INFORMATION TO USERS

This material was produced from a microfilm copy of the original document. While the most advanced technological means to photograph and reproduce this document have been used, the quality is heavily dependent upon the quality of the original submitted.

The following explanation of techniques is provided to help you understand markings or patterns which may appear on this reproduction.

1. The sign or "target" for pages apparently lacking from the document photographed is "Missing Page(s)". If it was possible to obtain the missing page(s) or section, they are spliced into the film along with adjacent pages. This may have necessitated cutting thru an image and duplicating adjacent pages to insure you complete continuity.

2. When an image on the film is obliterated with a large round black mark, it is an indication that the photographer suspected that the copy may have moved during exposure and thus cause a blurred image. You will find a good image of the page in the adjacent frame.

3. When a map, drawing or chart, etc., was part of the material being photographed the photographer followed a definite method in "sectioning" the material. It is customary to begin photoing at the upper left hand corner of a large sheet and to continue photoing from left to right in equal sections with a small overlap. If necessary, sectioning is continued again — beginning below the first row and continuing on until complete.

4. The majority of users indicate that the textual content is of greatest value, however, a somewhat higher quality reproduction could be made from "photographs" if essential to the understanding of the dissertation. Silver prints of "photographs" may be ordered at additional charge by writing the Order Department, giving the catalog number, title, author and specific pages you wish reproduced.

5. PLEASE NOTE: Some pages may have indistinct print. Filmed as received.

Xerox University Microfilms

300 North Zeeb Road
Ann Arbor, Michigan 48106

75-10,211

LANDWEHR, Carl Edward, 1946-
LOAD SHARING IN COMPUTER NETWORKS: A QUEUEING MODEL.

The University of Michigan, Ph.D., 1974
Computer Science

Xerox University Microfilms, Ann Arbor, Michigan 48106

THIS DISSERTATION HAS BEEN MICROFILMED EXACTLY AS RECEIVED.

LOAD SHARING IN COMPUTER NETWORKS:

A QUEUEING MODEL

by
Carl Edward Landwehr

A dissertation submitted in partial fulfillment of
the requirements for the degree of
Doctor of Philosophy
(Computer and Communication Sciences)
in The University of Michigan
1974

Doctoral Committee:

Professor Bernard A. Galler, Chairman
Professor Bruce W. Arden
Dr. Alfred B. Cocanower
Professor Ralph L. Disney
Associate Professor John F. Meyer

Acknowledgments

My thanks go to the members of my doctoral committee who have all given generously of their time and effort in the course of this work. Professor Galler has been especially helpful and encouraging, particularly during the final year of work. To Professor Disney, I am indebted for his aid on the queueing theory aspects of this research, and I thank Professor Arden for overseeing the initial phases of the work. Dr. Cocanower's helpful comments and prompt reading of the draft of this dissertation are gratefully acknowledged, and thanks are due to Professor Meyer for taking Professor Arden's place during the final period of work.

In addition, I have benefitted from many discussions with members of the University of Michigan Computing Center staff and the staff of the MERIT Computer Network. Both of these organizations provided a stimulating environment in which to work as well as financial support during the course of this research. I am also grateful to Mrs. Valerie Kepler of the MERIT staff for her patient and accurate typing of the manuscript.

Finally, I thank my wife, Jurate, not only for her patience and good humor, but also for the many times she participated in discussions of this research which helped bring it to completion.

TABLE OF CONTENTS

ACKNOWLEDGMENTS ii

LIST OF TABLES iv

LIST OF ILLUSTRATIONS v

CHAPTER
1.0 Introduction and Literature Survey 1

1.1 Introduction to the Problem 1
1.2 Survey of Previous Work 8

2.0 Single Host Model 26

2.1 Description and Discussion 26
2.2 Analysis 35
2.3 Data Collection 53
2.4 Validation 74
2.5 Additional Parameter Studies 88

3.0 Network Considerations 94

3.1 Network Structure 94
3.2 Job Model 98
3.3 User Model 100
3.4 Decision Maker Model 102

4.0 Two Node Model 115

4.1 Description and Discussion of the Model 115
4.2 Analysis 129
4.3 Parameter Studies 137

5.0 Multi-node Model 156

5.1 Description 156
5.2 Analytic and Computational Aspects of the Multi-node Model 163
5.3 Parameter Studies 169

6.0 Conclusions 191

6.1 Conclusions 191
6.2 Future Research 194

BIBLIOGRAPHY 196

List of Tables

Table		Page
2.1	Summary of MTS Measurements	73
2.2	Summary of Parameter Studies	82
2.3	Analytic Model - System Data Comparison	83
4.1	State Space Size and Cost of Analysis	138
4.2	Traffic Parameters	138
4.3	Transshipping Parameters	138
4.4	Turnaround Factor Function	146
5.1	Resource Costs for Fully Connected Three-node Networks	171
5.2	Multi-node Studies - Group 1	173
5.3	Multi-node Studies - Group 2	174
5.4	Multi-node Studies - Group 3	175

List of Illustrations

Figure		Page
2.1	Single Host Model	29
2.2	A-Matrix	42
2.3	C-Matrix	44
2.4	B-Matrix	45
2.5	Initial Block Structure	47
2.6	Block Structure after Transformations	48
2.7	University of Michigan Machine Configuration	57
2.8	Mean CPU Utilization by Hour of Day	60
2.9	Mean Batch Queue Length by Hour of Day	61
2.10	Batch Queue Statistics - Period 2	63
2.11	Batch Queue Statistics - Period 8	64
2.12	Batch Queue Length Distribution - Light Periods	66
2.13	Batch Queue Length Distribution - Moderate Periods	67
2.14	Batch Queue Length Distribution - Heavy Periods	68
2.15	Terminal Interarrival Distribution - Period 14	69
2.16	Batch Interarrival Distribution - Period 8	69
2.17	Batch CPU Service Distribution - Period 2	70
2.18	Terminal CPU Service Distribution - Period 13	71
2.19	Analytic Results - Period 2	78
2.20	Analytic Results - Period 8	79

2.21 Analytic Results - Period 13 80

2.22 Model vs. Data - Error Vectors 84

2.23 Model vs. Data - Std. Deviations 85

2.24 Model vs. Data - Interpolated Curves 86

2.25 Parameter Studies - Mean Batch Wait vs. Batch Arrival Rate 90

2.26 Parameter Studies - Mean Batch Wait vs. Terminal Arrival Rate 92

3.1 Charging Rates at Two MTS Installations, March 1974 107

3.2 Estimated Service Charges for Michigan Bell Telephone Service, Ann Arbor-Detroit (36 Miles), April 1974 108

3.3 Line Costs as Percent of Computing Center Budget 109

3.4 Streeter's Curves 111

3.5 Modified Express Curve 111

3.6 Turnaround Factor Curve 111

3.7 Modified Turnaround Factor Curve 111

4.1 Two Node Network 116

4.2 Finite State Two Node Model 128

4.3 Finite State Two Node Model - Block Structure ...133

4.4 Parameter Studies - Mean Batch Wait vs. Batch Arrival Rate 140

4.5 Light vs. Light State Space Size Studies (Batch) 141

4.6 Light vs. Light State Space Size Studies (Terminal) 142

4.7 Light vs. Heavy State Space Size Studies (Batch) 143

4.8 Two Lightly Loaded Systems, Identical Parameters 148

4.9 Two Lightly Loaded Systems, Varying Batch Arrival Rates 150

4.10 Two Lightly Loaded Systems, Different Resource Costs 152

4.11 One Lightly Loaded System, One Heavily Loaded System 154

5.1 Multi-node Model 159

5.2 Multi-node Model - Convergence Studies 172

5.3 Multi-node Model - Two Lightly Loaded Systems, Varying Batch Input Rates 179

5.4 Multi-node Model - Two Lightly Loaded Systems, Different Resource Costs 181

5.5 Multi-node Model - Lightly Loaded and Heavily Loaded System 183

5.6 Multi-node Model - One Lightly Loaded and Two Heavily Loaded Systems 186

5.7 Multi-node Model - Two Lightly Loaded and One Heavily Loaded Systems 187

5.8 Multi-node Model - One Lightly Loaded and Two Heavily Loaded Systems with Varying Batch Input Rates 189

Chapter 1 - Introduction and Literature Survey

1.1 Introduction to the Problem

The earliest computer networks typically consisted of a single computing facility connected to remote terminals by a communications network. These networks can be referred to as "centralized" since they distribute the resources of a single center among many users. More recently, networks which interconnect several computing centers have been implemented. Since the computing power in such networks is located at several sites, they are called "distributed" networks. Users of distributed networks thus can access several different computing systems. Probably the largest of these distributed networks is ARPANET[1], which included approximately 50 different host computers as of Spring 1974 (K3). One of the primary motivations cited for the creation of distributed networks is the opportunity afforded for the sharing of unique resources. Particular systems may have specialized software (or hardware) which can be made available to a much larger body of users through such networks. Another potential benefit that has been less emphasized is the possibility of sharing non-unique

[1] Advanced Research Projects Agency of the Department of Defense Computer Network

resources.

The notion of sharing a common resource is perhaps more familiar in the guise of the "computer utility." This term usually refers to a powerful computing system that provides a homogeneous service to a large number of users. Instead of a single system handling a large user community, we might picture a utility similar to the electric power network -- many interconnected power sources which distribute the current demand for computing among themselves. Computing power is certainly a much less homogeneous product than electric power; nevertheless, a group of machines all running the same operating system (or at least systems which maintain a common user interface) will appear quite similar to most users even if the machines are quite different. Such a utility could offer some of the same benefits to users that the electric utilities do: better utilization of equipment and more predictable service.

