time that server j is busy, thus $MU_j = 1 - \pi^*_{0,j}$. So

$$MU = \frac{1}{m} \sum_{j=1}^{m} (1 - \pi^*_{0,j}) . \qquad (9)$$

Continuing with the example, MU = 0.991014 .

The average queue length, $E[N_j]$ for queue j, for finite length queues is:

$$E[N_j] = \sum_{k=1}^{L+1} k \, \pi^*_{k,j} , \qquad L < \infty . \qquad (10)$$

For the running example above, $E[N_j] = 3.2661$ for all j. Which shows that the queue may often be full. For infinite length queues the Pollaczek-Khinchin formula may be used:

$$E[N_j] = \lambda_{e,j}t_s + \frac{(\lambda_{e,j}t_s)^2}{2(1-\lambda_{e,j}t_s)} \quad , \quad \text{if } \lambda_{e,j}t_s < 1 \ , \ L = \infty \ . \tag{11}$$

The final measure, packet delay time, is slightly more complicated to find. First consider the M/ D_g/ 1/ ∞ case. Little's equation may be used once $E[N_j]$ from equation (11) is known.

Let $T_{i,j}$ be the continuous random variable associated with the amount of time a packet from processor i spends in queue j, include service time in this measurement. Then

$$E[T_{i,j}] = \frac{E[N_j]}{\lambda_{e,j}} \ , \quad L = \infty \ .$$

Thus,

$$E[T_{i,j}] = E[T_{k,j}] = E[T_j] \quad \text{for all i, k} = 1, 2, \cdots, n \ .$$

So $E[T_{i,j}] = E[T_j]$, (T_j is simply the random variable representing the amount of time ***any*** packet spends in queueing station j) this shows that any routing matrix **P** and rate vector $\boldsymbol{\lambda}$ that lead to the same $\boldsymbol{\lambda}_e$ will exhibit the same $E[T_j]$'s . In particular this shows that this system model is not as sensitive to **P** and $\boldsymbol{\lambda}$ as are finite customer models [Hoo77, SeD79, Pat79, Rau79, MaG81, MaM81], i.e., systems where connections are circuit switched.

Using (11) the above may be re-written as:

$$E[T_j] = t_s\left(1 + \frac{\lambda_{e,j}t_s}{2(1-\lambda_{e,j}t_s)}\right) \ , \quad \text{if } \lambda_{e,j}t_s < 1 \ , \ L = \infty \ .$$

An aggregate packet delay time for processor i may be defined:

$$\overline{E[T_i]} = \sum_{j=1}^{m} P_{i,j}\, E[T_j] \qquad i = 1, 2, \cdots, n \ .$$

Or,

$$\overline{E[\mathbf{T}]} = \mathbf{P}\, E[\mathbf{T}] \ .$$

For finite length queues, the analysis is more complex because packets may try to enter a full queue in which case they are rejected and assumed to be resubmitted.

When a processor emits a packet for memory module j, the packet will be rejected with probability $\pi^{*}_{L+1,j}$ (the probability that queue j is full at submission time, the result $\pi^{*}_{L+1,j}$ requires the Markovian properties of the source processes). The rejected packet then returns to its processor where it will be resubmitted after a Δt delay (this delay models processor resubmission delay, it is a simple approximation to the resubmission process). Assume that resubmissions are handled by interface logic so that processors are free to continue processing even when packets get rejected.

If $\pi^{*}_{L+1,j}$ is small, the emission/resubmission process resembles a Bernoulli process. This approximation is supported by discrete time analyses and simulations, see [MaM81].

Define $E[T_j|A]$ to be the average time a packet spends in queue j given that it was accepted on its first entry attempt.

