NETWORK MODELS FOR LARGE-SCALE TIME-SHARING SYSTEMS

by

Charles G. Moore III

TECHNICAL REPORT NO. 71-1

30 April 1971

Prepared Under Contract
N00014-67-A-0181-0036 (NR 049-311)
For the Office of Naval Research

Stephen R. Kimbleton, Project Director

ISDOS RESEARCH PROJECT
PERFORMANCE MODELING GROUP

Department of Industrial Engineering
The University of Michigan
Ann Arbor, Michigan

Acknowledgements

First, acknowledgement must go to my dissertation committee: Bruce Arden, Larry Flanigan, and Bernie Galler of the Computer and Communication Sciences Department and Steve Kimbleton of the Industrial Engineering Department. In particular, Bruce Arden, my committee chairman, and Steve Kimbleton did much to shape the ideas presented here.

Much credit should also go to the University of Michigan Computing Center - both its equipment and its people. It has been an exciting and stimulating environment in which to work. Among the staff, Mike Alexander has been an invaluable source of suggestions and information. Tad Pinkerton (now at the University of Wisconsin) left behind the software instrumentation system which was crucial to this dissertation.

This thesis was prepared on a Model 37 Teletype, using a DEC PDP-10 time-sharing system. The actual text preparation was done by a computer program called CypherText (AFIPS Conference Proceeding, Fall 1970, 555-561). The computer services required for this effort were provided by the Cyphernetics Corporation of Ann Arbor, for which I am most appreciative. The special characters which contribute much to the readability of the mathematical sections were supplied by the Teletype Corporation.

And last, but certainly not least, I must thank my wife, Pat, who has waited (usuaully patiently) many years for the completion of this dissertation.

Table of Contents

Table of Contents (cont.)

List of Figures

List of Figures (cont.)

Chapter I

Introduction to the Problem

1. Analytic Time-Sharing System Models

There have been many attempts to develop mathematical models of computer systems. The goal has most often been to investigate various aspects of the design and performance of the system. Queue-theoretic models have been used to study various scheduling algorithms for central processors[A3,C5,C6,K4,K5,S7] and file storage devices [A1,A2,C4,D4,F4]. Probabilistic models have most recently been used in a number of papers dealing with memory management in virtual memory systems[B3,D1,D7,D8,F2,S6,W3].

In most cases, analytic models have been used to investigate relative performance of alternative system structures, rather than absolute performance. By this we mean, for example, that a model might be used to show that a particular scheduling algorithm has certain advantages over a different scheduling algorithm, under given conditions[K5]. Such papers require no reference to real systems, and no attempt is generally made to argue that the model used to compare the scheduling algorithms represents any real system.

Attempts to develop complete performance models of

real time-sharing systems have generally been made via simulation models. By the very nature of such models - great detail is invited because it is possible - the models developed tend to be special purpose. A model is generally developed for a particular time-shared operating system, operating on a particular type of equipment (if not a particular configuration of that equipment)[N2]. It is perhaps for this reason that such models have not received wide attention. Proponents of various simulation languages might argue that there is no need for general models, as it is easy to develop new special purpose models within the language.

2. Network Models

It is the goal of this dissertation to develop a class of analytic models suitable for the investigation of the absolute performance of time-sharing systems. We have chosen to call these network models, although the models investigated here are a very narrow sub-class of what are called queueing networks[S2]. Apologies are directed to those who find this narrow use of the term objectionable. There seems to be no conflict in usage within the computer modeling field, and the term is descriptive of the difference between the models developed here and previous ones.

Network models have an intuitively attractive feature in that they allow the representation of a time-sharing system as a set of independent resources. This seems to be a crucial characteristic of the current generation of large-scale time-sharing systems. The users of the system compete for resources. When the immediate demand cannot be met, the operating system (supervisor, monitor, executive, or whatever) automatically queues the requests that cannot be met, and the performance of the system degrades from the user's point of view. When queueing occurs, the user can no longer be given the impression that he "has the entire system to himself" because of the delays. Of course if the delays are short enough, this degradation may not be objectionable or even noticable to the user.

Network models are introduced as a set of independently operating queues. They are structured in such a way, however, that they can be represented as a time-continuous markov process. It is this characteristic which makes it possible to derive many results. On the other hand, considerable extra work is required to allow the characteristics of time-sharing systems to be represented within the model. The necessary mathematical development is carried out in Chapter III.

3. Performance and Load Measures

There are a variety of proposed performance measures for time-sharing systems. Many of them are based on some variant of the system response time. This quantity has been variously defined[S5,S7,S11]. It generally measures the time required by the system to initiate work on, partially complete, or finish, some user's request for service made from a terminal. As another example, the degree of utilization of system resources has also been suggested as a performance measure[C1]. This measure is less user oriented and of more interest to the system designer.

In batch-oriented computer systems, the terms throughput and turn-around-time are often used to denote performance measures. Throughput is usually defined as "jobs completed per unit time", while turn-around-time is the time required to complete the job. Equivalent concepts (though not as well-defined) exist for time-sharing systems.

Though the measures of performance differ greatly, they have one thing in common: each must be measured relative to some state of the system being analyzed or modeled. One would often like to get a graph of the performance measure versus some measure of how busy or loaded the system is. We will call this second quantity a load measure. The question of how system load should be defined, measured, and

represented in a modeling effort seems to have received little attention in the literature. One point of possible confusion is that some of the candidates for a performance measure are also candidates for a load measure (resource utilization is a prime example).

In an infinite-source queueing model, the arrival rate at the service facility divided by the average service rate (and usually called ρ, the load factor) is used as the measurement of the load in the model. In finite-source models, there is usually an arrival rate characteristic of each customer (and identical for all customers) so the equivalent load measure is the arrival rate per customer(λ), divided by the service rate of the service facility(μ), times the number of customers in the system(N). Or:

$$\rho = \frac{N\lambda}{\mu}$$

In Chapter IV we consider in depth a number of performance and load measures. They are discussed in light of available statistical data as well as the performance measures that can be derived from network models.

Further concepts developed in Chapter IV are those of balance and saturation. Each has its intuitive meaning. Saturation for a system is the load the system can absorb without experiencing significant degradation in performance.

A balanced system is one in which the available resources are well-matched to the demands of its customers. In a balanced system, as the saturation point is reached all resources become equally busy.

The concepts of saturation and balance both have natural definitions within the framework of network models. The characteristics of these concepts are explored in Chapter IV.

4. Using the Model

The proof of the worth of a model lies in its ability to be used as a tool in investigating an actual time-sharing system. There are two very different cases to be considered. The first is the case in which one wants to evaluate the expected performance of some system design which has not yet been implemented. In this case, the actual structure of the system is only imperfectly known. Such things have a tendency to change drastically between design and fruition. The behavior of the system users, who will eventually provide the load, is completely unknown. It may be possible to estimate the parameters of the model from previous experience and data available from other systems. However, this approach is fraught with danger. If the new system were just like some other one, there would be no need

to build it. The system characteristics will be different and presumably the system's users will behave differently in the face of the new (and hopefully improved) capability. As will be shown, relatively slight shifts in user characteristics can cause drastic changes in system performance. This is not to imply that there is no hope. One can perhaps find a reasonable range of user behavior and system parameters to use in a model. Using this range in the model (assuming it is possible) will yield a range of possible system performances. One should be prepared, however, to find that the predicted performance range runs from hopelessly unacceptable to quite adequate - with little assurance of where the actual operating point will exist.

In modeling existing systems the problems are much different. The system structure is in principle well known. The user behavior can be determined by the gathering of appropriate statistics, assuming instrumentation exists within the system for this purpose. The reason for modeling existing systems is generally to evaluate the effect on system performance of some proposed or expected change in the system structure.

The model can be developed using existing data, and can be validated using that data. The conditions which will allow the model to remain valid after the change in structure are usually well-defined. Checking these conditions may

or may not present problems.

For the particular case of network models, the parameters necessary to define a particular model are essentially unlimited, since there is such freedom in developing the structure of the model. Chapters V and VI are devoted to the development of two network models for time-sharing systems. The first is not intended to represent any real time-sharing system and serves as a vehicle to illustrate the use of network models and the concepts developed in Chapters III and IV.

Chapter VI contains the development of a model for MTS, the University of Michigan Terminal System. MTS is a large time sharing system upon which a wealth of statistical data is available. This makes it possible to demonstrate the development of a network model from a known system structure. In this case, we can also see clearly where the model does deviate from the system being modeled. Where empirical distributions differ from the distributions used in the model, the effect is evaluated. Where there are structural differences between the model and MTS, we again examine the seriousness of these necessary simplifications.

Having developed the model from MTS statistics, we evaluate the model's validity using other statistics which measure actual system performance. If the model had been faithful in every way to the structure of MTS, there would

be no difference between expected and observed performance. What differences exist are due to the differences that exist between the model and the system.

Having validated the model within the "operating range" of MTS, we now use the model to study system performance outside the normal operation load. For example, we predict response time under very heavy loads. In addition, we also consider possible alternatives to the current MTS hardware configuration. In making such studies, it is necessary to consider whether the model will remain valid under the new structure. For it to remain valid, the model parameters must have been developed from statistics and considerations which are independent of the change in structure or load. Finally, the small number of feasible alternative configurations for the MTS system makes it possible to search for a configuration which appears optimum within certain constraints.

Chapter II

Review of Previous Work

Introduction

The purpose of this chapter is first of all the straight-forward one - to review for the reader the previous work in the field of analytic models for time-sharing systems and their components. There is also a secondary goal. In developing a network model by the methods of Chapter III we may be able to take advantage of results obtained from less general models. The parameters which must be chosen to define a particular model are not trivially determined. We shall see that models for components of the system - such as processor scheduling, memory management and file operations - may be necessary to assign values to the parameters of the network model. Thus this chapter can also be viewed as a marshalling of tools which will prove useful in the modeling tasks ahead. The reader who is familiar with the literature on computer system models may easily skip this chapter. Appropriate references will be made when material outlined here is referenced in succeeding chapters.

The work to be reviewed is divided into several classes, with regard to the problems that are treated rather than the way they are treated. The emphasis is on results,

not methods. Work which is heavily mathematical in nature is referred to as "queue-theoretic" as opposed to "stochastic" or "probabilistic". The line of demarcation is not exact, but lies somewhere between an introductory course in probability theory and a good course in queueing theory (as required background for reading the paper).

1. Processor Scheduling Models

By processor scheduling models, we mean those which are concerned primarily with the effects of reordering the queue of requests for central processor (CPU) service. Much of the work reported here is queue-theoretic. Much of it limits itself to consideration of single-thread (non-multiprogramming) systems. This makes the work somewhat irrelevant to the modeling task at hand, since large-scale time-sharing systems by nature depend heavily on multiprogramming capability. Nevertheless, this work is of interest, since it was the first to appear in the area of computer system modeling and leads directly to some of the papers which are of more interest.

An excellent review of processor scheduling models by McKinney[M2] has appeared in Computing Surveys. Rather than repeat what is presented there, that paper is recommended to the reader if he is not familiar with the work in

this area.

In addition to the papers discussed by McKinney in his review, a general treatment of single-server, finite-source M/G/1 (exponentially distributed inter-arrivals, general service time distribution, single server) queueing systems is given by Jaiswal in his book Priority Queues[J3].

A paper by Kleinrock[K4] should also be singled out for special mention. Kleinrock investigates the simple finite-source M/M/1 queueing system. Customers at the source are viewed as "thinking" at their terminal. Arrival at the server is viewed as initiation of a request from the terminal. The time to complete this request is the response time. Let the mean think time be $1/\lambda$ and the mean service time be μ. Kleinrock solves the model and graphs the response time as a function of N, the number of terminals. He notes that response time increases slowly with N for $N < \lambda\mu+1$. For $N > \lambda\mu+1$, response time rises more quickly, approaching a slope of μ seconds per additional terminal. Kleinrock calls $N = \lambda\mu+1$ the saturation point for the system. Intuitively, we can see that if $N = \lambda\mu+1$, and one customer is in service, customers arrive at the server at the rate of one every $1/\mu$ seconds, which is exactly the rate at which they are served. Below this rate there will be excess server capacity and little congestion. Above this rate additional customers add to the delay by approximately their

service time. The argument is, of course, made more precise by Kleinrock. The same sort of phenomenon arises in the development of network models, and a similar argument for a saturation point will be made in Chapter IV.

2. On the Advantages of Finite-Source Models

In developing analytic models for processor scheduling or other aspects of time-shared computer systems, a fundamental choice must be made between the use of an infinite or finite source model. Time-sharing systems, unlike many other stochastic service systems, are inherently finite in the nature of their source of customers. The (effectively infinite) number of customers passing by a snack bar will be unaffected by the (finite) number of persons inside the snack bar. One may assume the number of customers in the snack bar is never a significant portion of the population. This is not so obviously true in time-sharing systems, since they have a maximum customer population limited by the number of available terminals which can be connected to the system. The number of terminals may be in the order of several hundred (though it is usually not). Even with that number, however, we can still forsee the possibility of a significant portion of the terminal users having requests pending simultaneously in the system. This means that they

no longer contribute to the stream of incoming requests, and the arrival rate goes down. This phenomenon provides a feedback in the arrival rate which is not present in infinite-source, constant arrival rate systems. One would expect to find different characteristics in these controlled and self-limiting systems.

We proceed now to investigate, in terms of time-sharing processor scheduling models, the extent to which infinite-source approximations may be valid. It would be possible to work within a more general framework, but the implications of the results presented here for other situations should be clear.

By a finite-source processor-scheduling model, we mean one in which there are a fixed number of customers, N, in the system. These customers, when they complete their CPU service, return to the source. The time before they return to the processor is then determined by some distribution function characteristic of the time-in-source for all customers. An infinite-source model is one in which there is a positive (and usually constant) arrival rate of customers to the service facility, no matter what its state.

Infinite-source models are generally much simpler to treat mathematically. For example, for exponentially distributed arrivals, the number of customers arriving in time t is Poisson distributed. Let the arrival rate be λ, and

denote the probability that n customers arrive in time t by $p_n(t)$. Then we have:

$$p_n(t) = \frac{e^{-\lambda t}(\lambda t)^n}{n!}$$

For a finite-source model this statistic is more complicated. If we assume the time-in-source for each customer is exponentially distributed with parameter λ, then the number of customers arriving at the service center in time t, assuming no customer completes service in the interval, is binomially distributed with parameters $(1-e^{-\lambda t})$ and k, where k is the number of customers at the source at the beginning of the interval t. Thus:

$$p_n(t) = \binom{k}{n} (1-e^{-\lambda t})^n (e^{-\lambda t})^{k-n}$$

This increased complexity exhibits itself throughout the entire treatment of otherwise-equivalent infinite and finite source models.

Since it is easier to derive results for infinite-source models, it is obviously advantageous to use infinite-source results wherever they do not differ significantly from finite-source results. The question that arises is under what conditions this is possible for time-sharing system models. There are two conditions which must

be met. Together they essentially insure that the arrival rate to the service facility remains close to its maximum.

(1) N is sufficiently large in the system being modeled.

(2) The load in the finite-source system is not too close to, or greater than 1.

We denote the load in either type of queueing system by ρ. For the infinite-source model, $\rho=\lambda\mu$, the arrival rate times the mean service time. For the finite-source model we have $\rho=N\lambda\mu$, the total arrival rate if all customers are at the source. In this case, λ is the arrival rate for a single one of the N identical customers. Of course in a finite-source system the instantaneous arrival rate depends on the number of customers at the source and is generally less than $N\lambda\mu$.

Condition (1) probably does hold for most time-sharing systems of interest. For $N > 25$ infinite source results for several important statistics are within 5% of the finite-source results for $\rho = .6$ or less (see Jaiswal[J1, p. 47]).

The second condition is not nearly so widely recognized. Neither is it so easily met. There are a number of reasons to expect that time-sharing system resources will operate under loads close to or greater than one. A manage-

ment goal in operating a time-sharing system must be to get the highest possible resource utilization (efficiency) consistent with satisfactory performance. In terms of processor scheduling models, this means keeping the probability of an idle server, the CPU, low, while keeping response time acceptable. The controllable parameter in determining the load is most often the number of terminals.

Taking a simple M/M/1/N queueing system (the model of Kleinrock[K4]) as our finite source queueing system and an M/M/1 infinite-source queueing system as its infinite counterpart, we investigate their relative performance. We let the expected waiting time in the model represent the response time. The exponential server represents the CPU or processor and has a mean service time of μ. Under any scheduling algorithm or service distribution, the CPU utilization will rise nearly linearly with the load, ρ, for ρ sufficiently less than 1. Response time, on the other hand, rises quite slowly with ρ, for ρ less than 1. This relation is illustrated by Fig 2-1 and Fig 2-2, where we have graphed CPU utilization and response time for both the finite and infinite source models. The dotted lines correspond to the finite-source model, the solid lines to the infinite-source model. We have let N=100 and μ=1 for these computations. Note that in going from ρ=0 to ρ=.6, the CPU utilization has gone from 0% to about 60%, while the

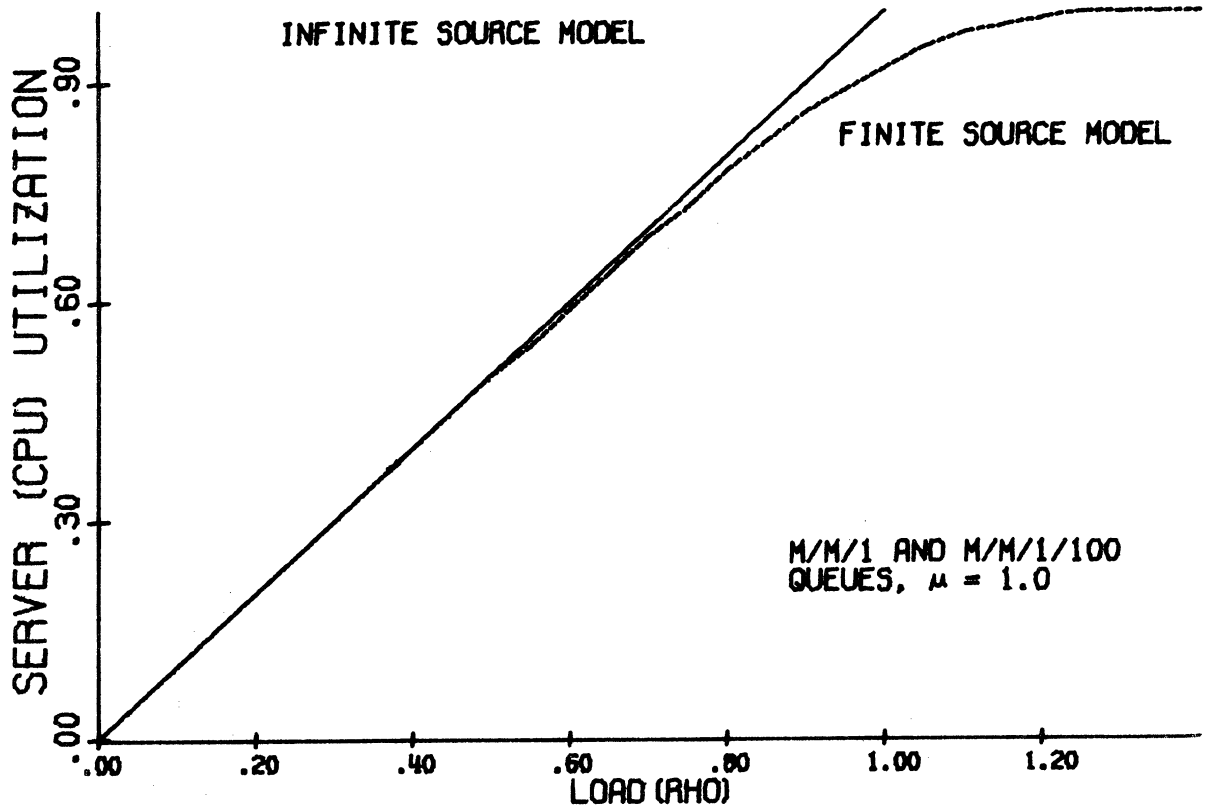

FIG 2-1: LOAD VS. UTILIZATION

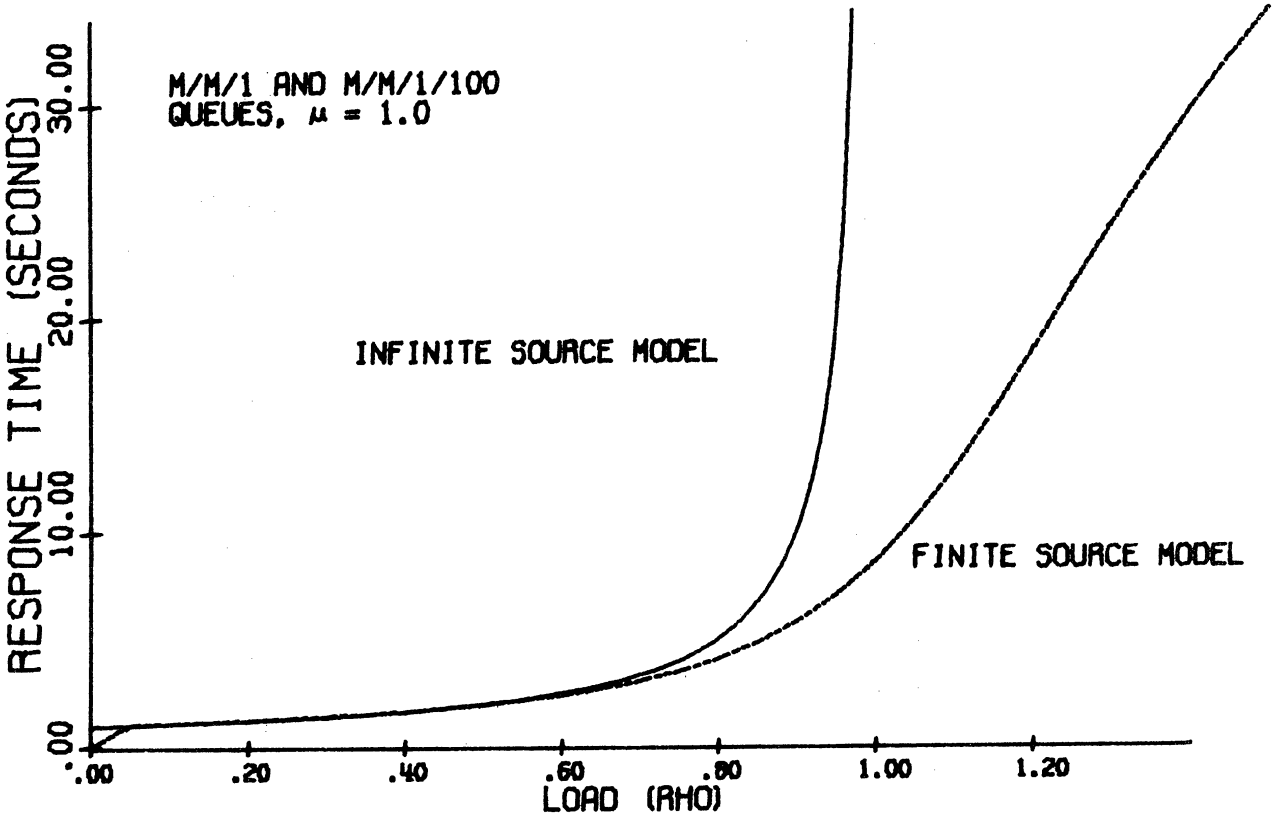

FIG 2-2: EXPECTED RESPONSE TIME VS. LOAD

response time has approximately doubled. Then we would certainly wish to operate such a system in the region where $\rho \geq .6$, for in this region there is an appreciable increase in revenue (billable CPU time), at little cost to the customer in terms of response time. In fact it may be possible to operate the system satisfactorily in the region with ρ greater than 1.

It should be noted that for less common statistics the difference between the finite and infinite-source models may be even more pronounced. This is illustrated by Fig 2-3 which graphs the response time variance for the two models, as a function of load. It is interesting that for sufficiently high loads ($\rho > 1.3$) the variance of the response time decreases with increasing load, for the finite-source model.

It might be suggested that as an alternative to abandoning the infinite-source model in the region $\rho > 1$, it might be possible to let ρ be different in the finite and infinite source models. Instead, choose ρ in the infinite-source model so that some statistic in the finite source model is matched. The futility of this approach is illustrated in Fig 2-4. The finite-source system has $\rho = 1.4$. For the infinite-source model, we chose $\rho = .966$, such that the expected response time (29 seconds) was equal in both models. The graph of the queue length distributions in

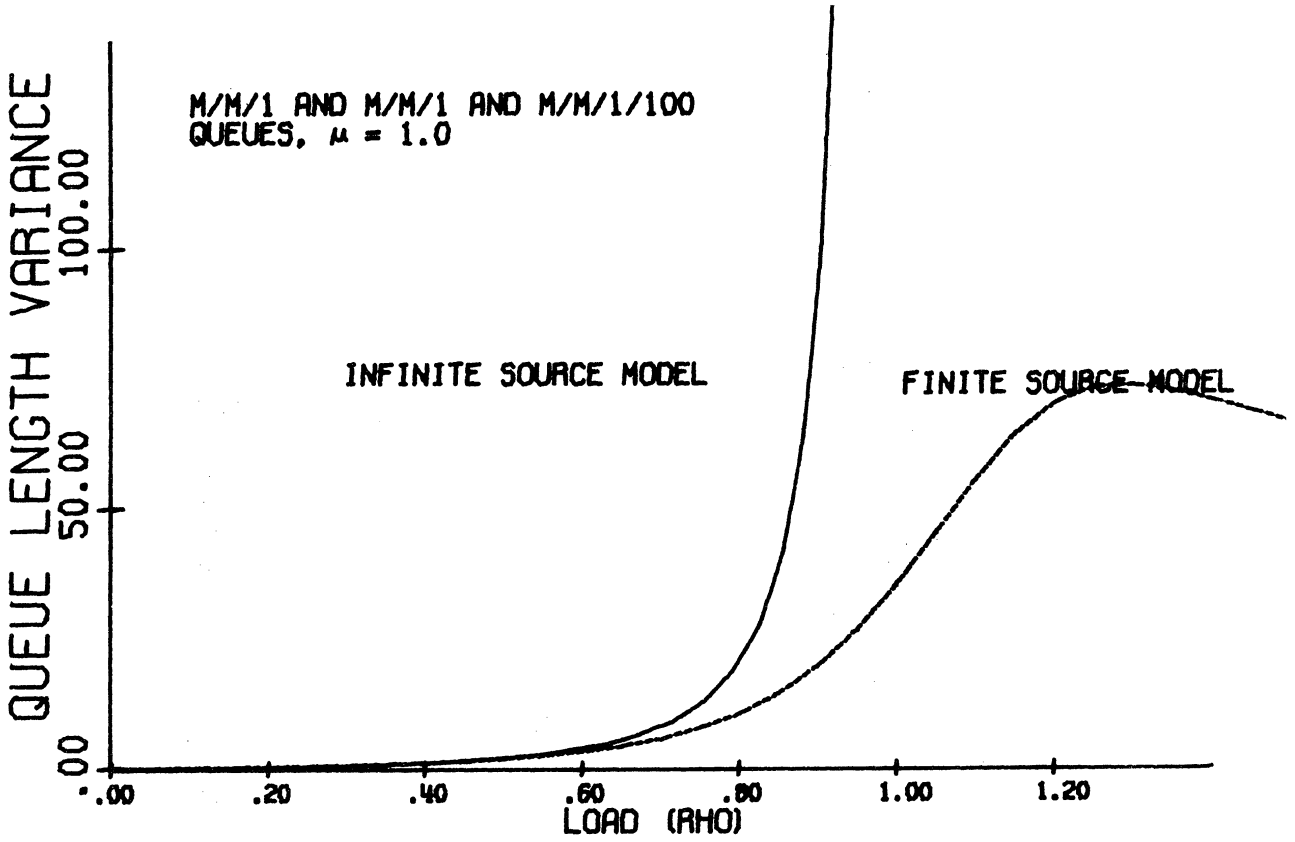

FIG 2-3: LOAD VS. QUEUE LENGTH VARIANCE

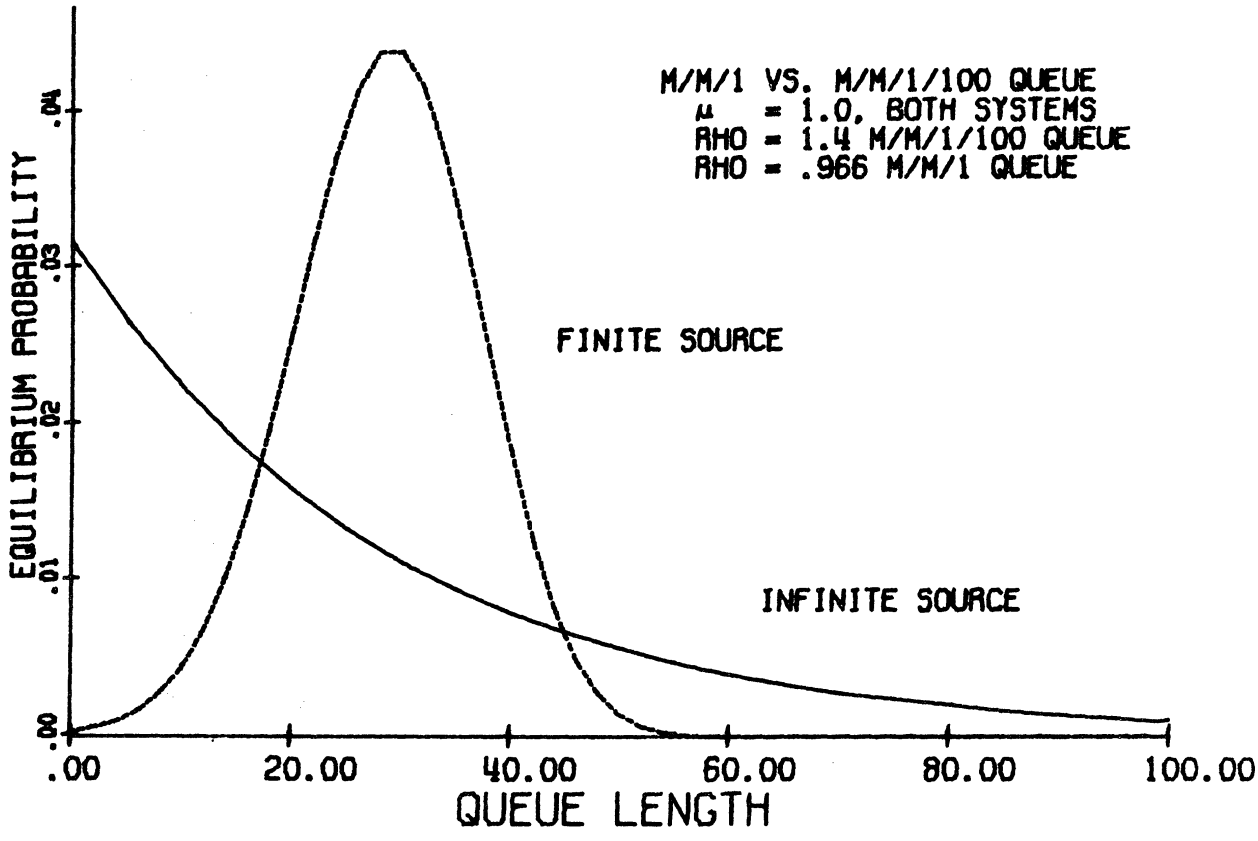

FIG 2-4: QUEUE LENGTH DISTRIBUTIONS

the two models in this case shows that the behavior of the two queueing systems is actually very different.

3. Models for Mass Storage Device Performance

In this section models for the operation of such peripheral devices as drums and disks are considered. Fewer papers have been published in this area than in the processor scheduling field. Perhaps the reason is that the range of reasonable scheduling algorithms is less broad. The goal is almost always to minimize the expected waiting time for requests, or some equivalent measure. There is generally a priori knowledge of the time required to complete a request, making optimal scheduling a practical possibility. This is in marked contrast to processor scheduling models. There the processing time required is at best known as a distribution function. More often, this distribution is assumed to be exponential, indicating that even after serving the customer for a period of time, the server has no additional information about his remaining service time requirement.

We will review several papers, emphasizing the types of systems models and the results obtained.

A paper by Denning[D2] was the first to appear in this area. It is based on work which appeared in his Master's thesis as a Project MAC Technical Report[D4]. The

differences between the paper and the thesis are primarily in the detail in which consideration is given to various assumptions made and positions taken. Denning's paper considers separately the problem of drum (fixed head) and disk (single moveable arm) scheduling. Both devices are assumed to be page-residence devices. This implies that operations are of a fixed length. Two drum scheduling algorithms are considered: FCFS (or random selection) and SATF (shortest access time first). SATF selects for service the page request in the queue for which the access time (rotational latency) is shortest. To do this a queue is kept for each sector on the drum. There are as many sectors as there are pages which can be read or written in one drum revolution. Approximate comparisions are given for the throughput of the FCFS and SATF scheduling algorithms, under the assumption that page read requests arrive uniformly distributed across sectors. The improvement gained by using the SATF algorithm increases with the arrival rate of page requests and the number of sectors on the drum. For a large number of sectors and a fairly heavy load, the improvement is impressive. Denning gives as an example a 64-sector(!) drum with an average of 10 requests outstanding. The drum utilization (actual time a transfer of data is in progress) is 3% for the FCFS scheduling algorithm and 53% for the SATF.

Denning's results are (as he is the first to state)

quite approximate and derived under conditions which limit the generality of their application. For example, Denning derives his results under the assumption that there is a fixed length queue of operations to perform at all times. This is stated as an "average" queue length, but is used as though it were fixed. This makes results difficult to compare with others who assume some random arrival process. If throughput is improved in some way in Denning's model, the load is effectively assumed to increase equivalently in order to maintain the fixed queue length.

At least two papers have appeared which offer improvement on the accuracy of the results obtained by Denning. Coffman[C4] essentially retraces Dennning's work on drum performance, but with considerably more attention to rigor. Coffman assumes Poisson arrivals to each of the drum sector queues, not necessarily at identical rates. He then derives the expected waiting time for arrivals and the queue length distribution generating function. His results cannot be compared directly with Denning's because of the aforementioned difference in their treatment of the arrival process. Coffman does show the same dramatic improvement in performance obtained by using SATF scheduling under heavy loads on a drum with a large number of sectors.

Pinkerton[P4] in his Ph. D thesis covers about the same ground, but his results are derived for discrete time

synchronized to the instants at which a sector boundary is passed by the drum heads. The results are similar to those of Coffman.

In treating disks, Denning ignores the equivalent sector problems which exist, and assumes one page can be written in each disk revolution. He then considers three scheduling algorithms which decide which of the cylinders having outstanding requests one should seek to next. The alternatives considered are FCFS, SSTF (shortest-seek-time-first) and SCAN (moving regularly in one direction across the cylinders, then back). SSTF has the advantage of being the optimal algorithm in terms of throughput under Denning's assumptions. It has the disadvantage that there are some outstanding requests which may never receive service. Since the shortest possible seek time is none at all (don't move the arm), under very heavy loads SSTF may never service any requests other than those for the cylinder on which it has become "stuck".

The SCAN algorithm is shown to be less efficient than SSTF, but it is fairer in the sense that it guarantees service for each cylinder once every "scan" of the arm. Denning's example for disk scheduling shows little improvement (10-20%) for SSTF and SCAN over FCFS. In this case it is partially due to a small disk (30 cylinders as opposed to 200 on the IBM 2314, for example). It is also clear from his

analysis that a large queue of operations (something approaching the number of cylinders) is necessary for SCAN or SSTF to show impressive improvement over FCFS.

Another paper in this area considers the performance of the IBM 2314 direct access storage unit. This device consists of 8 modules (disk packs), each with a single access arm. The 8 packs are served by a single I/O channel in the configuration analyzed. Seeks can be in progress on any number of modules at one time, while data transmission can be in progress on one.

This paper by Abate, et al[A2] treats this system as two separate queueing systems. There is a "seek queue", consisting of eight parallel servers, distributed as the seek time. The queue is actually treated as eight M/G/1 queues, each independent of the other. Expected waiting times are derived for a number of seek time distributions. Numerical inversion techniques for laplace transforms allow arbitrary seek time distribution functions to be analyzed.

The data transmission part of a file operation is treated as an entirely separate queueing system. This system consists of a single server, with a service time distribution equal to the convolution of a 0 to 25 millisecond uniform distribution (representing the rotational latency) and an arbitrary transmission time distribution. The arrival process is Poisson with rate equal to the sum of the arrival

rates at the seek queues. This is again a standard M/G/1 system for which results are easily obtained. The total waiting time for a requested 2314 operation may then be obtained by convolving the seek time wait and transmission time wait distributions.

There are clearly some dependencies which exist between the seek and data transmission queues. Therefore the results are optimistic, since they ignore this interference. The problem does not seem to be given sufficient treatment in the paper. The reader is referred to a book by Kleinrock[K6], which argues for this independent treatment of related queues in a rather different context.

Another paper co-authored by Abate and Dubner[A1] analyzes a drum containing variable length records. This is in contrast to the Denning and Coffman papers, which both assumed fixed length operations. Another interesting aspect of this model is that it assumes a least-rotational-latency scheduling algorithm for the drum. With variable length records this may lead to arbitrarily long waits for unlucky requests which are continually "masked" by a request starting earlier. It is stated in the paper that a manufacturer planned to implement a hardware queuer of this nature, but it seems hard to believe that it would ever gain wide acceptance. The variance in waiting times under a heavy load is exceedingly high for this algorithm, as is shown in the

paper.

As a final paper in this area, we mention a study of disk storage devices by Frank[F4]. Almost the entire paper is devoted to the synthesis of seek time distribution functions under various assumptions. Two particular disk storage devices, the Bryant 4000 and the IBM 2314 are analyzed.

Under various assumptions about the distribution of operation requests over the tracks of the disk, the average seek time is derived. No attempt is made to consider queueing problems except as they affect the length of the queue within which operations may be reordered to reduce seek times.

Frank also discusses several changes in disk storage structure and the effect these changes would have on performance. For instance he suggests a means for reducing rotational latency by placing two read heads, 180^{o} opposed, on each track. This reduces average latency to one quarter of a revolution, instead of one half a revolution. Of course there is also some increase in cost.

4. Memory Management Models

Before the advent of techniques for relocating memory addresses dynamically, memory management was a simple task. Multiprogramming was not generally possible, so all of the memory available was assigned to each user, and as much as he used was swapped in and swapped out[S5]. Base address registers, paging and segmentation hardware made multiprogramming possible and memory management an interesting problem.

There have been two general approaches to hardware relocation techniques: paging and segmentation. It is possible also to combine both. The segmentation approach uses one or more "base" registers so that different parts of the processes addressing space are relocated by different amounts. This approach, with variations, has been followed by Burroughs, CDC and GE. The base registers divide the program into one or more segments, where each segment is of arbitrary length. A given segment must reside in contiguous memory. If the hardware supports only one segment (as in the CDC 6600 and GE 635 machines), then the entire program must be contiguous in memory. The interesting problem in this type of implementation is one of packing segments into memory. Segments are of random length and may grow or shrink in size, as well as be released or allocated. The available

real memory is of fixed size.