The implementation of such a network would require a great deal of effort since most current systems are not easily interconnected and do not provide a standard interface to users. The purpose of this research is to determine under what conditions it would be feasible to interconnect systems with standard user interfaces so that workloads from different centers could be shared. We will attempt to minimize assumptions of features unavailable on

current systems, so that the models constructed will be applicable to present conditions. At the same time, it will be useful to keep the models general, so that results will not be confined to particular systems. A paradigm for the load-sharing network to be investigated is developed below, keeping these goals in mind.

A network which has the goal of distributing identical resources among users could operate in two ways: (1) users could distribute themselves among the different resource centers or (2) the systems on the network could distribute the users among the centers. The first type of resource distribution will be referred to here as manual or static load sharing since it involves no action by the network itself; the second type will be called automatic or dynamic load sharing, since the network actively allocates the load imposed upon it. In either case, if a particular center of the network were overloaded, users (or their workloads) would tend to leave that center and move to other, less busy systems. Dynamic load sharing seems preferable, since it does not depend on the rationality and knowledge of the users -- systems on the network could inform each other of current load conditions and move users to locations with excess capacity. The balance of this research will be concerned with modelling dynamic load-sharing networks.

In addition to the mechanisms by which load is

distributed, the composition of the load itself must be defined. If we consider the usual terminal session or batch job, it would be within the reach of current technology to allow different portions of the job to be executed at different systems on the network; however, the overhead and complexity involved in any such scheme would probably outweigh any advantages gained by allowing jobs in progress to be pre-empted and shipped elsewhere. The appropriate unit for sharing thus appears to be an entire batch job or terminal session.

If an entire job is to be executed at a given center, all of the programs and data used by the job must be available at that center. For batch jobs, this condition can be met without too much difficulty: as long as all systems in the network present the same interface to users (and thus have the same translators, utilities, etc. available) and as long as all of the private data required for the job is included in the job deck, the entire deck can be executed on any system in the network, with the results returned to the user's location. The assumption that interactive jobs are similarly self-contained is a far more severe one, since users that rely on interactive access will also often rely on information stored in private files. For jobs requiring data from local files to be executed on a remote system, some type of dynamic file access between the local and remote host must be allowed. Because of the unpredictable

nature of a typical interactive session, the user may not know in advance just what files he will require; thus, it would not be possible in general to ship only the required subset of his data files along with the job. Dynamic file access schemes are feasible technologically (and are under development in some networks), but because of the complexity inherent in attempting to model the transshipping of interactive jobs, this research will be confined to networks which transship self-contained batch jobs. Systems on the network may be assumed to be capable of handling interactive jobs, but these jobs will function only as a non-shareable background load on the system.

The paradigm for networks to be investigated in this research, then, is a dynamic load sharing network in which only batch jobs are transshipped. To a batch user, the operation of the network might appear as follows:

1. User submits a self-contained job deck to a particular system.
2. Network decides on which host to execute the job and sends the job to that host.
3. Job is executed.
4. Job results are returned to the user at the original site.

The user need not be aware of the particular host on which the job is executed.

Before proceeding with a review of literature which bears upon this work, it is useful to briefly define a number of common terms and to outline the method of attack which will be applied to the problem. First, a computer network will include any collection of computers which can communicate with each other only by transmission lines; they may not have mutually addressable storage. Thus, a multiprocessing system may be a host on a network, but it is not a network in itself. Each computing system on the network will be called a host or node. (Node is used by some authors to include both a computer system and a local miniprocessor which interfaces with the network transmission lines. The networks considered here may or may not include such interface processors, so host and node will be used interchangeably to designate both the computing system and any additional hardware required to interface to the network.) Different hosts may have different hardware configurations, but each host will be restricted to understand a common job control language or to handle translations from language to language internally. Hosts must be able to process batch jobs and may have interactive capabilities as well. Load sharing in this context means the transmitting of a job, together with its data, from one host to another, and transmission of the resulting output back to the first host. The sending of a job and its data between hosts is also referred to as the transshipping of the job. The local host for a given job is the host where the job

enters the network; all other hosts are remote in relation to that job. Other terms will be defined as they arise.

In order to investigate the feasibility of dynamic load-sharing networks for batch jobs, we will build a number of analytical models and study their behavior in a variety of projected situations. In order to remain tractable, analytical models are usually required to be considerably simpler in structure than simulation models, so some detail is lost in their construction. However, the results obtained from analytical models are not subject to the vagaries of random number generators or limitations due to fixed numbers of trials, as are simulation results. At present there are no existing dynamic load-sharing networks, so the perhaps less precise, but more generally applicable results of analytical models should provide greater insight into the overall behavior of such networks than could a detailed simulation of an imaginary implementation.

The approach to building a network model will be synthetic. In Chapter 2, a model for a single host is presented; Chapter 3 considers various aspects of modelling a network; and Chapters 4 and 5 describe models for two-node and multi-node networks, respectively. The network models of Chapters 4 and 5 are based on generalizations of the single host model and the network considerations introduced in Chapter 3.

1.2 Survey of Previous Work

In this section past work which relates to this dissertation will be reviewed. The survey reflects the approach of the research; we present (1) queueing models for single-computer systems, (2) models for users and cost functions, and (3) models for computer networks. Any overview of such broad areas must be incomplete; we will include primarily those references which have a direct influence on the topics of this dissertation, as well as references to surveys to which the reader is directed for further information and references.

Queueing Models

The area of queueing models for computer systems is a very broad one indeed. A recent survey by Wyszewianski (W5) lists over a hundred references on feedback queues alone used in computer models. Since in constructing a model for a single-host system we will be concerned with modelling an entire computing system, models designed specifically to study particular subsystems such as disks or drums will not be covered here. It is true that by reinterpreting some of these models they could perhaps be used to represent entire systems, but such investigations are beyond the scope of this work. Chang (C2) reviews a number of queueing results and indicates their applications to models of different

portions of computer systems.

Queueing models for whole systems may be divided into two broad categories: Those with only one resource type (single-resource models) and those with several different resources (multiple-resource models). The dissertation by Omahen (O2) includes a good survey of the field structured along these lines, and we will follow his organization of this area. Because of the decision to omit models which are solely concerned with peripheral equipment, the single-resource models mentioned here will all be models for central processor scheduling. McKinney's survey of such models (M1) still provides a good introduction and bibliography for this area, as does the survey by Estrin and Kleinrock (E1). Chapter 4 of the book by Coffman and Denning (C5) offers a general introduction to queueing theory with applications to operating systems.

One of the earliest queueing models applied to time-sharing systems is the machine interference model. The terminals active in the system are considered to be machines which break down (request service) at random intervals. The server (CPU) then acts as the repairman for the terminals. Ferdinand (F3) presents some results for this model as the number of machines, n, grows very large but the ratio of arrival to service rate ($\frac{n\lambda}{\mu}$) remains fixed. As Feller (F2) indicates, this model can be extended to multiprocessor

systems if more than one repairman is allowed.

In time-sharing systems, tasks are not run to completion; each task receives a unit of service (called a quantum) and is then suspended so that another task may receive its quantum. The goals of this multiplexing strategy are to make the system appear to be serving a number of tasks in parallel and to give rapid response to short requests. In a multiprogramming system which does not support time-sharing, tasks will usually be allowed to run to completion (that is, until they can no longer make use of the CPU). This strategy can increase utilization of hardware, but will allow a single compute-bound task to saturate the CPU for periods which would be intolerable in a time-sharing system.

The existence of a quantum for a task, with its implication of suspending and restarting the job later, leads to the idea of queues with feedback. A job receives service until its quantum is exhausted or until it fulfills its service requirement. If it has not completed its service needs, it rejoins the queue; as soon as it fulfills its service requirement, it leaves the system. The strategy just described is referred to as "round-robin" if jobs always rejoin the queue in the same order they exit the server. Considerably more complicated feedback schemes are possible; some implementations are described in Alexander (A5).

A good introduction to models for feedback-scheduling policies is the paper by Coffman and Kleinrock (C6). They investigate the round-robin model described above, as well as more complicated multiple level feedback queues, with and without priorities. In multiple level feedback queues, each time a job completes its quantum it joins the queue at the next higher numbered priority level, and the server processes jobs in order of priority, with lowest numbered levels first. If there is a finite number of levels, a job remains in the highest level queue if it still requires more service. Coffman and Kleinrock treat these cases under the assumptions of fixed quantum size and zero overhead time to pre-empt and resume jobs. They also present a model which involves a variable number of servers, the processor-sharing model. In this case there is no queueing; if there are n jobs in the system, each receives 1/n of the service rate. This model can be viewed as the limiting case of the round-robin model as the quantum size approaches zero, but average arrival and service rates are held constant. The Laplace transform of the equilibrium waiting time distribution for the processor-sharing model is developed by Coffman, Muntz, and Trotter (C7).

As mentioned above, an extensive survey and bibliography for feedback queues is available in (W5). A few references will be mentioned here to indicate the scope of

the field. Adiri and Avi-Itzhak (A3) study the round-robin queue with a finite number of customers allowed in the system, fixed length quanta, and a fixed length setup time at the start of each quantum; the same system was studied earlier by Krishnamoorthi and Wood (K8). Mean waiting times conditioned on amount of service demanded are obtained. Adiri has also obtained the mean waiting time for customers in a multi-level feedback queue with priorities, conditioned on the priority of the customer and his service requirement (A2). In this case, customers arriving at the jth priority level join the jth-level queue in the feedback system, fixed setup times are assumed, and quantum size depends on the priority level of the job in service. All of the above models assume exponentially distributed service times. Schrage (S1) has analyzed a system allowing a general service time distribution with an infinite number of levels of feedback. Quantum sizes at each level of feedback are allowed to have an arbitrary distribution.