In a Bernoulli process with probability of event occurrence $(1 - \pi^{*}_{L+1,j})$ (here the event is that a packet is accepted by queue j immediately, thus $1 - \pi^{*}_{L+1,j}$ is the probability of event occurrence), the average number of trials required before the first event occurrence is $\frac{\pi^{*}_{L+1,j}}{1 - \pi^{*}_{L+1,j}}$. Thus the average delay from the first emission to service completion, given that the packet was first rejected is:

$$\left(\frac{\pi^{*}_{L+1,j}}{1 - \pi^{*}_{L+1,j}} + 1 \right) \Delta t + E[T_j|A] .$$

The total average packet delay time for finite queues, to an approximation, is:

$$E[T_j] \approx (1 - \pi^*_{L+1,j})\, E[T_j|A] + \pi^*_{L+1,j}\left(\frac{1}{1 - \pi^*_{L+1,j}}\,\Delta t + E[T_j|A]\right)$$
$$= E[T_j|A] + \frac{\pi^*_{L+1,j}}{1 - \pi^*_{L+1,j}}\,\Delta t\,, \qquad L < \infty\,. \tag{12}$$

This approximation is expected to be accurate when $\pi^*_{L+1,j}$ is small enough that:

1) the Poisson nature of the processor model is not disturbed much, and

2) $\pi^*_{L+1,j}$ is the same for all resubmissions.

Evaluation of $E[T_j|A]$ can be found using Little's result where $\lambda_{e,j}$ is reduced to the rate of packet arrival and acceptance $(1 - \pi^*_{L+1,j})\lambda_{e,j}$. This yields:

$$E[T_j|A] = \frac{E[N_j]}{(1 - \pi^*_{L+1,j})\lambda_{e,j}}\,.$$

So (12) becomes:

$$E[T_j] \approx \frac{E[N_j]}{(1 - \pi^*_{L+1,j})\lambda_{e,j}} + \frac{\pi^*_{L+1,j}}{1 - \pi^*_{L+1,j}}\,\Delta t\,, \qquad L < \infty\,. \tag{13}$$

Note that as $\Delta t \to \infty$, $E[T_j] \to \infty$ (if $\pi^*_{L+1,j} \neq 0$) so memory queue behavior resembles a saturated $M/G/1/\infty$ queue. Alternatively, as $\Delta t \to 0$, $E[T_j] \to \frac{E[N_j]}{(1 - \pi^*_{L+1,j})\,\lambda_{e,j}}$ and if $\pi^*_{L+1,j}$ is small ($\leq 10^{-3}$) the approximation (13) again resembles an $M/G/1/\infty$ queue solution. Obviously if $\pi^*_{L+1,j} \to 0$, the approximation approaches the $M/G/1/\infty$ solution form. Care must be used when applying the waiting time approximation in (13).

5. Special Case Solutions.

This section describes closed form solutions obtainable for smaller sizes of the queue length L.

First the L = 1 case will be discussed, it will be seen that the results approximate discrete time analyses.

Let L = 1, that is, only a single buffer register is present at memory modules. This would be the situation where processors must resubmit requests until buffer registers are free to accept them. Note that this situation in general does not have a low $\pi^*_{L+1,j}$ so the resubmission process approximation will not hold very accurately. For simplicity let $P_{i,j} = \frac{1}{m}$ for all i and j, and $\lambda_i = \lambda$ for all i, also let there be one page size, z = 1. Then,

$$\mathbf{T} = \begin{bmatrix} q_0 & 1-q_0 \\ q_0 & 1-q_0 \end{bmatrix}.$$

Solving the relation $\boldsymbol{\pi} = \boldsymbol{\pi}\mathbf{T}$ yields:

$$\pi_0 = q_0 = e^{-\frac{n}{m}\lambda t_1}$$
$$\pi_1 = 1 - e^{-\frac{n}{m}\lambda t_1}$$

Then from (8.1) and (8.2):

$$\pi^*_0 = \left(\frac{1}{e^{-\frac{n}{m}\lambda t_1} + \frac{n}{m}\lambda t_1}\right) e^{-\frac{n}{m}\lambda t_1}$$
$$\pi^*_1 = \left(\frac{1}{e^{-\frac{n}{m}\lambda t_1} + \frac{n}{m}\lambda t_1}\right) \left(1 - e^{-\frac{n}{m}\lambda t_1}\right)$$
$$\pi^*_2 = 1 - \frac{1}{e^{-\frac{n}{m}\lambda t_1} + \frac{n}{m}\lambda t_1}$$

Evaluating the remaining measures gives:

$$PU = 1 .$$

From equation (9):

$$MU = 1 - \pi^*_0 = \frac{\frac{n}{m}\lambda t_1}{e^{-\frac{n}{m}\lambda t_1} + \frac{n}{m}\lambda t_1} .$$