In general, the segmentation-only approach does not allow the running of a partially loaded program. Especially in the common case where only a single base register is available, it is fruitless to attempt to run the program unless it is entirely in main memory. The problem then is to decide what programs to load, and when to shuffle memory so as to remove "holes" which have appeared. The problem is treated by Knuth[K9], and by Baskett, et al[B1].

A second approach to the memory allocation problem is through paging, possibly combined with segmentation[A6,D10]. This approach also allows the loading of more than one process at once into memory, and also has two other advantages:

(1) A user's program may be partially loaded, that is to say, not all of it need reside in real memory at once.

(2) The user's program need not be loaded contiguously in main memory, but may be scattered arbitrarily throughout it, in page-sized blocks.

In this case, there are two interesting and related problems. The first is the intra-process memory management problem. This is generally stated as follows: for a given program and number of available real pages, what subset of the virtual pages of the program should be assigned to the

available real pages? A related question asks how many real pages should be assigned to the task, which leads to the "meta-question" of how the set of available real pages should be partitioned.

Attempts at modeling memory management processes seem to have been quite successful. In particular, the analysis of intra-process page management has been a fruitful field of study. Denning[D9] and Belady[B3] were early workers in this field. Both considered various models for program behavior: the distribution of memory references across the virtual addressing space of the process. They then investigated the performance of various replacement algorithms, the decision process which determines which of the pages in real memory will be displaced when it is necessary to retrieve another page from back-up storage.

Denning suggested the working set algorithm, which makes no attempt to hold constant the number of pages of real memory assigned to a process. It rather attempts to keep all the pages which a process has referenced in the last t seconds in real memory. This set of pages is called the working set. Inter-process decisions are made as follows: run as many processes simultaneously as can be kept, with their entire working sets, in real memory, making allowance for variation in the size of these sets.

Belady's suggested algorithm for memory management

is similar to Shemer's. Both attempt to assign the next available page to the process which needs it most (or conversely, to remove the page least needed). In contrast to Denning's position that the page replacement decision should be made primarily within a process, both Shemer and Belady view the set of pages in real memory as a whole.

In Shemer's paper[S6] the decision is made on the basis of a function defined on the set of pages. This function defines the value of each page to the process(es) to which it belongs. The function is computed on the basis of information available from the past history of references and transactions involving the page.

Belady[B4], on the other hand, concentrates on the number of pages allocated to a process, rather than individual pages. He defines a function $V_i(k)$, where k is the number of pages assigned to process i. $V_i(k)$ measures the value of acquiring k real pages to process i. It is monotonically increasing, up to the size of the virtual memory allocated to process i. The page replacement policy is: give a new page to that process i such that $V_i(k+1)-V_i(k)$ is maximum, where k is the current number of real pages assigned to process i.

Other paging algorithms have been investigated (for example see Weizer[W3] or DeMeis[D1]). A review paper by Denning[D8] covers the entire field of memory management,

and considers many of the advantages and disadvantages of using various algorithms, as well as some of the practical implementation problems.

When it comes to measuring the performance of memory management policies, there is quite a bit of variation in the suggested measures to be used. Depending on the constraints applied, some measures may or may not be appropriate. For example, minimizing the number of page faults is a good goal within certain bounds. If that is the only goal, however, it leads to such absurd policies as running all programs which will fit entirely in memory to completion, and never attempting to multiprogram.

The models used to study various paging algorithms do have one feature in common. Given sufficient data about program behavior, one can predict the rate at which page faults will occur in running program. As we shall see, this will be a useful tool in the model development carried out in later chapters.

5. Models for Time-Sharing Terminals

Most analytic models for time-sharing systems have depicted requests for service as arriving from terminals in some exponential distribution. Though there is data which argues that this statistic is not perfect[C9], it seems satisfactory for most purposes. A paper by Denning[D6] however, develops a considerably more complex picture of the arrival and processing of terminal input. The results he derives are approximate, and the assumptions open to question, but the paper is notable for the fact that anything at all can be said about what is a fairly complex model.

The model is based on the MULTICS structure for processing terminal input. There, all incoming characters are routed to a single process which buffers them. When a complete line has been accumulated, it is passed to the process to which the terminal is attached. Denning characterizes the operation of this process which interfaces on a character basis to the terminals (he calls it a PSR - Protected Service Routine). There are several quantities which he derives: The size of the buffers required to accumulate characters, the waiting time of the process to acquire a line, and the number of processes which will be blocked waiting for the PSR to release a line of input to them.

The reader may note that since Scherr's early in-

vestigation of the Project MAC system[S5], and parallel work by Krishnamoorthi and Wood[K9], there have been no serious attempts to model complete time-sharing systems via analytic models. This has been due to the lack of suitable models, and the difficulty in obtaining data to validate these models from more and more complex systems. Difficulties have always existed in representing the variety of time-sharing system resources and their interaction. It is to the development of a modeling structure suitable for such tasks that we now turn.

Chapter III

Mathematical Development

1. Introduction

This chapter contains the mathematical development of the network models which will be used in later chapters to investigate performance measures and the MTS system. The model is a true network model in that it consists of a number of interacting, but independent, queueing systems. The general treatment of such networks is a difficult problem which has received some attention[S2]. However, little in the way of practical results has appeared in the literature. The networks we develop here avoid many of the general problems by their ability to be characterized as time-continuous, discrete-state markov processes. There is nothing inherently limiting in such an approach - by sufficient augmentation of the state space arbitrary queueing networks could be treated as markov processes, though not necessarily solved in any sense of the word.

It will not be at all obvious through much of the chapter that the results are applicable to the analysis of computer systems. The reader is asked to be patient (or skip to the next chapter). Some interpretations of the mathematical structure must be made to see the fruitful

approaches. This work appears in succeeding chapters. The mathematical structure of network models is rich enough so that the number of "free parameters" of a model are essentially unlimited. While this means that we have great flexibility in constucting a particular model, it is also a drawback. The basic model is so pliable that it is difficult to envision all the shapes into which it may be formed.

The markovian nature of the model leads to larger and larger state spaces as more complex models are developed. Even very small models will quickly outstrip the capabilities of a desk calculator. Most practical models lie within easy reach of (exact) computer solution. For very large models, where even this is not practical, approximation techniques become important. This is one of the reasons for our emphasis on such results in this chapter.

The primary references for this chapter are papers by Gordon and Newell[G3] and J.R. Jackson[J1,J2]. The models developed in this chapter are extensions of the work reported in these papers.

2. The Structure of Network Models

One sort of analytic model naturally suited to time-sharing system modeling is the type shown in Fig 3-1. The resources of the time-sharing system - disks, drums, processors, channels, etc. - are represented by the circles of Fig 3-1. Each resource may also have a queue associated with it. This is represented by the rectangle on top of the circle. If a customer is unable to acquire some needed resource, he is represented as waiting in the queue behind that resource. Together we will call the resource and its queue a node of the network. The lines connecting the nodes represent the transitions customers of the network may make. If there is a line coming from the bottom of a circle representing some resource and entering at the top of the queue at another resource, then a customer, when his request for the first resource has been fulfilled, may move to the second resource for his next request. The customers are not represented explicitly in Fig 3-1. They pass from node to node along the transition lines, according to as yet unstated rules. At each node, the customer may spend time in the queue and will spend some time receiving service from the resource. When the customer completes service, he moves instantaneously from the node at which he just completed service to another node.

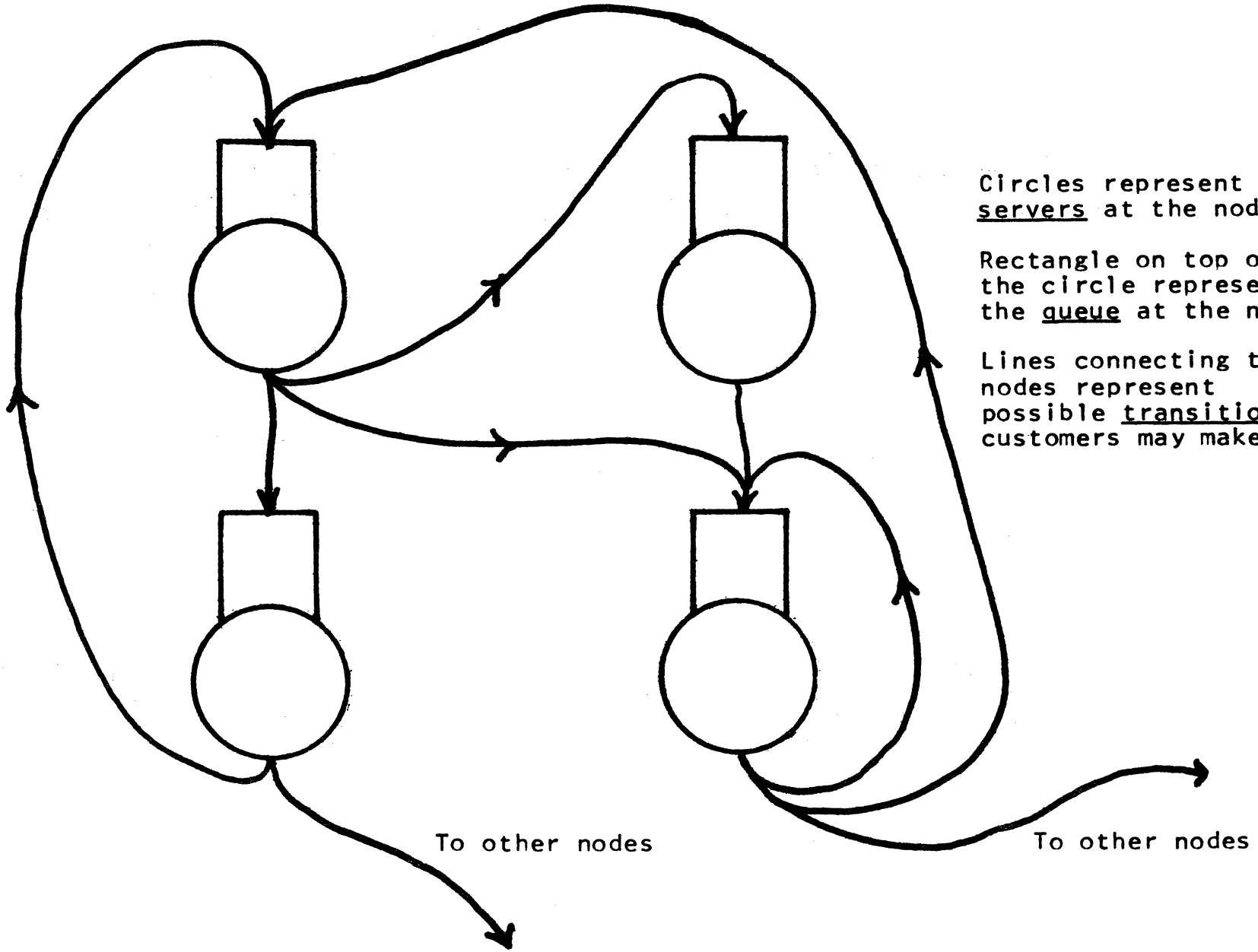

Fig 3-1: General Network Structure

The network is closed in the sense that the number of customers is fixed. New ones do not arrive from outside the network, nor do customers ever exit completely from the network. Exits from the network could be represented by an isolated node which if entered is never exited, or from which the only possible transition is back to same node. Customers entering such a node would effectively be lost to the system.

We have said nothing about the rules which govern the transitions of customers or the service at the nodes. Obviously, we have in mind some sort of stochastic process. We must then specify the following for the network:

(1) The rules governing the transition of customers from node to node.

(2) The rules governing the delays incurred in receiving service at each node.

(3) The rules governing the ordering and servicing of the queue at each node of the network.

(4) The number and characteristics of customers in the network.

The characteristics of customers (part of (4)) will of course be implied by the specification of (1)-(3). Of primary importance, however, is the decision as to whether customers may be of different types or are identical in the statistical sense. For example in order to have two priority

classes of customers we would need to specify how many customers were of each type. Then in (1)-(3), we could specify separately the treatment of each of the priority classes.

We require only that (1)-(4) be specified in terms of the state of the network and not in terms of information external to the network. Thus in the most general case the queueing discipline at each node i might depend on the number of customers queued at node j or on the elapsed service received at node k (a different node) by some customer over the last 10 minutes.

The above structure represents the most general type of network model. As it is stated there is nothing that can be said about such a structure analytically, since it is only partially specified. We will however derive analytical results for a subset of such models which follow certain rules in the specification of (1)-(4) above. This subset will allow us to model quite naturally the structure of large-scale time-sharing systems.

It should be noted that even in this general formulation, there is no means of representing the acquisition of more than one resource simultaneously. This is fairly common in a time-sharing system where for example a process must have acquired both some real memory and a processor to execute a program. Since a customer can only appear at one

node at a time in our network formulation, we cannot represent this phenomenon. This is the most serious drawback of the approach taken here and is considered further in the last section of this chapter.

We now proceed to the mathematical solution of several special cases of network models.

3. Derivation of Steady-State Results

In the network we consider, the customers of the network are assumed to be statistically identical. The transition probabilities are fixed, depending only on the node at which a customer has just received service. That is, the probability that a customer completing service at node i moves next to node j is given by p_{ij}, a constant. The service time distribution at each node is exponential, but the service rate (parameter of the exponential distribution) may vary arbitrarily with the number of customers at node i. That is, the service time distribution at node i is given by:

$$F_i(t) = 1-e^{-\mu_i(n_i)t}$$

where n_i is the number of customers at node i. The queueing discipline at the nodes will be assumed FCFS, though the exponential service rates allow other interpreta-

tions.

Let the number of nodes in the network be M. Let the number of customers distributed in the network be N. We denote the M nodes by $Q_1, Q_2, \ldots, Q_M$. The number of customers in service and in queue at node Q_i will be denoted by n_i. The service rate at Q_i varies arbitrarily with n_i and is denoted by $\mu_i(n_i)$. The probability that a customer completing service at Q_i moves next to Q_j for service is given by p_{ij}. Of course we require:

$$\sum_{j=1}^{j=M} p_{ij} = 1 \qquad \text{for } i=1,2,\ldots,M$$

Fig 3-2 illustrates the basic structure and parameters of such a network.

The matrix of transition probabilities p_{ij} is denoted P. If the state of the network is the M-tuple $(n_1, n_2, \ldots, n_M)$, then $(n_1, n_2, \ldots, n_M)$ may be seen to be a continuous-time discrete-state markov process. The state space of the process consists of all M-tuples $(n_1, n_2, \ldots, n_M)$ such that:

$$\sum_{i=1}^{i=M} n_i = N \quad \text{and} \quad n_i \geq 0 \text{ for } i=1,2,\ldots,M$$

We now give conditions under which the network will possess an equilibrium distribution of customers among the M nodes. We first require that the matrix P, viewed as the transition matrix of a discrete markov chain, be

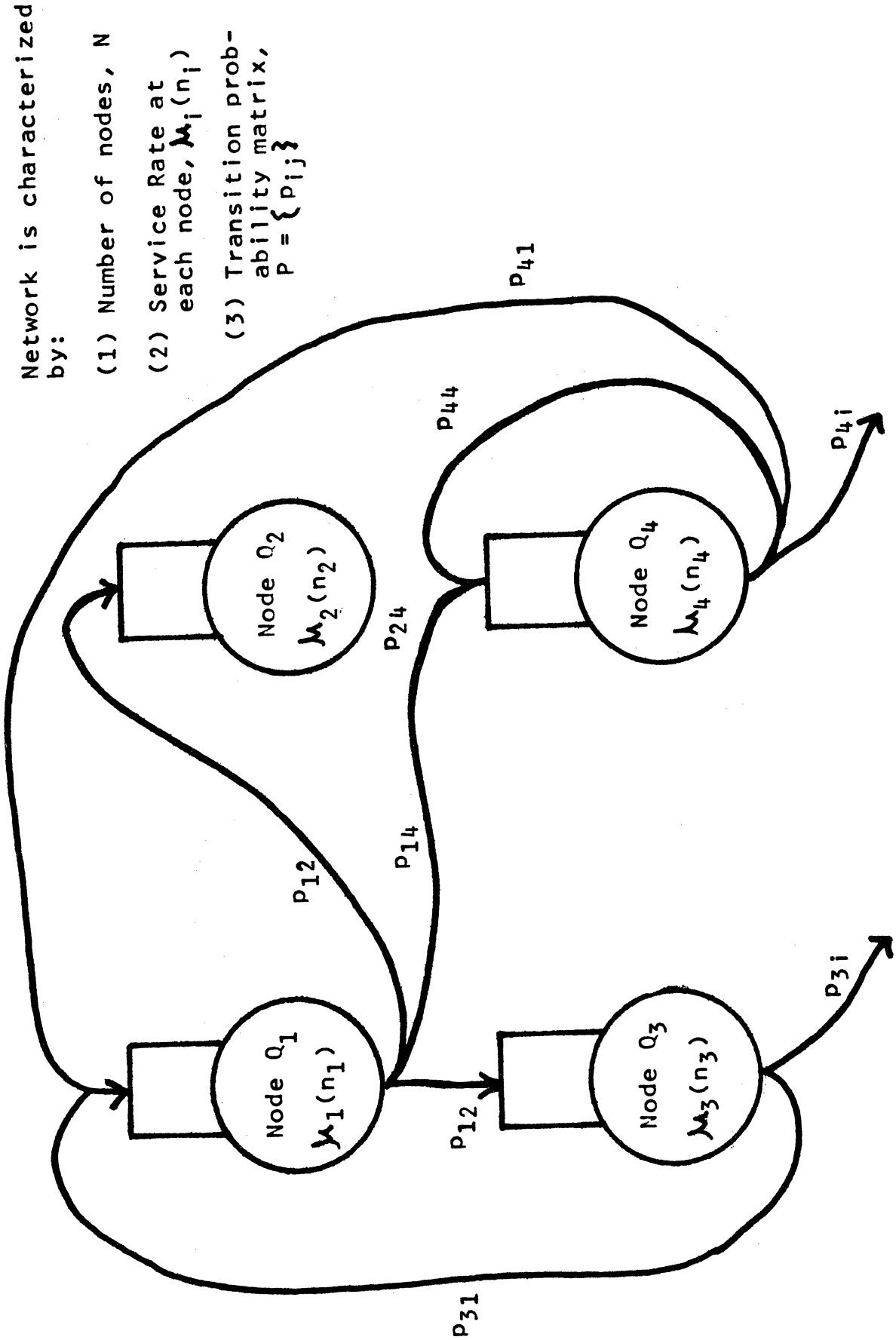

Fig 3-2: Markovian Network Structure

irreducible. For a finite chain this simply implies that it is possible to reach any state (node) from any other state, in a sufficient number of steps. Secondly we require that:

$$\mu_i(x) > 0 \text{ for all } x > 0 \qquad i=1,2,\ldots,M$$

This condition implies that no node "turns off" at some point allowing customers to gather behind it indefinitely. With these two conditions, it is clear that the process possesses an equilibrium solution. The state space is finite, it is possible to reach any state in the network from any other state, and therefore all states communicate[P1, p.276].

Let us denote the probability of being in state $(n_1,n_2,\ldots,n_M)$ in equilibrium by $p(n_1,n_2,\ldots,n_M)$.

We can now write the equilibrium balance equations for the network. These equations are based on the principle that the rate of transition into a state must be equal to the rate of transition out of that state if the system is in equilibrium.

$$\{\sum_{i=1}^{i=M} \mu_i(n_i)\}p(n_1,n_2,\ldots,n_M) = \sum_{i=1}^{i=M}\sum_{k=1}^{k=M} \mu(n_i+1)p_{ik}p(n_1,\ldots,n_k-1,\ldots,n_i+1,\ldots,n_M) \tag{4}$$

where we require $\mu_i(x) = 0$ for $x \leq 0$.

The left-hand side simply gives the transition rate

out of the state $(n_1,n_2,\ldots,n_M)$ as the sum of the service rates at the individual nodes. The right-hand side sums the transition rates into the state $(n_1,n_2,\ldots,n_M)$ over all states from which it is possible to reach $(n_1,n_2,\ldots,n_M)$ in one transition.

Let the state space of the network be denoted by S. The cardinality of this state space is:

$$\binom{N+M-1}{M-1}$$

The equilibrium equations if used directly, yield a system of linear equations with cardinality equal to the cardinality of S. For N=50 and M=5, for example, this is 293,046. The following reduction will allow the determination of the equilibrium probability distribution in a simpler fashion. Define:

$$f_i(x) = 1 \qquad \text{for } x = 0$$

$$f_i(x) = \mu_i(x)\mu_i(x-1) \qquad \text{for } x > 0$$

We then make the following change of variable in the balance equations:

$$p(n_1,n_2,\ldots,n_M) = \{\prod_{i=1}^{i=M} f_i^{-1}(n_i)\}Q(n_1,n_2,\ldots,n_M) \quad (2)$$

The transformed equilibrium equations then become:

$$\{\sum_{i=1}^{i=M} \mu_i(n_i)\}Q(n_1,n_2,\ldots,n_M) = \quad (3)$$

$$\sum_{i=1}^{i=M}\sum_{k=1}^{k=M} \mu_k(n_k)p_{ik}Q(n_1,\ldots,n_k-1,\ldots,n_i+1,\ldots,n_M)$$

Now let us assume that it is possible to express $Q(n_1,n_2,\ldots,n_M)$ as a product of powers of some set of M constants $x_1,x_2,\ldots,x_M$ as follows:

$$Q(n_1,n_2,\ldots,n_M) = \{\prod_{i=1}^{i=M} x_i^{n_i}\}\cdot C \quad (4)$$

Where C is some constant of normalization. Making this substitution into the modified balance equations (3), we have:

$$\sum_{i=1}^{i=M} \mu_i(n_i) = \sum_{i=1}^{i=M}\sum_{k=1}^{k=M} \mu_k(n_k)p_{ik}(x_i/x_k)$$

These equations may be rewritten as:

$$\sum_{i=1}^{i=M} \mu_i(n_i)\{1 - \sum_{k=1}^{k=M} p_{ki}(x_k/x_i)\} = 0$$

Note that we still have one equation for every possible state of the system. Since $\mu_i(N)$ is greater than 0, by hypothesis, and having N customers in queue and in service at Q_i is a possible state of the system, the term in curly brackets (which is independent of the n_i) must be identically 0. This yields the system of M linear equations:

$$\sum_{k=1}^{k=M} p_{ki}(x_k/x_i) = 1 \qquad i=1,2,\ldots,M \tag{4.1}$$

One solution of these equations is the equilibrium probability distribution of the markov chain defined by the matrix P [P1, p.249], which by the irreducibility of P is known to exist.

Substituting this result back into the original balance equations(1), we find:

$$p(n_1,n_2,\ldots,n_M) = \{\prod_{i=1}^{i=M} \mu_i(n_i)x_i^{n_i}\}\cdot G \tag{5}$$

G is a normalization factor introduced by the arbitrary constant in (4). It may be computed from the requirement that the equilibrium probabilities sum to 1.

4. Limiting Queue Length Distribution

If either the number of nodes in the network is large, or the number of customers is large, the state space of the network is so great as to make the calculation of the steady-state queue length distributions by the method of the previous section prohibitively expensive. For example, a 5-node network with N=75 required about one minute of IBM 360/67 time to compute the equilibrium queue length distributions.

We derive in this section the limiting queue length distributions in an unbalanced system as N becomes

arbitrarily large. An unbalanced system is informally defined as one in which a bottleneck exists limiting the behavior of the entire network. More precisely, we define a network to be unbalanced if there exists an i such that:

$$x_i/\mu_i > x_j/\mu_j \quad \text{for } j=1,2,\ldots,M \qquad (5.1)$$

$$\text{where } x_i/\mu_i = \lim_{n_i \to \infty} x_i/\mu_i(n_i)$$

We have implicitly assumed that the limit in (5.1) exists for all i=1,2,...,M. Note that for a node with $\mu_i(n_i) = kn_i$ for some constant k, the limit of x_i/μ_i will be 0. A necessary condition for the limit to exist is that for some n>0 and some k>0,

$$\mu_i(n_i) \geq k \quad \text{for all } n_i > n$$

This effectively prohibits a node from slowly "shutting off" as the queue gets longer and longer. Thus the conditions required for the limit to exist are slightly stronger than those of the previous section, in which we simply required $\mu_i(n_i) > 0$ for all $n_i > 0$.

We will call the node possessing the largest x_i/μ_i the limiting node or limiting resource of the network. In addition to our primary purpose of obtaining approximate formulas for the queue length distributions in this section, we will also gain some insight into the effect of the limiting resource on the system. In particular, the existence of

the limiting resource will allow us to express a number of the network statistics as functions of N, the number of customers in the network.

Let us rename the nodes of the network in such a way that the limiting node is Q_1. We can then write the normalization constant of (5) as:

$$G = \sum_S \{ \prod_{i=1}^{i=M} x_i^{n_i} / f_i(n_i) \}$$

where S is the set of all M-tuples $(n_1, n_2, \ldots, n_M)$ such that:

$$\sum_{i=1}^{i=M} n_i = N$$

We may also write G then as follows:

$$G = \sum_{S'} \{ \prod_{i=1}^{i=M} x_i^{n_i} / f_i(n_i) \} \{ x_1^{s'} / f_1(s') \} \qquad (6)$$

where S' is the set of all M-tuples $(n_1, n_2, \ldots, n_M)$ such that:

$$\sum_{i=2}^{i=M} n_i \leq N$$

and:

$$s' = N - \sum_{i=2}^{i=M} n_i$$

To arrive at (6) we have simply factored the terms involving x_1 and f_1 from the original expression for G.

Now as we let $N \to \infty$, we see that (6) must approach

some finite limit. To see this, we rewrite (6) again as:

$$\lim_{N\to\infty} G = F_1 \cdot \prod_{i=2}^{i=M} \{ \sum_{n_i=0}^{n_i=\infty} [x_i^{n_i}/f_i(n_i)] \} \qquad (7)$$

where we define F_1 as:

$$F_1 = \lim_{n\to\infty} \frac{x_1^n}{f_1(n)}$$

which by our choice of the normalization of the x-vector exists and is finite.

The second term of (7) is finite since for sufficiently large j:

$$x_i(j)/\mu_i(j) < 1 \qquad i \neq 1$$

again by our choice of a normalization for the x vector. The sum then is finite since we have the relation:

$$x_i^{n_i}/f_i(n_i) = (x_i/\mu_i(n_i)) \cdot (x_i^{n_i-1}/f_i(n_i-1))$$

making the sum a power series beyond some point.

Consider next the limit of the probability of being in a given state, $p(n_1,n_2,\ldots,n_M)$:

$$\lim_{N\to\infty} p(n_1,n_2,\ldots,n_M) = \{ \prod_{i=1}^{i=M} [x_i^{n_i}/f_i(n_i)] \}/G \qquad (8)$$

This is clearly zero, unless there are a finite number of customers at $Q_2,Q_3,\ldots,Q_M$. But if N is infinite, then (8) is non-zero only if n_1 is infinite. Let us therefore

consider the marginal probability $p(n_2,n_3,\ldots,n_M)$ which is given by:

$$p(n_2,n_3,\ldots,n_M) = \sum_{n_1=0}^{n_1=\infty} \{\prod_{j=1}^{j=M} [x_i^{n_i}/f_i(n_i)]\}/G \qquad (9)$$

Taking the limit of (9) as N becomes infinite, we find, using (7):

$$p(n_2,n_3,\ldots,n_M) = \frac{\prod_{i=2}^{i=M} [x_i^{n_i}/f_i(n_i)]}{\prod_{i=2}^{i=M}\{\sum_{n_i=0}^{n_i=\infty} [x_i^{n_i}/f_i(n_i)]\}}$$

The fact that such a result exists is of interest. It is rather surprising that as long as a largest x_i/μ_i exists, there is a steady-state limiting distribution for customers at nodes other than Q_i. This implies that, asymptotically, all additional customers added to the network appear at node Q_i, the limiting node. They are not "distributed" in some probabilistic fashion among the nodes.

The question of how fast the limiting distributions are approached is a difficult one. The rate of convergence depends on the size of the network and the spread in values of the x/μ-vector. For example, for a 4-node network with $x/\mu = (1.00,.995,.990,.800)$, the expectation of the queue length distributions had converged to within .1 of the limiting value for N = 130. We will return to the question

of rate of convergence in the section on network saturation.

5. Cycle Times in the Network

By our requirement that the matrix P be ergodic, and that $\mu_i(n_i) > 0$ for $n_i > 0$, we have ensured that every customer will visit every node infinitely often, given sufficient time. We define the cycle time of a customer in the network to be the time required to pass from a given node back to that node. We measure the time from the moment the customer leaves the node (completes service) until he returns to that node or the queue at that node. There will be a cycle time distribution associated with each node of the network. We denote the cycle time density function associated with node Q_i by $c_i(t)$. This density function is given by:

$$c_i(t) = \sum_S b_s(t)$$

where S is the set of all possible paths from Q_i back to Q_i and $b_s(t)$ is the probability of traversing a given path s in time t. The set S will in general be infinite, since there may be loops in the network which do not include Q_i. These loops may be traversed an arbitrary number of times before returning to Q_i.

We do not attempt to derive $c_i(t)$, but instead turn

our efforts to its expectation, which we denote by C_i,

$$C_i = E[\sum_S b_s(t)] \qquad (10)$$

We now argue that the expected time spent in traversing a path is the sum of the expectations of the time spent at each node on the path. That is, if we denote by s_i, a particular path from Q_i to Q_i, then the expected time to traverse that path, $E[s_i]$, is given by:

$$E(s_i) = \sum_{j=1}^{j=M} r_j w_j(N) \qquad (11)$$

where r_j is the number of times node Q_j appears on the path S_i, and w_j is the expected waiting time (in queue and in service) at node j, in equilibrium. The only way this can fail to be the case is if there are possible effects of serial correlations in the waiting times along the path. That this does not occur for the case of exponential servers is shown by Takacs[T1] for a similar situation.

From (10) and (11) then we have:

$$C_i = \sum_{j=1}^{j=M} {}_{j\neq i}\, r_{ij} w_j(N) \qquad (12)$$

where r_{ij} is the expected number of times node j will be entered in the cycle from Q_i to Q_i. This interchanging of expectation and summation is valid because of the markovian nature of the transitions in the network. The r_{ij} are easily computed from the elementary theory of markov

chains (for example, see Parzen[P1, p. 242]). Let us assume we are interested in C_M. Then the r_{Mj} are given by the solution of the following set of linear equations:

$$\sum_{i=1}^{i=M-1} p_{ji}(r_{Mj}/r_{Mi}) = 1 \qquad j=1,2,\ldots,M-1$$

Note that the r_{Mj} are a solution to the set of equations (4.1) which determined the x-vector in the derivation of the equilibrium distribution for the network.

The expected waiting time at node Q_j in equilibrium, $w_j(N)$, is computed from the steady-state queue length distributions as:

$$w_j(N) = \sum_{i=1}^{i=N} p_j(i)i/\mu_j(i)$$

The r_{ij} in (12) are constant for all N. The $w_j(N)$ in (12), for sufficiently large N, approach a limit which we denote w_j. This implies that we can write an asymptotic cycle time relationship for an unbalanced system. Let Q_k be the limiting node. Then we have:

$$C_k \cong \sum_{j=1}^{j=M} {}_{j\neq k}\, r_{kj}w_j$$

$$C_i \cong \{\sum_{j=1}^{j=M} {}_{\substack{j\neq k\\ j\neq i}}\, r_{ij}w_j\}+(N-K-1)(1/\mu_k)r_{ik} \qquad (13)$$

The first term in (13) is the asymptotic time spent at nodes $1,2,\ldots,i-1,i+1,\ldots,k-1,k+1,\ldots,M$ in a cycle from node i to node i. By the limits on the distributions of

$p_j(n_j)$ at these nodes, it approaches some constant. We define K as:

$$K = \lim_{N\to\infty} \sum_{j=1}^{j=M} {}_{j\neq k} \sum_{n=1}^{n=N} np_j(n) \qquad (13.1)$$

which is the expected number of customers "absorbed" by nodes other than Q_k as N becomes large. The limit, of course, exists by our assumption that the system is unbalanced. The cycle time for nodes other than the limiting node then becomes linear and is essentially:

$$C_i = K' + Nr_{ik}/\mu_k \qquad i\neq k$$

for some appropriate constant K'. C_k of course approaches some finite limit as $N \to \infty$.

6. Saturation for a Network

We now turn to a more detailed investigation of the performance of the network near the point at which the asymptotic formulas become valid. As mentioned previously, this point occurs as the service rate at the limiting resource approaches its maximum. As soon as it has reached this maximum, it can no longer "eject" customers at a faster rate, and it acts to constrict the flow of customers in the entire network. The analogy of flows in networks, from the work of Ford and Fulkerson[F5], is a rather good one. If one

views the network as a "flow graph" with the flow capacities being the service rates at the nodes, the asymptotic results presented in this chapter seem quite natural.

We will define here a saturation point for a network. We are interested more in gaining insight into the capacity of a network to support customers than in rigorously determining the point at which the asymptotic results of the previous sections become valid. The definition we choose is motivated by a definition of saturation used by Kleinrock[K3] in a simpler context, which was reviewed in Chapter II[p. 12].

Let Q_k be the limiting node and consider C_k, the expected cycle time of a customer from Q_k back to Q_k. From the previous section we have:

$$\lim_{N\to\infty} C_k = \sum_{j=1}^{j=M} {}_{j\neq k}\, r_{kj} w_j$$

which is finite. Then a customer exiting node Q_k will be "lost" to the system for a period of time averaging C_k. In addition, we know that Q_k will eject customers at an average rate μ_k, its limiting service rate. Thus in the period while a customer is cycling back to Q_k, Q_k will have served an average of $C_k \cdot \mu_k$ customers. Let us define:

$$N_s = C_k \cdot \mu_k + 1$$

We can argue then that Q_k has a "capacity" of N_s customers. Potentially it could serve this number without delay, since they arrive at a rate exactly equal to the rate at which they are served. We call N_s the saturation point for the network.

Under fairly general conditions, the network exhibits two general sorts of behavior, one above and one below the saturation point. For $N \ll N_s$, there is relatively little conflict at the nodes, since the flow into each node is less than its capacity. Therefore, statistics such as the cycle time expectations rise slowly with increasing N. For $N \gg N_s$, the asymptotic relations become valid, and the cycle time rises linearly with N for nodes other than Q_k, as shown by (13). For Q_k, of course, the cycle time becomes constant.

The behavior of the network in the region near N_s depends on two factors:

(1) The degree of imbalance in the system.

(2) The regularity of the $\mu_i(n_i)$.

If the system is nearly balanced, the network approaches its asymptotic queue length (and cycle time) distributions much more slowly than in a more unbalanced network. This is because a large number of customers are "absorbed" by the other nodes in the network. The rest of the network can at best eject customers back to the

saturated node just slightly faster than the saturated node can service them. Thus the node(s), which have x_i/μ_i values slightly less than that of the saturated node, must have long queues to guarantee that they are operating at near full utilization before they can drive the saturated node to its limiting behavior. An example of this phenomenon is given in the next chapter. It may also be helpful to return to equation (13.1) noting that the the K defined there is exactly N_s, the saturation point for the network.

The dependence on (2), the regularity of the $\mu_i(n_i)$, is obvious. All the asymptotic relations were derived in terms of the limits of the $\mu_i(n_i)$. If these service rates do not become stable until n_i is very large or approach the limit in some unusual fashion, the saturation point may be meaningless. For example, suppose we have a network in which Q_k is the limiting node and:

$$\mu_k(i) = 2000 \qquad i \leq 1000$$
$$\mu_k(i) = 1 \qquad i > 1000$$

In this case, there will be a great change in the behavior of the network as the number of customers at Q_k passes 1000. Q_k essentially "shuts off" if its queue ever gets that long, and at that point limits the network behavior. But the network may have a very different behavior below N=1000. In fact one can easily produce a network whose

computed saturation point in this case would be at $N \ll 1000$, but which did not begin to exhibit asymptotic behavior until $N \gg 1000$. As a rule of thumb we argue that the asymptotic service rate of the limiting node should be achieved for $N \leq N_s$ for the saturation point measure to be valid.

7. Extensions to the Model

The network model for which we have just derived analytic results is greatly simplified from the general model introduced in the first section of this chapter. The intent of the simplifications was to achieve a model which could be treated as a time-continuous markovian process with a reasonable state space. In this section we consider some extensions to this model, as well as some alternative structures which might be used.

Markovian networks have the theoretically attractive feature that by simply augmenting the state space, one can represent more and more complex behavior. Increasing the state space size however, has serious practical implications. The state space with which we have been working is already large, and capricious increases in the cardinality of that space will quickly render it incomputable by the methods of this chapter. Therefore, the possibilities for extension treated here must be viewed in the light of both

theoretical and practical applicability. Unfortunately, some of them will prove to be of little practical value.

Exponential Service Time Distributions

Though the service rate at each node may vary with the number of customers there, the service time distribution for a fixed number of customers is exponential. It is possible, however, to approximate arbitrary distributions within this framework. To do this it is necessary to represent a given service facility, with non-exponential service time density, by more than one node in the network.