The processor-sharing model has also been studied extensively. Some typical studies include the papers by Coffman, Muntz, and Trotter (C7) and by Kleinrock and Muntz (K5). The earlier work (C7) assumes the round-robin form of processor sharing -- each of n customers receives 1/n of the service rate -- with a Poisson arrival process and exponential service times. The equilibrium waiting time distribution is derived, and the authors note that the

variance in waiting times in the processor sharing system is potentially much higher than in the corresponding first-come, first-served (FCFS) system. The second paper (K5) examines processor sharing models for mixed scheduling disciplines. Possible disciplines are FCFS, round-robin, and feedback with infinite levels. Customers in the system are ordered into groups by the amount of service received so far, and a discipline is defined for each group. A feedback-with-infinite-levels discipline applied to a processor-sharing model results in all jobs with the lowest attained service sharing the processor equally. Service times in this model may have a general distribution. The results of the analysis are the expected waiting times for customers conditioned on service requirements.

A word on single-server priority models is appropriate here. A number of the studies mentioned above include priority levels for arriving jobs or for jobs cycling through feedback queues. In all cases the priorities of arriving jobs have been determined by the jobs themselves, independent of the state of the system when the job arrives. These are referred to as exogenous priority disciplines. Systems in which the priority of an arrival depends upon the system state are called endogenous systems. A good source on priority queues is Jaiswal (J1); he notes that relatively little is known about endogenous priority systems. The model we shall develop for a host system in Chapter 2 will, in

fact, be an endogenous priority system; it will be fully motivated and described in that chapter.

Before leaving the area of single-resource models, we note that expositions of basic queueing theory for single server models are available in a number of standard texts such as Takacs (T1), Prabhu (P3), and Cox and Smith (C9), as well as Coffman and Denning (C5). Single resource queues with multiple servers are also covered in these texts; perhaps because of the relatively small number of true multiprocessing configurations that have existed in the past few years, multiple server models are infrequently seen in the computing literature.

The body of literature on networks of queues, however, has been growing rapidly. Queueing networks are often used to model systems in which queueing may occur for several different types of resources; for example, a system may require a job to queue for the CPU, for I/O units of different types, and for storage. Typically, a network model will include a queue for each resource or for each resource type, with jobs departing from a service point proceeding to other queues in the network on the basis of a set of probabilities. The text by Conway, Maxwell, and Miller (C8) presents a brief overview of queueing network results. Probably the single most important work in this area is the paper by Jackson (J2). It presents a general structure for a

network of queues, and demonstrates sufficient conditions that the queues in the network be independent -- that is, the joint probability distribution for the state of the entire network is the product of the individual distributions for all the queues. These conditions can be understood by examining a typical subsystem in the network. It is allowed to have an input from outside the network which must be Poisson-distributed, as well as inputs which are taken from the outputs of other queues within the network. The processing rate must be exponential, but may be a function of the number in the system. The scheduling discipline may not use information about the processing time for a job or the routing of the job. The output of the subsystem is routed to other subsystem inputs (or leaves the network) according to a fixed set of transition probabilities. Under these conditions, the flows along all of the arcs in the network will be Poisson-distributed. We will make use of this decomposition property in the multi-node model presented in Chapter 5.

Jackson's work is for open models: jobs can enter and exit the system from outside. Gordon and Newell (G4) examine a similar closed system in which a fixed number of jobs circulate through the network. They find the equilibrium joint probability distribution for the queue lengths throughout the network. Moore (M4) has continued their work, developing a model and applying it to a specific system,

with special attention to system bottlenecks that may develop in such networks. Buzen (B4) has carried out research along similar lines.

Work on numerical techniques to determine steady state probability distributions for finite state Markov models has been conducted by Wallace and Rosenberg (W3). In his dissertation (W2), Wallace uses a similar technique to yield numerical solutions for the steady state distribution of a more general class of infinite state queueing systems. These techniques will be presented in detail and utilized in Chapters 2 and 4.

A number of efforts have appeared which apply the diffusion approximation to queueing networks. One of the earliest papers taking this approach to modelling a multiprogramming system is by Gaver and Shedler (G2). They apply the approximation in order to avoid the assumption of exponentially distributed service times for I/O devices. More recently, Kobayashi (K6, K7) has used the same technique applied to both open and closed queueing networks. Chandy, Herzog, and Woo (C1) approach the problem of examining the behavior of a particular network subsystem from an electrical engineering viewpoint: an "equivalent network" is defined to approximate the part of the queueing network not included in the subsystem to be studied.

In addition to the general approaches mentioned so far, a number of more specific models for multiple-resource systems have been investigated. Again, we will only be able to list a few of these efforts. One of the earliest studies is by Gaver (G1), who models a system as having one CPU and many I/O units, with jobs circulating between the two types of resources. CPU service times may have a general distribution, while I/O devices are restricted to exponentially distributed service times. Lewis and Shedler (L1) investigate a model with two servers (CPU and paging I/O) in a loop, but with each server dispensing service in three phases. The middle phase of each service represents useful work, while the first and third phases represent overhead functions. Avi-Itzhak and Heyman (A7) analyze a closed network with jobs circulating between a central server and several peripheral devices and use the results to develop an approximate model for a similar open system. Another model of a two-queue network is presented by Adiri, Hofri, and Yadin (A4). Again, the servers represent CPU and I/O service, but in this case the system is assumed to be heavily loaded at all times, so that the input queue in front of the I/O server is never empty. The finite capacity of the CPU queue restricts the number of jobs in the system. The steady state probability distribution for the system state is obtained. Shedler (S3) has analyzed a closed series of three queues in a loop, representing a system with both disk and drum services. The CPU server in this model employs

a simple round-robin feedback discipline. Jobs leaving the CPU receive service first from the disk and then the drum before returning to the CPU queue. Discrete distributions are used to model the service times, and results are obtained for the various device utilizations. For all of the previously mentioned models, a job could only acquire one resource at a time. Since, in many cases, a job must acquire two or more resources (e.g., memory, CPU, and disk) before it can proceed, this condition represents a significant restriction. The dissertation by Omahen (02) defines and investigates a model for multi-resource queues in which a job can request resources of several different types simultaneously.

Models for Users and Cost Functions

Literature concerning user behavior and cost functions is not as well developed or well organized as work on queueing models. Often a model for a computer system will not discuss users directly but will contain implicit assumptions of their behavior. Cost functions are grouped with user models here since decisions concerning what costs to assign to certain facilities and how to allocate resources should consider the effects of these policies on users; thus, they contain models for users, at least indirectly. We can subdivide the area into work which relates to relatively short term planning (e.g., priority

schemes for task scheduling) and research which treats longer periods (e.g., computing center planning).

Models for user behavior in the short term include work which relates to some of the queueing models presented above. A user-system interaction period can be defined for interactive systems, consisting of the time from the entry of a command into the system until the entry of the next command. This period can be partitioned into the time required for the system to receive the command, process it, output results to the user's I/O device, and the time for the user to consider the results, to decide on a next command, and to enter the new command. Jackson and Stubbs defined a more detailed model for this cycle in (J3) together with data on observed distributions for random variables representing different parts of the cycle. This work is continued by Fuchs and Jackson (F7); they find that most of the discrete distributions can be characterized as geometric, while continuous distributions observed are best approximated by the gamma distribution. The mean user "think time" period observed for users of four different systems was observed to be 5.0 seconds, with a standard deviation among the systems of 3.1. The paper by Yule (Y1) measures user response times in a graphics environment and finds that 50 percent of the times are less than 4.5 seconds and 90 percent are less than 16 seconds. Yule then constructs a cost function which, together with the user think time

distribution, can be used to determine at what point it will be profitable to move the user's program from core to disk.

An early paper by Greenberger (G5) introduces the notion of the cost of waiting time to users; different users may have different values for waiting time in this model. Building on this work in their simulation study of performance in a time-sharing system, Blatny, Clark, and Rourke (B1) introduce cost curves which depend on the mean delay to all jobs in the system. The relative response time for a job is defined to be the ratio of elapsed time in the system to time spent processing; the cost of delay for a job is then a function of the relative response time. The general question of scheduling tasks with deadlines is treated in Conway, Maxwell, and Miller (C8), but the usual assumption made is that processing requirements for the tasks are known in advance. This assumption is rarely fulfilled in computing systems. Bowden and Barr (B3) present a structure in which a user-specified penalty function may be used in scheduling tasks for a network of computers. Streeter (S5) presents some interesting results from user surveys which attempt to define the value of a job as a function of the delay incurred in processing it. His results will be examined in detail in Chapter 3.

Some work on accounting schemes for multiprocessing systems also utilizes measures of delay to jobs in the

system. Rettus and Smith (R1) define the multiprogramming degradation as the elapsed time for a job in the system minus the "productive" time -- time spent executing the user's program. They note the difficulty of making such a measurement and present a method for approximating the multiprogramming overhead. A model for charging users of a multiprocessing system according to consumption of total system resources rather than simply for CPU usage is presented in (W4). This accounting method implies a more complex model for a job than most analytical models can deal with. The two papers by Swoyer and Armstrong (S7, S8) present an interesting case study of a computing center which implemented a two-parameter classification scheme. Users were required to place their jobs into one of seven categories based on the proportion of CPU use to I/O processing, and rates for CPU and I/O equipment use were dependent on the category specified. After an initial learning period, users were found to be able to predict the behavior of their jobs with reasonable accuracy.