From equation (10):

$$E[N_j] = E[N] = \pi^*_1 + 2\pi^*_2$$

$$= 2 - \frac{1 + e^{-\frac{n}{m}\lambda t_1}}{e^{-\frac{n}{m}\lambda t_1} + \frac{n}{m}\lambda t_1} .$$

From equation (13):

$$E[T_j] = E[T] \approx \frac{\pi^*_1 + 2\pi^*_2}{(1 - \pi^*_2)\frac{n}{m}\lambda} + \frac{\pi^*_2}{1 - \pi^*_2}\Delta t .$$

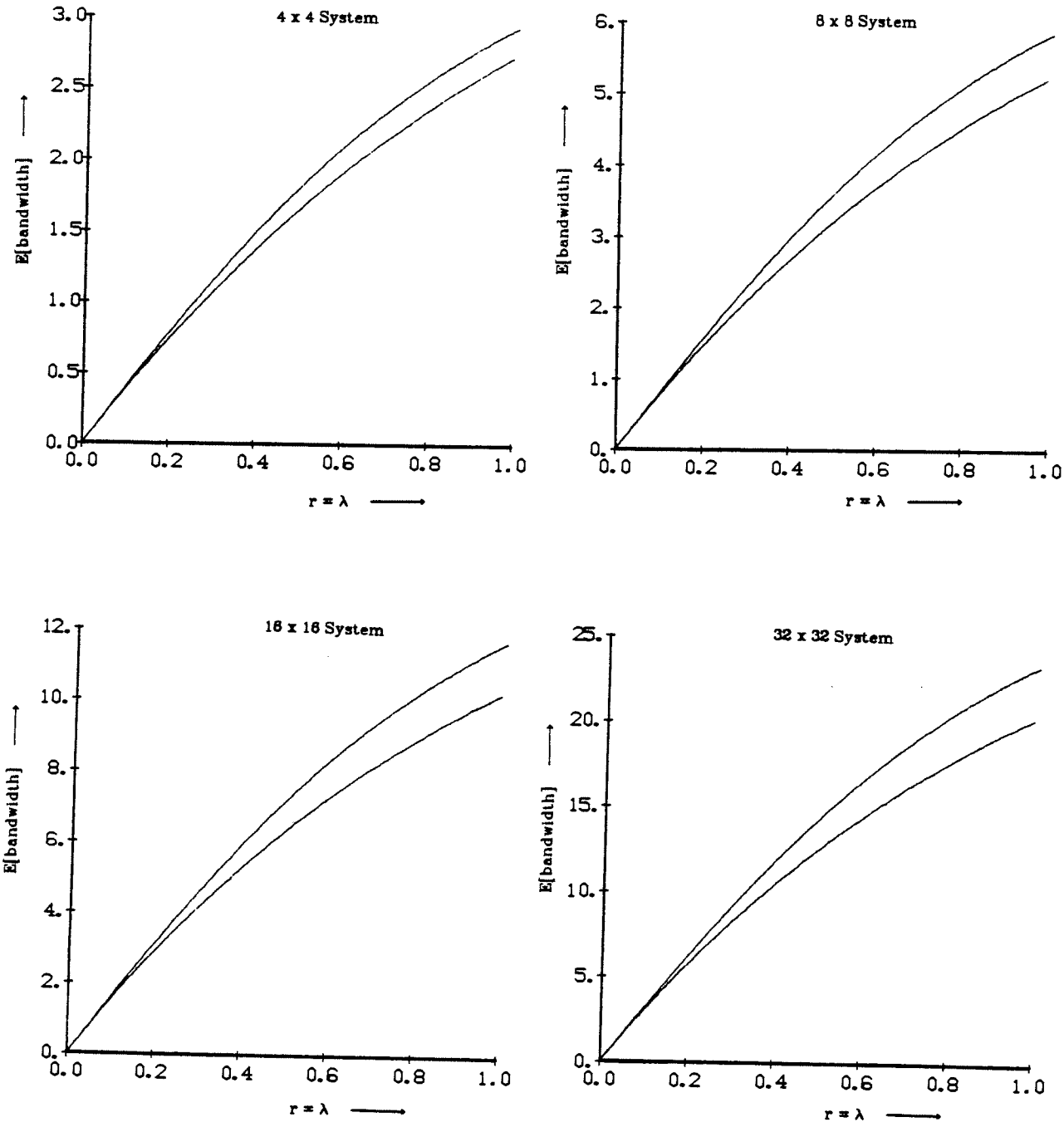
It is interesting to note that if $t_1 = 1$, and $0 \leq \lambda \leq 1$ then MU is close to E[bandwidth], defined as the average number of connections in use per unit cycle, for request rate $r = \lambda$ in discrete time model analyses [SkA69, BaS76, Hoo77, SeD79, Pat79, Rau79, MaM81]. (In discrete time model analyses, bandwidth has been defined as the number of connections available on a given system cycle. In discrete time model analyses, r is taken to be the probability of packet emission during a cycle, it is a request rate. Equating r and λ establishes a connection between the two modeling techniques.) If the bandwidth of a continuous time model is defined as the average number of busy servers, then the two models agree to some extent for large systems (n, m $\geq$ 32). The similarity occurs because the Poisson model of processor behavior is a continuous time extension of the Bernoulli process model of processor behavior. The two models should agree to some extent. This was noted differently by [BaS76]