Suppose we wish to represent in the network a service facility which has a single server, S, with a service time distribution function B(t). Let the moments of B(t) be denoted $B_1, B_2, \ldots$ A first approximation to B(t) would of course be to represent S by an exponential server with distribution function $1-e^{-t/B_1}$. This distribution function has moments $B_1, B_1^2, B_1^3 \ldots$ If this is not a satisfactory approximation, we may instead represent S by a <u>set</u> of exponential servers in series and parallel. Fig 3-3 illustrates such an approximation. All the nodes inside the box of dashes, taken together, represent the approximation to the server S. We call this approximation S'. We assume the service rates at each of the six nodes in S' are constant, and are given by the symbols $\mu_1, \mu_2, \ldots, \mu_6$ inside the

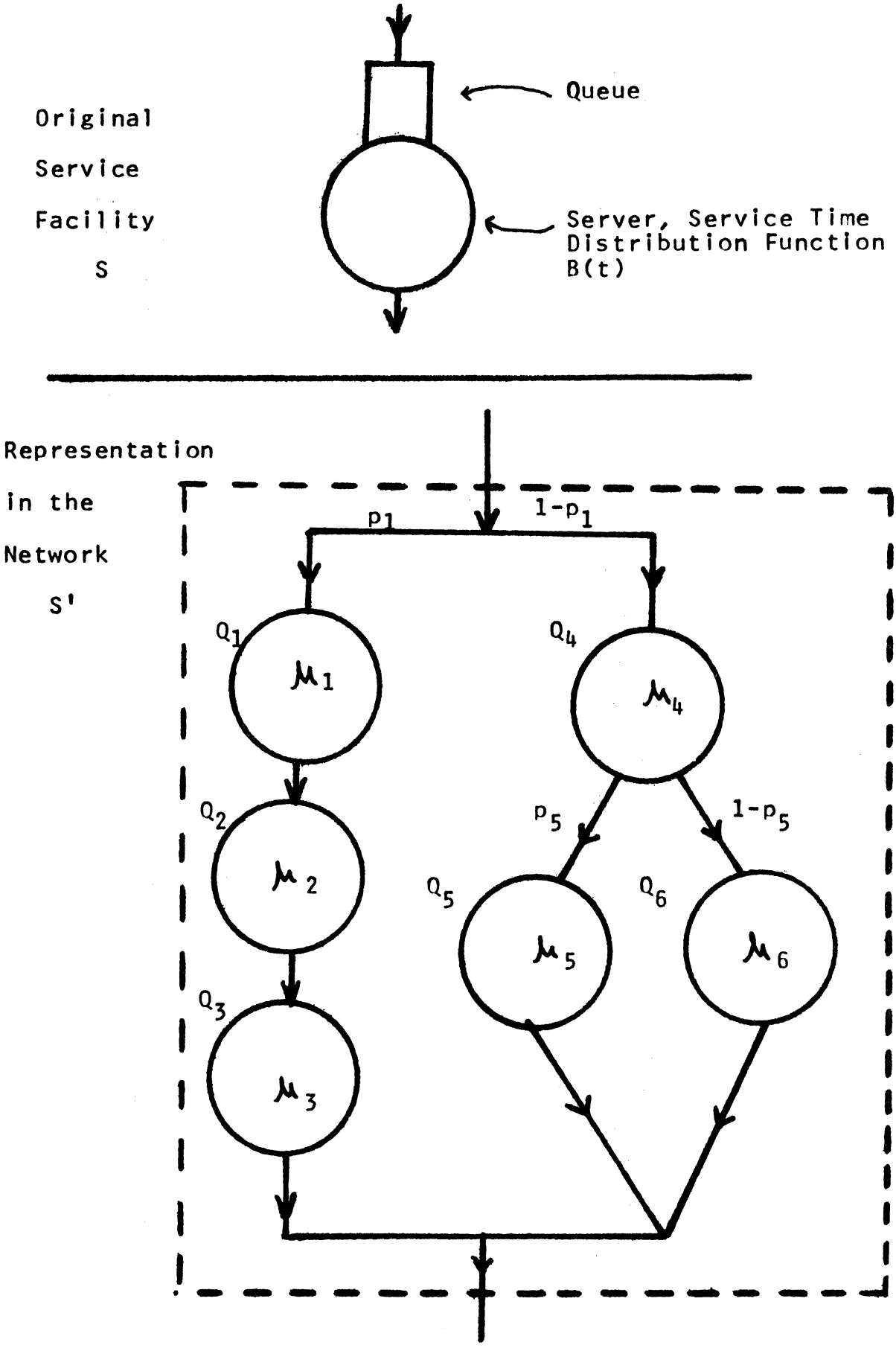

Fig 3-3: Approximation Network

circles representing the nodes of S'. p_1 is the probability that a customer enters node 1, $1-p_1$ the probability that he enters node 4, and so forth. The Laplace transform for the service time distribution of S', which we denote B'(s), is given by:

$$B'(s) = \frac{p_1\mu_1\mu_2\mu_3}{(\mu_1-s)(\mu_2-s)(\mu_3-s)} + \frac{(1-p_1)\mu_4}{\mu_4-s}\left\{\frac{p_5\mu_5}{\mu_5-s} + \frac{(1-p_5)\mu_6}{\mu_6-s}\right\}$$

The moments of B'(t) can easily be computed from B'(s), and one can quickly see that it is not an exponential distribution. The question that arises is how one constructs a sub-network of nodes, S', having given characteristics. How close an approximation to a given service time distribution B(t) can we formulate with a network of exponential servers? The Laplace transform of the service time distribution of the most general sub-network which can be constructed, assuming a single server with constant service rate at each node, will be of the form:

$$B'(s) = \sum_{i=1}^{i=M} p_i\left\{\prod_{j=1}^{j=n_i} (\mu_i(j)/(s-\mu_i(j)))\right\} \qquad (14)$$

This is the Laplace transform of the service time density of a weighted mixture of series of exponential servers. The general form of this sort of sub-network is illustrated in Fig 3-4. The restrictions on (14) are as follows:

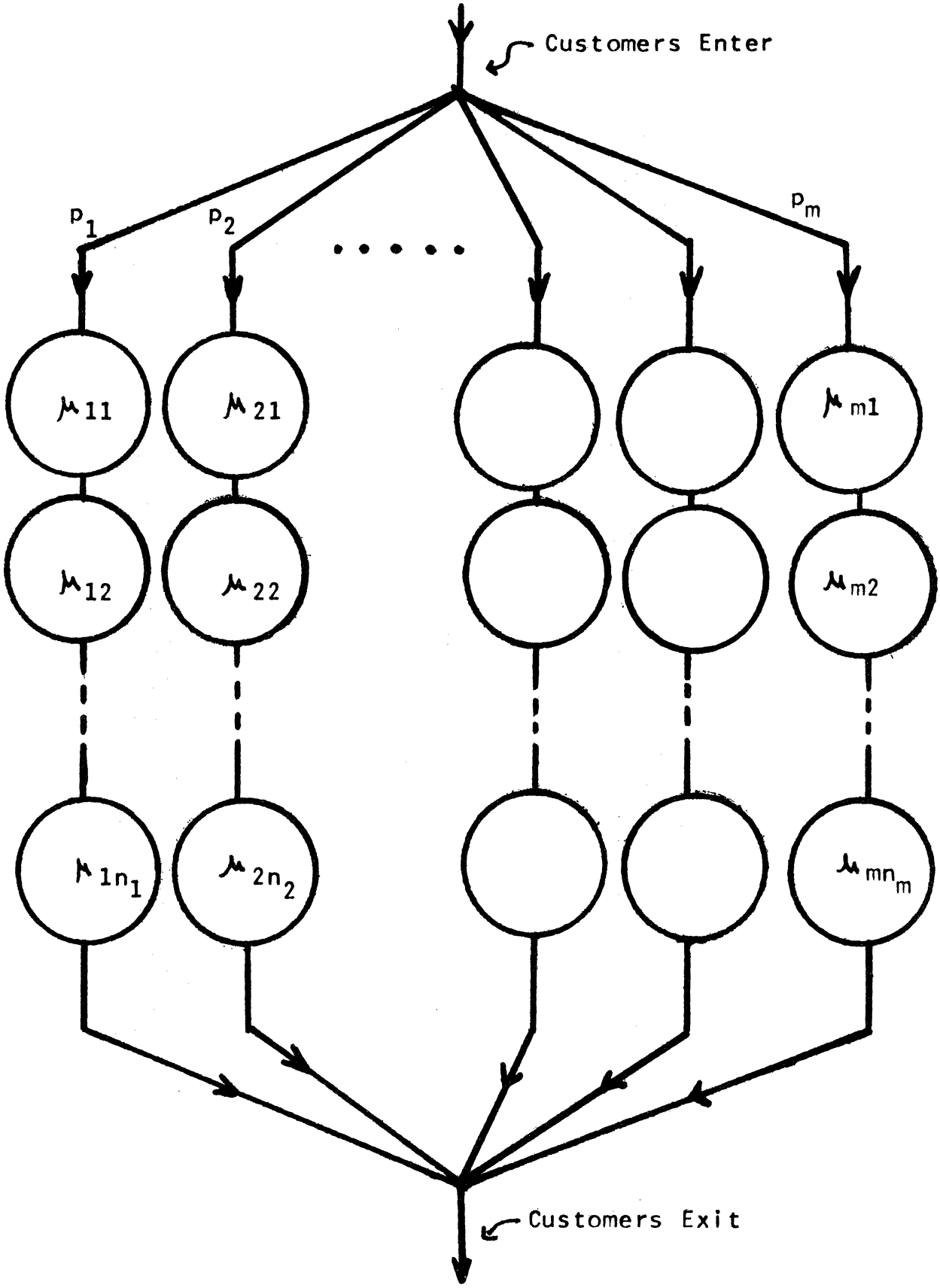

Fig 3-4: General Approximation Network

(1) $\sum_{i=1}^{i=M} p_i = 1$

(2) $\mu_i(j) > 0$ for $i=1,2,\ldots,m$; $j=1,2,\ldots,n_i$

(3) n_i a positive integer for $i = 1,2,\ldots,M$

We need not consider any more complicated structures than the one illustrated in Fig 3-4, since any other structure (such as that of Fig 3-3) can be represented in the form of Fig 3-4.

There are a number of questions we can now ask more precisely. First, can we choose parameters for (14) in such a way that $B'(s) \cong B(s)$ for any $B(s)$ which is the Laplace transform of a probability distribution? It seems reasonable that this should be so, but it is difficult to give either an existence or constructive proof. A constructive proof might be of practical interest.

A somewhat simpler approach is to attempt to construct a sub-network whose service time distribution approximates the first x moments of B(t), for some desired x. This has been done for $x \leq 3$, without any general pattern or algorithm becoming apparent. It is clear, however, that any general solution will require arbitrarily large numbers of nodes, making it of little practical value. Even to approximate the mean and variance of a deterministic distribution requires an infinite number of nodes; one must construct a single series of exponential servers, each with

parameter M/B_1, where M is the number of nodes in the string, and take the limit as M goes to infinity. In fact this is the general method for approximating the first two moments of a distribution whose variance is less than the variance of an exponential distribution with the same mean. In the other case ($B_2 > 1/B_1$), two nodes in parallel (a hyper-exponential distribution) will suffice.

It is felt that for most applications matching the first two moments should be satisfactory.

Independence of Nodes and Related Problems

There are certain restrictions imposed by the assumption that the nodes of the network operate independently. We do not allow the situation (queue length, for example) at one node to affect the behavior of any other node. This might be desirable in some modeling situations. Consider, for example, a disk and drum connected to a time-sharing system through a single channel. If the drum has priority over the disk, then we might like to represent them as separate nodes in the network, but have the service rate at the disk node become 0 whenever the drum node was busy.

Another extension of interest along the same line is the generalization of the transition structure. One might want to allow the transition pattern to change with the state of the system. As an example, consider a paged time-

sharing system. As the CPU queue length increases, more and more processes must share the available real memory. This may mean that the probability of a page fault in a process will increase. So if we represent the CPU as a node in the network, and a page fault as causing a transition to the drum node, we would want this transition probability to be a function of the CPU queue length -or perhaps of even more information about the state of the network.

Both of these problems fall in the class of those which can be solved by a straightforward - and impractical - augmentation of the state space. It would not destroy the markovian nature of the network to make the service rates at the nodes and the transition probabilities between nodes functions of the state of the network $(n_1, n_2, \ldots, n_M)$. We could then easily incorporate phenomena such as the two examples mentioned above into a network model.

With minor changes in notation, the balance equations for the network remain nearly the same, even with these changes in the treatment of service rates and transitions. Unfortunately, we can go no farther than the balance equations, since we now have $(M+1)M$ free parameters for every state of the network: M^2 transition probabilities and M service rates. Without solving the balance equations directly there is no hope for deriving analytic results for a network cast in this generality.

Setting our sights somewhat lower, there are some possibilities for less sweeping generalizations. One involves simply making the transition probabilities a function of the number of customers at the node, in the same manner as the service rates. That is, instead of the p_{ij} being constant, we would have N sets of M transition probabilities for each node. It is not too difficult to generalize the derivation of III.3 to this case, allowing us to compute the steady-state queue length distributions. The results for limiting queue length distributions, balance and saturation do not follow, however. Therefore it was not felt that it was worthwhile to present these results, which offered numbers perhaps, but very little insight.

Identical Customers

There are a number of reasons for wishing to represent non-homogeneous customers in the network. For instance, in modeling a computer system which served both terminal and batch users, we might wish to represent them as two separate classes of customers in the model. One would expect them to have different characteristics.

If one is interested in the performance of a system as a whole, this defect may not be too important, as long as a "virtual customer" can be synthesized. This virtual customer must have the weighted characteristics of the dif-

ferent types of actual customers. This concept is far from new, and has been used for example by Denning in modeling time-sharing system terminal users[D5]. With this approach it is not possible to study the treatment any particular class of actual customers receives from the system, for no actual customers are represented. This limits the kinds of questions we can ask of the model, and expect reasonable answers. Predicted queue lengths, resource utilization and statistics of this sort should be accurately predicted. On the other hand, a measure such as expected response time may be meaningless, since the "expected" customer is a virtual customer, who may not in fact resemble any of the actual system customers. For example, if half of the users of a system are "batch" users and the other half are "terminal" users, the virtual users would be, in a sense, half one and half the other - and not really like either.

There seems to be no reasonable way to extend the markovian network models of this chapter to allow the direct representation of non-homogeneous customers.

Lumped Resources

There are two problems with even the more general network models which make it difficult to represent the allocation and use of storage or real memory.

The first problem, which has already been mentioned

is that network customers cannot acquire, or be represented as needing, more than one resource. In fact a process in an actual time-sharing system may have real memory allocated to it, be using a processor, and at the same time have a disk operation in progress. There is no way to represent this behavior in a network model short of allowing customers to split into several customers. Though this splitting of customers is a possibility[J1], the result of the branching would be a set of independent customers, and there would be no way to tie them together again when the simultaneously allocated resources were released.

The second problem is that resources (nodes) in the network are discrete entities. They are allocated to customers in an all-or-none fashion. There is no way to represent the allocation of 10% of a resource (capacity of the node) to a customer. This of course would be the natural way to represent real memory allocation, where customers may be allocated 10 pages of a real memory which has a capacity of 100 pages. Another customer may be allocated 30 pages of the memory at the same time. This concept of a resource being allocated in unequal portions to the system customers is common to simulation languages such as GPSS, but seems to have received little attention in the field of queueing theory. It is certainly a problem which it would be worthwhile to attack.

Chapter IV

Performance and Load Measures

1. Modeling Goals

In this chapter we consider the goals that we are attempting to reach in developing an analytic model of a time-sharing system. We have already said we are interested in an absolute model rather than a relative one. This implies the goals of our modeling effort are to predict the actual performance of the system being modeled. We seek to make statements such as "the CPU utilization will be X%" rather than statements such as "the CPU utilization under structure A will be 10% greater than the utilization under structure B." We do not deny that "insight" is an important goal - and we certainly hope to acquire some insight along the way - but we hope also that we can achieve another modeling goal: predicting the performance of a time-sharing system under given conditions, without having to construct the time-sharing system in question and operate it under the specified conditions.

We will assume we are given the structure, or perhaps several alternative structures, for the time-sharing system to be modeled. We wish to develop an analytic model of the system and predict its performance under various con-

ditions.

In the mathematical development of Chapter III, we referred to customers of the network. In talking about time-sharing systems we have been consistently using the term user. The user's "representative" inside the time-sharing system, which carries out his requests, is commonly referred to as a process. Since there is generally a one-to-one correspondance between users and processes, we will hereafter use these terms interchangably, employing the one which seems most appropriate at the time. Users will type in requests, processes will be allocated CPU time. In the network models we develop, both the user and his process will be represented as customers, so actually all three of these terms should be considered equivalent, unless otherwise mentioned. We do not consider here the case where one user may initiate more than one process.

The conditions under which the system operates are generally described by specifying the behavior of the system users, i.e. - the requests they make of the system. Together, it seems reasonable to call the collective demands the users make of the system the load. The question of how one defines and measures the load is an important one which has received relatively little attention. The measure of load most commonly used for analytic modeling has been the number of simultaneous system users. This has the advantage

of being easily measured. The alternative would be some measure of the demand placed on the system resources by the users. In most cases, however, such a measure (assuming we are dealing with a single-valued random variable) would be linearly related to the number of users, as long as all users were drawn from the same population. Therefore we will argue in section IV.3 that the number of users, with some strong reservations, is an acceptable single-valued load measure.

The question of how to measure the performance of the system is equally important. In this chapter we will argue that response time is the most satisfactory single measure of performance. There are a number of possible alternatives. We suggest some of them, including the possiblity of a performance vector which recognizes the fact that a time-sharing system is a collection of resources on which varying demand is placed. Response time can be characterized as a weighted sum of the elements of this performance vector.

Before discussing measures of load and performance, we discuss the implications of the several ways in which these measures may be expressed in an analytic model.

2. Distributions, Moments, Expectations

An analytic model may be viewed as a functional operator of sorts. The structure of the time-sharing system - its hardware and operating system - determine the transformations to be made. The functions operated on are the load measures, the descriptors of user behavior. The results of the functional transformation are the performance measures. Thus the processor scheduling models reviewed in Chapter II incorporate the scheduling algorithm used in the time-sharing system as part of the functional operator. The input functions are the arrival rate for user requests and the distribution of the amount of CPU time requested by each arrival. The output function may be a response time distribution, a queue length distribution, a response time expectation, etc.

What does it mean to solve a model? Presumably it means that for some class of input functions the model produces one or more output functions. We can evaluate the worth and generality of a model on several levels.

(1) How varied and rich a system structure can we represent in the model itself, without having to "resolve" it? It is in this area that network models are particularly attractive, since one has great flexibility in representing the structure of the

system being modeled.

(2) How much freedom is allowed in the specification of the input functions - the load descriptors? For example, to return again to processor scheduling models, the input parameters may simply be the arrival rate and the mean of an exponential distribution (the CPU time demanded) [A3,C6]. Or an arbitrary distribution for the service demanded may be allowed[C2].

(3) How rich and varied are the output functions - the performance measures? These generally come in three varieties: distributions(or Laplace transforms thereof), conditional distributions, and expectations. Of course several performance measures may be available from one model: perhaps a CPU queue length distribution and a waiting time expectation, etc.

It is in this last area that network models are weakest. Equilibrium queue length distributions are available but important measures such as the cycle time (which will be the response time in our models) are available as expectations only. Since this is the case, we should consider the problems of working with expectations, instead of distributions, as performance measures. We now consider under what conditions, if any, an expectation might be a satisfactory substitute for a complete distribution as

a model output.

An expectation will be a satisfactory measure if the distribution of which it is the mean has a sufficiently small variance. Then the actual system performance will not deviate too much from its predicted value. Of course, if it were possible to determine exactly what the variance was, we could probably derive the distribution itself, so we must somehow estimate this variance.

We would expect the variance of the performance measures to be closely related to the variance in the user behavior. We would also expect larger numbers of users to reduce the effect of variance in the individual user's behavior. Given the Central Limit Theorem, this is of course a natural expectation. However, the relation of the variance in the performance measures to the variance in user behavior (the load measures), as a function of the number of users, is a complex one. Even in simple queueing systems, increased variance in the customer behavior (service demanded) causes changes not only in the variance of performance measures, but in the means of performance measures (expected queue lengths for example). Thus we must proceed with a good deal of caution when suggesting a simple expectation as a performance measure, since the variance associated with it could conceivably be very large.

3. Load Measures

In developing analytic models for computer time-sharing systems, the number of terminal users has almost universally been used as a measure of system load; the independent variable in the equations. In addition, the users have generally been assumed to be statistically identical.

Anyone with experience in the analysis of actual time-sharing systems will agree that actual system performance measures do not correlate particularly well with the number of terminal users. The reason for this does not lie in any basic defect in the number of users as a load measure. The problem lies rather in the behavior of these users. They are not statistically identical. In a general purpose system users may be engaged in a number of different activities: scientific computation, information retrieval, text editing and so forth. The demand users of different types place on the system may vary drastically. A user performing text-editing will tend to interact frequently but use little CPU time during each IR cycle. [An "IR cycle" is the time spanning a user's request (Interaction) for system service and the system's fulfilling of that request (Response)]. The user who is doing scientific computation (and has already debugged his program) may interact very infrequently but demand a great deal of CPU time. Thus the

user population of a general purpose time-sharing system at any given time will consist of such diverse types of users, possibly with little overlap in their "behavior parameters".

A reasonable way to represent different classes of users is via a load vector characteristic of each class of users. The load vector is a vector of distribution functions. Each function gives the distribution of time demanded from each resource during an IR cycle. An additional function in the vector gives the IR cycle time distribution. Thus if the resources are numbered 1,2,3,...,n, the vector of load functions for a user of class A is denoted:

$$L_A(f_1(t),f_2(t),f_3(t),\ldots,f_n(t),f_I(t))$$

where each of the $f_i(t)$ is the distribution function of a random variable x_i, which is the amount of time demanded from resource i in one IR cycle by a user of class A; and $f_I(t)$ is the distribution function of the user interaction time. There is a close relation between this concept and the one formulated by Wulf[W4] for batch multiprogramming systems. We can use this notation (and concept) for network models, with the restriction that each of the $f_i(t)$ be an exponential distribution, or one of the possible approximations mentioned in Chapter III. In addition, in network models, we are limited to a single class of users.

We might still assume all users are identical for

modeling purposes if we can construct a "virtual user" who is a composite of the various types of actual system users. If the virtual user so constructed has the characteristics of the expected user population as a whole, the model predictions should still be valid. We have not, however, measured the effect of the non-existence of this virtual user on the stability of the load. The virtual user is a sort of "expected composite" of the actual users. Wide variance from this expected user may be the rule rather than the exception. This will of course make predictions seem unreliable, as was discussed in the previous section. This effect will be especially noticeable in smaller time-sharing systems, for there the actions of one user may seriously effect the system load (and indirectly, performance). The differences between zero, one and two users requesting large numbers of disk operations may, for example, lead to noticeable differences in system performance in a small system. Yet the virtual user must represent the expected proportion of such heavy disk users: .207 or whatever is appropriate. Performance will be predicted on the basis of N virtual users, .207N of which are heavy disk users. Actual performance will be determined by the current number of heavy disk users, regardless of N. We reiterate that under such circumstances, it is best to develop load and performance "envelopes", representing the <u>range</u> of possible user

behavior parameters for a given number of users. This is more effective than simply representing the virtual user as having more variance. Variance injected in this way appears in variance of the user behavior over a few seconds or minutes (CPU time demanded per IR cycle, for example). The mix of user types using the system, however, will tend to vary more slowly, significant changes not occurring for many minutes or an hour or so. The load mix - types of users connected to the system at any one time - varies as users sign on and sign off. Presumably this happens at a rate which is slow compared to the interaction rate of the connected users. Thus it seems reasonable to assume that the system will reach near-equilibrium for a given user mix, once that mix has been established. This gives support to the concept of using load and performance envelopes, as mentioned above, rather than representing changes in the user mix by increasing the variance in the behavior parameters of the individual users. Of course with sufficiently large numbers of users, and in the absence of many clearly defined different types of users, the load mix may remain statistically quite stable, allowing a narrow load envelope to be used.

4. Performance Measures

The actual measure used in evaluating time-sharing system performance is most often "How many users can it support?". Taking advantage of the manner in which the question is phrased, the purveyor of such systems will generally respond with the number of terminals which can physically be connected to it. This of course doesn't really answer the question being asked, which is better stated: How many active users will the system support, doing the things we expect to do with it, and with an acceptable response time?

Response time then, defined as the time it takes for the system to complete a terminal user's service request, is a very intuitive measure of system performance. It can clearly be refined in several ways. A request to make a compilation will generally require more system resources than a request to add a line to an already-opened file. This leads to a need for <u>conditional</u> response time measures. The response time in this case to be conditioned on the resources needed to fulfill the request. For processor scheduling models, the only resource has been the CPU, so conditional response time has meant response time conditioned on the amount of CPU time requested. For network models, where several resources are represented as nodes in the network, the response time may be conditioned on a cycle path - the

path the user will take through the network to fulfill his request. For example, we might ask what the response time would be if the user's path through the system were:

(1) Drum, to be swapped in.

(2) CPU, 3 quanta.

(3) Disk, for a file operation.

(4) CPU, 2 more quanta.

(5) Drum, to be swapped out.

As was discussed in Chapter III, in the derivation of the expected cycle time, the path taken through the network is unimportant. Only the number of times each node appears on the path is of interest. Thus, instead of a path, we may condition on a vector, called the demand vector giving the number of times each node in the network appears on the path. We will call this vector $(d_1,d_2,\ldots,d_M)$, where d_i is the number of times node i appears on the path. Note the close relation to the demand vector and the load vector discussed in the previous section as a general load measure. We denote expected cycle time (response time) conditioned on taking this path by $E[c|(d_1,d_2,\ldots,d_M)]$. Using the same arguments used in section III.5 to derive the cycle time, we can show that:

$$E[c|(d_1,d_2,\ldots,d_M)] \cong \sum_{i=1}^{i=M} d_i w_i(N)$$

where $w_i(N)$ is the expected waiting time at node i when the number of customers in the network is N.

The unconditional response time is simply the conditional response time of the expected demand vector (see equation III.3.(10)). Thus the expected response time weights the performance of each of the individual resources of the system by the degree to which it is used, to produce a single-valued performance measure.

Chapter V

An Example Model

The models developed in this chapter are intended to illustrate the use of network models in the analysis of time-sharing systems. The system to be modeled will be hypothetical. This will relieve us of the clutter of practical considerations, allowing concentration on the use of the mathematical tools and concepts developed in Chapters III and IV. The modeling of an actual system is left to the next chapter. We first develop a fairly simple network model and then consider a number of major and minor changes to illustrate our points.

The Hypothetical System

Consider a time-sharing system of the swapping type. There is no virtual memory, but there is some sort of relocation hardware allowing multiprogramming to take place. The entire memory allocated to a process is swapped in from a drum, used only for that purpose, whenever the user types in a line of input. The memory is entirely swapped out whenever the system is ready to accept another line of input, indicating it has completed the service request initiated by the previous line. Up to 200 terminals may connect to the system. There is a single drum, a single processor, and suf-

ficient core memory so that there is always room to swap in another process. There is a disk available for file storage. The disk and drum are on separate channels. An outline of this system is given in Fig 5-1.

Suppose we have made the following observations concerning the users of this hypothetical system and their programs. We define an Interaction-Response cycle (IR cycle) to be the time (and events occurring) between the completed typing of one line by the user (indicated by his striking carriage return), and the time the system is ready to accept another line. The user's think time or interaction time is the time from the point at which system is ready to accept another input line, to the point at which the user completes the typing in of that line.

(1) All users and their programs behave independently and identically (in the probabilistic sense).

(2) The think time of a user is an exponentially distributed random variable with a mean of 15 seconds.

(3) The time required to swap a process in (or out) is an exponentially distributed random variable with a mean of 90 milliseconds.

(3) The processor time required to service an IR cycle is exponentially distributed with a mean of 500 milliseconds.

(5) The process makes a geometrically distributed number

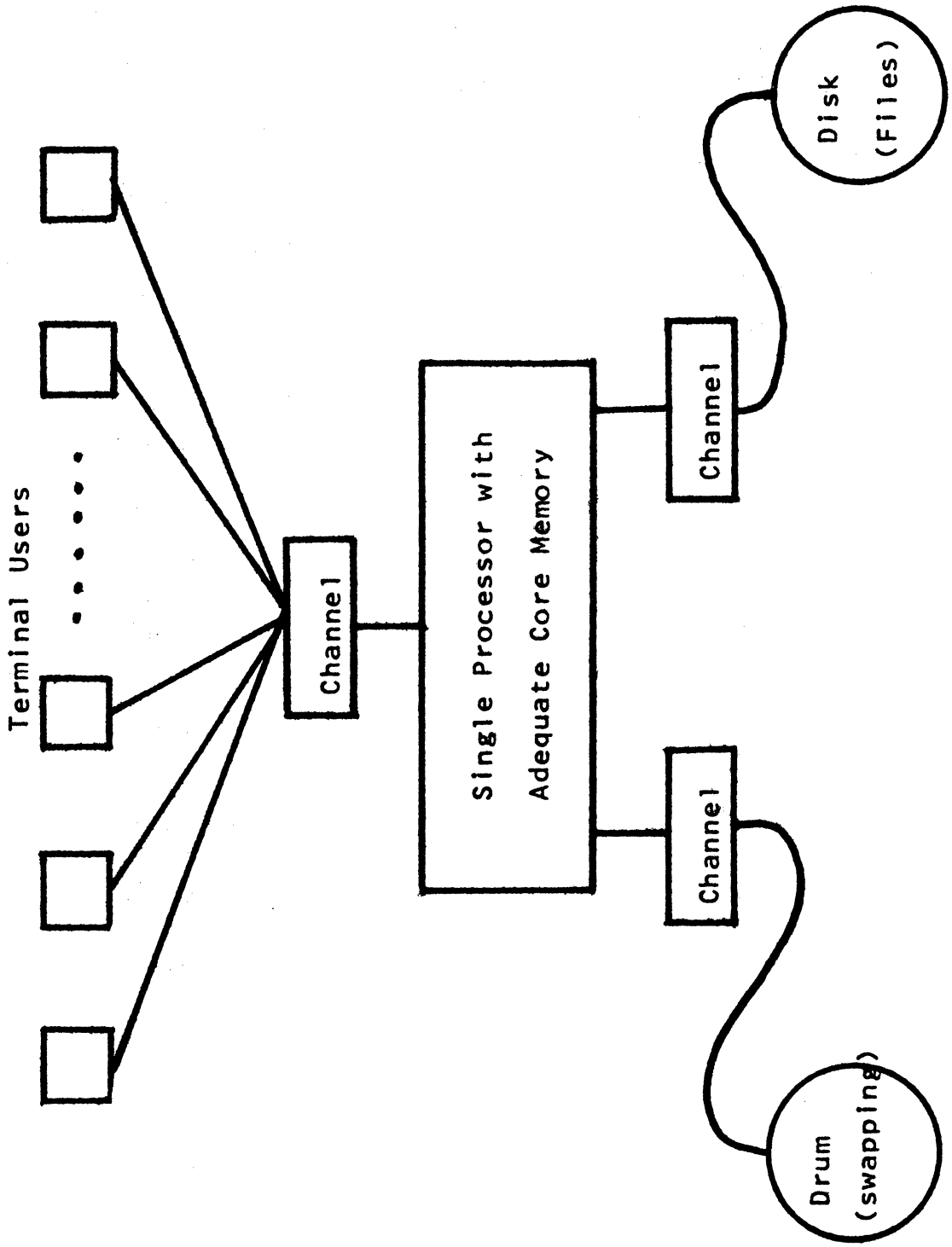

Fig 5-1: Block Diagram of Hypothetical System

of file requests in servicing an IR cycle. The mean number of requests made by a user process for file operations in one IR cycle is 1.5. The time required to service requests is an exponentially distributed random variable with a mean of 150 milliseconds.

Fortunately, these "experimentally" determined statistics will make the modeling tasks very easy. We can construct a network model which is exactly equivalent to the structure of our example system. The structure of the model is based on that of the GE-635 time-sharing system implemented by Dartmouth College.

The model we will start with is diagrammed in Fig. 5-2. Each circle in Fig. 5-2 represents a server and its associated queue - a node in the network. We will treat each node as though service there were performed by one or more identical exponential servers. Thus, each node is characterized by the two numbers indicated in Fig. 5-2: the mean service time of each of the servers and the number of servers at the node. This is a special case of the network model structure developed in Chapter III. There the service rate was allowed to vary arbitrarily with the number of customers at the node. In this case, the service rate varies as follows. Let n_i denote the number of customers at node i. Let k_i denote the number of servers at node i. Let m_i be the mean service time of each of the servers at node i. Then in

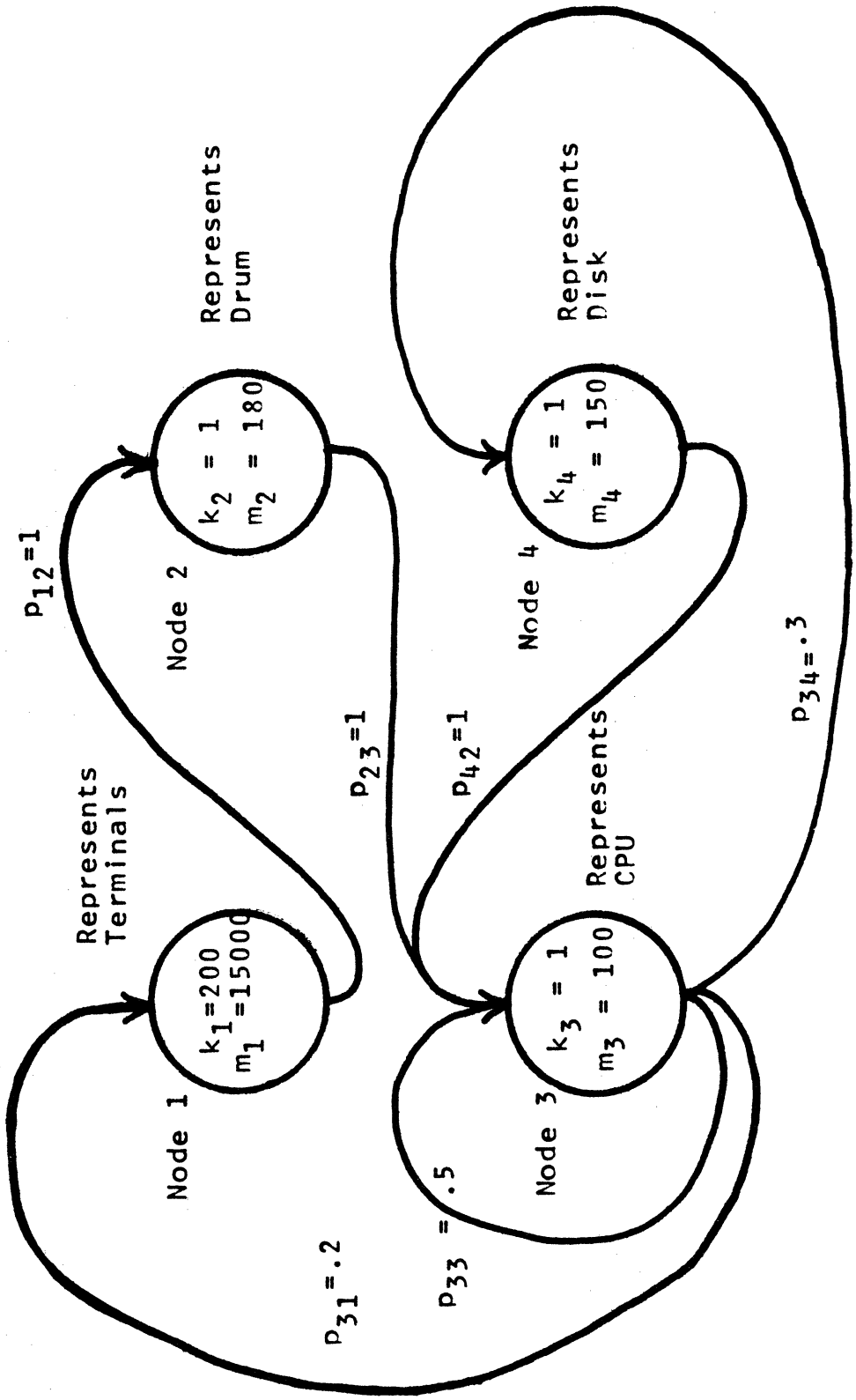

Fig 5-2: Initial Model of Hypothetical System

the notation of Chapter III we have:

$$\mu_i(n_i) = 0 \qquad \text{for } n_i = 0$$
$$\mu_i(n_i) = n_i/m_i \qquad \text{for } 0 < n_i < k_i$$
$$\mu_i(n_i) = k_i/m_i \qquad \text{for } n_i \geq k_i$$

Note the representation of the terminals as a node of the network. The servers at the terminal node represent the users of the time-sharing system. The service time represents the time it takes a user to think of, and input, his next service request (a line of input, in this case).

When a user (now a process) completes a service request by typing a carriage return, he exits from the terminal service node and passes to the node representing the drum. This represents the need to swap the process into core before it can receive service. We have made the service time here twice the mean swap time in the system being modeled. We will model both the swap in and swap out operations as occurring <u>before</u> the process receives CPU service. From a practical point of view this is unrealistic, of course. From a mathematical point of view this formulation is equivalent to one in which the process returns to the drum service node after completing its CPU service. It is not possible to model the swapping process in the more natural way. To do so, we would need to "tag" processes arriving at the drum node to indicate that they were being swapped in or swapped

out. Then on completion of drum service, they could correctly be routed back to the terminal service node or to the CPU node. The markovian nature of the model does not allow this sort of tagging to occur. Transition probabilities for a process exiting from a node may not depend on the past history of the process. In particular, it cannot depend on the previous node through which the process passed. There are several other mathematically equivalent ways of representing the swap-in, swap-out relation within the restrictions of the network model.

The rest of the representation is straightforward. The reader need only satisfy himself that the total CPU service time incurred by a process before returning to the terminal service node is exponentially distributed. The process will pass through the CPU node a geometrically distributed number of times, each time receiving service that is exponentially distributed with a mean of 100 msec. The density function for the total CPU time used in one IR cycle is of the form:

$$f(t) = \sum_{i=0}^{i=\infty} p^i(1-p)e_i(\mu,t) \qquad (1)$$

where p is the probability of returning to the CPU node before returning to the terminal node (.8) and μ is the parameter of the CPU service time distribution (1/100 milliseconds). The $e_i(\mu,t)$ represents an Erlang-i density func-

tion with parameter μ.

Using Laplace transforms one can verify that the transform of (1) is the transform of an exponential density function with mean 500 msec.

All the numerical results presented in this chapter and the following chapter were computed by a program written for this purpose. The program is written in FORTRAN and essentially automates the mathematical developments of Chapter III. The program is about 300 statements long, much of it devoted to presenting the results in a readable format. The program computes the statistics for either a fixed N or for the limiting case. The computation time required for the limiting case is relatively constant at between 15 and 20 seconds. For fixed N, the program will solve networks with no more than 200 customers and 16 nodes. The computation time in this case depends on the size of the state space, and is given approximately by:

$$\text{CPU Time Required} = 20 + S/17000 \quad \text{seconds}$$

where S is the size of the state space, and is equal to:

$$\binom{N+M-1}{M-1}$$

Further discussion of the computation program may be found in the Appendix

2. Results for a Fixed Number of Customers

We now present the results of solving the model of Fig 5-2 for N =30. This is nearly equal to N_s, the saturation point for this model. In Fig 5-3 the output of the computation program for this model is reproduced. Most of the entries should be self-explanatory. The expected cycle time for the terminal node (2.915 sec.) is the expected response time. Cycle times for the other nodes are of relatively little interest. The "demand per interaction" entry gives the expected amount of time a user will spend in service and in queue at each node during one IR cycle.