Long range planning for computing centers is still an art as much as a science, but some papers from this area have appeared in the literature. The work on accounting systems just reviewed has implications for computing center policies in the short to medium term. Longer range plans are discussed in the work by Hanssman, Kistler, and Schultz (H1). A framework is provided for modelling a center which

is a modification of one by Gaver (G1). Long range planning is performed by projecting the total system elapsed time required to process the anticipated load. Oldehoeft and Halstead (O1) relate "Grosch's Law" (the computing power of a single system is proportional to the square of the cost) to the possibility of decentralized networks and find that a single central machine is likely to be more cost effective than a network. Streeter (S6) examines the issue of centralization with a somewhat more detailed model and also finds that large decentralized networks seem to be the less desirable alternative. Note, however, that these two papers do not treat the case in which a number of machines already exist and the choice to be made is whether or not to interconnect them.

Models for Networks

Computer networks have been the subject of increasing attention over the past several years. Literature describing networks in general, software design of networks, network measurements, and models for networks will be briefly surveyed in this section. Peterson and Veit (P2) present an introduction to computer network terminology and possible configurations for networks. The article by Farber (F1) provides good descriptions of several current network implementations. The largest and best documented of these, ARPANET, is also described in several other papers. Probably

the most complete description is given by the group of papers presented in the spring of 1970 (R2, K2, H3, F6). The MERIT network[2], as well as ARPANET, Farber's ring-oriented network, CDC's CYBERNET, two IBM networks, and the Lawrence Livermore Laboratory OCTOPUS network, is described in (R3). Further description of MERIT may be found in (C3); a more recent exposition on OCTOPUS is (F4). The terminal interface processor (TIP) which allows terminals to interface directly to the ARPA network (without first connecting to a particular host system) is described by Ornstein et. al. in (O3).

The internal design of packet-switching computer networks usually includes considerations of buffer size, protocols, and software structures similar to those involved in the construction of any operating system. The degree of asynchronism that can occur between two completely independent computer systems, however, can introduce some unique problems. The protocols designed to implement host-to-host communications in networks must cope with these problems. ARPANET protocols are discussed in Crocker et. al. (C10). The MERIT host-host interactive protocol is described in (M2), and the MERIT protocol for remote batch transmission is outlined by (M3). A proposed generalized task-to-task communication protocol is presented by Walden

[2] A distributed computer network oriented toward university computing, currently serving three universities in Michigan

in (W1). Davies describes a protocol used by the English National Physical Laboratory network in (D1).

Measurements of activity in computer networks have only recently been published. Facilities available for measuring ARPANET activity are described by Cole (C4) and results of a one-week measuring experiment are given by Kleinrock and Naylor (K3). One of the primary findings of this experiment is that very short messages tend to dominate ARPANET traffic; as a consequence smaller packet sizes are suggested. In addition, a significant degree of favoritism seems to exist in the network -- that is, a large percentage of network traffic is accounted for by a relatively small number of hosts.

Literature on network flows in general is quite extensive, and will not be reviewed in detail here. Computer network models have generally been constructed in order to study aspects of the performance (or projected performance) of particular network configurations. Typical goals for such models include minimizing message delays and line costs and maximizing network reliability. Since these goals are contradictory (increased reliability and decreased delays generally lead to higher line costs), The normal compromise is to fix the value of one of the parameters to meet a minimum specification while another is varied (e.g., minimize cost subject to a maximum host-host delay of T

seconds, given the routing strategy and traffic distribution). The book by Kleinrock (K1) is one of the earliest and most basic works in this field. Queueing networks (as described above) are used to model the communications network. Work performed in designing configurations, channel capacities, and routing methods for ARPANET has been well documented in (K2, F5, F6). The thesis by Gerla (G3) treats the modelling and design of store-and-forward networks in detail and includes a good general bibliography of that area; the interested reader is referred to his work for further references. Schultz (S2) analyzes a model for message assembly in a network processor. A model for a circular network has been constructed by Hayes and Sherman (H2). Finally, we cite the work by Bowden (B2) in which both host computers and the communication network are represented. The communications network model in this case does not involve any queues; queues are used only to represent the processing of jobs by the host machine. This model reduces to a multi-server queue with a single, fixed maximum length queue. Several priority classes of units are assumed, and a non-pre-emptive priority service discipline is employed.

Chapter 2 - Single Host Model

2.1 Description and Discussion

Analytical models of computer systems will differ according to their intended applications. This fact explains, in part, the profusion of queueing models for computer systems which have been developed in recent years. Some models have been designed to evaluate the performance of particular aspects of the system; for example, models have been developed to study disk scheduling policies, job dispatchers, and storage allocation schemes. Other models have attempted to capture the behavior of an entire system to discover bottlenecks or to assess the impact of changes in one part of a system on another part. A model of a complete system will necessarily contain a less detailed treatment of a particular component than a model which treats that component in isolation. Consequently, different measures are available from models at different levels of detail. The choice of the level of detail to be used in a particular model is thus a critical one. An overly detailed model may be difficult to comprehend and to analyze, while a model which is too coarse will fail to reflect accurately important system properties.

Before presenting the model to be analyzed, then, it is

important to outline the measures which will be required of it and the goals it is intended to fulfill. Since one of the goals of this research is to model a network of two or more computers, it will not be practical to support the level of detail often found in a model which incorporates only a single computer system. In fact, in the process of modelling a computer network, it will be necessary to model not only the host computer systems, but also the units to be transmitted among hosts (jobs), the medium of transmission (data communications), and the user being serviced. Thus, in order to keep the entire network model comprehensible, it is essential that the model of the host system be as simple as possible.

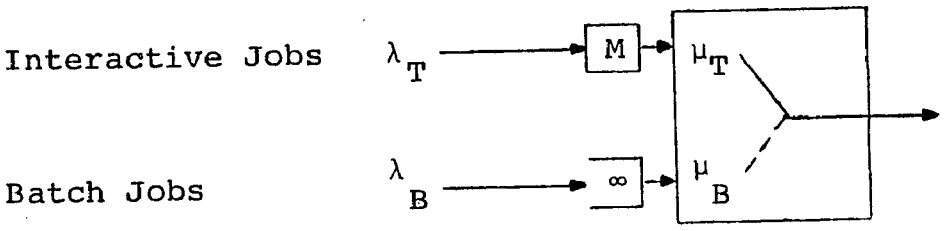
At the same time, there are several characteristics that must be preserved in the model. First, the model must be capable of representing systems which include both batch processing and interactive access. In such a system, while interactive users will not be considered as candidates for load sharing, they will have an important effect on the need for sharing batch jobs. A degradation of response time that is negligible to a batch user may be intolerable to a terminal user who is not free to perform other tasks while his job is running. Consequently, it is desirable that systems which combine batch and interactive service place a priority on interactive service.

The model should also have some way of distinguishing among centers with different processing capacities. This sort of distinction should not be in terms of actual numbers of CPUs, storage boxes, disks, etc., since such detail would overcomplicate the model, but there should be a way of indicating that hosts have varying rates of servicing users.

Finally, the host system model must allow some definition of the load imposed on the system and some way to measure the performance of the system under the given load. These measures are necessary since differing systems may well perform differently under similar loads, and these differences in performance may influence transshipment decisions. The most important aspect of system performance will be the response time that the user sees, particularly the turnaround time for batch jobs.

Keeping these goals in mind, we will now describe a queueing model which we have developed for the host systems. It is a mixed model, containing one finite queue and one potentially infinite queue. There is one server for these two queues. The server chooses a job from one of the two queues according to a priority scheme described below, and services the job to completion (no pre-emptions). (See Figure 2.1.)

Single Node Model Structure



Batch Queue - infinite capacity

Maximum # terminals in system = M

Single Server, which has a state associated with it:

State	Meaning
idle	no jobs in system
0	serving batch job
1	serving 1st consecutive terminal
2	serving 2nd consecutive terminal
⋮	⋮ ⋮ ⋮
L-1	serving L-1st consecutive terminal

System State: (T,B,S)

Fig. 2.1 Single Host Model

The modelling of the interactive part of the system is centered in the finite queue. The maximum size of the queue is intended to represent the maximum number of terminals that the system can support at one time (i.e., the number which can be physically connected to the host). The population of terminals which are capable of connection to the host system is considered to be infinite; the number of terminals actually connected at a given time is limited by the number of available ports at the host. Terminals which attempt to call in to the system when all ports are busy (the queue is full) are lost to the system (depart without queueing or service). The aggregate arrival distribution of terminals is assumed to be Poisson-distributed with parameter λ_T. The amount of service required by the terminal form the system is assumed to be exponentially distributed with mean $1/\mu_T$. The arrival of a job to the terminal queue, then, is considered to be analogous to a terminal calling in to the system. The departure of a terminal from the server is analogous to the completion of a terminal session. On departing from the server, the terminal re-enters the population of terminals which may call in to the system.

The batch component of the host system is represented by the potentially infinite queue in front of the server. Batch jobs are assumed to arrive at this queue according to a Poisson-distribution with rate λ_B. The source for these jobs is assumed to be infinite. Once in the queue, the batch

job waits for all batch jobs in front of it to be completed; then it receives service and departs from the system. The amount of service required by a batch job is assumed to be exponentially distributed with mean $1/\mu_B$.