where they found that if n and m $\rightarrow \infty$, then the discrete time model approximates the continuous time model. Here it is seen that the two models are similar in results (to within about 15%) for E[bandwidth] even when n and m are not large. Figure 5 shows the two E[bandwidth]'s.

The primary difference between the two models is that in the discrete time model, processors emit memory requests and wait for service, so the discrete time model is a finite customer queueing model. The continuous time model is an infinite customer queueing model. It seems reasonable that if $\frac{1}{\lambda_i} > \sup_j \{E[T_j]\}$ for all i , that is, the mean time between packet emissions is greater than the maximum possible average packet delay time, the continuous model is "approximately" a finite customer model. On the average each memory reference is completed before the next packet is emitted. See Figure 6 for a timing description of the situation. Obviously, since the Poisson process is memoryless, the approximation may be inaccurate, but for low emission rates it may well be close to a finite customer model. This may be seen by observing the curves of Figure 5 in the low rate region. The two analyses are very close for low emission rates.

Another difference between the two models is that of service discipline, in the continuous time model, the server operates on a FCFS basis for those packets in the queue. Whereas in the discrete time model the next request packet to be serviced is chosen randomly from those packets present at the server. The difference does not affect MU much but does affect $E[T_j]$'s.

Consider the case where L = 2. Again let $P_{i,j} = \frac{1}{m}$ for all i and j, $\lambda_i = \lambda$ for all i, and z = 1 . Then,



Lower curves are discrete time analysis results,
upper curves are continuous time analysis results.

Figure 5. E[bandwidth] for discrete and continuous time models.

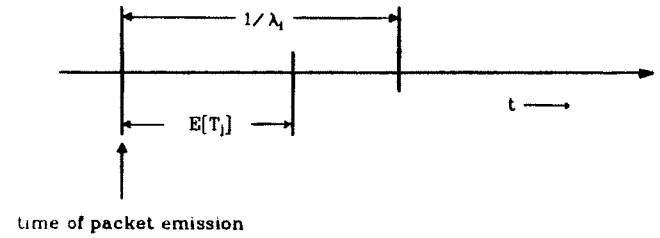


Figure 6. Average timing of packet emission and delay.

$$\mathbf{T} = \begin{bmatrix} q_0 & q_1 & c_1 \\ q_0 & q_1 & c_1 \\ 0 & q_0 & c_2 \end{bmatrix}.$$

Again, solve $\pi = \pi \mathbf{T}$ to get the steady state queue length probabilities, due to symmetry $\pi_j = \pi_i$ for all i, j :

$$q_0(\pi_0 + \pi_1) = \pi_0 ,$$
$$q_1(\pi_0 + \pi_1) + q_0\pi_2 = \pi_1 ,$$
$$\pi_0 + \pi_1 + \pi_2 = 1 .$$

Solving this gives:

$$\pi_0 = \frac{q_0^2}{1 - q_1} ,$$
$$\pi_1 = \frac{q_0(1 - q_0)}{1 - q_1} ,$$
$$\pi_2 = \frac{1 - q_0 - q_1}{1 - q_1} .$$

Where,

$$q_0 = e^{-\frac{n}{m}\lambda t_1} ,$$
$$q_1 = \frac{n}{m}\lambda t_1 e^{-\frac{n}{m}\lambda t_1} .$$

The π^* vector, E[N], and E[T] are lengthy and as such will not be written here (see the Appendix).