The utilization of a resource (node) is computed by one of two methods. The arbitrary μ_i at each node may be interpreted as either a single server whose service rate may vary, or as one or more identical servers, as was the case in our definition of the terminal node in this model. In the case where the service rate of a single server varies, the resource utilization is simply the probability that the node has one or more customers at it. In the case where the node represents K identical servers, the utilization is computed

*** INITIAL EXAMPLE MODEL FOR CHAPTER V ***

NUMBER OF USERS = 30

	TERMINAL	DRUM	CPU	DISK
MEAN WAIT AT NODE (INCLUDES SERVICE)	15000.000	257.608	459.594	240.006
MEAN SERVICE TIME AT THIS NODE	591.183	180.000	100.000	150.000
MEAN QUEUE LENGTH (INCLUDES THOSE IN SERVICE)	25.373	0.431	3.596	0.600
MEAN NUMBER OF TIMES THROUGH NODE (IN 1 INTERACTION)	---	1.000	5.000	1.500
UTILIZATION FACTOR FOR THIS RESOURCE	0.846	0.304	0.846	0.381
EXPECTED CYCLE TIME FOR THIS NODE	2915.585	17657.977	3123.523	11703.717
DEMAND PER INTERACTION FROM THIS NODE	15000.000	257.608	2297.968	360.009

STEADY-STATE QUEUE LENGTH DISTRIBUTIONS

CUSTOMERS	TERMINAL	DRUM	CPU	DISK
0	0.00000	0.69553	0.15424	0.61941
1	0.00000	0.21402	0.14785	0.23927
2	0.00000	0.06428	0.13711	0.09022
3	0.00000	0.01881	0.12287	0.03314
4	0.00000	0.00535	0.10626	0.01184
5	0.00000	0.00148	0.08855	0.00410
6	0.00000	0.00039	0.07101	0.00138
7	0.00000	0.00010	0.05470	0.00045
8	0.00000	0.00003	0.04041	0.00014
9	0.00001	0.00001	0.02857	0.00004
10	0.00003	0.00000	0.01930	0.00001
11	0.00009	0.00000	0.01242	0.00000
12	0.00023	0.00000	0.00760	0.00000
13	0.00053	0.00000	0.00441	0.00000

• • • • •

Fig 5-3: Program Output for Initial Model

as:

$$U = 1 - \sum_{n=0}^{n=K} p(n)(K-n)$$

where p(n) denotes the probability of having n customers at the node in equilibrium.

The queue length distributions for each of the four nodes are graphed in Fig 5-4. The term queue length is perhaps misleading since what is actually computed is the mean number of customers at the node. This will include both those customers in service and in queue. We reiterate that from a mathematical point of view, the interpretation of those in queue and in service at a node is unimportant.

Approximating Other Distributions

Now let us consider a modification to the system/model, in which we approximate a non-exponential distribution. Suppose, for example, that the disk (Node 4) does not have an exponential service time distribution, but rather an Erlang-3 distribution with the same mean (150 milliseconds). That is to say, the disk is best represented by 3 exponential servers in series, each with a mean service time of 50 milliseconds. This reduces the standard deviation of the (total) disk service time from 150 to 50 milliseconds. The revised network model is diagrammed in Fig 5-5.

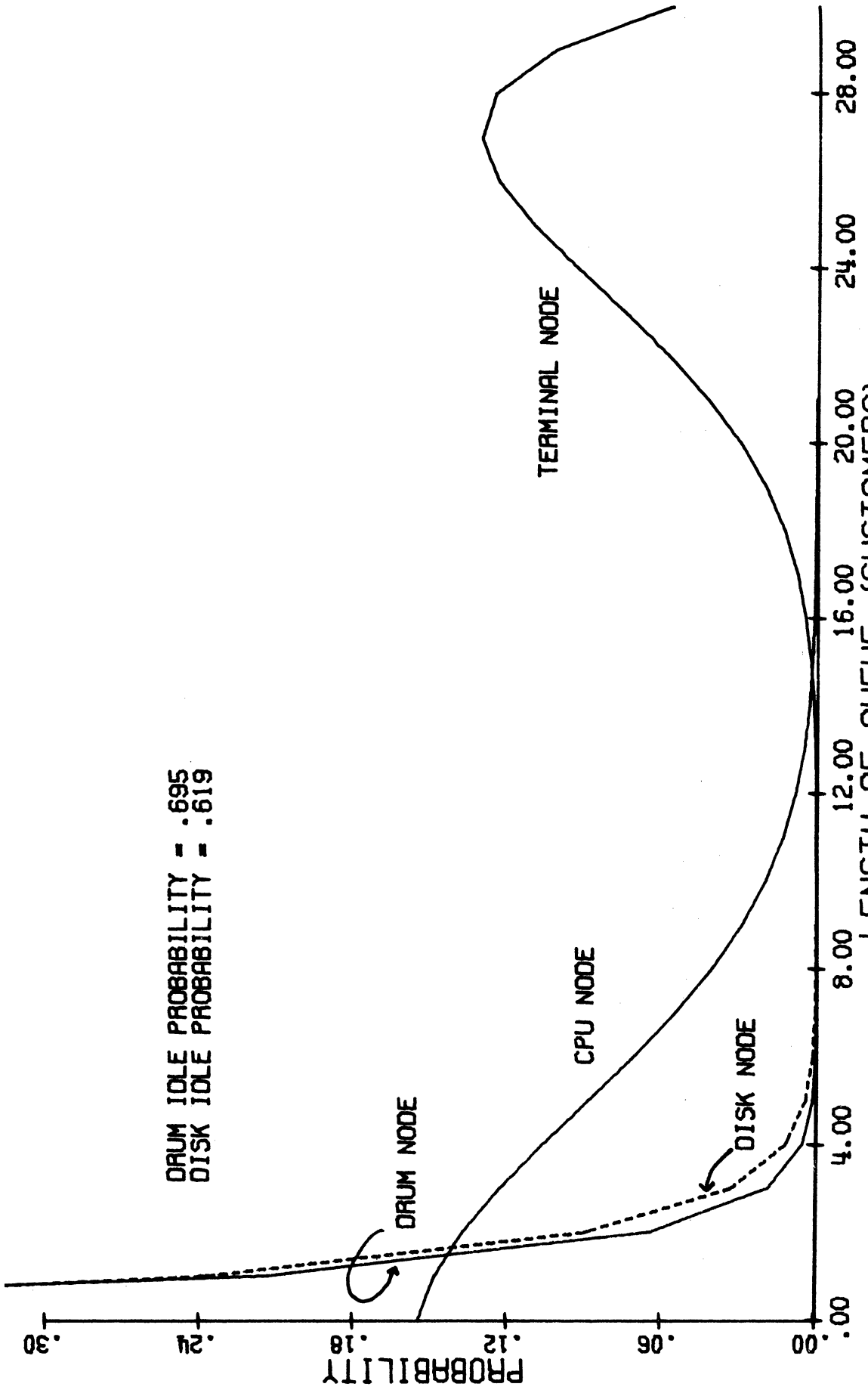

FIG 5-4: QUEUE LENGTH DENSITIES (FIG 5-2 MODEL. N = 30)

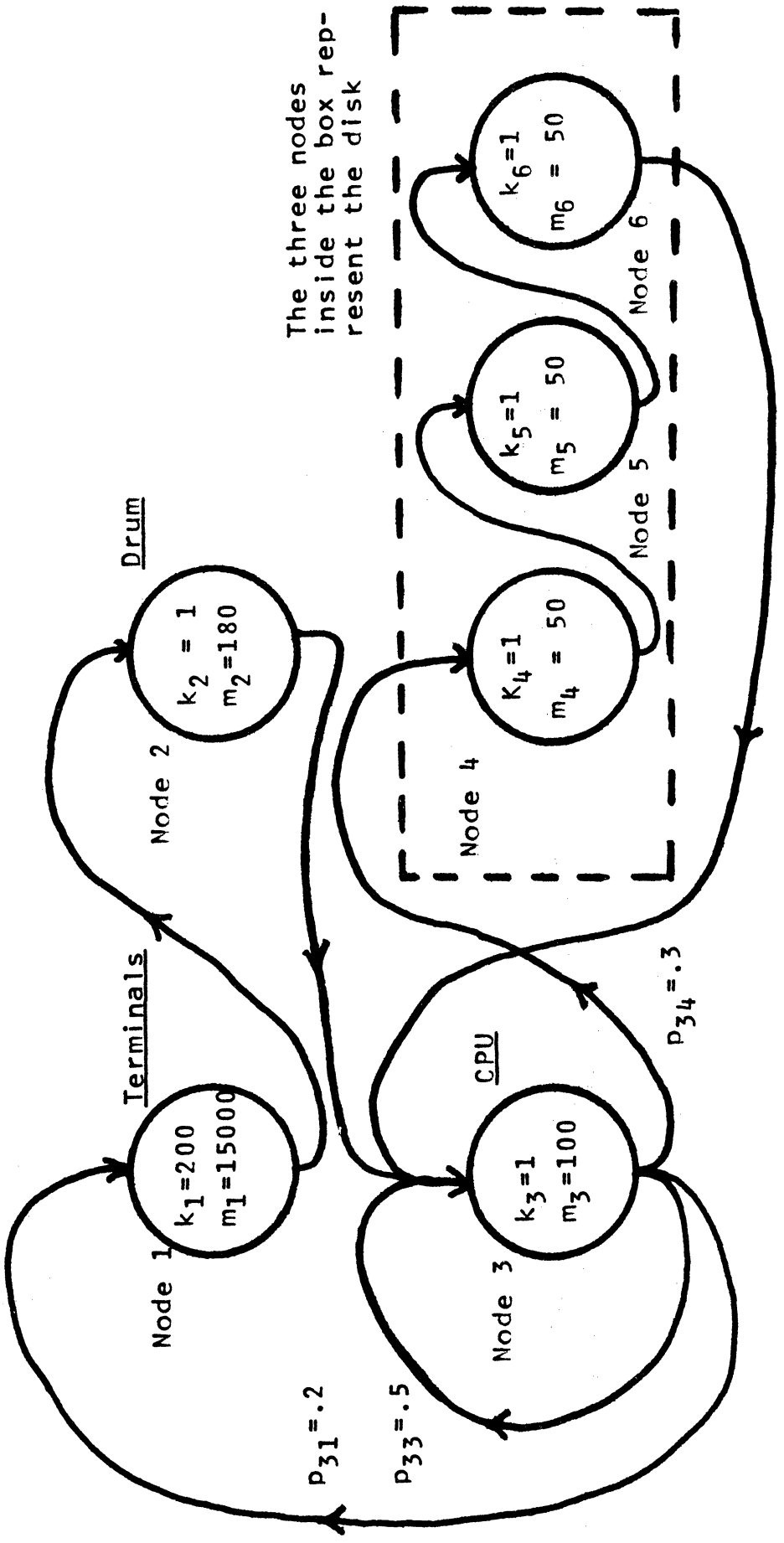

Fig 5-5: Hypothetical System with Modified Drum Node

Solving this model, again for N = 30, we find (Fig 5-6) that the model statistics have changed very little from those of the original model. The actual drum queue now appears as 3 queues - one at each of the nodes representing the drum. Fig 5-7 graphs the queue length distributions for this revised model. The three drum node queues have been added together for this purpose.

A Linear Equivalent Model

Another interesting transformation to make on our original model is to "linearize" the transitions. That is to say, we restructure the model so that all the transitions are deterministic. At the same time, the service rate means are changed so that the service time requested at each node during an IR cycle remains the same. For example, in the original model, the CPU node was passed through an average of 5 times with a mean service time each cycle of 100 milliseconds. Thus the total CPU time requested each IR cycle was exponentially distributed with mean 500 milliseconds. We then change the service rate at the CPU node to 1/500 and the transition matrix so that it is passed through once. The resulting model is depicted in Fig 5-8. When we solve this model, again for N=30, it is surprising that all the statistics are almost exactly the same as those in the original model. These statistics are presented in Fig 5-9

*** EXAMPLE MODEL WITH ERLANG-3 DISK SERVICE ***

NUMBER OF USERS = 30

	TERMINAL	DRUM	CPU	DISK(1)	DISK(2)	DISK(3)
MEAN WAIT AT NODE (INCLUDES SERVICE)	15000.000	258.072	464.955	57.271	57.271	57.271
MEAN SERVICE TIME AT THIS NODE	588.686	180.000	100.000	50.000	50.000	50.000
MEAN QUEUE LENGTH (INCLUDES THOSE IN SERVICE)	25.480	0.434	3.650	0.145	0.145	0.145
MEAN NUMBER OF TIMES THROUGH NODE (IN 1 INTERACTION)	---	1.000	5.000	1.500	1.500	1.500
UTILIZATION FACTOR FOR THIS RESOURCE	0.849	0.306	0.849	0.127	0.127	0.127
EXPECTED CYCLE TIME FOR THIS NODE	2840.562	17582.490	3103.158	11836.438	11836.438	11836.438
DEMAND PER INTERACTION FROM THIS NODE	15000.000	258.072	2324.773	85.906	85.906	85.906

Fig 5-6: Program Output for the Erlang-3 Model

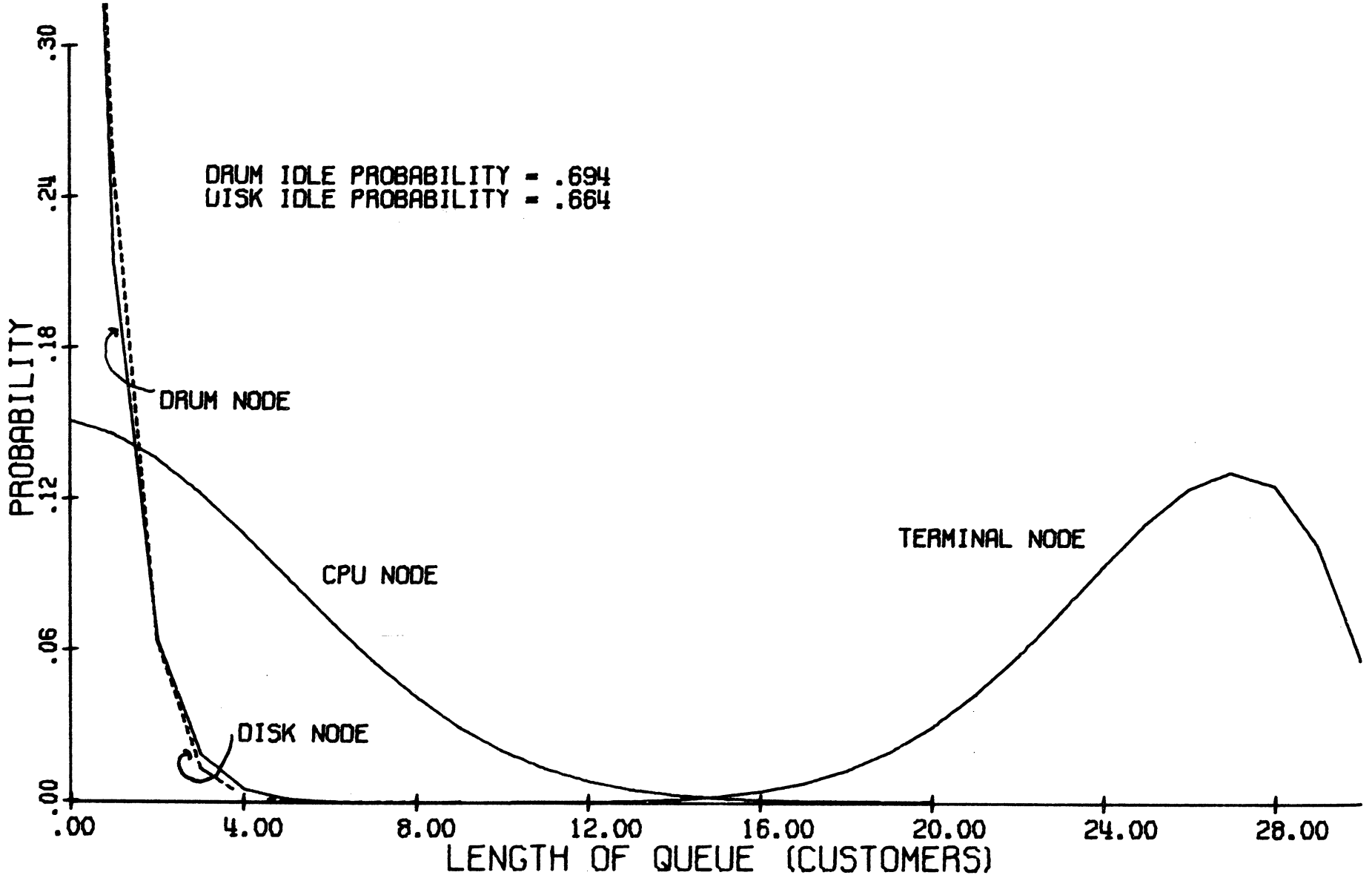

FIG 5-7: QUEUE LENGTH DENSITIES (FIG 5-5 MODEL. N=30)

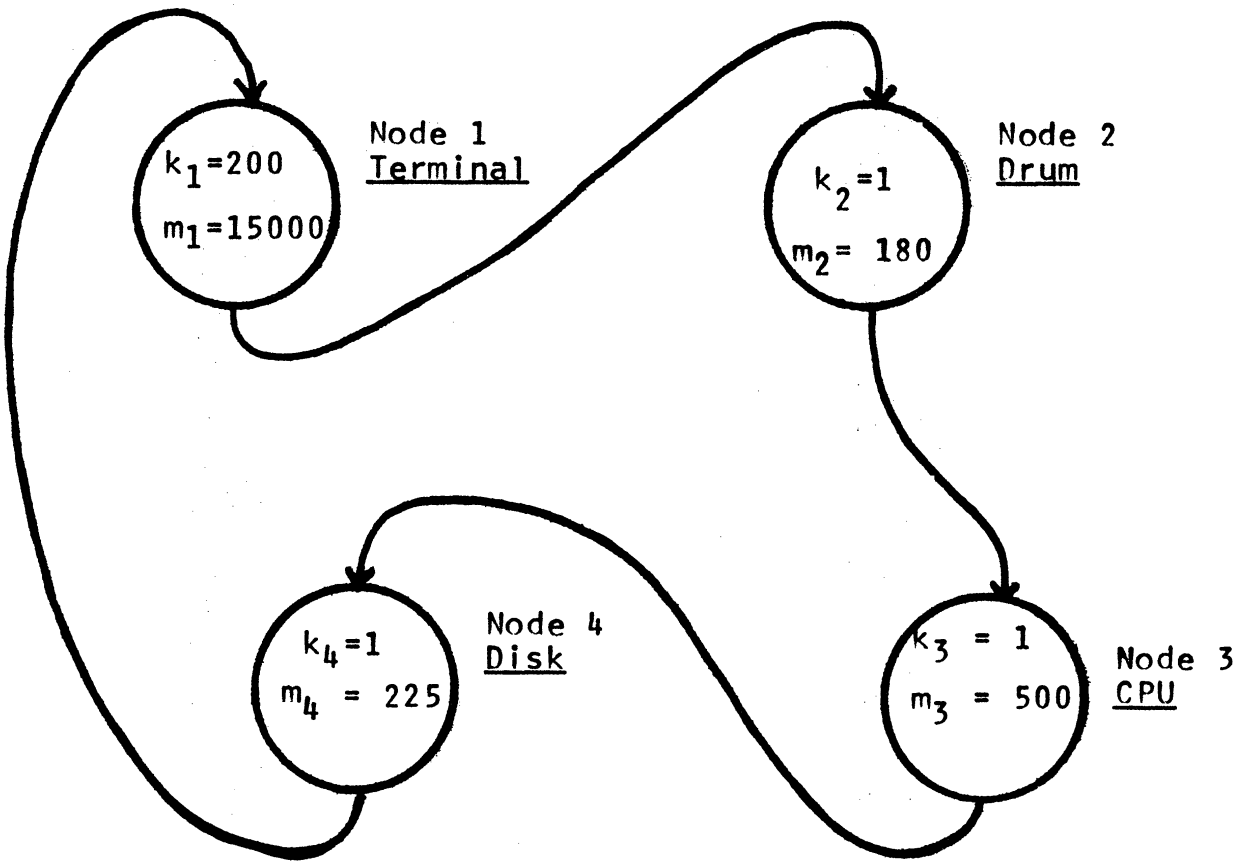

Fig 5-8: Hypothetical System Modeled by Linear Model

*** LINEAR EXAMPLE MODEL FOR CHAPTER V ***

NUMBER OF USERS = 30

	TERMINAL	DRUM	CPU	DISK
MEAN WAIT AT NODE (INCLUDES SERVICE)	15000.000	257.608	2297.968	360.009
MEAN SERVICE TIME AT THIS NODE	591.183	180.000	500.000	225.000
MEAN QUEUE LENGTH (INCLUDES THOSE IN SERVICE)	25.373	0.431	3.596	0.600
MEAN NUMBER OF TIMES THROUGH NODE (IN 1 INTERACTION)	---	1.000	1.000	1.000
UTILIZATION FACTOR FOR THIS RESOURCE	0.846	0.304	0.846	0.381
EXPECTED CYCLE TIME FOR THIS NODE	2915.585	17657.977	15617.617	17555.576
DEMAND PER INTERACTION FROM THIS NODE	15000.000	257.608	2297.968	360.009

STEADY-STATE QUEUE LENGTH DISTRIBUTIONS

CUSTOMERS	TERMINAL	DRUM	CPU	DISK
0	0.00000	0.69553	0.15424	0.61941
1	0.00000	0.21402	0.14785	0.23927
2	0.00000	0.06428	0.13711	0.09022
3	0.00000	0.01881	0.12287	0.03314
4	0.00000	0.00535	0.10626	0.01184
5	0.00000	0.00148	0.08855	0.00410
6	0.00000	0.00039	0.07101	0.00138
7	0.00000	0.00010	0.05470	0.00045
8	0.00000	0.00003	0.04041	0.00014
9	0.00001	0.00001	0.02857	0.00004
10	0.00003	0.00000	0.01930	0.00001
11	0.00009	0.00000	0.01242	0.00000
12	0.00023	0.00000	0.00760	0.00000
13	0.00053	0.00000	0.00441	0.00000

• • • • •

Fig 5- 9: Program Output for the Deterministic Model

and Fig 5-10. Compare these statistics with those of Fig 5-3 and Fig 5-4.

This result seems to show that the stochastic nature of the node transitions is not too important. Restructuring a network in this form also makes it very clear why one node may control the entire performance of the network after saturation is reached. The network in this form has the appearance of a ring. One would expect the flow in such a ring to be naturally limited by the maximum flow through the node with the minimum capacity.

Reordering Queues for Faster Service

It is often possible to reduce the expected operation time for a disk or other mass storage device by reordering the queue of requests. This reordering is done to reduce the average seek time or latency time. Several studies have been made of such reordering schemes and were reviewed in Chapter II[A1,D2,F4]. From such models of disk and drum scheduling one can generally derive an operation time distribution for the mass storage device, conditioned on the length of the queue of operations to be performed. Thus the service time for the next request (to be served, not the next to arrive) is denoted $f(t|n)$, where n is the queue length. Using the expectation of this distribution, $E[f(t|n)]$, we can represent this sort of service structure

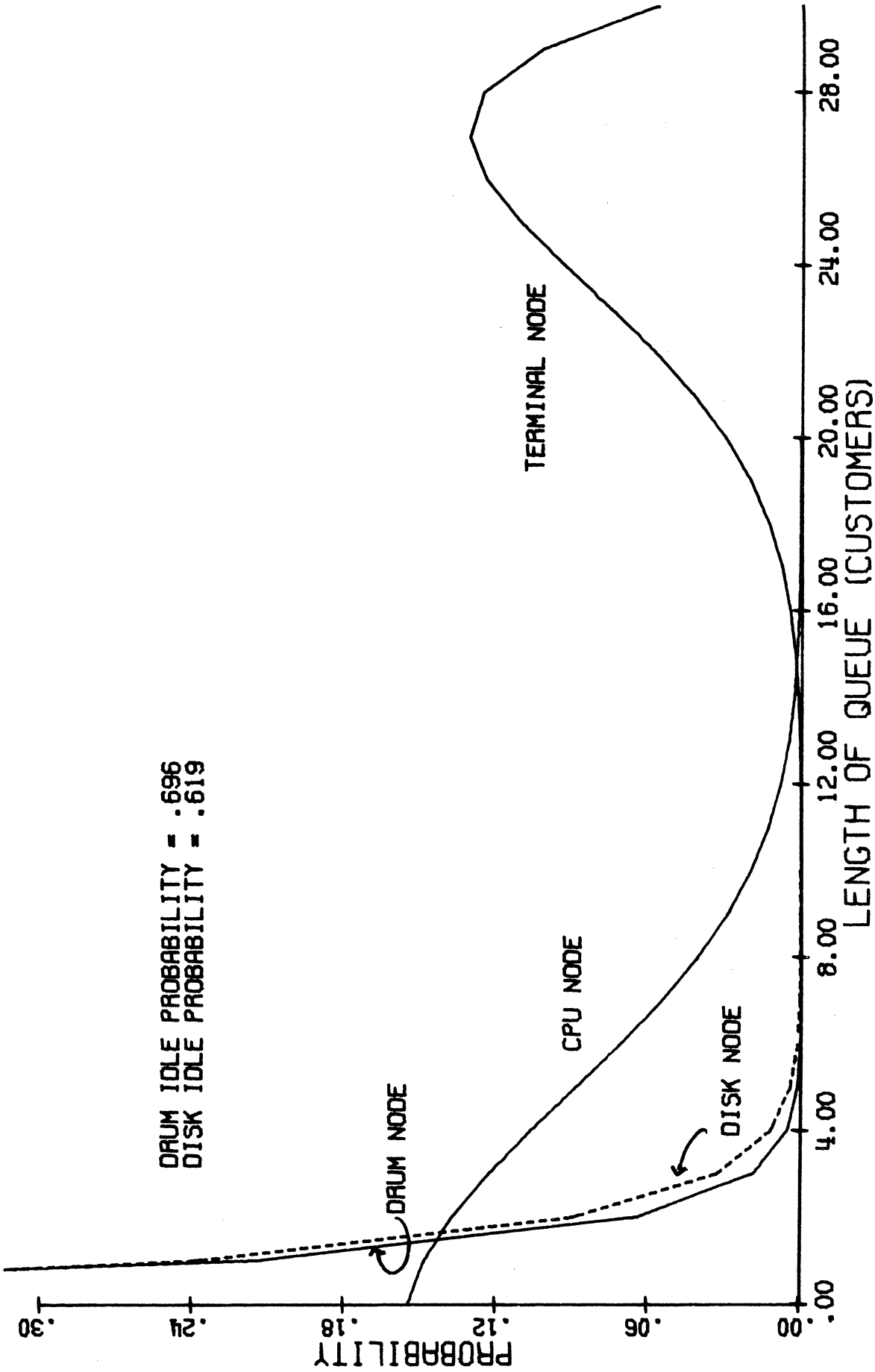

FIG 5-10: QUEUE LENGTH DENSITIES (FIG 5-8 MODEL, N=30)

in a network model by the scheme:

$$\mu_1(n) = 1/E[f(t|n)] \qquad n=1,2,\ldots,N$$

That is, we make the service rate at the node dependent on the number of customers there, in a way which reflects the throughput improvement to be gained by reordering the queue.

We illustrate this sort of service structure by another modification of the initial model of Fig 5-2. We modify only the service rates at the disk node, in the following way:

$$\mu_4(n) = 1/(170-20n) \qquad \text{for } n \leq 5$$
$$\mu_4(n) = 1/70 \qquad \text{for } n > 5$$

Thus with one customer at the node, the service time mean will be the same (150 milliseconds) as it was in the initial model. However, as the queue gets longer, the server, taking advantage of the choice of operations, reduces his average service time until a maximum average service rate of 1/70 is reached for n = 5.

The statistics for this modified model are presented in Fig 5-11 and Fig 5-12. Since the disk node is not heavily loaded, the effects of reordering are not too great. Since the probability (in the initial model) that the queue length at the node was greater than one is only .14, the reordering

*** EXAMPLE OF EFFECT OF REORDERED DISK QUEUE

NUMBER OF USERS = 30

	TERMINAL	DRUM	CPU	DISK
MEAN WAIT AT NODE (INCLUDES SERVICE)	15000.000	257.894	462.810	210.462
MEAN SERVICE TIME AT THIS NODE	589.636	180.000	100.000	140.334
MEAN QUEUE LENGTH (INCLUDES THOSE IN SERVICE)	25.439	0.433	3.628	0.500
MEAN NUMBER OF TIMES THROUGH NODE (IN 1 INTERACTION)	---	1.000	5.000	1.500
UTILIZATION FACTOR FOR THIS RESOURCE	0.848	0.305	0.848	0.357
EXPECTED CYCLE TIME FOR THIS NODE	2887.636	17629.742	3114.717	11714.628
DEMAND PER INTERACTION FROM THIS NODE	15000.000	257.894	2314.049	315.693

STEADY-STATE QUEUE LENGTH DISTRIBUTIONS

CUSTOMERS	TERMINAL	DRUM	CPU	DISK
0	0.00000	0.69473	0.15202	0.64300
1	0.00000	0.21435	0.14643	0.24838
2	0.00000	0.06455	0.13640	0.08117
3	0.00000	0.01894	0.12274	0.02187
4	0.00000	0.00540	0.10655	0.00469
5	0.00000	0.00149	0.08911	0.00076
6	0.00000	0.00040	0.07169	0.00012
7	0.00000	0.00010	0.05539	0.00002
8	0.00000	0.00003	0.04103	0.00000
9	0.00001	0.00001	0.02909	0.00000
10	0.00003	0.00000	0.01969	0.00000
11	0.00009	0.00000	0.01270	0.00000
12	0.00022	0.00000	0.00778	0.00000
13	0.00052	0.00000	0.00452	0.00000

• • • • •

Fig 5-11: Program Output for the Reordered Disk Queue Model

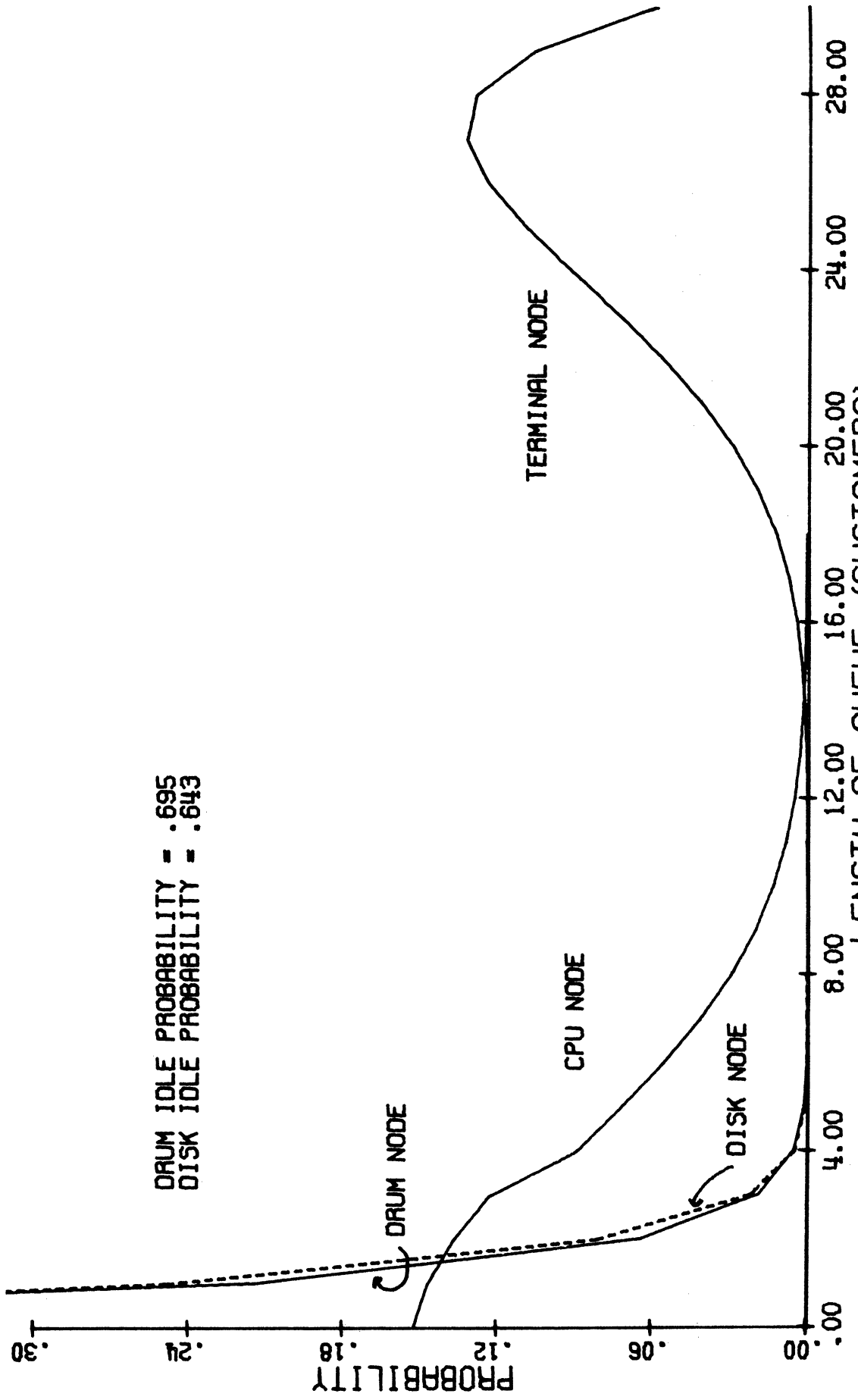

FIG 5-12: QUEUE LENGTH DENSITIES (REORDERED DISK QUEUE)

only reduces the average service time by about 10 milliseconds. The response time, however, is reduced by 28 milliseconds, a somewhat surprising result. This is because the probability of long waits at the disk node has been significantly reduced, reducing the mean wait at the disk node by 30 msec.

If the average queue length at the disk node had been quite long, the performance improvements might have been dramatic. The purpose of the example, however, was simply to illustrate the way in which the effects of such a reordering could be represented in a network model.

Conditional Response Time

We mentioned in the previous chapter the concept of a response time conditioned on a path through the network. This gives a measure of the response to a particular type of service request. Note that the conditional response time assumes no discrimination on the part of the servers, based on past history. Thus if scheduling algorithms such as feedback-N[C6] or priority disciplines[J3] are in operation in the system being modeled, the conditional response time as derived from the network model should be regarded as approximate at best.

Suppose we wish to estimate the response time of a service request requiring 8 seconds of CPU time and 20 disk

operations, in the model of Fig 5-2. 8 seconds of CPU time will require a mean of 80 passes through the CPU node (8 seconds divided by 100 milliseconds at each pass). 20 passes will be required through the disk node and 1 through the drum node. Then we have:

$$\sum_{i=1}^{i=M} d_i w_i = 1*257.6+80*459.6+20*240 = 41825.6 \text{ msec.}$$

where we have taken the w_i from Fig 5-3. This response time consists of the requested service time (11.18 seconds), plus approximately 1.8 seconds of delay at the disk node and 28.8 seconds of delay at the CPU node.

3. Performance and the Number of Users

We now consider again the model of Fig 5-2, but this time present some of the statistics as a function of N, the number of users. This will demonstrate the behavior of the network as it approaches and passes through the saturation point.

In Fig 5-13 the response time is graphed as a function of N. In this example the minimum expected response time is simply the expected service time request of .805 seconds. At saturation (31.4 users) the expected response time has increased to 3.2 seconds. For N = 40 the expected response time is within 350 msec. of the asypmtotic response

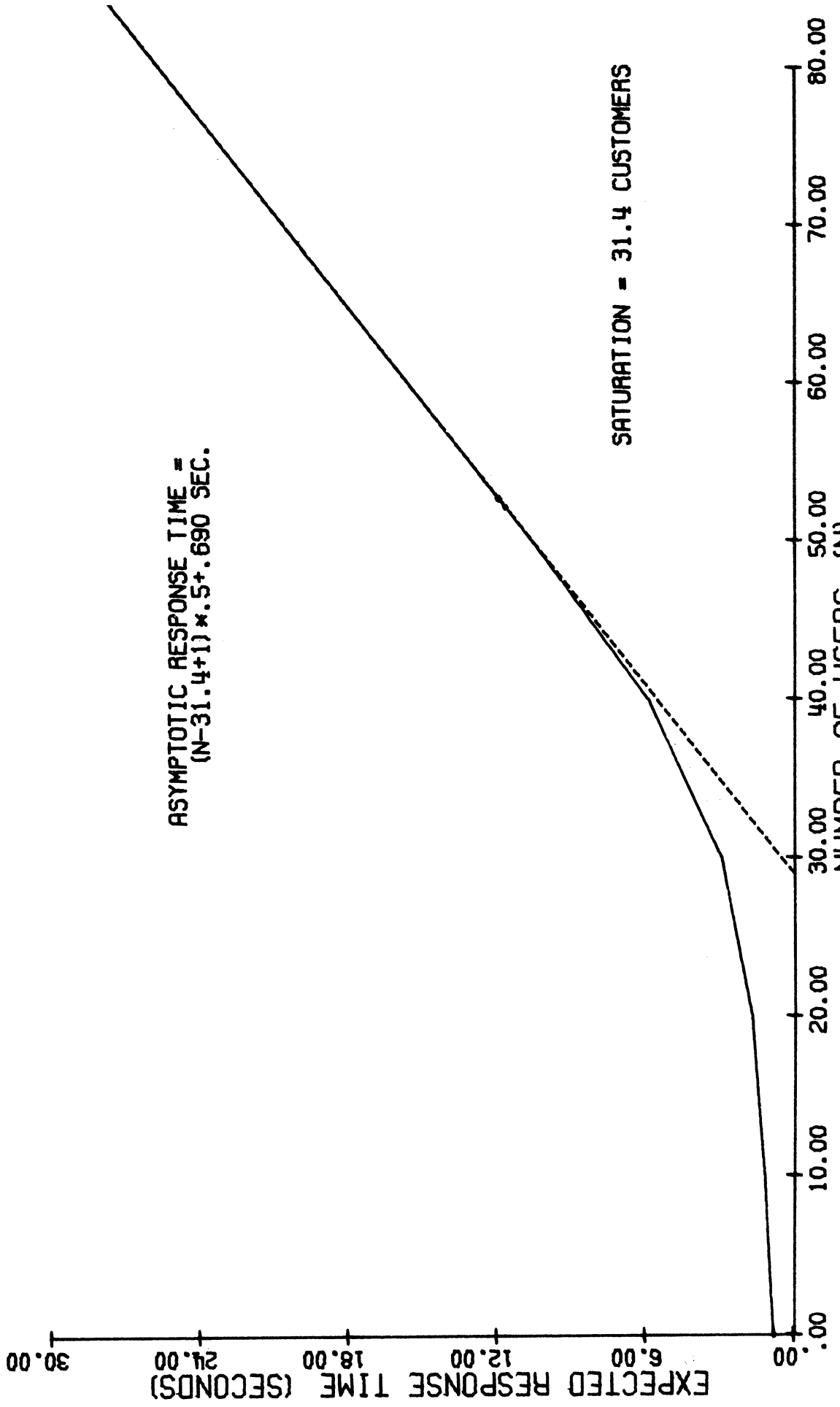

FIG 5-13: RESPONSE TIME VS. NUMBER OF USERS (FIG 5-2 MODEL)

time. For N = 50 it is within 10 milliseconds. This form of the response curve is a function of the finite nature of the network. For instance, in an infinite-source model, 100% utilization of any resource would imply infinite waiting times. The finite source model shows that 100% utilization of a resource is quite attainable and may well occur at a practical operating point of the system. In the example here, if 10 seconds is an adequate response time, the system can be operated with a 50-user population. This is well beyond the saturation point, and 100% CPU utilization is guaranteed.

The node utilization curves, as a function of N, are given in Fig 5-14. All utilizations are within .1% of their asymptotic values for N = 50 (with the exception of the terminal node). The implication of this figure is somewhat startling at first glance: no matter how much the load (number of users) is increased beyond some point, the utilization of the non-saturated resources does not increase. What is happening is illustrated by Fig 5-15, which shows the expected queue lengths at each node as a function of N. Shortly after saturation is reached, all additional users added to the network are effectively added to the CPU queue - a result implied by the existence of a limiting queue length distributions at the non-saturated nodes (Chapter III, section 4).