There is a single server in the system which services both queues. The server has associated with it a state, called the server state. The server may be idle, or in one of L other server states, numbered from 0 to L-1. Also associated with the server is a vector of L breakpoints $(b_0, b_1, \ldots, b_{L-1})$. The first breakpoint, b_0, is defined to have the value 1, and the last breakpoint, b_{L-1} is defined to have the value $+\infty$. There is no requirement that the breakpoint sequence be non-decreasing; however, all of the sequences considered in this work will be non-decreasing. Given this structure, the server implements the following priority scheme at each service completion:

1. When both queues are non-empty and the server is in state j, then
   a. If the number of jobs in the terminal queue is greater than or equal to b_j, select the next job to be serviced from the terminal queue and change the state to j+1.
   b. If the number of terminals in the queue is less than b_j, choose the next job from the batch queue and change the state to 0.

2. If only the batch queue is empty at a service completion, a terminal job is chosen and the state is incremented by one (unless the state is already L-1).

3. If only the terminal queue is empty, the state is set to 0 and a batch job is selected.

4. If both queues are empty, the server enters the idle state.

A brief example may clarify this scheme. Suppose we are given:

$L=3$ $b_0=1$, $b_1=3$, $b_2=\infty$

n_T=number of jobs in the terminal queue

n_B=number of jobs in the batch queue

s = server state

Then if the server has just completed a batch job (s=0) and there are one or more jobs in the terminal queue ($n_T \geq b_1$) the server does a terminal job next and enters state 1. At the completion of this job, if there are 3 or more jobs in the terminal queue ($n_T \geq b_1$) (or if $n_B=0$), another terminal job is selected and s=2. Otherwise, a batch job is chosen and s is reset to zero. If three consecutive terminal jobs are serviced, then s=3, and a batch job must be selected next since $n_T < b_3=\infty$. Thus, the typical mode of operation for the

server is to service one or more consecutive terminal jobs, dependent on the number in the queues, followed by a single batch service, followed by one or more terminal services, etc.

In summary, the host system model is defined by the following parameters:

M = maximum number of terminals supported by the system

L = number of server states (excluding the idle state)

$(b_0, b_1, \ldots, b_{L-1})$ = sequence of breakpoints

λ_T = arrival rate of terminal jobs

λ_B = arrival rate of batch jobs

μ_T = service rate of terminal jobs

μ_B = service rate of batch jobs

The state of a particular model at any time is specified by:

n_T = number of terminal jobs in queue

n_B = number of batch jobs in queue

s = state of server

Before proceeding with the analysis, we review briefly how this model fulfills the goals set for it above. Both batch and interactive processing are represented in the model. It might be argued that while batch jobs may indeed queue up for service, execute for a while, and then depart, interactive jobs have a much more complex behavior --

periods of "think time," user requests, and system responses, occur in cycles (J3). A long wait in a queue followed by a single service burst does not portray the detailed behavior of interactive jobs. The complexity of terminal behavior is not, however, the central concern of this research. Rather, terminal processing is of interest only insofar as it affects the system response to batch jobs, thus influencing decisions on transshipment. A model in which two queues contend for a single server can reflect this influence without introducing too much detail. The priority scheme is chosen to favor terminal processing when there is a high level of interactive demand on the system. At the same time it enforces a minimal level of batch throughput (one batch job for every L-1 terminal jobs) even at the highest periods of terminal usage. These goals are appropriate for an actual system, although they could not be implemented as they are in the model since service is not actually dispensed in one burst. The measures of load and performance in the model are relatively straightforward. The load on the system is captured by defining the queueing load factor (the ratio of aggregate arrival rate to aggregate service rate) for the system:

$$\rho = \frac{\lambda_T + \lambda_B}{\mu_T \frac{\lambda_T}{\lambda_T + \lambda_B} + \mu_B \frac{\lambda_B}{\lambda_T + \lambda_B}}$$

This ratio approximates the mean server utilization and thus reflects the overall system load. The hardware capabilities of the host as well as the services requested by jobs will be reflected in the values of μ_T and μ_B. Performance of the system can be measured by the equilibrium queue lengths and the mean waiting time for batch jobs.

2.2 Analysis

The first part of the analysis of the host model deals with the derivation of the equilibrium joint probability distributions for the lengths of the queues and the server state. The method is due to V. L. Wallace (W2). A brief sketch of his work will be presented before the application to this particular model is discussed.

Wallace examines continuous parameter Markov chains whose transition intensity matrices (P1, p.289) have a block tri-diagonal form. If the states of the Markov chain are ordered, the steady state probability distribution can be considered as a vector of probabilities, where each element of the vector indicates the equilibrium probability of the corresponding state. Call this vector Π. If the transition intensity matrix is U, then

$$\Pi \cdot U = \underline{0} \qquad (1)$$

since Π is the steady state probability distribution.

In many interesting cases, the state space of the process is infinite and, consequently, π and U are infinite dimensional. Wallace shows how to arrive at an arbitrarily large portion of π if certain conditions on the structure of U are satisfied. These conditions are most easily understood by looking at a picture of U:

$$U = \begin{bmatrix} E & \hat{A} & & & & 0 \\ \hat{C} & B & A & & & \\ & C & B & A & & \\ & & C & B & A & \\ & & & \cdot & \cdot & \cdot \\ & & & & \cdot & \cdot & \cdot \\ 0 & & & & & \cdot & \cdot & \cdot \end{bmatrix}$$

where a Q-matrix is a square matrix whose non-diagonal entries are non-negative and whose row sums are non-positive

and E is an $m_o \times m_o$ Q-matrix

C is an $m \times m$ non-negative matrix

A is an $m \times m$ non-negative matrix

$\hat{A} = \begin{bmatrix} A \\ 0 \end{bmatrix}$ $m_o \times m$ matrix

$\hat{C} = [C \; 0]$ $m \times m_o$ matrix

B is an $m \times m$ Q-matrix

In words, U is a block tri-diagonal matrix with identical submatrices along a diagonal after the first submatrix. Wallace dubs processes which satisfy such a constraint "Quasi-Birth-and-Death" or QBD processes.

If U has such a structure, then π can be partitioned

into blocks corresponding to the submatrices of U:

$$\Pi = [\phi_0, \phi_1, \phi_2, \ldots]$$

ϕ_0 = m_0 component row vector

$\phi_1, \phi_2, \ldots$ = m component row vectors

Given this structuring, the equations (1) can be rewritten as:

$$\phi_0 \cdot E + \phi_1 \cdot C = \underline{0}$$

$$\phi_0 \cdot A + \phi_1 \cdot B + \phi_2 \cdot C = \underline{0} \qquad (2)$$

$$\phi_{n-1} \cdot A + \phi_n \cdot B + \phi_{n+1} \cdot C = \underline{0} \qquad n=2,3,\ldots$$

Next, Wallace assumes there is a solution $[\phi_0, \phi_1, \phi_2, \ldots]$ of the form

$$\phi_1 = \phi_0 \cdot \hat{R}$$

$$\phi_n = \phi_1 R^{n-1}$$

R=m x m matrix (3)

$\hat{R} = \begin{bmatrix} R \\ 0 \end{bmatrix}$ m_0 x m matrix

and ϕ is not all zero

Substituting (3) into (2) yields

$$\phi_o (E+\hat{R}\hat{C}) = \underline{0}$$

$$\phi_o(A+\hat{R}B+\hat{R}RC) = \underline{0} \qquad (4)$$

$$\phi_o \hat{R}R^{n-2} (A+RB+R^2C) = \underline{0}$$

The bulk of Wallace's work is spent in determining the conditions under which an R exists which satisfies (4). He demonstrates that sufficient conditions for the root matrix R to exist are that the QBD process is irreducible and positive recurrent.[1] In addition, Wallace defines an iteration which will converge to R if the QBD process is irreducible and positive recurrent. He also suggests an iterative technique to determine ϕ_o and a method for normalizing ϕ_o so that

1. A Markov chain is <u>irreducible</u> if all pairs of states communicate. It is <u>positive recurrent</u> if the mean recurrence time for all the states is finite.

$$\sum_{i=0}^{\infty} \phi_i \cdot \underline{1} = 1$$ where $\underline{1}$ is a column vector of the appropriate dimension containing all ones

Thus, if the host system model can be shown to represent an irreducible, positive recurrent QBD process, Wallace's techniques can be applied to find a root matrix R and an initial vector ϕ_0, from which the joint equilibrium probability distribution can be found as far out as desired, by using the relation $\phi_n = \phi_{n-1} R$.

To show that the host system model is an irreducible Markov chain, it is sufficient to show that from any state (n_T, n_B, s) any other state (n_T', n_B', s') is reachable through some sequence of transitions. To prove irreducibility, observe that from any state the system must be able to reach the idle state (this event will occur if there are no arrivals during the service of the current jobs enqueued). Further, any state (n_T, n_B, s) can be reached from the idle state as follows:

1. s=0

   A batch job arrives and begins service. During its service n terminal jobs and n batch jobs arrive, leaving the system in state $(n_T, n_B, 0)$ as required.

2. s>0

A terminal job arrives and begins service. During its service, s-1 more terminal jobs arrive. No batch jobs arrive until the last of these terminal jobs enters service. While the s^{th} terminal service is in progress, n_T terminals and n_B batch jobs arrive, leaving the system in state (n_T, n_B, s).

Given only the structure of the host system model, it is not possible to demonstrate that the model will be positive recurrent; this property depends on the particular values of λ_T, λ_B, μ_T, and μ_B chosen. Criteria for these parameters are not of concern, however, since Wallace (p. 95) concludes that if the QBD process is irreducible and his solution technique converges, then the process is necessarily positive recurrent. Thus, if a numerical solution can be computed for a host system model, the Markov chain represented must be irreducible and positive recurrent.