Figure 7 shows the effects of increasing L on MU for the equal rate, uniform memory reference distribution case and the example $\beta(t)$ used in the running example. As suggested by such graphs, it may be seen that the analysis allows L to be chosen so that MU is arbitrarily close to its maximum. In general, as L gets larger, MU gets larger. As L increases, the cost of memory modules increases also. There are many values for L that make MU as large as possible (for a given $\mathbf{P}$ and λ) but some of these values of L may violate a cost (or reliability) constraint. The smallest L that satisfies the desired MU and $E[\mathbf{T}]$ constraints may be found iteratively but not analytically.

For example, to find the smallest L satisfying a desired lower bound on MU and a desired upper bound on $E[\mathbf{T}]$ (if an L satisfying the desired constraints exists) use a simple linear search on L:

(1) Choose a test set of program, processor, and memory module model parameters:

$$\{ (\mathbf{P}, \boldsymbol{\lambda}, \beta(t), \Delta t) \}$$

The test set represents those algorithms that comprise the majority of programs to be run on the system.

(2) Choose the minimum acceptable measures MU and packet delay time for the model parameters chosen in step (1). Call these MU_{min} and $E[\mathbf{T}]_{max}$.

(3) Set L = 1 .

(4) Compute MU from equations (1), (3), (7), (8.1) and (8.2) for all points in the test set defined by step (1).

(5) If some MU computed in step (4) $< MU_{min}$, go to step (8).

(6) Compute for all test points defined by step (1) $E[\mathbf{T}]$'s from equations (10) and (13) using the results from step (4).