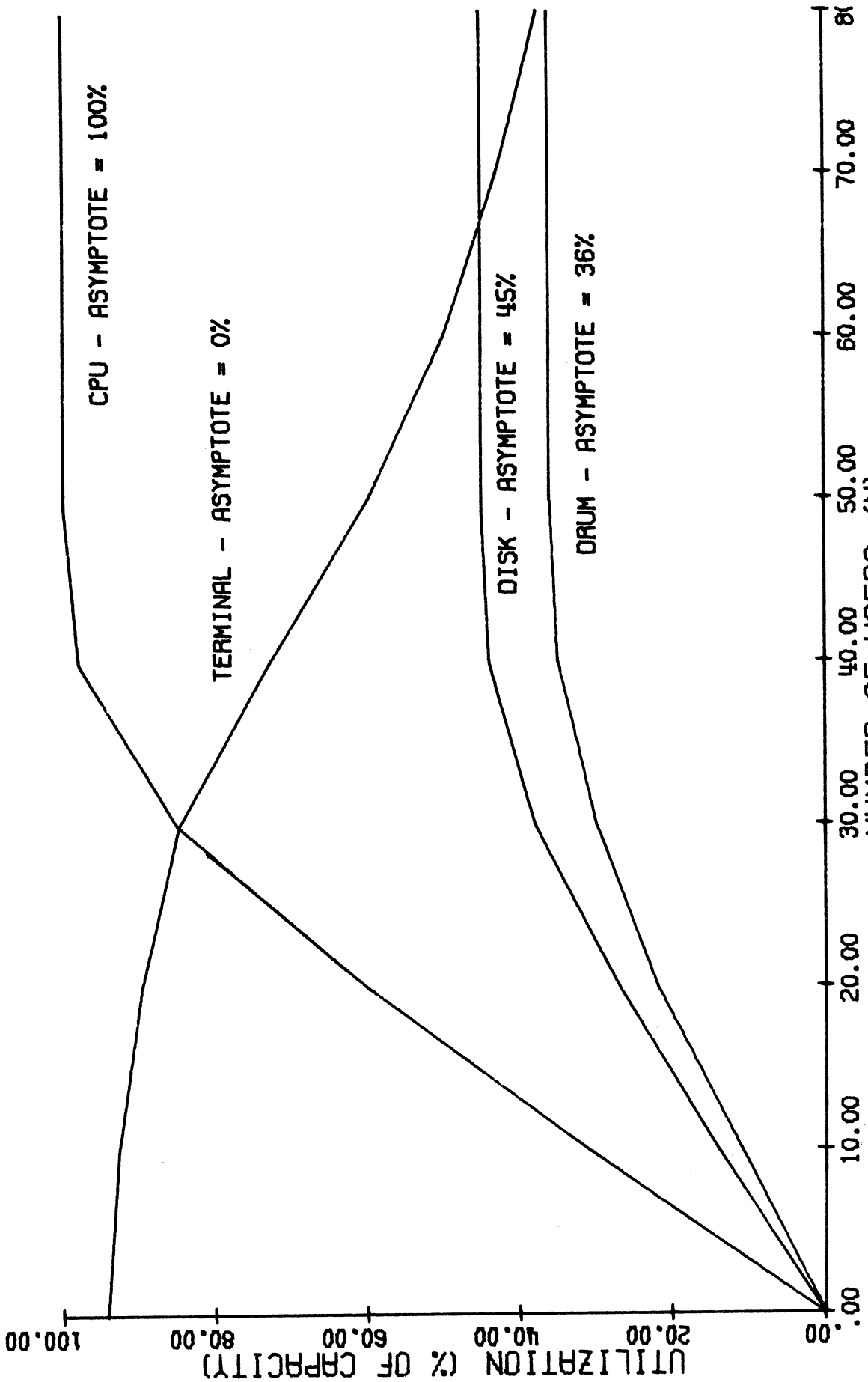

FIG 5-14: NODE UTILIZATION VS. NUMBER OF USERS

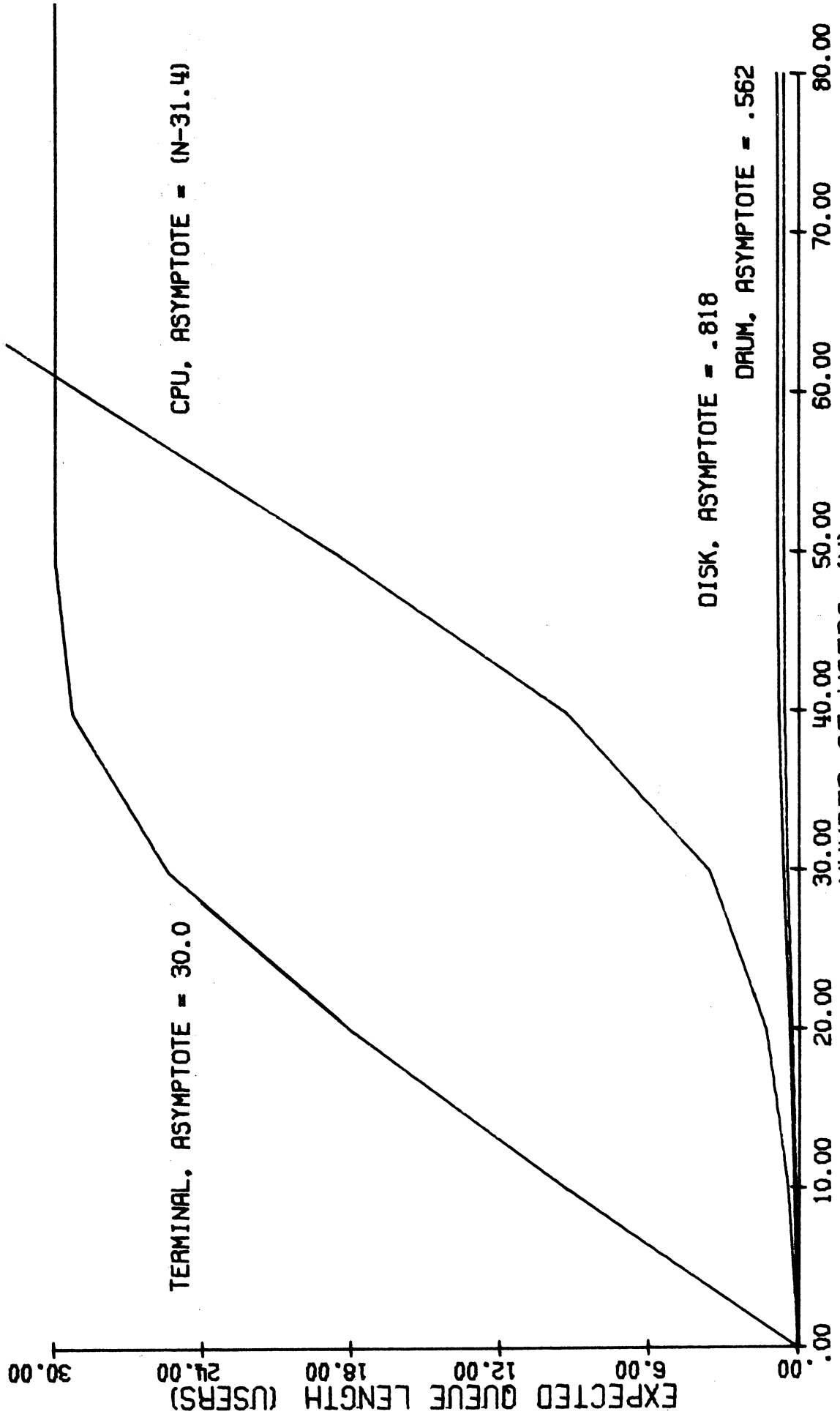

FIG 5-15: EXPECTED QUEUE LENGTHS VS. NUMBER OF USERS

This should not be taken to mean that the only way to improve system performance is via improvements at the CPU node. Decreasing the drum service time, for example, will improve response time, even when the CPU is saturated. However, the improvement to be gained is limited by the amount the drum node contributed to the response time. The drum node contributes a minimum of 180 msec., for N=1, and a maximum of 281 msec., for N>40. Thus if the drum were infinitely fast, it would reduce response time by at most 281 msec. The 281 msec. is the asymptotic expected waiting time at the drum as $N \to \infty$.

Performance Envelope

There is considerable empirical evidence that time-sharing systems are very sensitive to the loads their users place on them. This behavior is generally observed when one or more users usurp a very large proportion of a system resource: memory, the CPU, disk channel, etc. The performance of the system seems to degrade to a surprising degree, considering the incremental load added to the system. It is for this reason that we have suggested the use of performance and load envelopes in the modeling process. The load envelope should be bounded by the expected bounds on user behavior. The performance envelope is then computed by solving the model for the load envelope bounds.

In Fig 5-16 this principle is illustrated for the model of Fig 5-2. The high envelope was derived assuming:

(1) The mean think time had decreased from 15 to 10 seconds.

(2) The mean number of disk operations requested in one IR cycle increased from 1.5 to 3.5 operations.

(3) The mean amount of CPU time used per IR cycle increased from .5 to 1.0 seconds.

With these changes incorporated in the model, the saturation point was computed to be 11.3 users. This is as opposed to 31.4 users in the original model. The response time curve for this model is labeled "HIGH ENVELOPE" in Fig 5-16.

The low envelope was derived by making the following changes in the load parameters:

(1) The mean think time increased from 15 to 20 seconds.

(2) The mean number of disk operations requested in one IR cycle decreased from 1.5 to 1.0 operations.

(3) The mean amount of CPU time used per IR cycle decreased from .5 to .3 seconds.

In this configuration of the network, the saturation point increases from 31.4 to 61.8 users.

The magnitude of these changes in the user behavior parameters (in either direction) does not seem to be too large. Changes of this magnitude might well occur in an

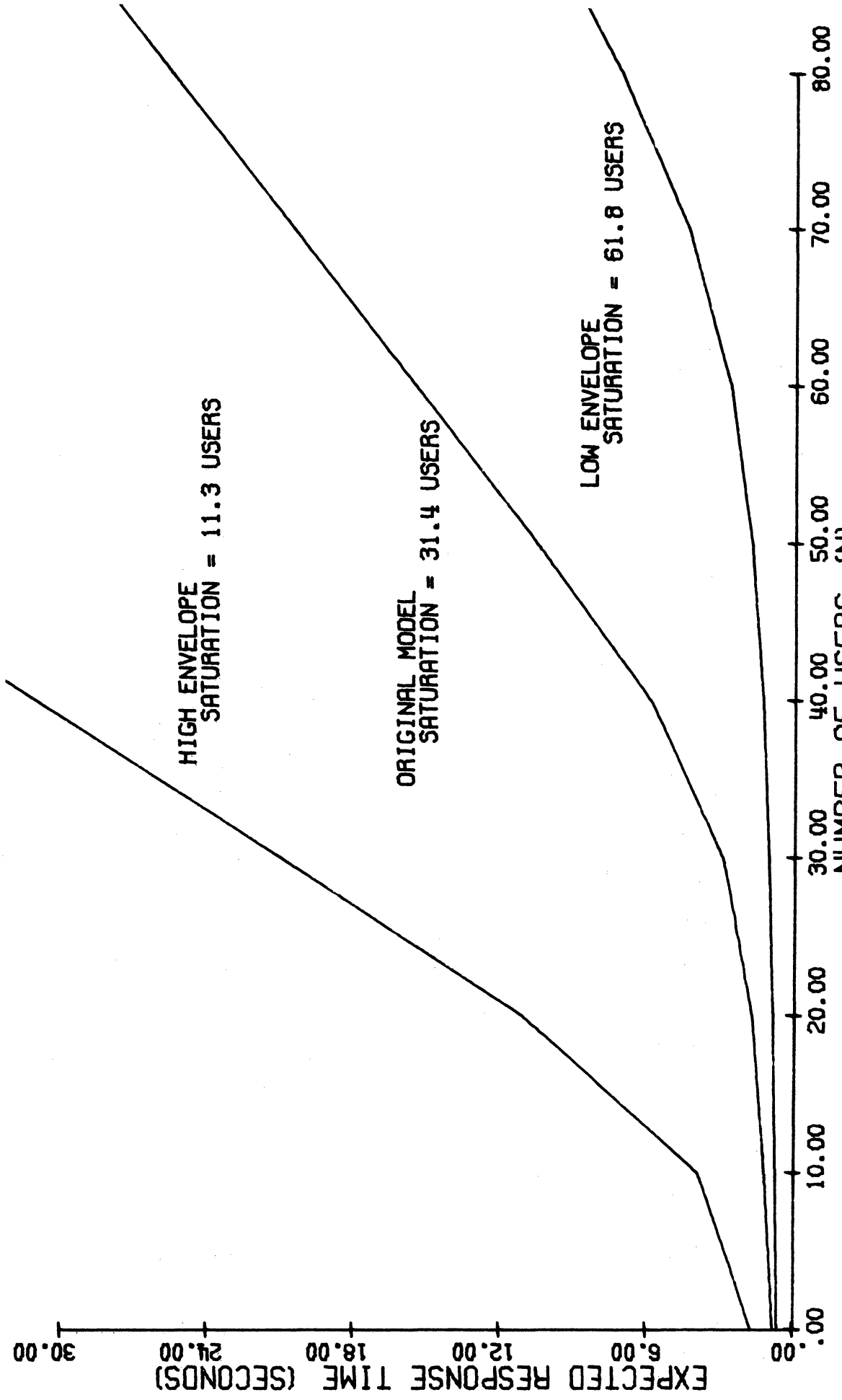

FIG 5-16: RESPONSE TIME ENVELOPE

actual time-sharing system, either as a result of a change in the user mix, or changes in the behavior of the attached users. Yet the range in system performance (as measured by the response time) between the upper and lower bounds of the envelope is very large. For N = 40, the expected response time for the low bound of the envelope is 1.7 sec, while for the high bound it is 31.0 sec. This sort of spread is in part responsible for the seeming lack of success in modeling or "seat of the pants" estimating of system performance before the system is built. Slight miscalculations of the system load (user behavior) can snowball into orders of magnitude differences in the predicted performance measures. We will return to this problem, with some concrete evidence of the shifts in user behavior, in the next chapter.

4. Balancing the System.

As a final example in this chapter, we present a balanced version of the system/model we have been using. In an actual system the options for balancing the system are limited. A continuum of processor speeds is not available. At best the options are to acquire another unit of some type - processor, channel, memory, etc. Being unconstrained by such practical problems, we have developed an almost-balanced model from the original model of Fig 5-2. To do

this, another identical processor was added (making node 3 a 2-server node), and the mean service times at the disk and drum nodes were arbitrarily increased to 170 and 245 milliseconds respectively. The resulting model is diagrammed in Fig 5-17. In this configuration the values of the x_i/μ_i, the parameters which determine the system balance, are 0, .9609, .9804, 1.000 respectively for nodes 1,2,3,4. The terminal node always has a limiting x_i/μ_i of zero, since we assume the number of servers at that node is always equal to N.

The statistics for this model are presented in Fig 5-18 and Fig 5-19. The computed saturation point for this system (by the method of Chapter III) is 206 users. This high saturation point is due to the number of customers absorbed by the CPU node and drum node before the disk utilization reaches 100%. For example, at 150 customers, the expected response time is 23.8 seconds (a point reached at 78 users in the original model). Thus, the derived measure of saturation is not as intuitively satisfying for a balanced system as it was for an unbalanced system. In a sense the model counts balance as a positive factor in the saturation measure. Thus it is a sort of combined measure of efficiency and throughput. We can increase the saturation point by either adding more "power" to the system, in terms of faster servers, or we can balance the system better, increasing the utilization of the available resources.

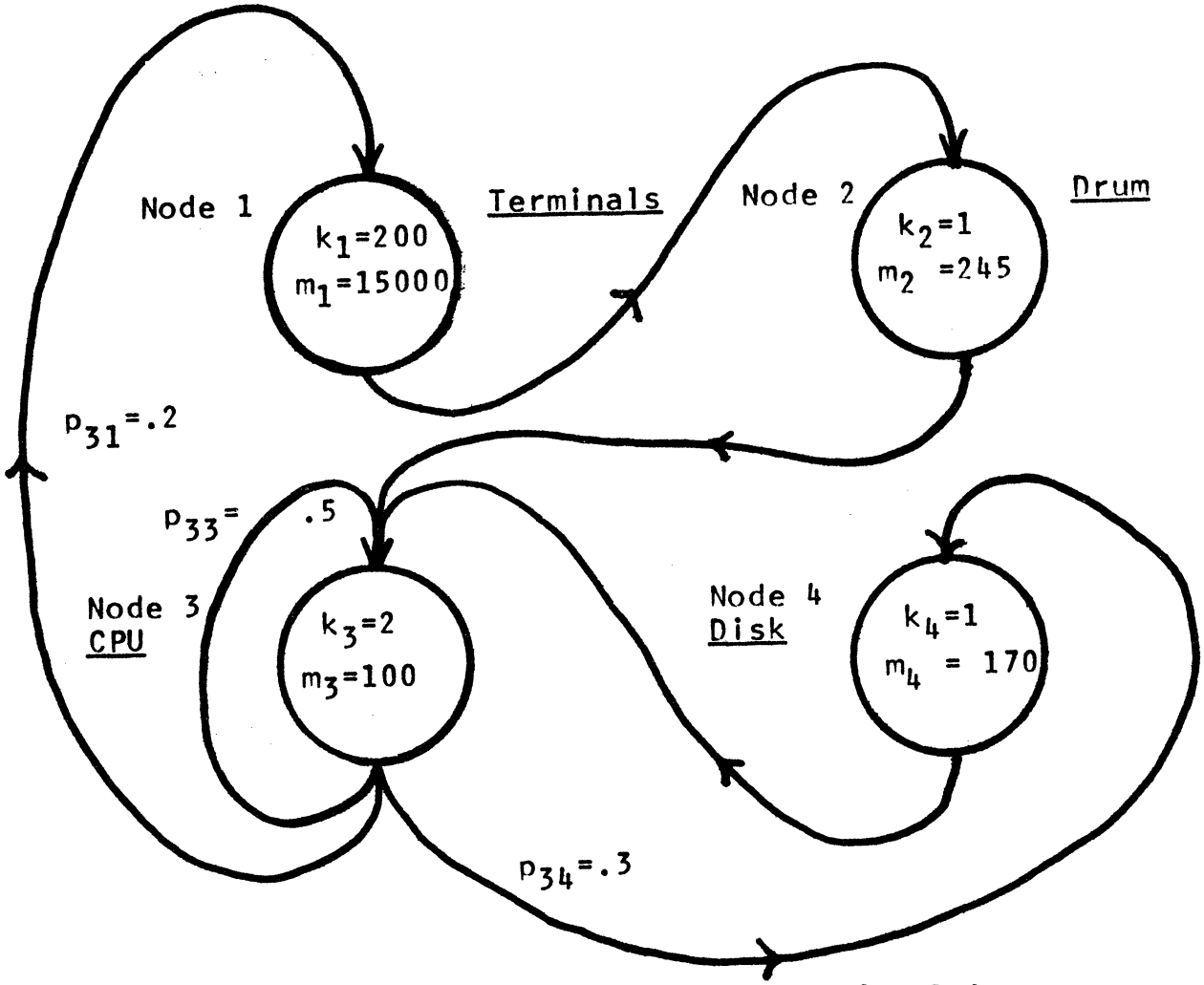

Fig 5-17: Initial Model Modified for Balance

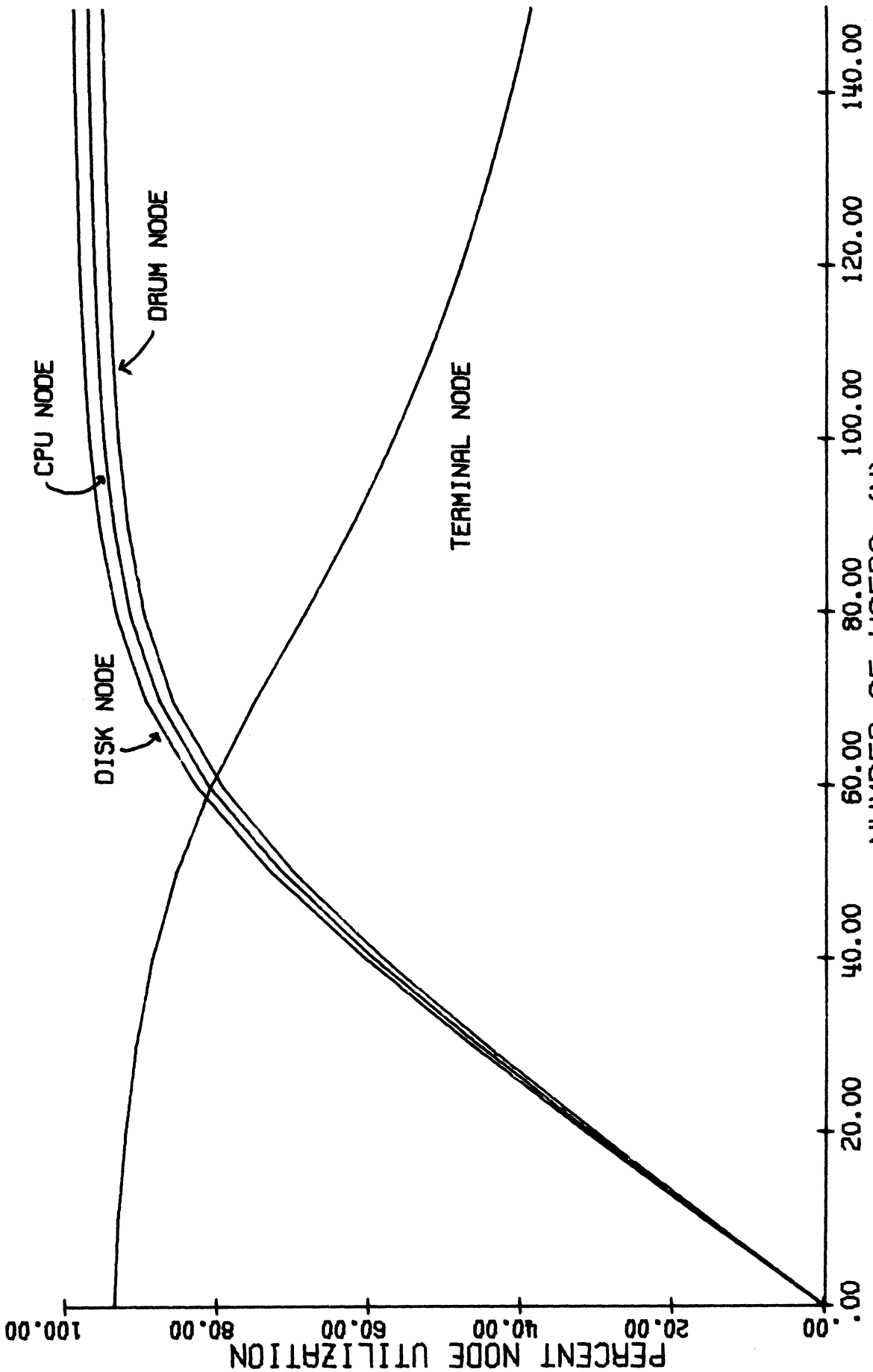

FIG 5-18: BALANCED SYSTEM NODE UTILIZATION

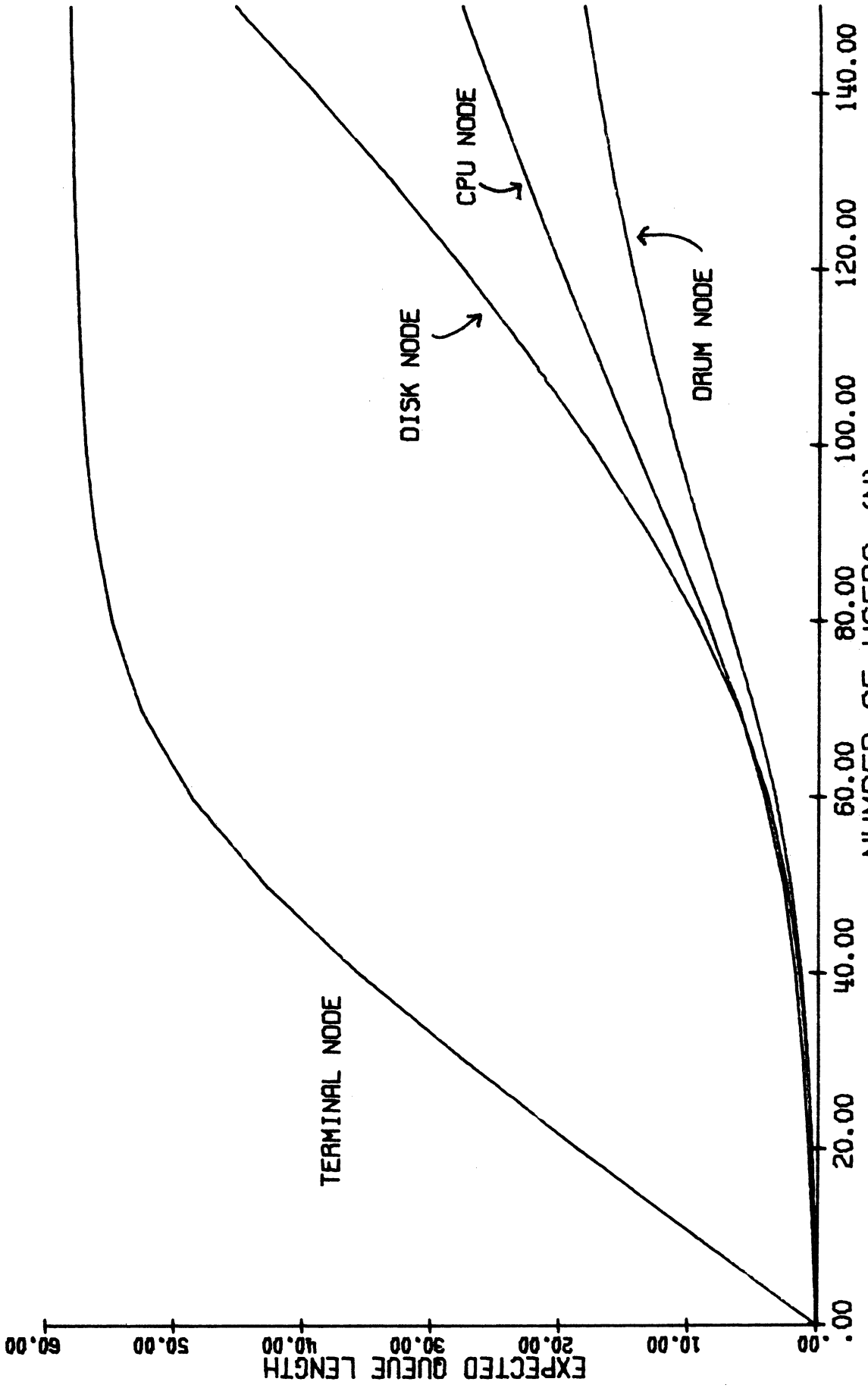

FIG 5-19: BALANCED SYSTEM EXPECTED QUEUE LENGTHS

Chapter VI

A Model for MTS

In this chapter we develop a network model for the University of Michigan Terminal System(MTS)[M4]. We give first a brief description of the system structure, though other references are available[A4,A5,M4]. The rest of the chapter is divided into three major sections,, covering the actual modeling process, and consisting of:

(1) Development of the model from knowledge of the system structure and available statistics. The statistics yield information both about user behavior and the actual performance of hardware modules.

(2) Validation of the model, comparing the predicted performance of the system with its empirical performance. We will be able to investigate a number of performance measures, as well as the concepts of system balance and system saturation developed in Chapter IV.

(3) Investigation of possible changes in the system structure and prediction of the system performance outside the normal operating range of the system. In view of the limited number of alternatives for the hardware configuration, it will also be possible to search for an optimal configuration within given constraints.

1. A Brief Description of MTS

MTS is a large-scale time-sharing system, developed at the University of Michigan. The system provides both batch and terminal services to the students and faculty at the university, as well as to a number of research projects. The system can be "partitioned", so that not all the equipment is operating at one time (and in fact this is not infrequently the case). The full system configuration is diagrammed in Fig. 6-1. The discussion which follows outlines the system structure from a hardware and operating system point of view. It assumes the entire configuration of Fig. 6-1 is operational.

Processors(CPUs): 2-IBM 360/67 processors.

The 360/67 processor is basically a Model 65 with segmentation and paging hardware added[A6]. The segmentation and paging (as used by MTS) allow the user a virtual addressing space of 1280 pages, or about 5 million bytes. The limitation is imposed by the operating system, not the hardware. Actual average program size is in the order of 30 pages[A5]. The internal cycle time of the processor is 200 nanoseconds. The processors are identical; there is no master-slave relationship in terms of hardware or software. Major

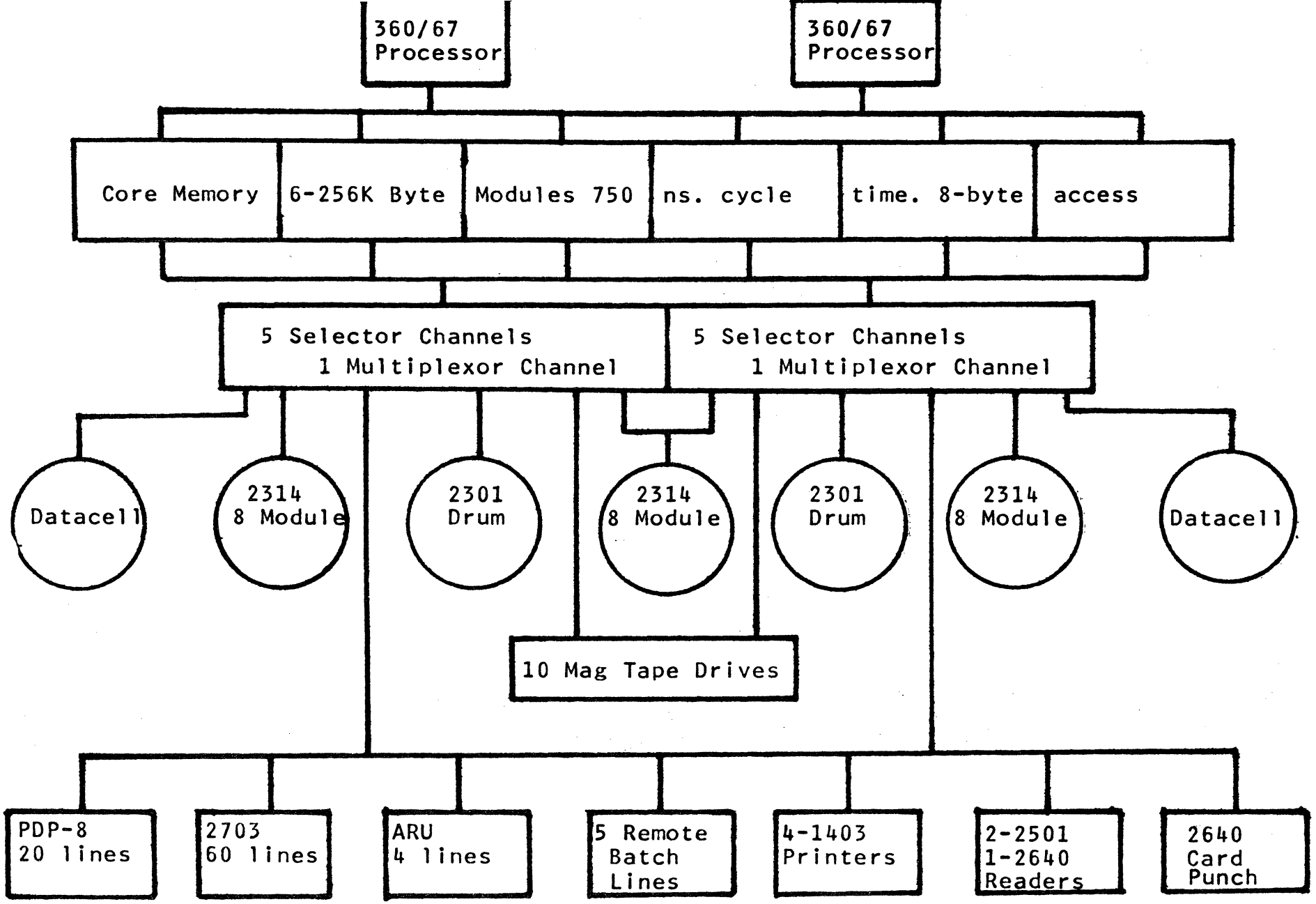

Fig 6-1: MTS SYSTEM BLOCK DIAGRAM

system software components are reentrant, allowing them to be executed simultaneously by both processors.

Core Memory: 6 - 256K byte modules.

Access to the memory is independent for each of the six modules, allowing considerable interleaving in memory accesses. In addition there is two-way interleaving within each 256K byte module. Each access to memory fetches a double word (8 bytes). Memory cycle time is 750 nanoseconds. Under these circumstances, processor speed degradation because of memory access conflicts is not a serious problem.

Of the 385 available core memory pages, approximately 36 are allocated to the supervisor and a number of other resident support routines. Another 34 pages are allocated to shared, reentrant programs. They may be paged out, but most are effectively resident. The rest of core memory is available for dynamic allocation to tasks.

Input/Output Channels: 2 Multiplexor and 10 Selector Channels.

In the 360 system architecture, the multiplexor channels are used to simultaneously manage a number of low speed input/output devices, such as printers, card

equipment, terminals, etc. The selector channels attach the drums, mass storage devices and tape drives to the system. Since there are as many channels as devices, channel contention is not a problem. Thus the channel structure will be ignored in modeling MTS: we simply consider each I/O device of Fig 6-1 as having its own path to memory.

Paging Storage: 2 - IBM 2301 Drums.

The primary residence device for pages swapped out of core are the 2301 drums. They are used only as paging storage. Their capacity is 900 pages each. The revolution time of the drum is 18 milliseconds. It is possible to write 4.5 pages in one revolution. Because there are not an integral number of pages written around the drum, it is often more convenient to treat the drum as having a revolution time of 36 milliseconds, and 9 pages written around it (9 sectors).

If the drum space is exhausted, paging activity may also take place on one or more 2314 disk modules. This overflow procedure is currently necessary only under abnormal conditions, as the usual system load does not exceed the drum capacity.

File Storage Devices: 3 - IBM 2314 Disk Storage Units
2 - IBM 2321 Datacells

Each 2314 contains 8 disk modules. Each module is accessed by an independent arm. Each module stores about 28 million characters. The rotation time is 25 milliseconds. The arm movement time ranges from 25 milliseconds for a one cylinder seek to 135 milliseconds for a full 200 cylinder seek. 140,000 characters can be accessed from one arm position of one module. There is one data transmission path from each 2314 unit to memory. Arm movement may be in progress on all modules simultaneously, since this requires no data transmission.

Larger capacity storage, at slower speeds, is provided by the 2321 Datacells. They have a capacity (together) of about 800 million characters, stored on magnetic strips which can be retrieved, wrapped on a drum, and read automatically by the 2321. Access time ranges from 95 to 600 milliseconds. There is one path to memory from each 2321.

Terminal Equipment

The majority of terminals are low-speed hard-copy devices such as teletypes or IBM 2741-like typewriters. There are 80 available lines for such terminals. There are also 6 lines for terminals in the 1200-2000 baud speed range. Four lines to an audio response unit allow touch-tone phones to be used as terminals. Two on-line displays (IBM 2260's) may also be used as terminals.

Terminals may connect to MTS through IBM communications equipment or through a greatly augmented PDP-8[M3]. The PDP-8 (called the "Data Concentrator") serves as a buffering device, and also performs a number of communication services such as line editing, code translation and so forth.

Batch Support

Batch service, which is carried on in parallel with terminal user support, is controlled by a "spooling" program called HASP which queues batch job card decks on the 2314. Printed and punched output is also placed on the 2314 before actual processing by the unit record equipment. Thus all the printers, card punches,

and card readers are operated by HASP, not by user programs. The unit record equipment available consists of 4 high speed printers, 3 - 1000 card per minute card readers and a single card punch. HASP also supports up to five 2000 baud remote batch terminals operating on the switched network. They are treated as extensions of the local unit record equipment.

Other Equipment

Other peripheral equipment attached to the system may be allocated to terminal or batch tasks on request. These devices include 10 magnetic tape drives, a paper tape reader, and two paper tape punches (both operated as peripherals of the Data Concentrator).

System Operation

What is normally called a "process" in most literature on time-shared operating systems is called a "task" in MTS. We shall adopt the "task" terminology, since it is consistent with other documentation. The reader should feel free to think "process" every time he sees "task" if that concept is more familiar to him. Tasks come in two major varieties in MTS: virtual memory tasks and real memory tasks. Real memory tasks are primarily "overhead" tasks such as the tasks which manage HASP (the spooling system), tasks

which handle paging, and so forth. The real memory tasks are resident and cannot be paged, though they may acquire additional real memory for their use, reducing the number of core pages available for other tasks. The real memory tasks represent a relatively constant overhead in terms of system demand. They operate primarily on dedicated peripherals and use fairly fixed percentages of real core and processor time. Therefore we shall ignore them in the modeling process, since their effect is simply to make the effective real memory somewhat smaller and the processors seem slower.

The virtual memory tasks consist primarily of terminal and batch tasks. There is one task for each terminal user, and one task for each batch job. Once the task has been started, the system does not make any distinction between a batch-job-task and a terminal-user-task. No attempt is made to discriminate for or against either type of task in terms of resource allocation. There is however discrimination at task startup time. MTS may run anywhere from 1 to 10 batch tasks simultaneously. The number to run is based on the system load factor. The load factor is a number which is computed from the following system statistics:

(1) Average CPU queue length and CPU utilization.

(2) Average number of page-in operations per second.

(3) Average number of disk operations per second.

(4) Number of virtual memory pages allocated.

If the load factor is computed to be 85, for example, at most three batch tasks will be run simultaneously. If the load factor is over 100, only 1 batch task is run. Further discussion of the load factor concept may be found in the Appendix. Only the last two statistics sessions occurred after the installation of this feature, so its strong stabilizing effects were absent from the bulk of the statistics.

Processor Scheduling.

The processors are scheduled on a preemptive basis with a quantum of 27 msec. Each task, when it becomes ready to use a CPU, generally preempts the current task, and is assigned the CPU until it in turn is preempted, or its quantum is exhausted. When the quantum is exhausted, the task is placed at the end of the CPU queue and receives a new quantum. Tasks which preempt the processor are not assigned a new quantum, but continue to decrement their current one. Much of the motivation for the CPU scheduling algorithm is to give fast service to real memory tasks controlling unit-record equipment such as card readers and printers. These tasks require fast access to the CPU to ensure

high utilization of the devices they service.

A special case is made for the scheduling of tasks with very high virtual/real memory demands. This case is covered in the virtual memory management section following. All other tasks (including real memory tasks) are treated uniformly without regard to type or other characteristics.