The next task in applying Wallace's solution to the host model is to demonstrate an ordering of states which will give the transition intensity matrix U the required block tri-diagonal form. The ordering to be used is organized by blocks of size ML+1 (as above, M = maximum number of terminals, L = number of server states). Each block contains all the states of the system in which exactly

n jobs are in the system, i.e., either queued or being served ($n_T+n_B+1=n$). Within a block, states are ordered by increasing number in the terminal queue (from 0 in the queue to M) and within these sub-blocks, by increasing number of the server state (0 to L-1). Thus, a typical block (for n=21) in a system for which L=3 and M=4 would be ordered:

n_T	n_B	S
0	20	0
0	20	1
0	20	2
1	19	0
1	19	1
1	19	2
2	18	0
2	18	1
2	18	2
3	17	0
3	17	1
3	17	2
4	16	0

states in block = ML+1 = 13

M=max # terminals = 4

L = # server states = 3

The initial blocks (where $n \leq M$) will not be as large as ML+1; for example, for n=2, would be:

n_T	n_B	S
0	1	0
0	1	1
0	1	2
1	0	0
1	0	1

block size = 6 =nL

n=2

The initial blocks thus each contain nL states and these blocks increase in size to ML (at n=M) after which all blocks are of size ML+1.

The A, B, and C matrices can now be defined. All of them are of dimension ML+1 x ML+1 and lie past the initial blocks of states. In all of these matrices, a non-zero entry defines the intensity of the transition from the state

indicated by the row to the state indicated by the column. The A-matrices, lying above the diagonal, represent arrivals to the row states (Fig. 2.2):

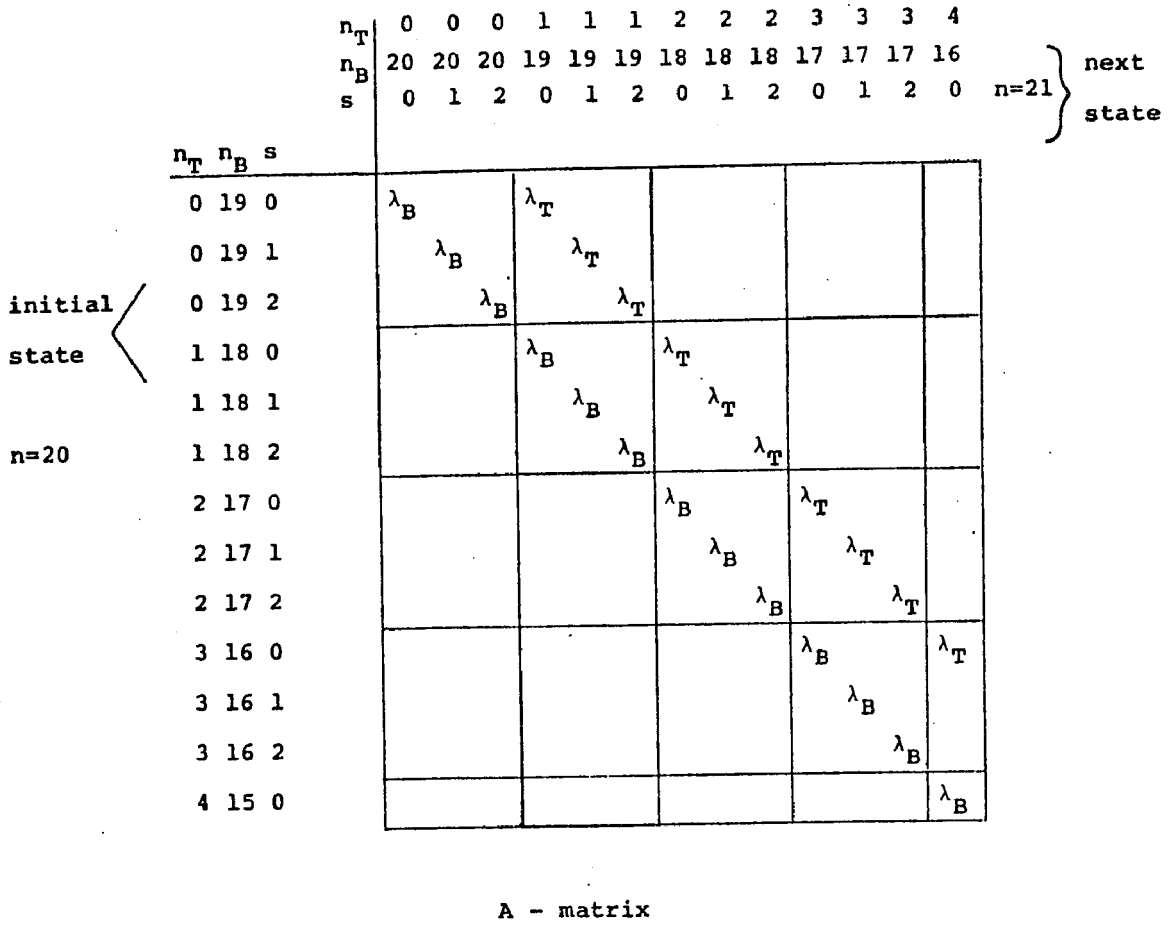
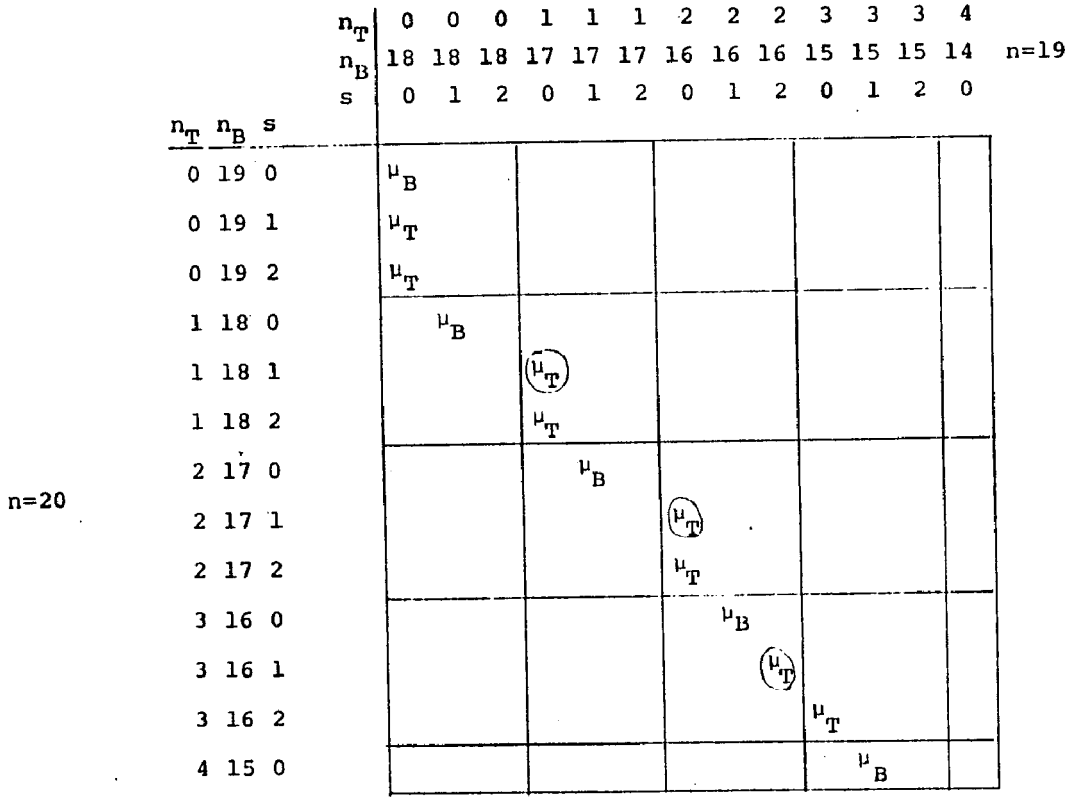


Fig. 2.2 A-Matrix

No terminal arrivals (rate λ_T) can occur in state (3,16,1) through (4,15,0) since all ports are already filled. Batch arrivals can occur in any state since there is no limit on batch queue size.