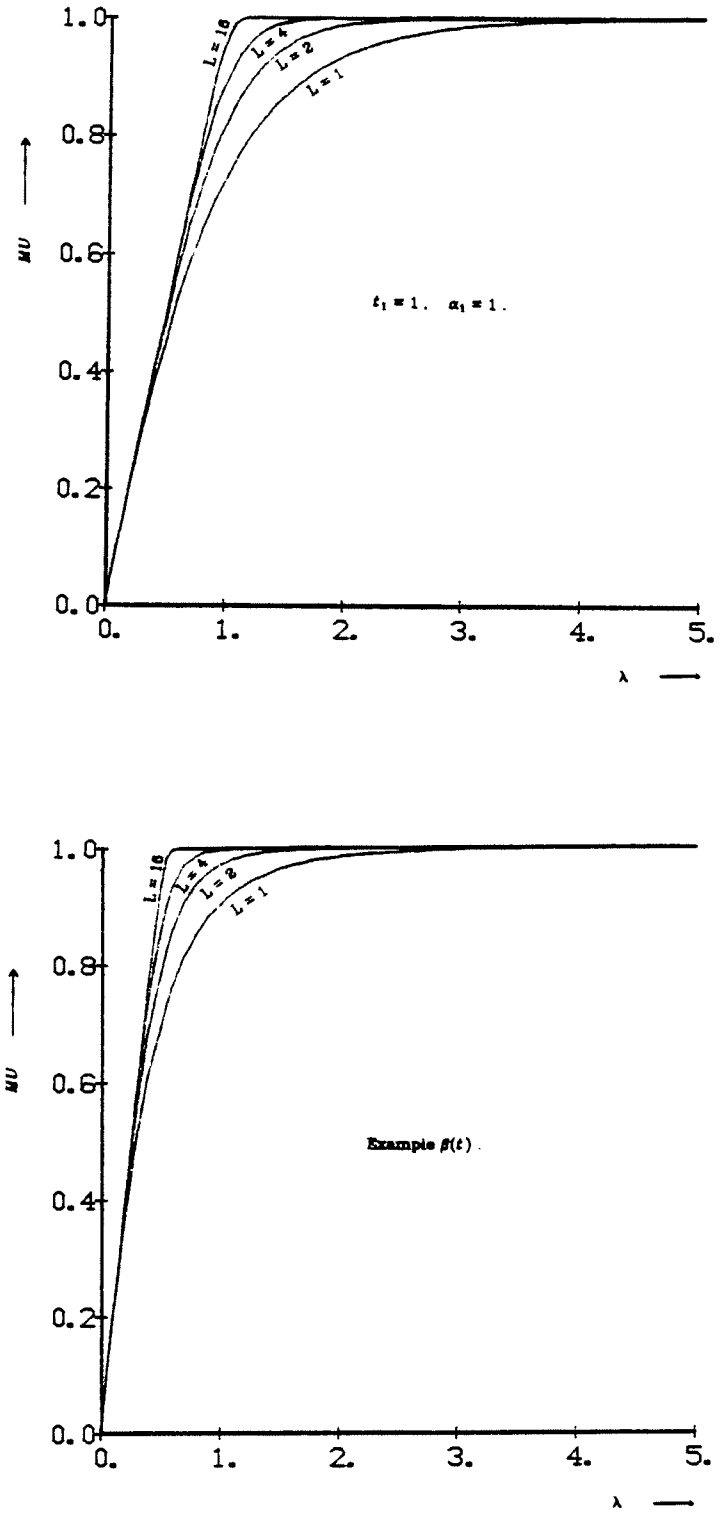


Figure 7. MU as a function of L.

(7) If all E[T]'s computed in step (6) $< E[T]_{max}$, stop. L now is the smallest queue length that satisfies the desired constraints. Check to see if all $\pi^{*}_{L+1,j}$'s are small ($\leq 10^{-3}$), if so E[T] computed is sufficient. If some $\pi^{*}_{L+1,j}$ is large ($>10^{-2}$) E[T] computed may be inaccurate.

(8) $L \leftarrow L + 1$, go to step (4).

6. Concluding Remarks.

This report has described a model and analysis of a crossbar-based, packet switched multiprocessor computer system. The model was discussed in the context of two systems: a general purpose multiuser system; and a numerically oriented multiprocessor system (suitable for sparse array processing) based on MAP-200 like processors. Performance measures for the model were defined. A solution for the performance measures was derived. Using the model, queue lengths that satisfy system requirements may be found.

The authors wish to thank D. G. Furchtgott for his careful proof-reading and critique of this report.

7. References.

[BaS76]
F. Baskett, and A. J. Smith, "Interference in Multiprocessor Computer Systems with Interleaved Memory," *CACM*, Vol. 19, No. 6, June 1976, pp. 327-334.

[Ben65]
V. E. Benes, *Mathematical Theory of Connecting Networks and Telephone Traffic*, Academic Press, New York, 1965.

[Bha75]
D. P. Bhandarkar, "Analysis of Memory Interference in Multiprocessors," *IEEE TC*, Vol. C-24, No. 9, Sept. 1975, pp. 897-908.

[Cin75]
E. Cinlar, *Introduction to Stochastic Processes*, Prentice-Hall Inc., Englewood Cliffs, N.J., 1975.

[Clo53]
C. Clos, "A Study of Non-blocking Switching Networks," *The Bell System Technical Journal*, Vol. 32, March 1953, pp. 406-424.

[Coo72]
R. B. Cooper, *Introduction to Queueing Theory*, The Macmillan Company, New York, 1972.

[CoS81]
E. U. Cohler, and J. E. Storer, "Functionally Parallel Architectures for Array Processors," *Computer*, IEEE, Sept. 1981, pp. 28-36.

[FuB75]
S. H. Fuller, and F. Baskett, "An Analysis of Drum Storage Units," *JACM*, Vol. 22, No. 1, Jan. 1975, pp. 83-105.

[GrH74]
D. Gross, and C. M. Harris, *Fundamentals of Queueing Theory*, John Wiley and Sons Inc., New York, 1974.

[Hoo77]
C. H. Hoogendorn, "A General Model for Memory Interference in Multiprocessors," *IEEE TC*, Vol. C-26, No. 10, Oct. 1977, pp. 998-1005.