Virtual Memory Management

The paging policy is relatively simple. Pages are brought in on demand, that is whenever a program attempts to access a page which is currently swapped out. Normally any such page request is serviced immediately (or queued for service immediately). Tasks which have exceeded a page threshold are treated specially. The page threshold is dynamically determined by the supervisor. When the real pages allocated to a task pass the page threshold, the task is placed in the privileged state. There is a single queue of privileged tasks. Only a small number of them (based on the current system configuration) are allowed to be active at one time. The currently active privileged task is allowed to make (and is granted) arbitrary page requests until a special quantum runs out. This quantum is about 200 milliseconds of CPU time. Tasks

which have exceeded the page threshold and are currently being prevented from competing for system resources are called non-privileged. When a privileged task's quantum expires, the task is placed at the end of the privileged task queue, and the next privileged task begins its quantum of special treatment.

Actual page-in requests are handled by a separate real memory task, the paging drum processor(PDP). In addition to swapping in requested pages, the PDP also writes out inactive pages, making room in memory for pages requested more recently. For a complete discussion of the operation of the PDP, see Alexander[A4].

The 2314 disk is used as a paging device only if the drums are full. Page-out operations initiated when the drum is fully allocated automatically go to the disk. No attempt is currently made to select particular pages for disk paging. No significant amount of disk paging took place during any of the statistics gathering sessions.

File Management

Files are stored on disk or datacell at the request of the user creating the file. The files are allocated, read, and written by shared reentrant subroutines, not (as in the case of the PDP) by a separate and independent task. This means that no queue of file operations is maintained. A task wishing to perform a file operation waits until the appropriate module is free, initiates the desired operation, then waits until its operation has completed. All files are randomly accessed and buffered in core. Buffer space allocated to a file ranges from 2 to 5 pages.

All permanent files, public or private, reside on the 2314's or datacells. Most public (library) files are kept on the 2314's. Compilers, utility routines, subroutine libraries, etc. are all kept as relocatable object modules on the 2314's. This (as we shall see) generates considerable 2314 traffic during the normal processing of an MTS task.

Terminal Management

Terminals are managed directly by the MTS task associated with each device. Shared reentrant code is used and appears in the address space of each task. This is in direct contrast to the method proposed for MULTICS[D6]. There seems to be no obvious advantage of one scheme over the other, as long as reentrant code is used. No queueing or buffering is done for terminals, except for those connected to the data concentrator, in which case queueing facilities are provided in the PDP-8 memory. There is little difference in the treatment of the various types and speeds of terminals, except those necessitated by any differences in functional capabilities. In particular, a device connecting to MTS through a 2000 baud line is indistinguishable (from a resource allocation viewpoint) from the usual 110-150 baud range terminals.

Other Devices

In this context, "other devices" means magnetic and paper tape equipment. Printers and card equipment are managed exclusively by HASP, and rarely addressed directly by MTS tasks. Tape equipment is assigned dynamically to individual tasks on request. It

receives most of its usage from batch users, rather than terminal users. All these devices are characterized by the fact that the device is dedicated to a single task. Thus no queueing takes place at these devices: waiting times are determined completely by the operation requested, not by other activity in the system. There is one exception to the preceding statement: there are only two paths from memory to the 10 tape drives, so channel queueing may occur, even though the device itself (the drive) is dedicated to a single task.

2. MTS Model Development

There are two "inputs" to the modeling process when one is modeling an actual time-sharing system. One is the structure of the system. This structure is essentially that imposed by the hardware characteristics and the operating system policies. The structural component of MTS was outlined in the previous section. We now proceed to integrate that information with the "stochastic" element of the modeling input: the user behavior. We use the term broadly enough to include both the actual behavior of the users, such as how long it takes them to type in an input line and the "behavior" of the process the user invokes in the system.

A basic characteristic of a general purpose time-sharing system is the stochastic nature of the knowledge about a user's process. In a "dedicated" or "real-time" (as opposed to "general-purpose") system the future course of a process or task is generally well known. In fact the system is often carefully designed to take advantage of this highly predictable behavior. For instance, in an airline reservation system, knowledge about the limited types of possible input requests may make it possible for requests of a particular kind to be gathered and processed simultaneously for efficiency. In a general-purpose system (such as MTS), on the other hand, the operating system generally makes no fixed assumptions about the future demands a given process will make on the system. When assumptions are made they are made probabilistically on the basis of information on the past behavior of the process. In systems in which the future demand is fairly well known or in which deadlines must be met, the modeling techniques suggested here may well prove unsatisfactory.

The rest of this chapter depends heavily on statistics gathered about the operation of MTS. These statistics were collected during fifteen "statistics sessions", at various times and under various loads. A complete discussion of the statistics gathering procedure is presented in the Appendix. A summary of some of the relevant

statistics from each of the sessions is presented near the end of this chapter, in Fig 6-19.

Deciding on the Nodes

The first decision to be made in modeling MTS is that of the system resources to be modeled in the network. We view the interaction-response cycle (IR cycle) of a user in the system as consisting of two major parts. The interaction portion consists of the period while the user is typing in his next service request at the terminal, and is making little or no demand on system resources. The second, or response, portion of the cycle covers the period while the system is fulfilling the request indicated by the input line (or triggered by the input line). It can itself be broken into several sub-segments: periods when a resource is in use and periods when a resource is needed, but unavailable. The response time is the sum of these queueing and service delays. The candidates for the nodes then are the queueing points in the system, the points at which a process may be delayed for some reason: either waiting for the resource to be made available, or using the resource. In MTS, such delays may occur for any one of the following reasons:

(1) CPU Wait

The process is waiting for a CPU, or is currently using one.

(2) Input/Output Wait

The process is waiting for some input/output operation to complete, or for a device to become available so that an operation may be initiated. The wait may be for disk, datacell, magnetic tape, etc.

(3) Page Wait

The process is unable to continue because a needed page of virtual memory is unavailable. The page may be unavailable because (1) there is no real memory available to assign to the virtual page, (2) the requested page has not yet been read in from the drum or disk, or (3) The process has been relegated to the non-privileged state, making it ineligible to acquire further real memory.

(4) Lock Byte

The process is waiting for the state of some byte in memory to change. This indicates that some other task is currently using some resource which is non-shareable, and may be a file, subroutine, table in memory, etc. An example is the "accounting lock" which prevents more than one process from modifying the system accounting file at any given time.

There are certain other more esoteric types of waits - for instance a wait for a certain amount of (real) time to

pass - but these occur infrequently enough in MTS processes to be ignored here.

A number of alternatives were considered for the set of resources to be represented in the model. Some choices were clear, but in other cases experimentation was necessary. It is not possible to simply represent each possible cause of delay in the system as a separate node - exact solution to such a large problem would be impractical, even with computer techniques. Thus some sort of "minimal" model which still provided satisfactory results was sought. When any serious question arose, both alternatives were tried, and if the simpler alternative proved satisfactory, it was adopted. Eventually a model was developed which proved adequate and contained nodes representing the following system resources:

(1) The CPU

(2) The 2314 Disks

(3) The 2301 Drums

(4) Terminal Output and Other Delays

(5) Terminal Input

Each of the above resources will be represented by one or more nodes in the network model of MTS. Since they may not be represented by a single node, we will refer to the set of nodes representing a resource as a meta-node. Thus, as we shall see, the terminal input behavior of the

system users will be represented by two nodes, and these nodes together will be called the "terminal input meta-node".

The relationship between these nodes and the various causes of delays in the system are now discussed separately for each of the nodes.

The CPU Node

Except for the preemptive scheduling algorithm used for CPU scheduling, the CPU queue and service are fairly well represented by a 2-server facility with exponential service time distribution. This is not surprising since interrupts often cause preemption of the CPU, and there are generally 50-60 independent potential interrupts (mostly I/O operations) outstanding at any one time. These interrupts are likely to arrive in something resembling a poisson pattern, yielding exponential inter-arrival times, since the superposition of a large number of independent stochastic processes will appear to be poisson.

The observed distribution of CPU 'active intervals' is shown in Fig 6-2. An active interval is defined as the length of time the processor was continually allocated to a single task. It should be noted that under certain circumstances, a task may be active on <u>both</u> CPU's at once, with the supervisor performing some service for the task on one

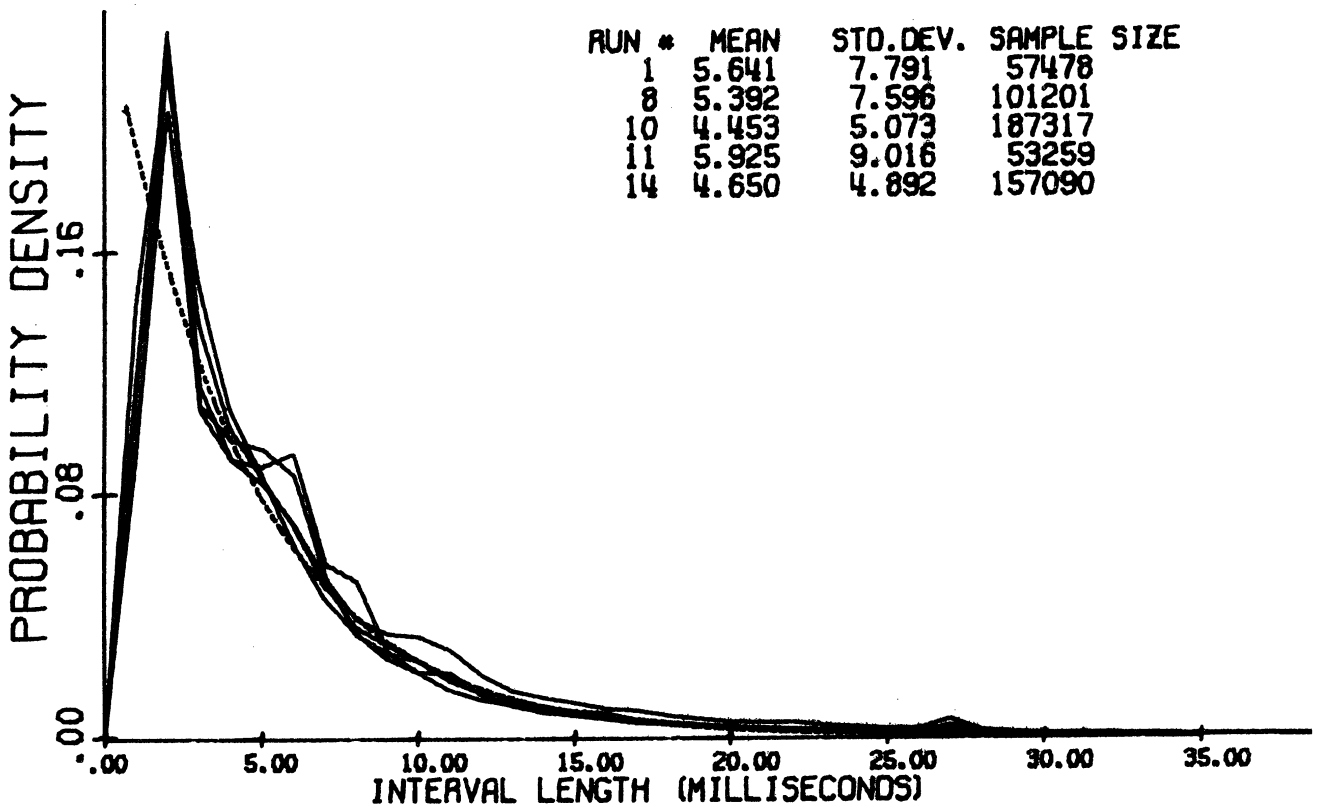

FIG 6-2: CPU SERVICE TIMES (ACTIVE INTERVALS)

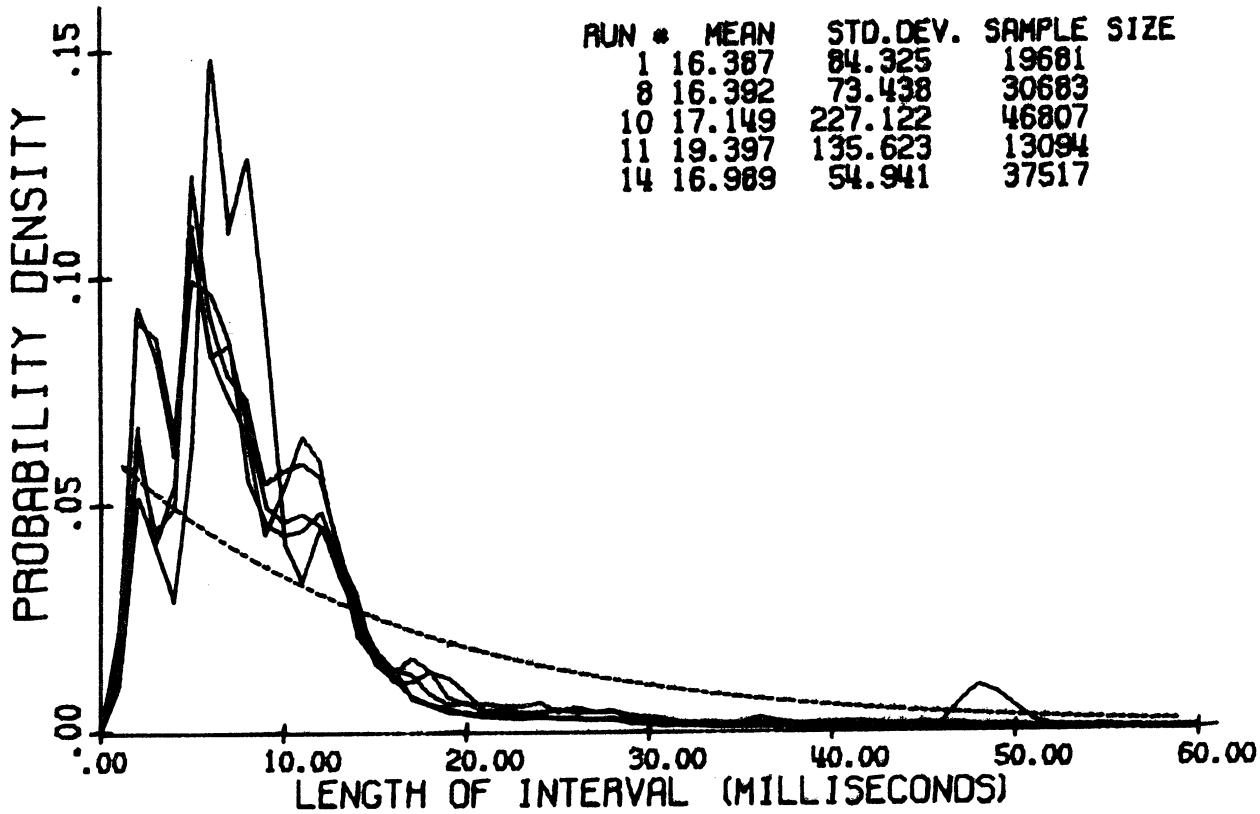

FIG 6-3: REQUESTED CPU ACTIVE INTERVALS

CPU, and the user task running on the other. In this case an active interval is computed for the use of each CPU.

In Fig 6-2, and in many of the subsequent presentations of empirical density functions, the observed density curves for four or five of the statistics gathering sessions are graphed as solid lines. The only reason for not presenting more of the 15 density functions available (see the Appendix) is that the graphs become increasingly indecipherable. Summary data for all 15 sessions is presented in Fig 6-19 of this chapter and in the Appendix.

The means of the active intervals tend to decrease with an increase in the system load, as one might expect with a preemptive scheduling algorithm. It would be possible to represent this directly in the CPU meta-node of the network model by making the service rate depend on the queue length. However, in view of the results of the previous chapter, another alternative is possible. We may compute the service rate parameter at the CPU meta-node on the basis of requdo<9yRA(d CPU intervals. This is what the distribution o the active intervals would have been if the scheduling algorithm was FCFS and not preemptive. The observed distribution of the requested active intervals is given in Fig 6-3. Note that the effect of the preemptive scheduling of the CPU's is to increase the number of task switches by a factor of 2 to 4, but that the quantum limitation of 27 mil-

liseconds is not often reached, even in the requested interval distribution. The dashed line in the figure is an exponential density function with the same mean as the Session #10 density.

For our initial model, we will use the actual (rather than the requested) CPU active intervals to compute the model parameters. As Fig 6-2 shows, the mean and standard deviation in each of these distributions is approximately equal. Thus the CPU meta-node in the MTS model consists of a single node with two exponential servers, one representing each of the CPU's

The 2314 Disk Node

The 2314 disks were chosen for separate representation because of the heavy use of the 2314's by MTS tasks. The total observed causes of input-output delays in MTS tasks (over all statistics sessions) were as follows:

Cause of Wait	Number	Percentage
Disk (2314)	152,087	51.7
Drum (Pagewait)	67,163	22.8
Datacell (2321)	8,837	3.0
Terminal Output	37,392	12.7
Terminal Input	9,975	3.4
All Others	18,916	6.4

It seems clear then, that if any I/O devices are to

be represented individually, one should be the 2314's. Decisions for representation were not made completely on the basis of such subjective judgements. If there was any question about the importance of some system resource or cause of delay, a version of the model was run with and without the questionable representation, and the results compared for significant differences.

The 2314 operation time distributions are not well represented by an exponential distribution, as can be seen from the observed density functions plotted in Fig 6-4, for five of the statistics runs. Despite the rather strange appearance of the density functions, note that the mean and standard deviation of each of the densities are approximately equal. Thus to match the first two moments of the distribution, a satisfactory choice is a simple exponential. We thus represent the 2314's in the network model by a single node, with 3 servers, representing the 3 simultaneous paths available from the 2314's to memory.

Actually, it is not possible to access each of the twenty-four modules from each of the three channels. Each set of 8 modules is served by a single path to memory. However, because of the relatively large number of modules and the fact that arm movement may take place simultaneously with data transmission, channel interference is not normally a significant problem. If it were a problem, the 2314 meta-

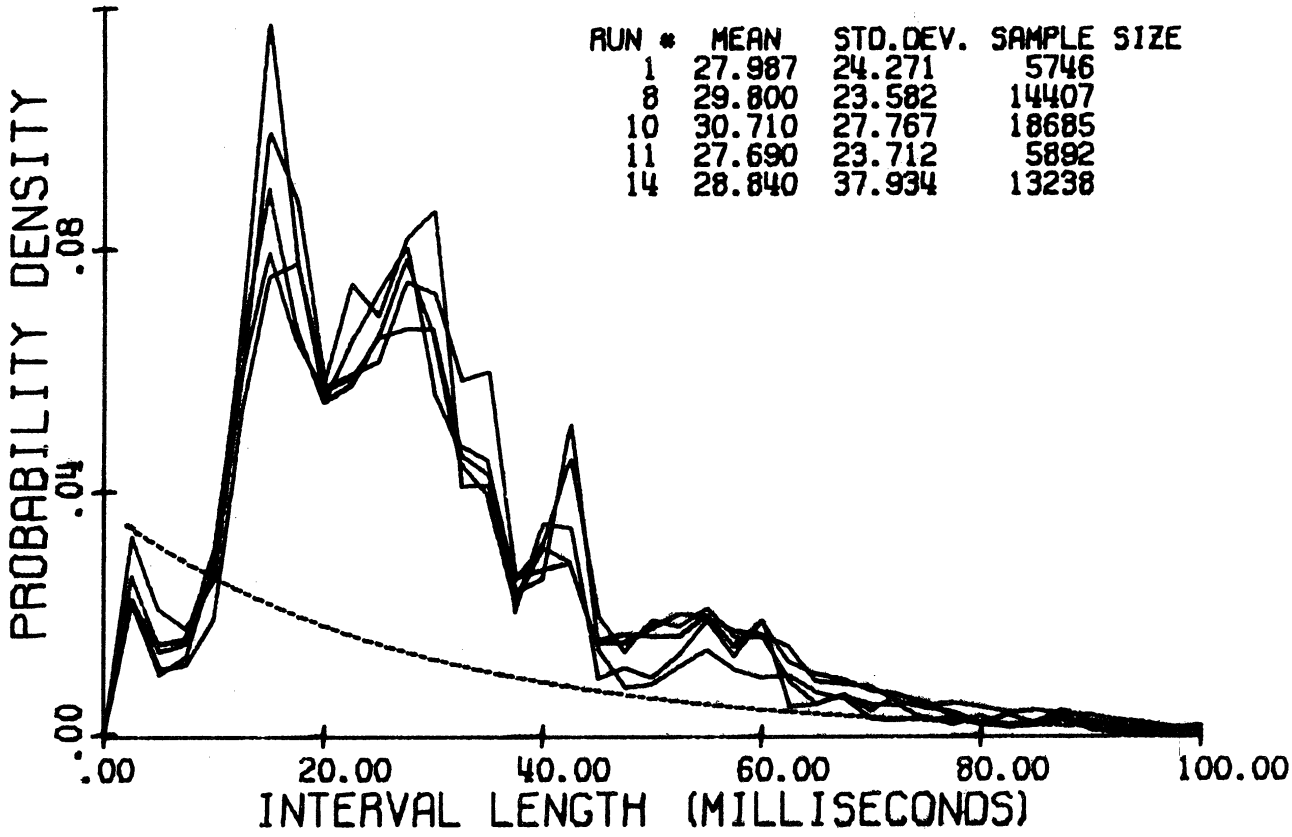

FIG 6-4: 2314 OPERATION TIME DENSITY

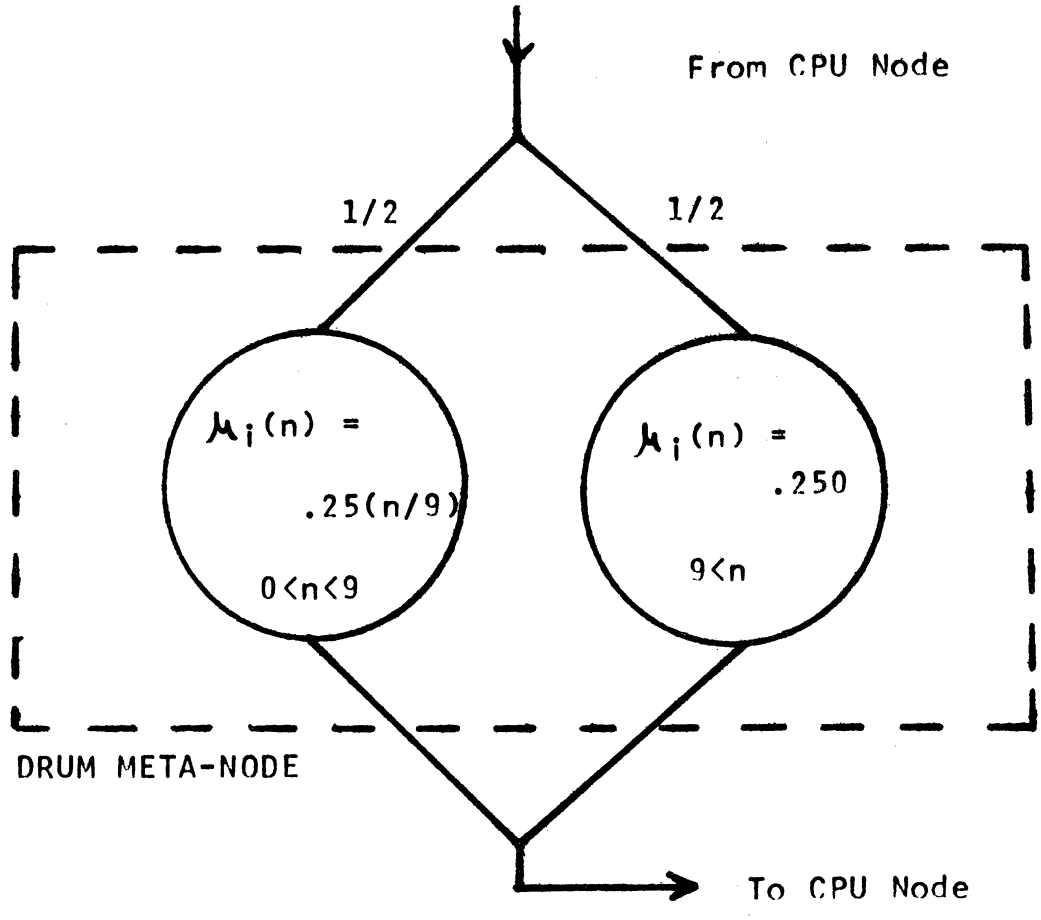

Fig 6-5: Representation of Drum in Model

node could be constructed from three nodes, each with a single server. The transition probabilities to each of the nodes could be chosen on the basis of the observed transition probabilities, or assumed random.

The availability of the statistics of Fig 6-4 removes the need for a model of the 2314 system to predict operation time distributions. If such statistics were not available, we might use the model of Abate[A2] to predict this distribution.

The Drum Node

The node(s) in the network which represent the drum must in fact represent the entire paging process and virtual memory management. Since it is not possible to represent real memory as a resource, the drum meta-node also serves to represent any delays incurred due to conflicts over the availability of real memory, as well as the actual paging mechanisms.

The operation of the MTS virtual memory management was studied by Pinkerton[P4] and is described by Alexander[A4]. We represent a page fault in a task by the occurrence of a transition to the drum meta-node. Since the transition probabilities in the network are fixed, the page fault rate must be linearly related to the rate at which customers exit from the CPU node. The validity of this

assumption is considered at the end of this section when the transition probabilities for the network are developed.

In addition to the page-in operations requested by a page fault, MTS must also write pages out to free real memory for incoming pages. The algorithm used to do this is designed to cause as little interference as possible with the page-in operations[A4]. There is considerable evidence that virtually all the page-out operations are done at times when the drum channel would otherwise be idle. Thus we may effectively ignore the page-out process in modeling the paging process.

The rate at which pages are read from the drum depends heavily on the length of the operation queue. The drum in MTS is treated as a 9-sector drum, with separate queues for each sector. For the discussion of the performance of such drum scheduling schemes see any one of Denning[D2,D8], Coffman[C4] or Pinkerton[P4].

We model the drum as two nodes each with a single server whose service rate parameter varies with the length of the queue at the node. Since there are two drums, on separate channels, we represent each drum as a node (Fig 6-5). The probability of entering either one of the identical nodes in the meta-node is one half. This is an assumption which the page placement policy of MTS is designed to make valid. The rotation time for the drum is 18

milliseconds and there are 4.5 sectors in one revolution. Thus the maximum page transfer rate per drum is approximately 250 pages per second, or 1 page every 4 milliseconds. We will use a linear approximation to the increase in transfer rate with queue length predicted by Denning[D2]. The mean service rates, $\mu_i(n_i)$ at each drum are then as follows:

$$\mu_i(n_i) = .250(n_i/9) \qquad 0<n_i<9$$
$$\mu_i(n_i) = .250 \qquad 9\leq n_i$$

There are a number of discrepancies between our model of the drum and its actual operation in MTS. For example, a minimum of 4.0 msec. is required to transfer a page from the drum. This lower bound on the service time cannot be represented in the network without an arbitrarily long series of exponential servers appearing at the drum metanode.

An alternative structure which is also more realistic is to represent each drum by nine parallel nodes, one for each of the drum sectors. This is in fact the organization which exists in MTS, where a separate queue is maintained for each node. The service rate parameter at each node would then reflect the logical rotation time of the drum, which allows the transfer of one page every 32 milliseconds for a given sector.

The MTS network model was run with three alternative structures, one being the 9-parallel-node representation just discussed. The other two were the final one, discussed at the beginning of this section, and an Erlang-4 version used to achieve a more deterministic service time. All three representations yielded virtually identical results, so the simplest version - two identical parallel nodes - was chosen. If the drum node received heavier use in MTS, one of the more accurate representations might be required.

Terminal Output and Other Delays

This meta-node serves as a "catch all" to represent all the other possible delays incurred by a process in the system. These may be of several types, and we list them here with their observed probability of occurrence (combining again all statistics runs):

Terminal Output	63.1%
Datacell Operation	10.5%
Magnetic/Paper Tape	5.6%
Lock Byte	.1%
Other	20.7%

The overwhelming majority of these miscellaneous waits are clearly for terminal output. The "Other" entry (20.7%) covers some waits which may have been actually for one of the preceding entries, but were not classifiable on

the basis of the information available on statistics tapes. It also includes those waits caused by the occasional use of unit record equipment (printers, primarily) by terminal tasks.

We will represent the service here as occurring in parallel fashion to all customers at the node (i.e. - the service rate is proportional to the number of customers at the node). The service rate parameter is simply the weighted average of the waiting times for each of the various delay causes listed above.

The parallel service structure implies that we must ignore any queueing effects occurring at the resources represented by this node. For terminal output, of course, there is no queueing, since the terminal is a dedicated device assigned to the user. The same is true for magnetic and paper tapes.

The only causes of waits at this node which might involve queueing are the datacell and lock byte. These waits occur infrequently enough so that the probability of an actual queue developing is extremely small, and we may ignore the queueing effects. If this were not the case, it would be necessary to represent the datacell (for example) by a separate meta-node in the network, as we did with the 2314 disks.

Not surprisingly, the composite density function for

terminal output and other waits has a standard deviation higher than the mean. It is a mixture of distributions with very different means. Terminal output operation time means are in the order of 1-2 seconds, while datacell and other waits are in order of .1 to .2 seconds (Fig 6-19). To get a more accurate fit to the observed distribution variance, a hyper-exponential service time distribution was used in the model. A hyper-exponential distribution is a mixture of two exponential functions, with the density function given by:

$$f(t) = p\mu_1 e^{-\mu_i t} + (1-p)\mu_2 e^{-\mu_2 t}$$

where $0 \leq p \leq 1$ and μ_1 and μ_2 are both positive. The representation of such a service time distribution in a network model requires two nodes and is straightforward.

We will then set the service time mean at this meta-node as:

$$1/\mu_i(n_i) = (.631T + .105D + .056M + .001P + .207R) \cdot n_i$$

where T is the mean wait for a terminal output operation, D is the mean wait for a tape (magnetic or paper) operation, P is the mean length of a process interlock and R is the mean time for other delays. Since the terminal output operation time dominates this node, we present the observed terminal output operation time density function in Fig 6-6.

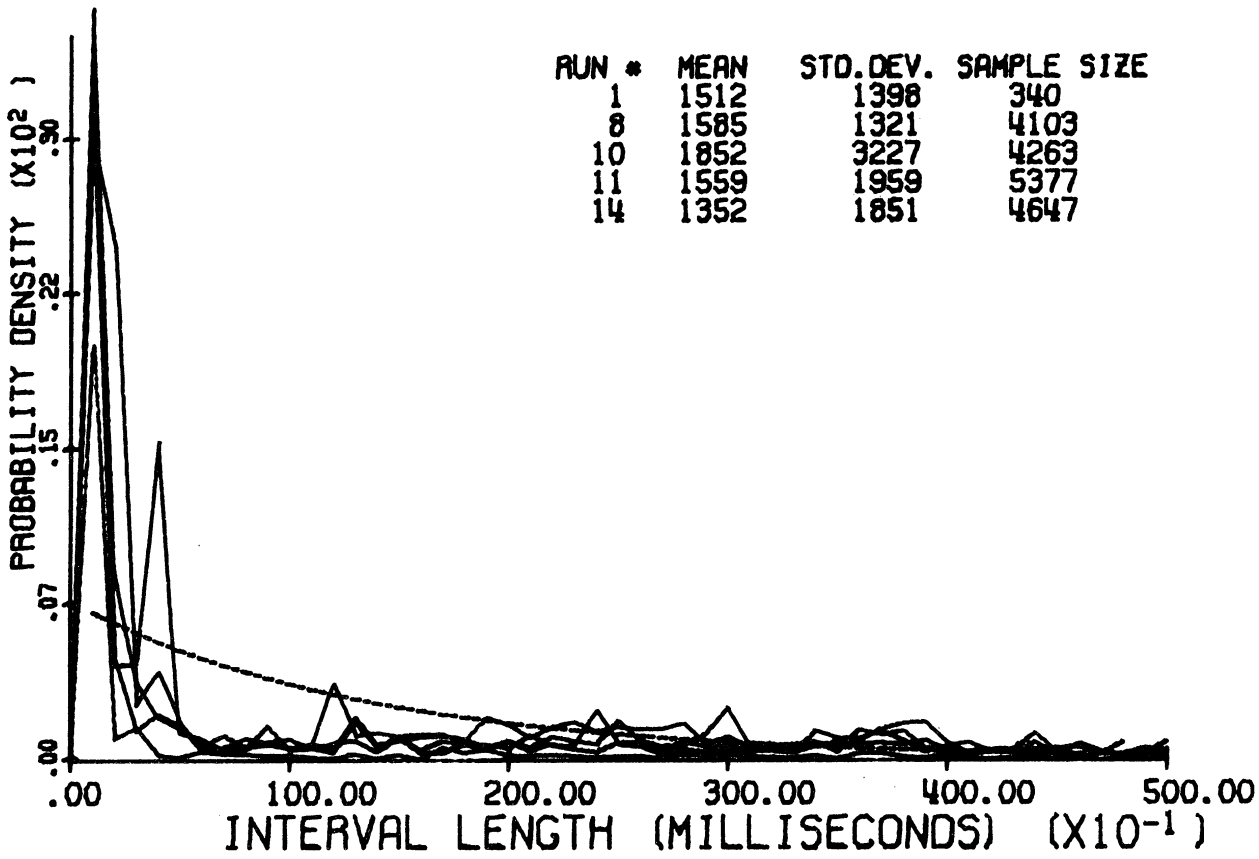

FIG 6-6: TERMINAL OUTPUT OPERATION TIME DENSITY

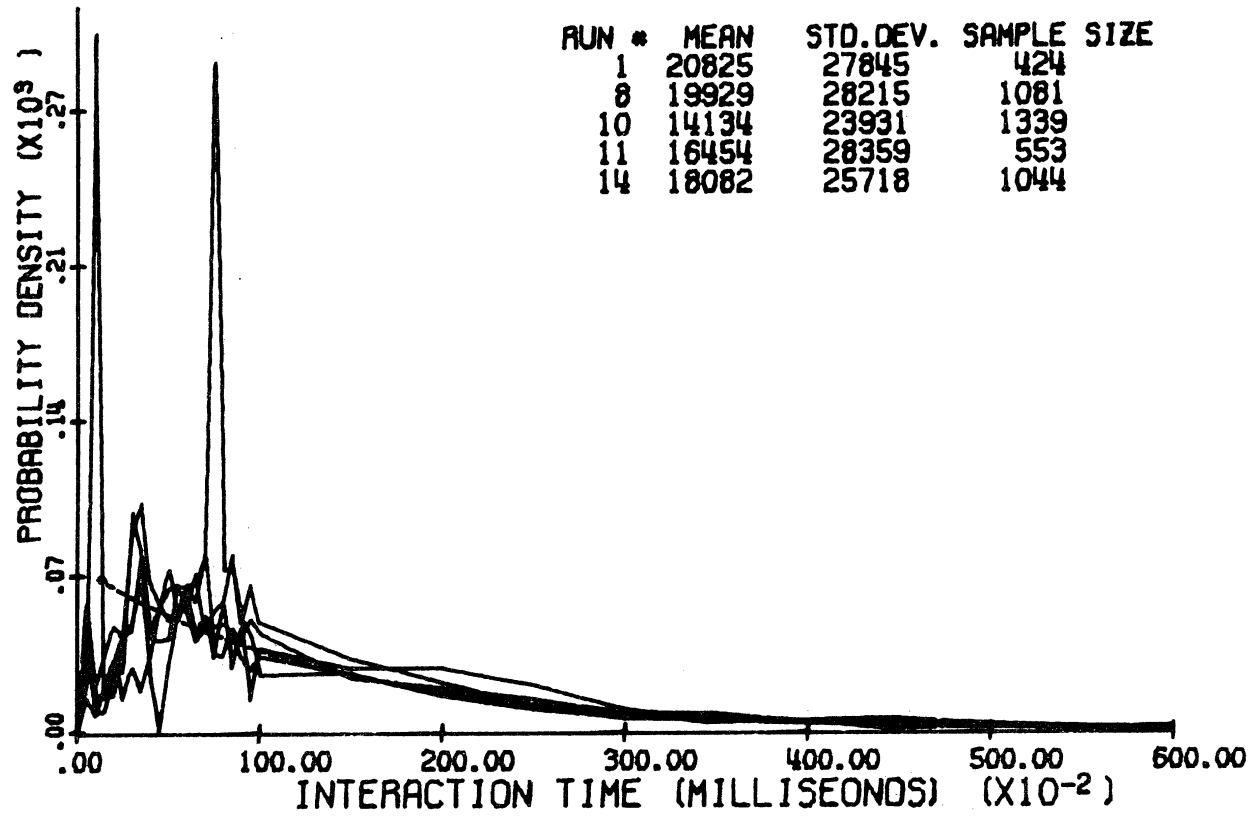

FIG 6-7: TERMINAL INTERACTION TIME DENSITY

The Terminal Input Node

To complete the network model we must represent the terminal input process - the interactions of the users. The basic scheme for doing this has already been presented in the examples of the previous chapter. It is the cycle times of customers in the network, the time it takes them to return to the terminal node after leaving it, that will provide us with our basic performance measures.

Coffman and Wood[C9] have studied the interaction time distribution in the SDC time-sharing system, and found it well-fitted by a hyper-exponential distribution. In MTS a hyper-exponential provides a good match for the first two moments of the distribution, though the actual "fit" is not too good. Fig 6-7 shows some representative observed terminal interaction time densities and a hyper-exponential fit to one of the curves. The peaks occurring in several of the densities were caused by the regular, patterned, interactions of "terminals" which were actually small remote computers connected to MTS through 2000 baud phone lines.

Since the hyper-exponential distribution provides an acceptable approximation, we shall take the approach taken in the "terminal output and other delays" node and let the terminal input meta-node consist of two nodes, with service rate parameters proportional to the number of customers at

the node. The parameters at the nodes will be computed from the mean and variance of the interaction time of the users.

The structure of the MTS model, as it has been developed to this point is diagrammed in Fig 6-8. The dotted boxes represent the groupings of nodes into meta-nodes. The actual nodes are represented by circles. For example, two circles (nodes) appear in the box (meta-node) marked "terminal input", indicating these two nodes together represent the interaction process of MTS users.

To complete the model specification we must yet determine:

(1) Numerical service rate parameters at each node.

(2) Values for the transition probabilities between nodes.

(3) The number of customers for which the model is to be solved.

We delay for a moment the assignment of actual numbers to the parameter values, and consider the general means by which such numeric values can be assigned to the transition probabilities from statistics available about the system.

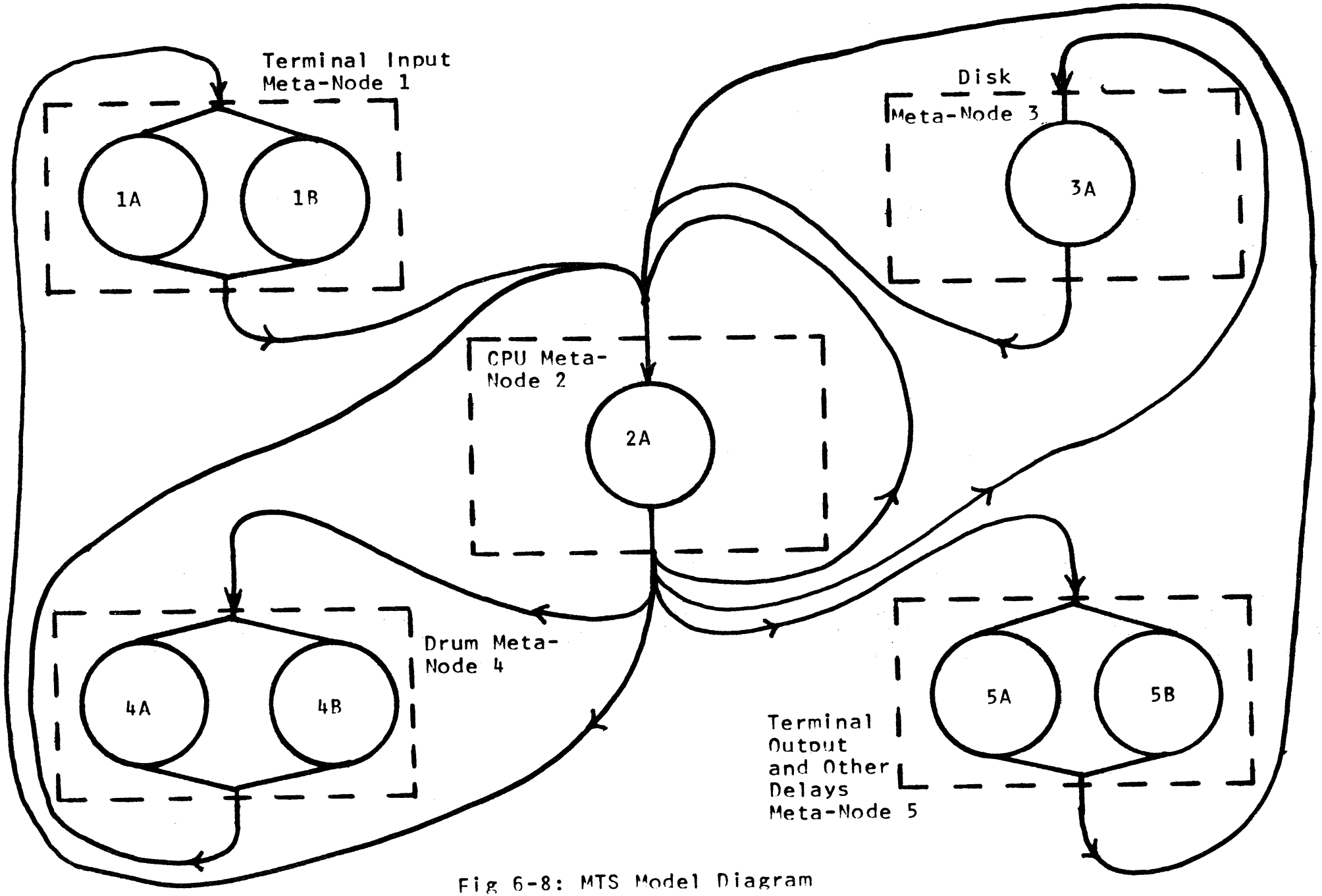

Fig 6-8: MTS Model Diagram

Transition Probabilities

With the available statistics there are several algorithms for assigning transition probabilities to the paths in the network. The most obvious is the direct method: record the actual transition probabilities occurring in MTS tasks, and transfer these to the model. This information is available from MTS statistics so that method can be used (see Fig 6-15 and Fig 6-16 below). However, if such detailed information were not available, alternative procedures could be used.

If one can measure the number of times a task appears at each node during an interaction, then approximate transition probabilities can be computed from this information. For instance if it is known that the average task performs 7.2 disk operations, causes 6.3 page faults and goes through the CPU queue 25 times in a response cycle, one can compute transition probabilities such that these numbers are matched. The assignment may not be unique, but as we showed (by example) in the previous chapter, these sorts of differences seem unimportant to the network performance.

A third possible method for assigning transition probabilities requires knowledge of the service rates at each node and the amount of time spent by a customer at each node during an interaction. Given this information the ex-

pected number of times the customer will visit each node during an interaction can be computed, and the problem reduces to the previous case.

Assigning Numeric Values to the Model Parameters

We have avoided assigning numeric values to model parameters up to this point for a good reason. We must decide which of the available values might best be assigned. The statistics sessions revealed that many of the measured statistics such as the transition probabilities were not particularly stable; they varied significantly from session to session. This is illustrated by the user behavior statistics of Fig 6-19 below.

The example of Fig 5-16, in the previous chapter, showed that fairly small changes in user behavior can lead to large changes in system performance measures (and we might add, in system performance itself!). Thus if we choose some "representative" set of parameters, a set of "representative" performance predictions can be obtained. These statistics unfortunately may be meaningless, however, since wide variance from the representative load may be the rule rather than the exception (see Fig 6-20).

When we are interested in predicting system performance then, we will use again the idea of a load envelope, determined from the range of observed values of service time

means and transition probabilities.

On the other hand, for model validation we use the data from a single statistics run to compute the model parameters. Then we can ask whether the performance of the system predicted by the model on the basis of these statistics from a single run agrees with the performance observed during the session.

3. Model Validation

For initial validation of the model we use the statistics from session #10 (see Fig 6-19 and the Appendix). The statistics gathered at this session were gathered under fairly typical conditions, with an average of 47 terminal users and a number of batch jobs ranging from 1 to 3 running during the period. This provided an average equivalent load of 69 users during the session. (See the Appendix for a discussion of the treatment of batch jobs during the statistics gathering process, and their mapping into "equivalent users").

The statistics gathered during Session #10 were used to choose numeric values for the parameters of the MTS network model of Fig 6-8. The values of these parameters are tabulated in Fig 6-9. This model was then solved for N = 69. The results of this solution are reproduced in Fig 6-10. The

Transition Probabilities

		To Node						
		1A	1B	2A	3A	4A	4B	5A	5B
	1A	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0
	1B	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0
	2A	0.0045	0.0030	0.7880	0.1065	0.0275	0.0275	0.0305	0.0120
From	3A	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0
Node	4A	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0
	4B	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0
	5A	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0
	5B	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0

Service Rates (Customers per Millisecond)

$\mu_{1A}(n) = n/20825$ $\qquad$ $\mu_{1B}(n) = n/4098$

$\mu_{2A}(1) = 1/4.453$ $\qquad$ $\mu_{2A}(n) = 2/4.453 \quad (n>1)$

$\mu_{3A}(n) = n/30.710 \quad (n\leq 3)$ $\qquad$ $\mu_{3A}(n) = 3/30.710 \quad (n\geq 3)$

$\mu_{4A}(n) = \mu_{4B}(n) = .25(n/9) \quad (0<n<9)$

$\mu_{4A}(n) = \mu_{4B}(n) = .25 \quad (9\geq n)$

$\mu_{5A}(n) = n/1552$ $\qquad$ $\mu_{5B}(n) = n/50.3$

Fig 6-9: Parameters for MTS Model

*** SINGLE SESSION MODEL FOR SESSION #10 ***

NUMBER OF USERS = 69

	TERMINAL	CPU(S)	DISK(S)	DRUM(S)	OTHER
MEAN WAIT AT NODE (INCLUDES SERVICE)	14134.000	14.535	42.000	26.311	1128.00
MEAN SERVICE TIME AT THIS NODE	323.291	2.400	17.645	17.245	61.57
MEAN QUEUE LENGTH (INCLUDES THOSE IN SERVICE)	43.719	5.057	1.380	0.526	18.31
MEAN NUMBER OF TIMES THROUGH NODE (IN 1 INTERACTION)	---	125.000	13.375	6.875	5.25
UTILIZATION FACTOR FOR THIS RESOURCE	0.634	0.861	0.424	0.367	0.26
EXPECTED CYCLE TIME FOR THIS NODE	8481.548	166.389	1648.882	3263.223	3179.72
DEMAND PER INTERACTION FROM THIS NODE	14134.000	1816.909	561.749	180.891	5922.00

STEADY-STATE QUEUE LENGTH DISTRIBUTIONS

CUSTOMERS	TERMINAL	CPU(S)	DISK(S)	DRUM(S)	OTHER
0	0.00000	0.07215	0.27001	0.63328	0.00000
1	0.00000	0.13396	0.34852	0.24388	0.00000
2	0.00000	0.12259	0.22242	0.09285	0.00000
3	0.00000	0.11055	0.09354	0.02465	0.00000
4	0.00000	0.09821	0.03886	0.00458	0.00002
5	0.00000	0.08594	0.01595	0.00067	0.00007
6	0.00000	0.07405	0.00646	0.00008	0.00025
7	0.00000	0.06282	0.00258	0.00001	0.00073
8	0.00000	0.05245	0.00102	0.00000	0.00189
9	0.00000	0.04309	0.00040	0.00000	0.00431
10	0.00000	0.03482	0.00015	0.00000	0.00877
11	0.00000	0.02768	0.00006	0.00000	0.01611
12	0.00000	0.02163	0.00002	0.00000	0.02690
13	0.00000	0.01661	0.00001	0.00000	0.04108

Fig 6-10: Program Output for MTS Model

results for the individual nodes have been condensed into results for meta-nodes in this figure. The results for the limiting performance (under this load profile) are not given in the figure. The limiting results indicated a saturation point of 89.5 customers. The CPU node is the limiting node, as is already obvious from the results for N = 69. The values of the x_i/μ_i vector were CPU = 1.000, Disk = .491, Drum = .100. These values for the x_i/μ_i vector were fairly typical of those for all sessions for which models were individually developed. In every case, the CPU node was indicated as the saturated node, though the saturation point varied considerably from session to session (see Fig 6-19).

Having solved the model with these parameters and N = 69, we can compare the performance predicted by the model with the actual performance statistics gathered during the session. The following statistics were computed and compared:

(1) Density function of CPU time and 2314 time used during an interaction (Fig 6-12 and Fig 6-13). Both these densities are assumed to be exponential by the model.

(2) Density function of the number of visits to the 2314 and CPU meta-nodes (resources) during an interaction (Fig 6-14 and Fig 6-15). This density is assumed to be geometric by the model.

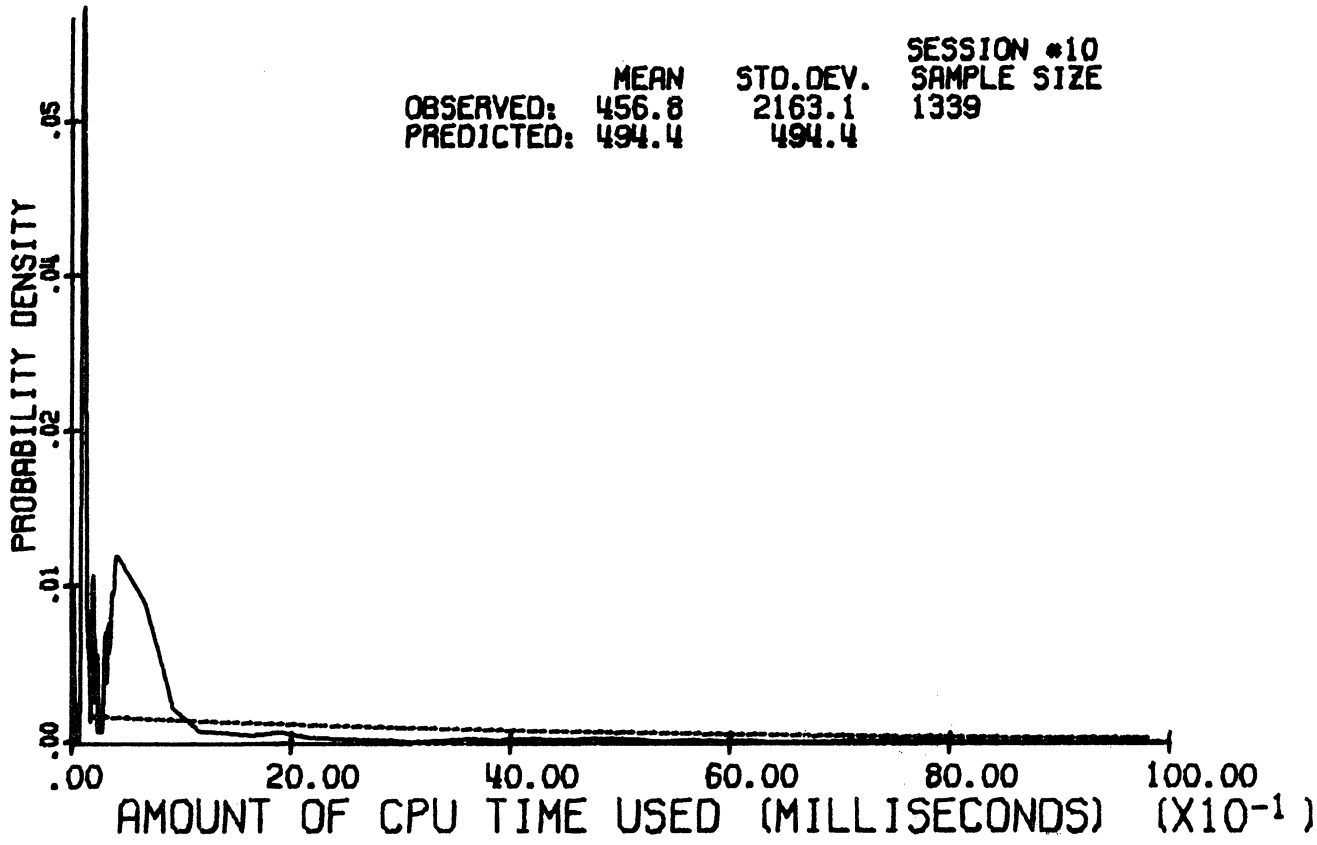

FIG 6-11: CPU TIME USED PER INTERACTION

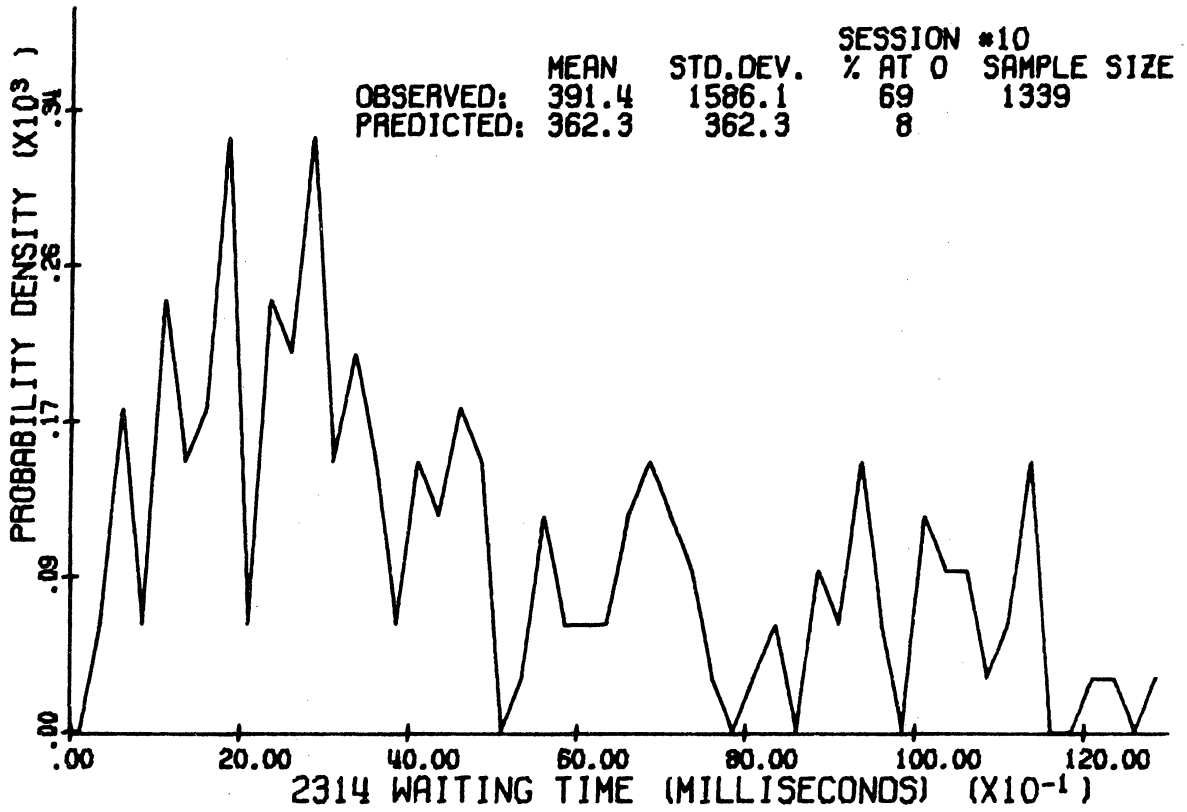

FIG 6-12: 2314 TIME USED PER INTERACTION

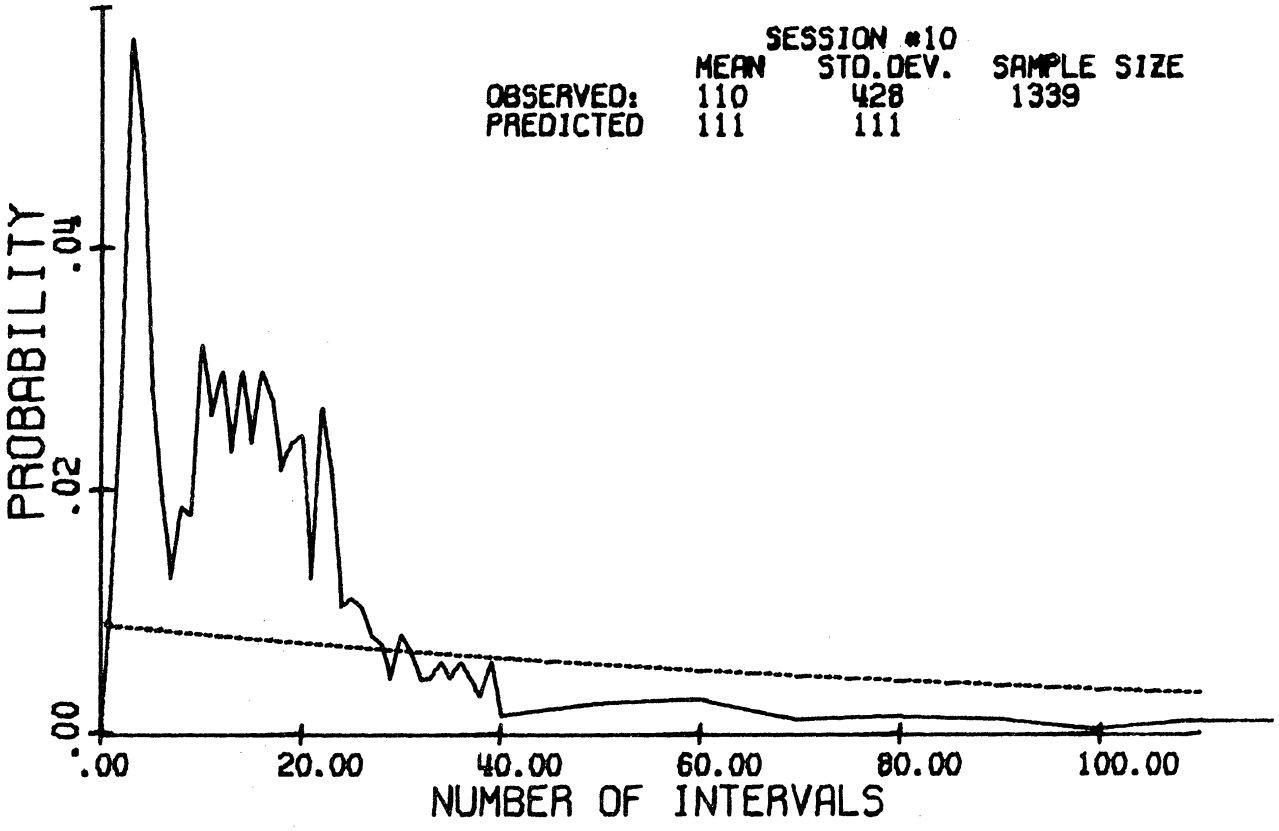

FIG 6-13: NUMBER OF CPU ACTIVE INTERVALS PER INTERACTION

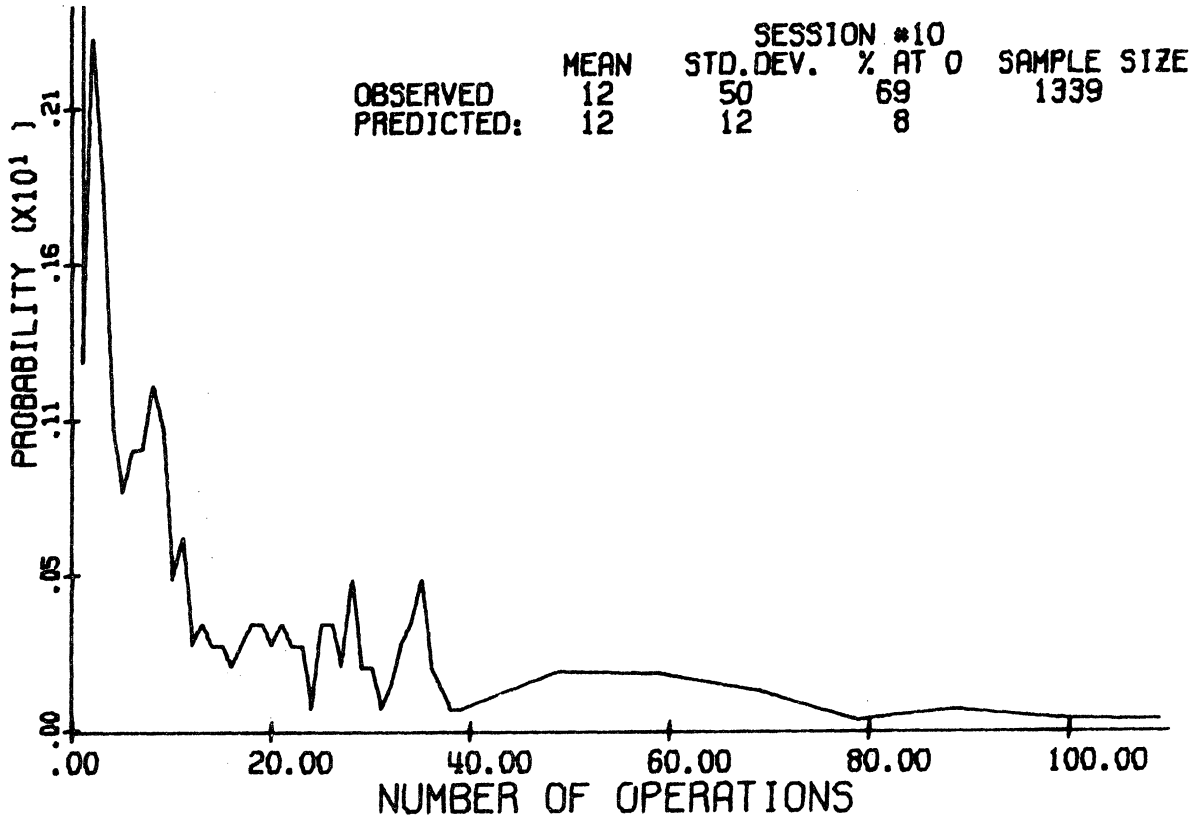

FIG 6-14: NUMBER OF 2314 OPERATIONS PER INTERACTION

(3) Correlation of transitions from node to node. One of the assumptions in the model development was that customer transitions from node to node were Markovian, and independent of the path leading to their arrival at the node. This assumption is certainly not likely to be valid for a given task. For example, a task which has been "cycling" in the CPU queue for a long period, without doing any input or output, is likely to continue doing so in the near future. This implies that the probability a customer exiting from the CPU node will return immediately to that node is higher if that is how the customer arrived at the CPU node for his current service. This intra-task correlation is illustrated by Fig 6-15. In this figure is given the probability of transitions to various resources (nodes) in the system, from the CPU. The transition probabilities given are unconditional (the first column) and conditional probabilities based on which node the task was at before it arrived at the CPU. Remember that these transition probabilities are for a single task. Markovian transitions would be indicated by rows with equal entries all the way across. However, the assumptions of independent customer transitions made in developing the model need not have been as strong as it was. If one obser-

	Abs.	CPU	Term. Input	Disk	Data-cell	Page-wait	Term. Out.	Other
CPU	.725	.818	.587	.506	.382	.376	.562	.569
Term. Input	.004	.002	.195	.000	.000	.019	.002	.001
Disk(2314)	.141	.103	.014	.448	.007	.026	.021	.052
Datacell	.005	.003	.000	.000	.536	.003	.004	.001
Page-in Wait	.074	.040	.104	.016	.053	.488	.172	.038
Term. Output	.031	.024	.079	.001	.007	.080	.237	.009
Other Delays	.020	.012	.022	.028	.015	.008	.003	.331

Fig 6-15: Intra-task Transition Correlations

	Abs.	CPU	Term. Input	Disk	Data-cell	Page-wait	Term. Out.	Other
CPU	.725	.690	.776	.817	.815	.834	.794	.818
Term. Input	.004	.004	.020	.002	.002	.002	.003	.001
Disk(2314)	.141	.169	.071	.080	.047	.048	.061	.079
Datacell	.005	.006	.004	.002	.026	.004	.003	.001
Page-in Wait	.074	.073	.063	.069	.071	.089	.088	.052
Term. Output	.031	.035	.047	.018	.028	.016	.034	.018
Other Delays	.020	.023	.020	.012	.011	.008	.019	.031

Fig 6-16: Inter-task Transition Correlations

ves the balance equations for the network (see Chapter III), it is clear that we need only require that there be no correlation of the next transition in the system with the last transition in the system. That is, the next node-to-node move of a customer must not depend on the previous node-to-node move of a customer. In this case, we are statistically much better off, as Fig 6-16 illustrates. Here we have exactly the same data as in Fig 6-15, but computed now over all jobs. That is, the sequence of transitions is taken within the entire system, not just within a single task. Note that in this case, the row entries are approximately equal, with no indication of significant correlations.

The lack of agreement between the observed and predicted statistics of Fig 6-12 and Fig 6-13 can be easily explained. The expected exponential form of the densities is predicated on an exponential service time distribution at the node. This is clearly not the case for the 2314 (Fig 6-4) and is only approximately true for the CPU (Fig 6-2). It is not surprising then that the form of the cumulative distribution densities is not exponential.

The data of Fig 6-14 and Fig 6-15 can be explained by the existence of non-homogeneous users. Heavy CPU and disk users do in fact exist. They tend to increase the

variance in these visits-per-interaction densities. The small sample size involved in these empirical distributions also tends to make the match look poor.

We note however that the means of the observed and predicted densities are in reasonable agreement in all four figures.

A further statistic that might be of interest for validation purposes is the utilization of each node - the amount of time it is busy. In MTS, this information is available only for the CPU, and tends to be very unstable. We illustrate this instability of resource utilization in Fig 6-17. This shows the percentage utilization of the CPU capacity averaged over 18 second intervals for a period of 10 minutes. Also shown are the average number of page-in operations per second, again averaged over 18 second periods. Given this pattern of behavior, it is not surprising that there tend to be discrepencies between predicted and observed node utilization. With that disclaimer, the predicted CPU utilization for Session #10 was 86%, while the observed value was 97%. Note that some of the excess CPU utilization can be accounted for by the overhead jobs (HASP, primarily), which were not modeled but do use CPU time.

The overall system performance measure - response time - was predicted to be 8.5 seconds and was observed to be 7.9 seconds, an acceptably close agreement. This general

Time (Minutes)	Average CPU Idle Time(%)	Average Page-in Operations per sec.	Number of Terminal Users	Number of Batch Users
0.19	3.83	25	41	7
0.37	1.70	20	42	6
0.55	1.70	20	42	6
1.13	.00	27	40	7
1.31	.64	21	41	5
1.49	.05	10	42	3
2.07	.05	10	42	3
2.25	.05	27	40	4
2.43	.03	12	39	4
3.00	1.14	16	39	4
3.18	36.42	8	40	5
3.36	30.54	7	41	3
3.54	58.67	12	40	3
4.11	39.09	8	41	3
4.29	14.55	9	41	4
4.47	10.21	13	40	3
5.05	2.63	11	40	3
5.23	19.25	7	40	4
5.41	22.81	11	40	3
5.58	.05	19	40	5
6.16	1.76	12	40	5
6.34	3.39	12	40	6
6.54	3.39	12	40	6
7.10	13.14	13	40	4
7.28	9.65	15	38	6
7.46	16.60	16	39	5
8.03	.92	17	39	5
8.21	.09	16	38	5
8.39	.04	12	38	5
8.57	.00	17	39	5
9.32	1.63	7	38	5
9.50	2.18	8	39	4
10.05	.72	11	40	3

Fig 6-17: System Utilization Statistics

pattern repeats itself for other sessions, with quite close agreement on means and expectations, but considerable difference in the distributions. The predicted and observed response time and CPU utilization for each of the sessions are tabulated in Fig 6-19.

4. Performance as a Function of N

Despite the fact that the number of users is not a particularly good predictor of system performance, we now develop a relation between the number of users and system performance as embodied in the expected response time measure.

The first question that must be asked is how constant the model parameters - service rates and transition probabilities - remain with changes in the number of users. The statistics from the 15 sessions were subjected to correlation analysis to see if there was a trend in the transition rates or service time means which correlated with changes in the number of equivalent users. With the exception of the CPU active intervals, no significant correlation was found. The CPU active interval length mean was found to decrease with increasing numbers of users, as has already been mentioned. Therefore to develop a model which was valid for all N, we have used the requested CPU interval lengths

(Fig 6-3), which were uncorrelated with N, in conjunction with the requested CPU time per interaction, to compute the CPU node parameters.

The load envelope was formulated from the data of Fig 6-19, and the resulting parameter ranges are tabulated in Fig 6-18. Instead of transition probabilities, the amount of CPU time used per interaction, expected number of disk operations per interaction, etc. are given. These figures tend to be more meaningful, and from them unique transition probabilities may easily be computed. The "average load" column of Fig 6-18 gives the expected values of these parameters as averaged over all the sessions.

Using the data of Fig 6-18, three models were constructed. One used the low envelope parameters, one the high envelope parameters, and a third the expected parameters. Each of these three models was then solved for N = 5,10,15,...,140. This yielded three sets of statistics one for each version of the model. In Fig 6-20 we present the expected response time curve, as a function of N, for each of the models. The computed saturation point for each model is also listed. The saturated node in each case is the CPU. In this figure, the terminal output delay times have been subtracted from the response time so that only the delays in response due to shared resources are shown.

It is unlikely that MTS would often operate at the

	High Envelope	Average Load	Low Envelope
CPU Time Used per Interaction	853.18	649.97	460.17
Disk Operations per Interaction	22.21	15.25	10.85
Mean Disk Operation Time	32.33	30.36	26.91
Page-in Operations per Interaction	10.22	6.73	.01
Terminal Output Operations per Interaction	2.89	3.75	5.82
Terminal Output Operation Time	1164.5	1684.0	2110.5
Mean Interaction Time	14134	18653	25670

* All times in milliseconds

Fig 6-18: Parameters for Performance Envelope Models

	Statistic		Run #1	Run #2	Run #3
1.	Interaction Time Distribution (msec.)	Mean	20825	18529	23340
		SD	27845	29411	56915
2.	CPU Active Interval Distribution (msec.)	Mean	5.641	6.371	10.100
		SD	7.791	8.773	27.568
3.	Disk Operation Time Distribution (msec.)	Mean	27.987	27.809	28.865
		SD	24.271	24.344	23.555
4.	Page-in Operation Time Distribution (msec.)	Mean	29.174	24.881	16.112
		SD	14.391	11.454	8.041
5.	Terminal Output Wait Distribution (msec.)	Mean	1512.2	1905.3	1254.8
		SD	1398	2307	1658
6.	Datacell Operation Time Distribution (msec.)	Mean	147.76	101.70	78.41
		SD	178.06	139.66	95.59
7.	Other Delays Distribution (msec.)	Mean	342.9	797.9	140.3
		SD	4354.3	3140.8	434.6
8.	Number of CPU Active Intervals per IR Cycle	Mean	135.55	93.91	64.04
		SD	295.00	261.00	135.00
9.	Number of Disk Operations per IR Cycle	Mean	14.42	12.39	16.36
		SD	28.00	26.00	29.00
10.	Number of Page-in Operations per IR Cycle	Mean	5.52	2.59	.01
		SD	12.00	4.00	0.00
11.	Number of Terminal Output Operations per IR Cycle	Mean	3.44	3.41	4.02
		SD	19.00	7.00	8.00
12.	Datacell ops./IR Cycle	Mean	1.12	.70	1.24
13.	Other Delays/IR Cycle	Mean	3.04	.63	.76
14.	Average Response Time (msec.)	Mean	7686	7204	6804
		SD	22800	21310	14245
15.	Predicted Response Time	Mean	8718	8167	6267
16.	Equivalent Users	Mean	41	50	18
17.	Predicted Saturation (Users)		70.0	88.9	91.4
18.	CPU Utilization (Actual)		84.5	89.2	18.5
	(percent) (Predicted)		56.5	54.7	19.3

Fig 6-19: Summary of Statistics Sessions and Models

Stat.		Run #4	Run #5	Run #6	Run #7	Run #8	Run #9
1.	Mean	25670	16895	16404	16163	19929	20446
	SD	33526	19337	23054	19984	28215	29557
2.	Mean	6.130	6.323	5.172	6.073	5.392	5.048
	SD	6.685	9.094	6.422	8.446	7.596	5.443
3.	Mean	28.733	29.348	28.276	31.456	29.800	26.907
	SD	26.404	29.158	22.026	30.163	23.582	23.037
4.	Mean	23.480	25.131	28.671	24.994	25.708	39.424
	SD	10.501	11.603	13.922	11.210	11.433	25.395
5.	Mean	1851.5	2110.5	1990.0	1420.4	1585.5	1164.5
	SD	2163.0	1725.0	2659.0	2482.0	1321.0	1476.0
6.	Mean	119.24	88.67	132.76	231.20	180.57	146.28
	SD	144.75	128.56	167.12	318.63	276.81	210.10
7.	Mean	761.6	288.7	519.1	506.0	504.1	336.2
	SD	2591.6	961.0	140.0	1549.3	2609.5	1279.2
8.	Mean	129.42	126.25	133.51	100.49	86.28	181.67
	SD	288.00	414.00	332.00	478.00	198.00	588.00
9.	Mean	17.59	15.11	16.08	10.85	14.75	22.21
	SD	26.00	31.00	39.00	28.00	34.00	124.00
10.	Mean	2.63	2.49	6.45	2.63	3.38	8.34
	SD	3.00	4.00	9.00	4.00	5.00	12.00
11.	Mean	3.57	3.17	3.88	2.89	3.80	5.82
	SD	4.00	7.00	8.00	21.00	10.00	18.00
12.	Mean	.22	2.13	.36	.79	.58	1.59
13.	Mean	1.11	1.05	1.57	10.94	.78	1.58
14.	Mean	8393	9171	10056	7101	8171	11585
	SD	39250	28215	30381	25360	30794	35437
15.		8800	10189	12194	6513	7364	14423
16.		40	59	65	54	55	63
17.		87.1	61.8	71.2	70.3	118.6	60.2
18.	act.	83.0	89.9	81.9	88.9	66.0	98.7
	pre.	44.8	87.3	85.1	73.1	45.6	93.0

Fig 6-19: Summary of Statistics Sessions and Models (cont.)

Stat.		Run #10	Run #11	Run #12	Run #13	Run #14	Run #15
1.	Mean	14134	16454	20345	16456	18082	18720
	SD	23931	28359	33458	19568	25718	25470
2.	Mean	4.453	5.925	3.738	5.914	4.650	4.810
	SD	5.073	9.016	4.394	8.310	4.892	4.845
3.	Mean	30.710	27.690	28.160	18.600	28.840	32.330
	SD	27.767	23.712	23.073	26.762	37.934	43.013
4.	Mean	28.610	21.180	103.320	21.090	36.670	32.660
	SD	17.254	13.039	267.795	12.136	23.830	52.477
5.	Mean	1852.0	1559.4	1741.6	1958.7	1351.6	2038.8
	SD	3227.0	1959.0	2306.0	1815.0	1851.0	2476.0
6.	Mean	110.80	155.84	151.33	112.09	107.27	171.87
	SD	159.98	177.39	163.49	186.91	135.57	227.37
7.	Mean	278.2	343.2	2028.5	852.6	641.9	1357.6
	SD	881.9	1408.7	3992.1	2952.8	2445.4	15489.1
8.	Mean	134.96	77.67	121.83	144.27	131.35	144.03
	SD	428.00	154.00	193.00	453.00	665.00	430.00
9.	Mean	14.49	11.41	12.76	18.60	12.37	19.52
	SD	49.00	23.00	26.00	46.00	33.00	42.00
10.	Mean	7.55	3.08	39.04	3.36	8.72	10.22
	SD	14.00	4.00	66.00	8.00	15.00	18.00
11.	Mean	3.18	3.48	3.90	4.45	4.45	3.09
	SD	7.00	7.00	12.00	8.00	7.00	6.00
12.	Mean	.54	.33	.23	2.12	1.36	.48
13.	Mean	2.07	.85	.58	1.58	1.45	1.90
14.	Mean	7909	6573	19760	12084	10434	11892
	SD	22971	20781	53593	32445	36894	40278
15.		8481	6757	11122	12648	8797	11096
16.		69	50	57	74	73	70
17.		74.7	98.8	64.4	64.4	87.2	102.5
18.	Actu	96.7	98.1	100.0	89.9	96.6	99.9
	Pred	86.1	49.5	82.6	85.4	79.7	66.4

Fig 6-19: Summary of Statistics Sessions and Models (cont.)