[GoL73]
G. R. Goke, and G. J. Lipovski, "Banyan Networks for Partitioning Multiprocessor Systems," *Proc. First Annual Symp. on Computer Architecture*, IEEE, Dec. 1973, pp. 21-28.

[Kle75]
L. Kleinrock, *Queueing Systems Volume I: Theory*, John Wiley & Sons Inc., New York, 1975.

[Law75]
D. H. Lawrie, "Access and Alignment of Data in an Array Processor," *IEEE TC*, Vol. C-24, No. 18, Dec. 1975, pp. 1145-1155.

[MaG81]
M. A. Marsan, and M. Gerla, *Markov Models for Multiple Bus Multiprocessor Systems*, Report No. CSD 810304, Computer Science Department, UCLA, Feb. 1981.

[MaM81]
B. A. Makrucki, and T. N. Mudge, *Probabilistic Analysis of a Crossbar Switch*, SEL Report No. 150, Department of Electrical and Computer Engineering, University of Michigan, March 1981.

[Pat79]
J. H. Patel, "Processor-Memory Interconnections for Multiprocessors," *Proc. 6th Annual Symp. on Computer Architecture*, IEEE, April 1979, pp. 166-177.

[Rau79]
B. R. Rau, "Interleaved Memory Bandwidth in a Model of a Multiprocessor Computer System," *IEEE TC*, Vol. C-28, No. 9, Sept. 1979, pp. 678-681.

[SeD79]
A. S. Sethi, and N. Deo, "Interference in Multiprocessor Systems with Localized Memory Access Probabilities," *IEEE TC*, Vol. C-28, No. 2, Feb. 1979, pp. 157-163.

[Sie77]
H. J. Siegel, "Analysis Techniques for SIMD Machine Interconnection Networks and the Effects of Processor Address Masks," *IEEE TC*, Vol. C-26, No. 2, Feb. 1977, pp. 153-161.

[Sie80]
H. J. Siegel, (Ed.), *Proc. Workshop on Interconnection Networks*, Purdue University, April 21-22, 1980.

[SkA69]
C. E. Skinner, and J. R. Asher, "Effects of storage contention on system performance," *IBM Systems Journal*, No. 4, 1969, pp. 319-333.

[Smi81]
J. E. Smith, Private communication, Department of Electrical and Computer Engineering, University of Wisconsin, Nov. 1981.

[Str70]
W. D. Strecker, *Analysis of the Instruction Execution Rate in Certain Computer Structures*, Ph.D. dissertation, Carnegie-Mellon University, Pittsburgh, 1970.

8. Appendix.

From section 5, the results for L = 2 are written here.

$$\pi^*_0 = \frac{q_0^2}{q_0^2 + \frac{n}{m}\lambda t_1(1 - q_1)}$$

$$\pi^*_1 = \frac{q_0(1 - q_0)}{q_0^2 + \frac{n}{m}\lambda t_1(1 - q_1)}$$

$$\pi^*_2 = \frac{1 - q_0 - q_1}{q_0^2 + \frac{n}{m}\lambda t_1(1 - q_1)}$$

$$\pi^*_3 = \frac{q_0^2 + (\frac{n}{m}\lambda t_1 - 1)(1 - q_1)}{q_0^2 + \frac{n}{m}\lambda t_1(1 - q_1)}$$

$$E[N] = \pi^*_1 + 2\pi^*_2 + 3\pi^*_3$$

$$= \frac{q_0(2q_0 - 1) + q_1 + 3\frac{n}{m}\lambda t_1(1 - q_1) - 1}{q_0^2 + \frac{n}{m}\lambda t_1(1 - q_1)}$$

$$E[T] = \frac{q_0(2q_0 - 1) + q_1 + 3\frac{n}{m}\lambda t_1(1 - q_1) - 1}{\frac{n}{m}\lambda(1 - q_1)} + \frac{q_0^2 + (\frac{n}{m}\lambda t_1 - 1)(1 - q_1)}{\left[q_0^2 + \frac{n}{m}\lambda t_1(1 - q_1)\right](1 - q_1)}\Delta t$$