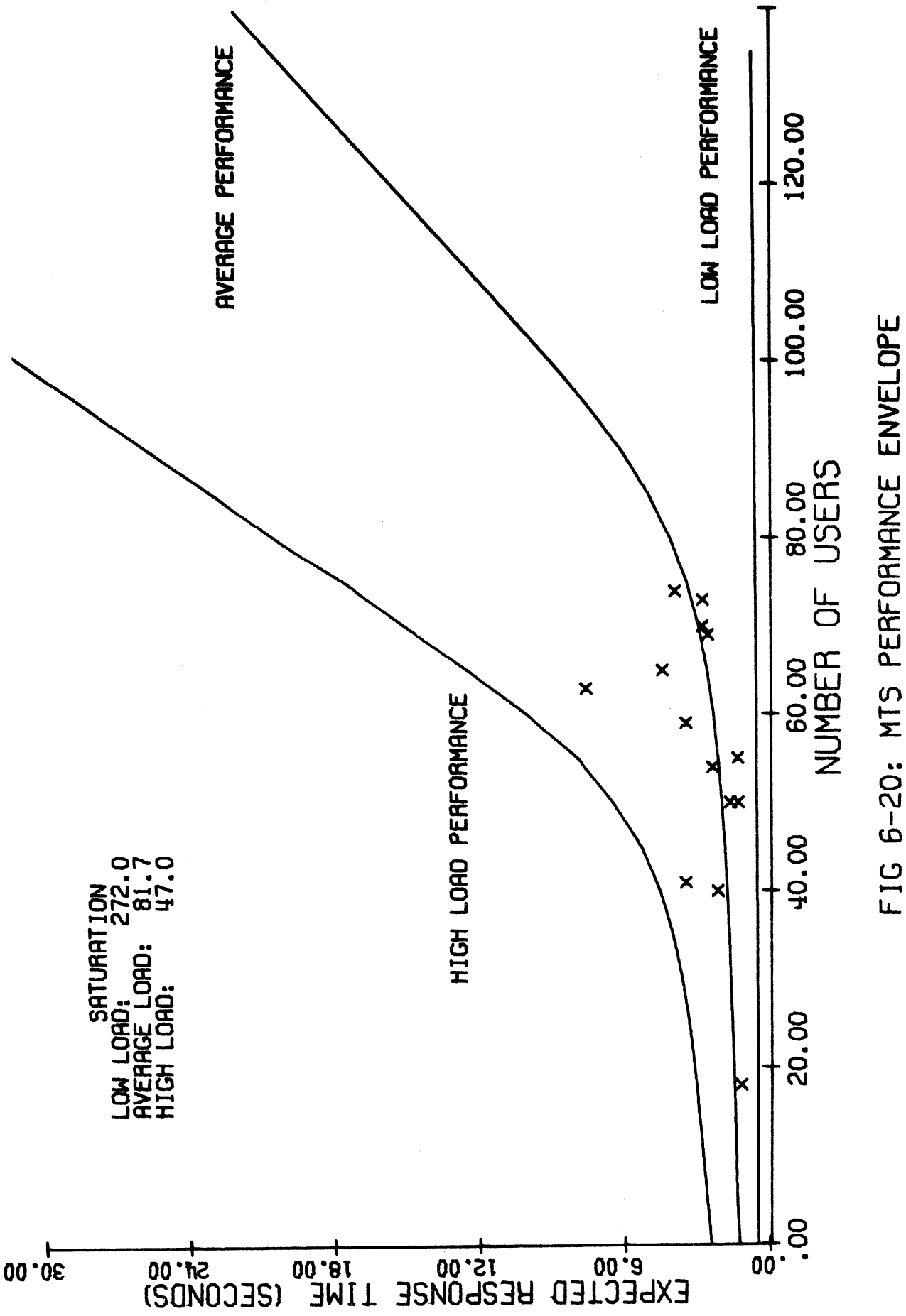

FIG 6-20: MTS PERFORMANCE ENVELOPE

boundaries of the load envelope (and therefore, performance envelope). Too many worst-case conditions would have to coincide. On the other hand, MTS does not often operate close to the median performance line, as is indicated by the X's in Fig 6-20. These X's represent the observed response time, plotted versus the number of equivalent users for the 15 statistics sessions. There is no strong correlation between the number of users and the observed response time, as we have previously mentioned.

The implication of Fig 6-20 is not that there is no hope for predicting system performance from a model. It is rather that this performance cannot be predicted accurately as a function of the number of users of the system. The number of users is useful as a load measure primarily because it is easy to measure and is unmistakably correlated with the system load. A better measure, as has already been suggested, is a vector giving the load on each of the system resources. In the case where a limiting resource is well defined, it may be possible to accurately predict the form of the performance curves (response time, for example), based on some measure of the load on that resource. To get absolute predictions for the performance measures, however, a complete load vector is required. Thus, though the CPU is the limiting resource, consider the average load MTS model. For N = 80, the average load waiting time at the 2314 node

per IR cycle was 639 milliseconds, and it remained relatively constant all the way out to N = 140. Thus it contributes little to any increase in the response time after saturation is reached. However it does continue to contribute 639 milliseconds to the absolute average response time. One must be careful in concentrating on the limiting resource to the exclusion of all other factors. Even well into saturation, at N = 100 in the average load model, the CPU node contributed only 40% to the expected response time of 15 seconds. An expected .656 seconds were spent at the disk node, 7.5 seconds at the terminal output/other node and .160 seconds at the drum node.

5. Balancing the MTS Configuration

There are a number of ways in which to define an "optimum configuration" for MTS. Within given constraints (presumably financial) one could search for a minimal response time configuration, based on a particular set of statistics. However, this configuration will depend on N, the number of users. For small loads (in the current MTS configuration), the disks contribute most to the response time delay, among the shared resources. As the load increases, the CPU's contribute more and more, eventually surpassing the disks.

A more general optimality criterion is one involving the concepts of balance and saturation, as discussed in Chapters III and IV. So we ask what alternate MTS configuration would be more balanced than the current one. There are two options - delete current equipment or add more. Since a reduction in equipment is rarely considered a viable alternative, we first consider the addition of equipment. The obvious algorithm is simply to add another component to the saturated node. In the case of MTS this means adding another CPU. For the model developed for statistics session #10 (Fig 6-10), adding a third CPU yielded the following values for the x_i/μ_i vector elements:

CPU = .915
Disk(2314) = .736
Drum(2301) = .074

The disk and CPU are more nearly in balance now (and the saturation point increases from 89.5 to 121 users!), but clearly the drum is still out of balance. Consider then a second alternative: keep the two CPU's, and discard the drums all together. Instead, let paging take place entirely on the disk. In this case, the only two non-zero components of the x_i/μ_i vector are those for the CPU and the disk, and they have values of 1.05 and .744 respectively. The saturation point is reduced slightly from the original configuration model, to 76 users.

Given the fairly limited space of feasible alternatives, it should be clear that most reasonable alternatives can be exhaustively tested for balance and saturation to achieve an optimal configuration (on either basis). For example, for MTS there are perhaps a dozen reasonable configurations, considering only disks, drums and CPU's.

We should also note, however, that caution must be exercised in considering alternative system structures based on the model. Are the parameters measured in the original model still valid in the alternate configuration? In the case of the 3-CPU modification to the MTS model, the answer is probably yes. For the configuration in which we removed the drums, the answer is almost undoubtedly no.

There are several reasons to question the validity of the no-drum model. In the first place, we admitted to ignoring the virtual memory management problem in the entire modeling process. It would be necessary to investigate the effects of increased page transit times (from the disks) on the page fault rate. Since increased page transfer times effectively decrease the size of real memory[D4], the effect may not be easily predictable. Secondly, we ignored the problem of page-out operation because of the known low paging load in MTS. If the paging device (the disk now) were heavily loaded, it would be necessary to reconsider this assumption. In summary, the validity of the statistics

available should be carefully considered before transferring them to a model involving a new configuration.

It seems clear, however, that the addition of a third CPU would greatly improve MTS performance. According to the model, one might expect it to support 30 to 40 more terminal users with no degradation in the current level of performance. Adding a fourth CPU would (again according to the model) make the disk node the saturated node (for the average load). At this point though, the question of memory conflicts becomes a serious one. Would a fourth processor (or even a third) run at the same speed as the first ones, or would the memories become saturated, unable to deliver information to the CPU's at any faster rate? Once this condition is reached, more CPU's are of little use. The question is unfortunately outside the scope of the model.

Chapter VII

Conclusions

We have considered to this point only the use of network models in modeling time-sharing systems. We have presented examples of two time-sharing systems, with very different structures, which can be satisfactorily modeled by this approach. We now discuss the general applicability of the technique.

We first admit that the limitations of the model are such that it is not applicable to the modeling of <u>all</u> time-sharing systems. One could probably take any time-sharing system, operating under a given load, and develop a reasonable network model for the system. This was essentially what was done in the first part of Chapter VI, for particular statistics sessions taken from MTS. This exercise may provide some insight into the system operation, but it is really useful only if the model developed is valid under loads other than that for which it was developed. If this is the case, then one can use the results which are a function of N to get asymptotic measures of response time, balance and saturation. Even in the case where the load did not correlate well with the number of users, we were able to develop load and performance envelopes which gave some idea of the region in which the system would operate.

When the model parameters, such as the transition

probabilities, change with the load, this approach cannot be used. One may develop a model for each set of load parameters one is interested in, but little new information will be produced. The model predictions may not be valid outside the range in which the actual performance measures are already known, so that the predictions are not needed.

One case in which this sort of problem arises is in virtual memory systems where the real memory is in some sense the actual saturated resource. In this case, as has been pointed out by Denning[D7] and others, it is necessary to limit the number of tasks allowed to compete for system resources, and in particular the scarce resource: real memory. Thus only some maximum number, K, of the tasks wishing CPU service are actually allowed into the CPU queue. A particular case of this sort of mechanism is the privileged - non-privileged task distinction in MTS. In MTS, however, enough real memory is available so that the number K is almost always large enough to include all those tasks currently requesting CPU service. If this is the case, or if K is large enough so that the probability of CPU idle time occurring with K tasks in the system is very small, the effect of this limitation may safely be ignored for modeling purposes. In systems with less real memory (in particular, for MTS operating with a smaller amount of real core), this would not be true. This limitation on the number of tasks in

the system causes CPU idle time to occur, and there is no way to represent this limitation in a network model.

An extreme example of this sort of limitation occurs in a non-virtual memory system of the type studied by Scherr[S5]. Here only a single task is allowed in memory at one time - again a limitation which cannot be represented in a network model. We should note, however, that the model Scherr did use with some success is a simple sub-case of the network models studied here.

Within these limitations, though, network models have general applicability to the study of time-sharing systems. Certainly they provide far more flexibility in representing the actual system structure than has been available to date.

Other Areas of Applicability

Network models of the sort presented here were first studied by J.R. Jackson[J1,J2] and Gordon and Newell[G3] in connection with the scheduling of jobs through a shop, where each job had to go through a randomly selected set of steps. Thus there are obviously applications of this type of model outside the computer field.

Within the computer field there is at least one other application area which would appear to be quite fruitful. That is the study of batch multiprogramming systems.

Chapter VII - Conclusions

Most of the concepts developed in this dissertation may be carried over directly to such a study, with only minor changes in definitions. N, the number of terminal users, becomes the degree of multiprogramming (number of jobs multiprogrammed). Response time becomes turn-around-time. The parameters of such a model could be determined in exactly the same way they were determined for the MTS model. Some further discussion of this problem may be found in the Appendix, and (in a different context) in Kimbleton and Moore[K2]. Many batch systems have the advantage (from a modeling viewpoint only!) that they are not virtual memory systems, and do not allow the dynamic acquisition and release of real memory within a task. A task is not initiated in this type of system unless sufficient memory for all its needs is available. Thus the problem of storage representation, which may occur in time-sharing system modeling, does not arise.

However, the small number of batch jobs multiprogrammed (in the order of 5-10) in most systems, coupled with some reduced constraints, make the variance in program behavior in batch systems quite large. Terminal users in a time-sharing system do not initiate very large jobs (either of the computational or file-shuffling variety) simply because they do not wish to sit at a terminal long enough to let it finish. In a batch system a very "patterned" sort of

program mix may exist for a long period of time. A payroll program, for example, may access a tape regularly every 30 milliseconds for several minutes. Such non-stochastic behavior may be expected to reduce the validity of a network model analysis, though to what extent it is hard to predict.

Further Work

The most successful aspect of the network model approach is the ability to model the response portion of the interaction-response cycle in a realistic and intuitive manner. This allows one to talk about what is actually happening while the user is in the response segment of the cycle. We have shown that a CPU model is not adequate. The average customer, during statistics session #10, experienced a response time of 7.9 seconds. The time was spent as follows:

* Receiving CPU service:	.556 seconds
Waiting for CPU service:	1.260 seconds
* Waiting for a memory page:	.180 seconds
* Waiting for terminal output:	6.896 seconds
Waiting for a disk:	.183 seconds
* Waiting for a disk operation:	.410 seconds
* Other delays:	.022 seconds

The above "user profile" was reflected almost exactly in the model, and in fact the starred(*) items formed the basis on which many of the model parameters were determined.

The ability to predict performance, at least in terms of response time, from this set of statistics, leads naturally to the evaluation of a vector of these measures (averaged over all users) as the "system load measure".

If we can construct such a load vector dynamically for each customer, we have a chance to balance the system dynamically. Instead of being disturbed by non-homogeneous user behavior, we can take advantage of it.

Let us construct for each user a user load vector as follows. Over some appropriate "window" - time period from t-ω seconds to t, where t is the present time, and ω is the window size - measure the percentage of time the user task (process) was allocated each resource. For example, suppose the window size was 10 seconds, and the user had used one second of CPU time during this past 10 seconds. Suppose he had also initiated disk operations requiring 400 milliseconds to complete. Then in a one CPU system with a single disk channel, this user has used 10% of the CPU capacity and 4% of the disk capacity. If these were the only two components of the load vector then (an over-simplified case), it would be (.10,.04) for this user. The system load vector then is simply the sum of all the user load vectors. It will have a maximum value of 1.0 in any component. The components of the load vector will vary from system to system, but in general any shared resource which might be in

scarce supply should be included in the load vector. One might well wish to include the percent allocation of real memory to this user in the load vector, for example.

Suppose now we have the ability to suspend any task temporarily from competing for system resources. In the case of MTS, this can easily be done by making the task non-privileged. Let us assume the current load vector for a task is a good short term predictor of its future load vector. Then if we have enough tasks to choose from, we can practically guarantee near-100% percent utilization of system resources and at the same time a balanced system (no saturated resource). To do this, the operating system goes through all the available tasks and chooses the set such that the sum of the load vectors in that set (the system load) is closest to 1.0 in every component. This is the set of tasks which it "releases", allowing them to compete for system resources. The other tasks are suspended until the next choice of an "optimal task set" is made.

This sort of idea is not entirely new, and a similar approach has been suggested by Wulf[W4]. There are also some practical problems involved in implementation. The overhead in maintaining the load vectors must not be excessive. The selection algorithm must also be constrained by some sense of fairness - it cannot refuse indefinitely to allow some task into the set of tasks allowed to compete for system

resources. Particularly for terminal users, some minimal service rate must be allowed the task. However, initial experiments with a very primitive form of this scheduling system (for batch tasks only), which has been implemented in MTS, have proved very successful. The implementation is described briefly in the Appendix.

In summary, we feel that though network models certainly do not satisfy all modeling needs for computer systems, they give satisfactory results in a number of cases, and offer considerable insight into the functioning of a time-sharing system.

Appendix

Statistics Gathering and Model Computations

This appendix describes the means used to gather and process the data used in the MTS model development of Chapter VI, and the computer program written to compute the solutions for the network models presented in Chapters V and VI.

1. The Statistics Gathering Facilities

The facilities available in MTS for performance monitoring and statistics gathering are extensive. Primary use in this dissertation was made of the instrumentation system developed by Pinkerton and Alexander, and described in detail in the dissertation and two other papers of Pinkerton[P2,P3,P4]. In [P4], the overhead incurred by the statistics gathering facilities and the details of its operation are described. What is collected is essentially a very raw but detailed chronology of the system operation. The information is buffered in memory and then written on magnetic tape for later processing. The basic unit of information recorded is the occurrence of an "event", the time the event occurred, the task(s) affected, and perhaps a few bytes of additional information. For example, a statistics

item is recorded every time a processor switches from one task to another. The time of the switch, and the tasks switched from and to are also recorded.

It is necessary to do further (computer) processing of this data before it becomes meaningful. A program was written to do this by the author (other programs have also been written by Pinkerton and others). Since the statistics gathered are very detailed - a full 2400-foot reel could easily be gathered in a half-hour under heavy system load - subsequent processing of statistics requires about one minute of 360/67 processor time for every minute of MTS monitoring. A decision was thus made to limit collection periods to 10-20 minute sessions. This limitation turned out to have some interesting effects, revealing the basic instability of the user behavior parameters (or the system load, to use the more general term). Over longer periods this instability might have been masked by the long averaging period inherent in the statistics processing. Yet the sessions were long enough so that some sort of "steady-state" could be achieved with the given user mix. That is, the measured system performance was not merely a response to a momentary transient in the load. In fact, no terminal was included in the statistics for a session unless two separate and distinct input messages had been received. The shorter periods also had the advantage that the user mix

remained relatively constant during a session. The change in the number of users signed on over the period averaged a 10% between maximum and minimum, though no statistics were kept about changes in the actual set of customers signed on.

The sessions themselves were scheduled with no pattern other than an attempt to avoid a pattern, except that periods of very light usage were avoided. The sessions extended over a six month period from September 1970 to February 1971. Fig A-1 lists all the statistics sessions and their time and duration. The average number of terminal and batch users signed on during the session is also listed. Two of the sessions deserve special comment. During session #9 the system "crashed" near the end of the period. The exact state of the system in the moments preceding the crash is unknown, so the statistics from the session are somewhat suspect. Session #12 was taken on a partitioned system. Only one processor, one drum, and two-thirds of the normal 6 core boxes were available. The system was under a heavy load during this session, as is obvious from the figures for this session in Fig 6-19. Because both these sessions were atypical, they were omitted from most of the summary and average statistics computed over all sessions.

No significant changes in hardware or the structure of the system were made during this period, and only two significant software changes were made. One software change

Run #	Date	Time of Day	Duration (sec.)	Average # Terminals	Average # Batch Jobs	Terminal Jobs = 1 Batch Job	Interactions Observed
1	9/15/70	13:14	598	29	1.3	11.5	424
2	9/16/70	16:35	554	29	2.1	10.0	517
3	10/2/70	18:12	1127	11	.4	17.2	298
4	10/6/70	12:32	614	27	2.0	6.6	307
5	10/7/70	10:26	600	30	2.8	10.5	575
6	10/8/70	15:23	742	45	1.8	11.3	955
7	10/15/70	11:41	511	25	3.7	7.8	484
8	10/26/70	16:30	868	39	1.2	13.1	1081
9	10/27/70	17:01	634	34	4.2	6.8	554
10	12/3/70	14:01	719	47	2.0	11.0	1339
11	12/15/70	10:42	578	26	4.7	5.0	553
12	12/15/70	21:41	957	20	6.3	5.8	342
13	12/30/70	10:24	1318	17	6.8	8.4	596
14	1/28/71	14:34	920	38	5.4	6.4	1044
15	2/4/71	12:22	1053	32	6.9	5.5	906

Fig A-1: History of Statistics Sessions

was made to the paging algorithm, between Sessions #10 and #11. The effect of the change was to reduce the average waiting time for a page-in operation under low paging loads. The need for the change was revealed by the paging statistics gathered in the first ten sessions. The effect of the change can be seen in Fig 6-19. In sessions #1-#10, the page-in waiting time never averaged less than 23.48 milliseconds, except for Session #3, during which only two page-in operations took place. After session #10, the minimum average waiting time was reduced to about 21 milliseconds. Since the drums in MTS are so lightly loaded, it is not felt that this change affected system performance in any significant way.

The second software change was made in February 1971, again partly in response to the information revealed by the statistics gathered in the preparation of this dissertation. The change made affected the number of batch jobs run simultaneously with the terminal tasks. Before the change, the number of batch tasks allowed to run was determined by the number of terminal users connected to the system. This criterion has now been discarded in favor of a weighted average of CPU utilization, number of paging operations per second, number of virtual memory pages in use and CPU queue length. The components to be used in computing this load factor are still being actively experimented with.

Even with the initial settings used, however, the system load (as measured by the load factor itself!) showed a marked increase in stability, as well as increased batch job throughput . It is felt that this increased throughput has been achieved at little or no cost in service to terminal users (i.e. - no increased response time), though this conjecture is still in the process of being verified.

2. Batch-Terminal Task Equivalences

The general problem of the treatment of batch tasks in the terminal modeling environment of Chapter VI was one which could have proved difficult. In the case of MTS, however, the potential problem proved amenable to a fairly simple solution. The statistics gathered for batch and terminal tasks were very similar, except for the interaction times. Thus the statistics of Fig 6-19, extended to cover batch tasks during the same periods, would look much like those for terminal tasks, except that the interaction time is essentially zero for the batch tasks and terminal output operations disappear from the list of delay causes in the response cycle. Batch tasks merely need to fetch the next input card image from a memory buffer (the usual case) or the disks.

One noticeable difference is the increased variance

in the distributions. This is partly due to the smaller sample size encountered. Only a few batch tasks were run during most of the session, so that unusual behavior on the part of one of the tasks could easily skew the distributions significantly.

Given this basic similarity, we may make the following equivalence between batch and terminal tasks. Suppose the expected response time for a terminal task is X seconds, while the expected user interaction time (for a terminal task) is Y seconds. Further suppose an average terminal task spends Z seconds waiting for terminal output during the response segment of an interaction-response cycle. Then a single terminal task spends a proportion (X-Z)/(X+Y) of its time in the system, competing for system resources. On the other hand, the batch task, with an effectively zero interaction time and no terminal output time, may be assumed to be always competing for system resources. So the number of terminal tasks equivalent to one batch task is just that number K which yields an expectation of exactly one terminal task competing for system resources at all times:

$$K = \frac{X+Y}{X-Z}$$

For example, if the average interaction time is 15 seconds, and the average response time is 6 seconds, 3 seconds of which is spent waiting for terminal output, then one batch task is equal to (6+15)/(6-3) = 7 terminal tasks.

This equivalence measure has two interesting properties. One is the generally low value of the ratio (see Fig A-1). One would expect a batch task to put as much load on the system as perhaps 15-20 users, in view of the fact that time-sharing systems typically support 50-100 customers, while batch multiprogramming systems have a degree of multiprogramming in the order of 4 or 5. This interesting relation may well be unique to MTS, but it has some interesting implications for the virtues of a virtual memory structure for a strictly batch multi-programming environment.

The other interesting property of the equivalence measure is that it is load dependent. As the load goes up, the average response time goes up, so the ratio goes down. For example, under a particular load, response time may average 5 seconds and terminal output time 2 seconds while interaction time averages 19 seconds. Thus 8 terminal jobs are equivalent to one batch job. But suppose the load increases, and response time becomes 10 seconds. Interaction

time is presumably unchanged (though one could question that assumption) and now 3.625 terminal tasks equal one batch task.

The general applicability of such measures and criteria is open to question. It is suggested here primarily as a tool allowing us to convert batch tasks to terminal tasks for modeling goals, and served adequately for that goal at least.

3. The Model Computations Program

Computer solution of the network model equations is almost essential. For a 6-node network with 10 customers the state space size is already 2376, and the probability calculations for each state require 6 multiplications.

The program written to perform the calculations for this dissertation was written in FORTRAN and run under MTS. The input to the program is the definition of a model structure, consisting of:

(1) The number of nodes.

(2) The transition probabilities between the nodes.

(3) The service structure at each node.

The service structure can be given as a specified service time mean for each queue length at the node, i.e. - as $\mu_i(1) = K_1$, $\mu_i(2) = K_2, \ldots, \mu_i(j) = K_j$. Any number, j, of

service time means may be given, and the program automatically assumes that for $j < n \leq N$, $\mu_i(n)$ is equal to $\mu_i(j)$. Alternatively the node can be specified as containing j servers, each with a mean service time of K. Provision is made for specifying an arbitrarily large number of servers, as was the case at the terminal input and output nodes of the MTS model.

The output of the program consists of two parts: the solution of the model for the limiting case, consisting of the asymptotic queue length distributions, waiting times at the nodes, node utilization, etc.; secondly an exact solution for any desired values of N can also be requested, e.g., N = 50, 55, 60. The program output is reproduced directly in several of the figures in Chapter V (5-2,5-6,5-9,5-11). Most of the other figures of Chapter V, and many of those of Chapter VI were derived directly from program output.

Since the program was fairly complicated and there were some possibilities of round-off error causing inaccuracies, a validation procedure for the program was required. The procedure used was to solve several sub-cases of the general model, where alternative solution methods were known. The finite-source, single-exponential-server models of Kleinrock[K4], Scherr[S5], and others is one which is a special case of a network model, but for which many

numerical results are available. These results were checked against the program output when the program was given input describing these models. These checks, along with some built-in internal checks in the program, were felt to sufficiently validate the program. In addition, the program always prints the model structure of the model it is solving so that one can check and make sure that the model being solved is the intended one.

References and Bibliography

Abbreviations used:

ACM	Association for Computing Machinery
AFIPS	American Federation of Information Processing Societies
CACM	Communications of the Association for Computing Machinery
Conf.	Conference
FJCC	AFIPS Fall Joint Computer Conference
IBM Systems J.	IBM Systems Journal
IEEE	Institute of Electrical and Electronics Engineers
IFIPS	International Federation of Information Processing Societies
JACM	Journal of the Association for Computing Machinery
Proc.	Proceedings
ORSA	Operations Research Society of America
SJCC	AFIPS Spring Joint Computer Conference
Trans.	Transactions

A1 Abate, J. and Dubner, H. Optimizing the Performance of a Drum-Like Storage. IEEE Trans on Computers C-18,11 (November 1969), 992-997.

A2 Abate, J., Dubner, H. and Weinberg, S. Queueing Analysis of the IBM 2314 Disk Storage Facility. JACM 15,4 (October 1969) 577-589.

A3 Adiri, I. and Avi-Itzhak, B. A Time-Sharing Queue with a Finite Number of Customers. JACM 16, 2 (April 1969), 315-323.

A4 Alexander, M.T. Time-Sharing System Supervisor Programs. in Advanced Topics in Systems Programming, University of Michigan Engineering Summer Conferences 7016 (June 1970).

A5 Arden, B. and Boettner, D. Measurement and Performance of a Multiprogramming System. Proc. 2nd ACM Symposium on Operating System Principles (October 1969) 130-146.

A6 Arden, B.W., Galler, B.A., O'Brien, T.C., and Westervelt, F.H. Program and Address Structure in a Time Sharing Environment. JACM 13, 1 (Jan 1966), 1-16.

B1 Baskett, F., Browne, J.C. and Raike, W.M. The Management of a Multi-Level Non-Paged 1memory System. AFIPS Conf. Proc. 36, (1970 SJCC) 459-465.

B2 Batson, A., Ju, S., Wood, D. Measurements of Segment Size. Proc. 2nd ACM Symposium on Operating System Principles (October 1969), 25-29.

B3 Belady, L.A. A Study of Replacement Algorithms for Multiprogramming. IBM Systems J. 5,2 (1966), 78-101.

B4 Belady, L.A. and Kuehner, C.J. Dynamic Space Sharing in Computer Systems. CACM 12, 5 (May 1969), 282-288.

B5 Belady, L.A., Nelson, R.A. Shedler, G.S. An Anomaly in the Space-Time Characteristics of Certain Programs Running in Paging Machines. CACM 12, 6 (June 1969), 349-353.

B6 Bryan, G.E. and Shemer, J.E. The UTS Time-Sharing System: Performance Analysis and Instrumentation. Proc. 2nd ACM Symposium on Operating System Principles, (October 1969) 147-156.

C1 Calingaert, Peter. System Performance Evaluation: Survey and Appraisal. CACM 10, 1 (January 1967), 12-18.

C2 Chang, W. Single-Server Queuing Processes in Computing Systems. IBM Systems J. 9, 1 (1970) 36-71.

C3 Chang, W. and Wong, D.J. Analysis of Real Time Multiprogramming. JACM 12, 4 (October 1965), 581-588.

C4 Coffman, E.G. Analysis of a Drum Input/Output Queue under Scheduled Operation in a Paged Computer System. JACM 16, (Jan 1969), 73-90.

C5 Coffman, E.G. Stochastic Models of Multiple and Time-Shared Computer Operations. Rep. No. 66-38, Dept. of Engineering, UCLA, June 1966.

C6 Coffman, E.G. and Kleinrock, L. Feedback Queuing Models for Time-Shared Systems. JACM 15, 4 (October 1968), 549-576.

C7 Coffman, E.G. and Krishnamoorthi, R. Preliminary Analysis of Time-Shared Computer Operation. SP-1719. System Development Corp., Santa Monica, Calif. August, 1964.

C8 Coffman, E.G. and Varian L.C. Further Experimental Data on the Behavior of Programs in a Paging Environment. CACM 11, 7 (July 1968), 471-474.

C9 Coffman, E.G. and Wood, R.C. Interarrival Statisitcs for Time Sharing Systems. CACM 9,7 (July 1966) 500-503.

C10 Cox, D.R. and Lewis, P.A.W. The Statistical Analysis of Series of Events. Methuen, London, 1966.

D1 DeMeis, W.M. and Weizer, N. Measurement and Analysis of a Demand Paging Time Sharing System. Proc. 24th National Conf. ACM (1969), p. 201.

D2 Denning, P.J. Effects of Scheduling on File Memory Operations. AFIPS Conf. Proc. 30 (1967 SJCC), 9-21.

D3 Denning, P.J. Equipment Configuration in Balanced Computer Systems. IEEE Trans on Computers 18,11 (Nov 1969) 1008-1012.

D4 Denning, P.J. Queueing Models for File Memory Operations. MIT Project MAC Technical Report MAC-TR-21 (October 1965, Master's Thesis).

D5 Denning, P.J. Resource Allocation in Multiprocess Computer Systems. MIT Project MAC Technical Report MAC-TR-50 (May 1968, Ph.D. Thesis).

D6 Denning, P.J. A Statistical Model for Console Behavior in Multi-User Computers. CACM 11, 8 (August 1968), 605-612.

D7 Denning, P.J. Thrashing: Its Causes and Prevention. AFIPS Conf. Proc. 33, (1968 FJCC), 915-922.

D8 Denning, P.J. Virtual Memory. Computing Surveys 2, 3 (September 1970) 153-190.

D9 Denning, P.J. The Working Set Model for Program Behavior. CACM 11, 5 (May 1968), 323-333.

D10 Dennis, J.B. Segmentation and the Design of Multiprogrammed Computer Systems. JACM 12, 4 (Oct 1965), 589-602.

E1 Estrin, G. and Kleinrock, L. Measures, Models and Measurements for Time-Shared Computer Utilities. Proc. 22nd National Conf. ACM, (1966), 85-96.

F1 Fife, D.W. An Optimization Model for Time Sharing. AFIPS Conf. Proc. 28 (1966 SJCC), 97-104.

F2 Fine, G., Jackson, C.W., and McIssac, P.V. Dynamic Program Behavior under Paging. Proc. 21st National Conf. ACM (1966), 223-228.

F3 Ford, J.R.Jr. and Fulkerson, D.R. Flows in Networks Princeton University Press, Princeton, 1962.

F4 Frank, W. Analysis and Optimization of Disk Storage Devices for Time-Sharing Systems. JACM 16, 4 (October 1969) 602-620.

F5 Fuchel, !K. and Heller, S. Considerations in the Design of a Multiple Computer System with Extended Core Storage. CACM 11, 5 (May 1968), 334-340.

G1 Gaver, D.P. Jr. Diffusion Approximations and Models for Certain Congestion Problems. J. Applied Probability 5, (1968) 607-623.

G2 Gaver, D.P. Jr. Probability Models for Multiprogramming Computer Systems. JACM 14,3 (July 1967) 423-438.

G3 Gordon, W.J., and Newell, G.F. Closed Queuing Systems with Exponential Servers. ORSA Journal 15,2 (March 1967) 254-265.

H1 Howard, R.A. Dynamic Programming and Markov Processes. MIT Press, Cambridge, 1960.

J1 Jackson, J.R. Jobshop-like Queuing Systems. Management Science 10, 1 (October 1963), 131-142.

J2 Jackson, J.R. Networks of Waiting Lines. ORSA Journal 5, 4 (August 1957), 518-521.

J3 Jaiswal, N.K. Priority Queues. Academic Press, New York, 1968.

K1 Karush, Arnold D. Two Approaches for Measuring the Performance of Time-Sharing Systems. Software Age 4, 3 (March 1970), 10-13 and (April 1970) 15-20.

K2 Kimbleton, S and Moore, C. A Probabilistic Framework for Systems Performance Evaluation. Proceedings of the ACM-SIGOPS Workshop on Systems Performance Evaluation (April 1971).

References and Bibliography

K3 Kleinrock, L. Analysis of a Time-Shared Processor. Naval Research Logistics Quart. 11, 10 (March 1964) 59-73.

K4 Kleinrock, L. Certain Analytical Results for Time-Shared Processors. Proceedings of IFIPS-68 (August 1968) D119-D125.

K5 Kleinrock, L. A Continuum of Time-Sharing Scheduling Algorithms AFIPS Conf. Proc. 36, (1970 SJCC) 435-458.

K6 Kleinrock, L. Communications Nets: Stochastic Message Flow and Delay. McGraw-Hill, New York, 1964.

K7 Kleinrock, L. Sequential Processing Machines Analyzed with a Queuing Theory Model. JACM 13, 2 (April 1966), 179-193.

K8 Kleinrock, L. Time-Shared Systems: A Theoretical Treatment. JACM 14, 2 (April 1967), 242-261.

K9 Knuth, D.E. The Art of Computer Programming Vol. I. Addison-Wesley, Reading, Mass., 1968.

K10 Krishnamoorthi, B. and Wood, R.C. Time-Shared Computer Operations with both Interarrival and Service Times Exponential. JACM 13, 3 (July 1966), 317-338.

L1 Lewis, P.A.W. A Computer Program for the Statistical Analysis of Series of Events. IBM Systems J. 5, (1966), 202-225.

M1 MacDougall, M.H. Computer System Simulation: a Survey. Computing Surveys 2, 3 (September 1970), 191-210.

M2 McKinney, J.M. A Survey of Analytical Time-Sharing Models. Computing Surveys 1, 2 (June 1969) 105-116.

M3 Mills, D.L. The Data Concentrator. Proceedings of the ACM Symposium on Problems in the Optimization of Data Communication Systems (October 1969) 291-316.

M4 MTS: Michigan Terminal System. Vol. 1 to 9, University of Michigan Computing Center.

N1 Nielson, N.R. The Simulation of Time-Sharing Systems. CACM 10, 7 (July 1967) 397-412.

P1 Parzen, E. Stochastic Processes. Holden-Day, San Francisco, 1962.

P2 Pinkerton, T.B. The MTS Data Collection Facility. Univ. of Michigan CONCOMP Proj. Memorandum No. 18 (August 1968).

P3 Pinkerton, T.B. Performance Monitoring in a Time-Sharing System. CACM 12, 11 (November 1969), 495-500.

P4 Pinkerton, T. Program Behavior and Control in Virtual Storage Computer Systems. Univ. of Michigan CONCOMP Proj. Rep. No.4 (April 1968, Ph.D. Thesis).

P5 Prabhu, N.U. Queues and Inventories. Wiley, New York, 1965.

S1 Saaty, T. Elements of Queueing Theory. McGraw-Hill, New York, 1961.

S2 Saaty, T. Stochastic Network Flows: Advances in Networks of Queues. North Carolina Symposium on Congestion Theory, Chapel Hill (1964) 86-105.

S3 Saltzer, J.H. Traffic Control in a Multiplexed Computer System. MAC-TR-30 (Thesis) MIT, Cambridge, Mass. July 1966.

S4 Saltzer, J.H. and Gintell, J.W. The Instumentation of MULTICS. CACM 13,8 (August 1970), 495-500.

S5 Scherr, A.L. An Analysis of Time-Shared Computer Systems. MAC-TR-18 (June 1965, Ph.D. Thesis) MIT, Cambridge, Mass.

S6 Shemer, J. and Gupta, S.C. On the Design of Bayesian Storage Allocation Algorithms for Paging and Segmentation. IEEE Trans. on Computers c-18,7 (July 1969) 644-651.

S7 Shemer, J.E. Some Mathematical Models of Time-Sharing Scheduling Algorithms. JACM 14, 2 (April 1967), 262-272.

S8 Shemer, J. and Shippey, B. Statistical Analysis of Paged and Segmented Computer Systems. IEEE Transactions on Electronic Computers EC-15, 6 (Dec 1966) 855-863.

S9 Smith, J.L. An Analysis of Time-Sharing Computer Systems using Markov Models. AFIPS Conf. Proc. 28, (1966 SJCC) 87-95.

S10 Stevens, D.F. On Overcoming High-Priority Paralysis in Multiprogramming Systems: A Case History. CACM 11, 8 (August 1968) 539-541.

References and Bibliography

S11 Stimler, S. Some Criteria for Time-Sharing System Performance. CACM 12, 1 (January 1969), 47-53.

T1 Takacs, L. A Single-Server Queue with Feedback. Bell System Technical Journal, (March 1963) 505-519.

T2 Totschek, R.A. An Empirical Investigation into the Behavior of the SDC Time-Sharing System, SP-2192, Systems Development Corp., Santa Monica, Calif. (August 1965).

W1 Wallace, V.L. and Mason, D.L. Degree of Multiprogramming in Page-on-Demand Systems. CACM 12,6 (June 1969) 305ff.

W2 Wallace, V.L. and Rosenberg, R.S. Markovian Models and Numerical Analysis of Computer System Behavior. AFIPS Conf. Proc. 28 (1966 SJCC) 141-148.

W3 Weizer, N. and Oppenheimer, G. Virtual Memory Management in a Paging Environment. AFIPS Conf. Proc. 34, (1969 SJCC), 249-256.

W4 Wulf, W.A. Performance Monitors for Multiprogramming Systems. Proc. 2nd ACM Symposium on Operating System Principles (October 1969) 175-181.

DOCUMENT CONTROL DATA - R & D

(Security classification of title, body of abstract and indexing annotation must be entered when the overall report is classified)

1. ORIGINATING ACTIVITY *(Corporate author)*
ISDOS Research Project/Performance Modeling Group
Department of Industrial Engineering
The University of Michigan

2a. REPORT SECURITY CLASSIFICATION
Unclassified

2b. GROUP

3. REPORT TITLE

Network Models for Large-Scale Time-Sharing Systems

4. DESCRIPTIVE NOTES *(Type of report and inclusive dates)*
Technical Report

5. AUTHOR(S) *(First name, middle initial, last name)*
Charles G. Moore III

6. REPORT DATE
30 April 1971

7a. TOTAL NO. OF PAGES
205

7b. NO. OF REFS
83

8a. CONTRACT OR GRANT NO.
N00014-67-A-0181-0036

b. PROJECT NO.
(NR 049-311)

c.

d.

9a. ORIGINATOR'S REPORT NUMBER(S)
TR 71-1 (U)

9b. OTHER REPORT NO(S) *(Any other numbers that may be assigned this report)*

10. DISTRIBUTION STATEMENT
Reproduction in Whole or in Part is Permitted for any Purpose of the United States Government

11. SUPPLEMENTARY NOTES

12. SPONSORING MILITARY ACTIVITY
Information Systems Program
Office of Naval Research
Arlington, Virginia

13. ABSTRACT

This paper develops a class of analytical models for large computer time-sharing systems. The models developed allow the various resources of the time-sharing system, such as central processors, paging devices, and mass storage equipment to be represented as independent queueing centers. The success of the method depends on developing the queueing network so that it may be treated as a time-continuous markov process.

Consideration is given to a number of proposed measures for system load and system performance. These measures are examined in the light of the model. In addition, concepts of system saturation and system balance are introduced and defined in terms of the model.

As an example of the use of network models, a model is developed for the University of Michigan Terminal System (MTS). Statistics available from that system are used to develop and validate the model. Finally, possible changes in that system's structure, in light of the insights gained from the model, are explored.

14. KEY WORDS	A		LINK B		LINK C	
	ROLE	WT	ROLE	WT	ROLE	WT
Network Models						
Time-share						
Computers Systems Design						
Queueing Networks						
Multi-programming						
Multi-processing						
Analytic Models						
Performance Evaluation						