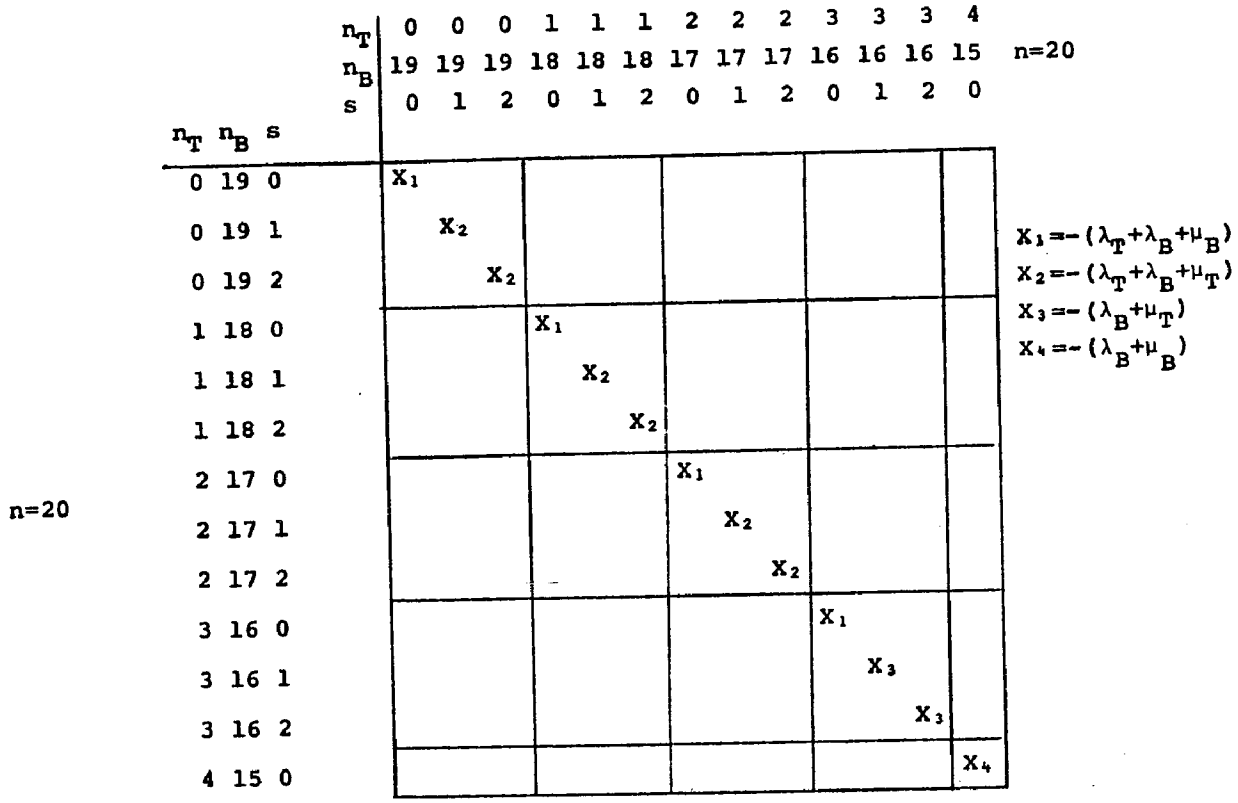
The sub-diagonal C-matrices are somewhat more complex, since they embody the breakpoint information. Each row of a C - matrix has only one entry, since the entry indicates departures from the system. Given the server state, the type of job being serviced is known; therefore, the intensity of the transition is known. The state to be entered after the departure must be in the previous block of states (since n is being decreased by one); the particular state entered is defined by the priority scheme. In the example under consideration suppose $b_1=3$ ($b_0=1$ and $b_2=\infty$; see Figure 2.3). Notice that the breakpoint b has effect only on the circled states. In the first two cases $n_T<b_1$, so the next server state will be zero (a batch job is chosen for execution), but in the final case, $n_T\geq b_1$, so a terminal job is selected.

The B sub-matrices, lying on the diagonal of U, simply give the intensities relating to the probability of no state change. Thus, as seen below, a B - matrix is diagonal, and the intensity in each row is the additive inverse of the sum of the intensities in the same row of the A and C matrices



C - matrix

Fig. 2.3 C-Matrix



B - matrix

Fig. 2.4 B-Matrix

(see Figure 2.4).

With the description of the B - matrix, the repetitive part of the U matrix is now defined. After the initial state blocks, each row-block of states corresponds to a C, B, and A matrix as described above:

$$U = \begin{bmatrix} E & \hat{A} & & & 0 \\ \hat{C} & B & A & & \\ & C & B & A & \\ & & \cdot & \cdot & \cdot \\ 0 & & & \cdot & \cdot & \cdot \end{bmatrix}$$

Now it is necessary to define the E - matrix so that the constraints $\hat{C} = [C\ 0]$ and $\hat{A} = \begin{bmatrix} A \\ 0 \end{bmatrix}$ are satisfied. If the state ordering described so far is obeyed strictly, the structure of non-zero blocks in U looks as shown in Figure 2.5. The blocks in E correspond to the initial blocks of increasing size. C and A do not meet the constraints in this form, but if E is expanded by one block in each direction and some of the initial states are re-ordered, then C and A will have the proper form (see Figure 2.6).

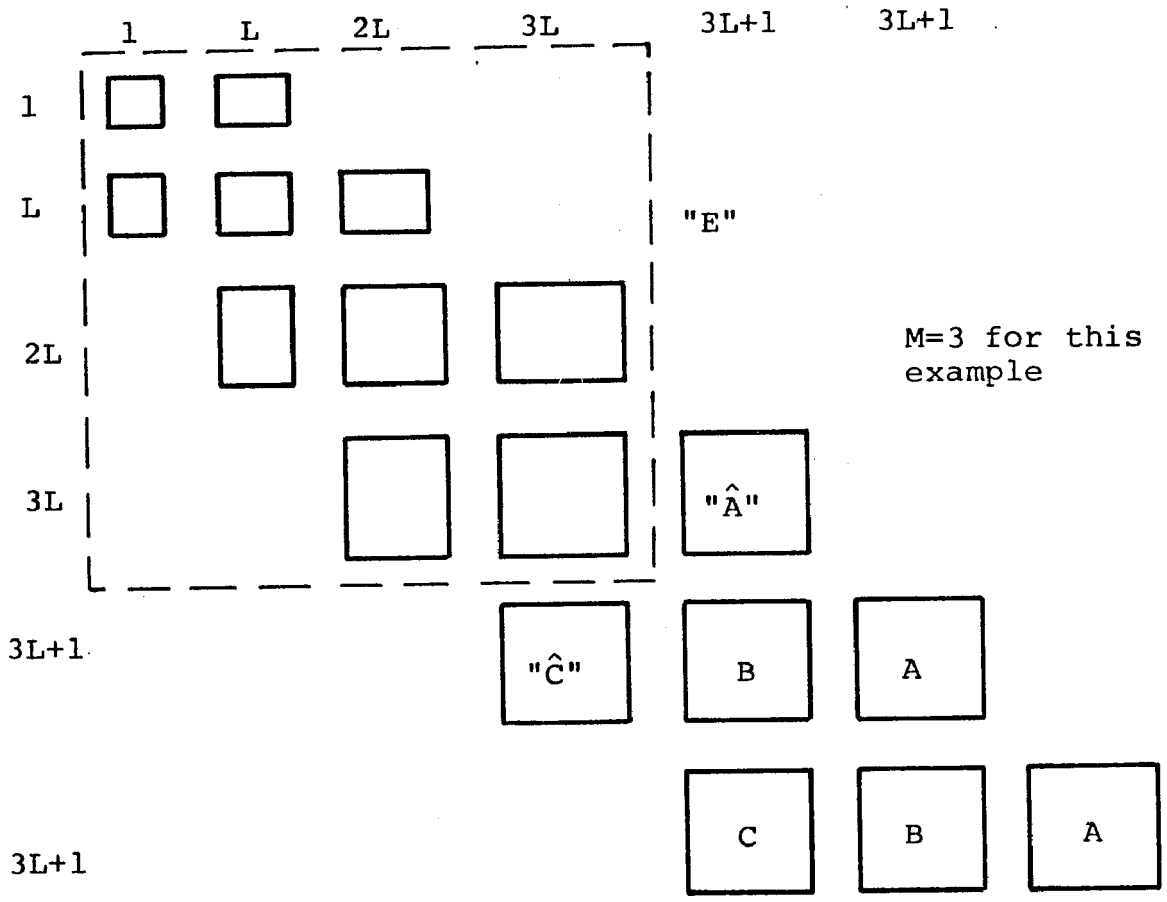


Fig. 2.5 Initial Block Structure

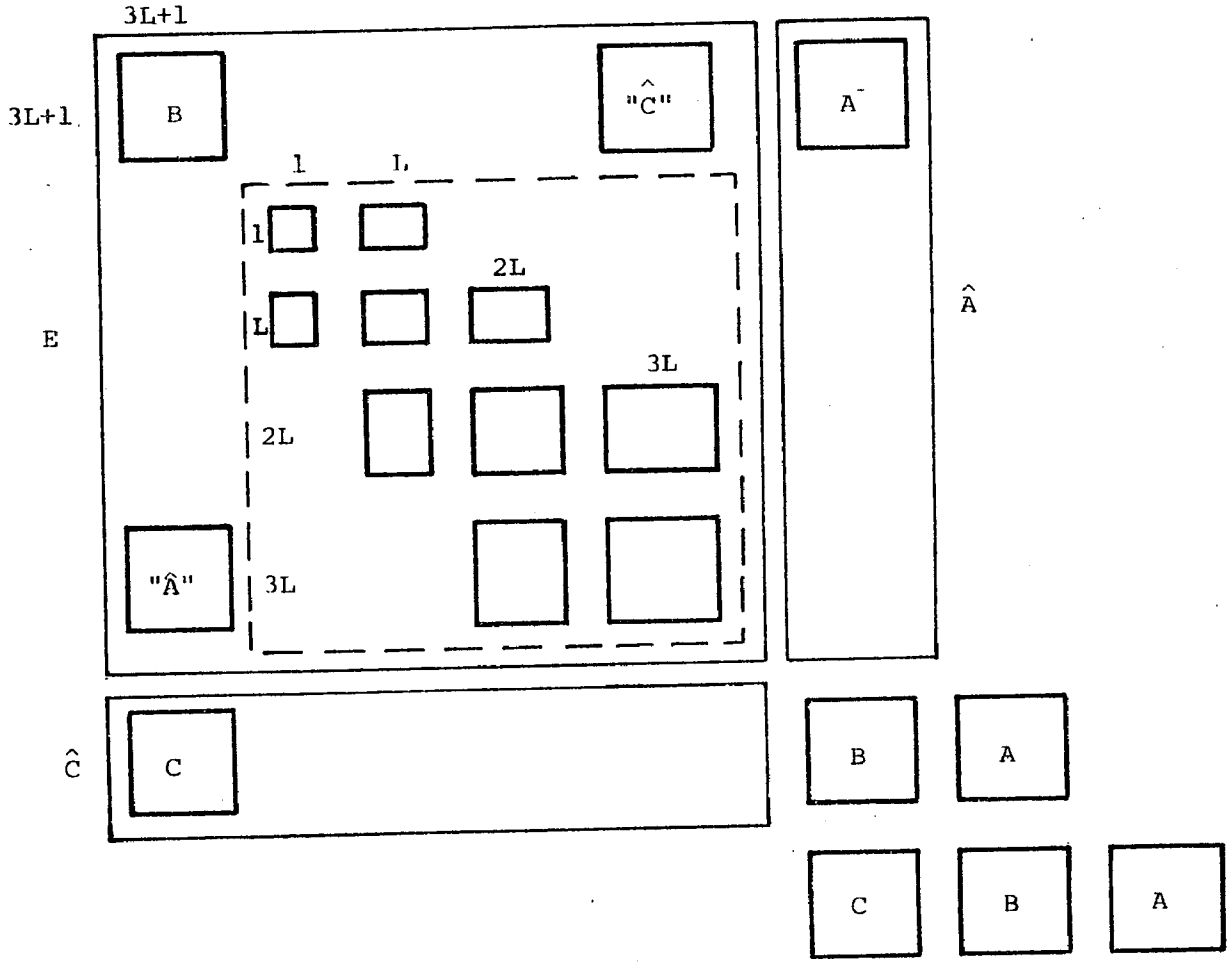


Fig. 2.6 Block Structure After Transformation

This transformation corresponds to example 2.1 (p. 13) in Wallace. Precisely stated, the final state ordering in terms of blocks is:

1. n=M+1
2. n=0 (idle state)
3. n=1
4. n=2

.
.
.

M+1. n=M

M+2. n=M+2

M+3. n=M+3

.
.
.

The number of rows in E, m_o, can now be computed. For a given n, if $0<n\leq M$, there will be nL rows, and if n=0, there is one row. For n>M, there are ML+1 rows (=m) as noted above. Then

$$m_o = \sum_{n=1}^{M} nL + 1 + ML + 1$$

$$= M(M+1)L/2 + ML + 1$$

$$= M(M+3)L/2 + 1$$

It has now been shown that the host system model is an

irreducible Markov chain (p. 39) and that, using the ordering of states described above, its transition intensity matrix satisfies the conditions Wallace defines for a QBD process. Thus, Wallace's techniques may be used to determine the joint equilibrium probability distribtuion of n_T, n_B, and s.

In order to compare various host systems operating under different conditions, a measure of system performance is required. The quality of service offered by a system in terms of the number and variety of different facilities (language translators, statistical packages, etc.) which are available is often of primary importance to the users of a system, but since we are dealing with a network of homogeneous operating systems, identical facilities are assumed to be available at all nodes. The question of performance then becomes one of how rapidly these services can be made available to users. For the individual batch user, the performance of the system is summarized by the turnaround time for this particular job. In the queueing model described above, batch turnaround time consists of the time a batch job spends in the batch queue (the waiting time) plus the service time for the job. The sum of these is called the system residence time for a batch job. Other measures of the system (such as the CPU utilization and queue length distribution) may be of interest, but the user is most directly aware of the system residence time of his

job.

The most complete predictions of system residence times would be provided by the equilibrium probability distribution for that statistic. Because of the complexity of the priority structure in the single host model, the derivation of that distribution would, however, be extremely difficult. The mean system residence time is not so difficult to derive and still provides a significant measure of performance. Two systems may reasonably be compared on the basis of their respective mean system residence times, and transshipment decisions could be made on the basis of the least expected residence time. Definitions:

L_{B_t} = number of batch jobs in system at time t

$\bar{L}_B$ = mean number of batch jobs in system in steady state

$\bar{L}_{B_{BD}}$ = mean number of batch jobs in system, measured at batch departure points

$\bar{L}_{B_{BA}}$ = mean number of batch jobs in system, measured at batch arrival points

$\bar{L}_{B_{AD}}$ = mean number of batch jobs in system, measured at all departure points (batch and terminal)

W_{B_n} = system residence time of the n batch job

$\bar{W}_B$ = mean system residence time for batch jobs

Suppose batch job b arrives at time a and finds L batch jobs already in the system. All of these jobs must be

serviced before job b_n if a FCFS (i.e., first-come, first-served) service discipline is assumed for the batch queue. At time d_n, job b_n finally departs from the system. At that time, the number of batch jobs in the system must be equal to the number of batch arrivals which occurred during b_n's system residence time. That is,

$$L_{B_{d_n}} = \text{\# batch jobs which arrived during } W_{B_n}$$

If the system is observed over a long period, there will be many jobs which have nearly equal system residence times ($t<W_{B_i}<t+dt$). Since arrivals are Poisson-distributed with rate λ_B, the mean number of batch arrivals during a period of length t must be $\lambda_B t$. So, if all observations for which $W_B=t$ are grouped, the mean number of batch jobs in the system just after the departure of a batch job whose system residence time was t seconds is

$$\overline{L}_{B_{W_{B_i}=t}} = \lambda_B t$$

Now the proportion of observations for which $t<W_{B_i}<t+dt$ is simply the probability that a batch job will have a system residence time between t and t+dt seconds. So, the mean number of batch jobs in the system at an arbitrary batch departure is given by:

$$\overline{L}_{B_{BD}} = \lambda_B \int_0^\infty p(t<W_{B_i}<t+dt)\, t\, dt = \lambda_B \overline{W_B}$$

Since the single host model is Markov, the equilibrium probability distribution derived above yields $\overline{L_B}$ (see K4), and since batch arrivals to the system are Poisson-distributed,

$$\overline{L}_{B} = \overline{L}_{B_{BA}}$$

The remaining question, then, is

$$\overline{L}_{B_{BA}} \stackrel{?}{=} \overline{L}_{B_{BD}}$$

If the model forms an irreducible, positive recurrent Markov chain, for every arrival that occurs with exactly n batch jobs in the system, there must be a batch departure which leaves exactly n batch jobs in the system. Consequently,

$$\overline{L}_{B_{BA}} = \overline{L}_{B_{BD}} = \overline{L}_{B}$$

and so,

$$\overline{L}_{B} = \lambda_{B} \overline{W}_{B} \quad .$$

This relation is commonly known as Little's theorem (L2) and has been shown to be valid for wide classes of queueing systems. As we have shown above, the model forms an irreducible Markov chain which is positive recurrent in every case for which a solution can be computed by Wallace's algorithm. Thus, the theorem will apply to all such cases.

2.3 Data Collection

In order to test the validity of the single node model, it is appropriate to investigate how well the model fits data from some particular computing system. Since the University of Michigan Terminal System (MTS) is the most easily available system which satisfies the general constraints of the model (i.e., it is a large scale system that processes batch and interactive jobs), MTS run on the

duplex IBM 360/67 at the University of Michigan Computing Center was chosen for this exercise. The general procedure followed was to (1) collect appropriate data from the system, (2) use some of the data to determine the parameter values for the model, and (3) examine how well the analytic results match the observed behavior of the system, using the rest of the data.

Descriptions of aspects of MTS may be found in (A5,A6,M4,M5). We will give only a brief overview of the system to clarify its relationship to the model. The basic software unit in the system is referred to as a task (analogous to the more commonly used process). There are separate tasks to handle various overhead functions such as paging and spooling, and there is an MTS task for each user signed on to the system. These MTS tasks implement the basic computing environment in which the user runs his programs. An interactive user gains access to the system by dialing one of several possible telephone numbers. If there is a port available which has an MTS task attached to it, it is assigned to his terminal. A user desiring batch access submits his card deck (i.e., his job) to a reader. The deck is read and spooled onto disk by HASP. The job is assigned a priority according to its time estimate, and at some later time it is retrieved from the disk according to its priority and assigned to an MTS task for execution. The results of the run are returned to HASP and spooled for printing and/or

punching, with a priority determined by the amount of output generated. When executing in MTS, a batch job competes for resources on an equal basis with terminal jobs -- the system makes no distinction in this respect.

In order to prevent batch jobs from saturating the system and causing intolerable response times for interactive users, there is a control on how many batch jobs may be executing simultaneously. A particular task is designated to collect statistics on system load approximately every twenty seconds. These statistics include measures of activity in various parts of the system -- channel activity, disk and drum activity, number of jobs in the batch queue, number of active tasks, number of virtual and real pages allocated, hardware configuration, CPU idle time, etc. From these measures, the task constructs several different load factors. These factors are combined and weighted by the previous values observed to form a final index of the current system load and performance. From these calculations, the task determines the preferred number of batch streams to be run in order to keep the system "balanced." The current number of batch streams (i.e., MTS tasks executing batch jobs) may be larger or smaller than the preferred number. If it is smaller, more batch streams will be started up; if larger, currently executing batch jobs are allowed to complete execution (there is no preempting), but at termination of the current job, the

stream will be shut down. During the data collection periods in this research, the preferred number of batch streams was always at least one, so that a minimum level of batch throughput was maintained. The algorithm has since been altered to allow zero batch streams.

In fact, the decision making process is somewhat more complex than this, since each batch stream only services jobs in a certain priority range, but the general structure is as described above. The algorithms and thresholds involved in the decisions are modified and tuned from time to time in order to keep the system running efficiently. However, this technique of regulating batch service depending on the impact of the terminal load is similar in motivation to the breakpoint priority scheme in the single host queueing model developed above. If adequate response times for interactive users are to be maintained, some such strategy is required.

The hardware on which the system runs is shown in Figure 2.7. Briefly, there are two CPU's (IBM 360/67), eight storage boxes (two megabytes), and two channel controllers through which selector and multiplexor channels are interfaced to the CPU's. Many peripherals, including drums, disks, tape drives, and front end I/O processors are connected to the various channels. Because the system

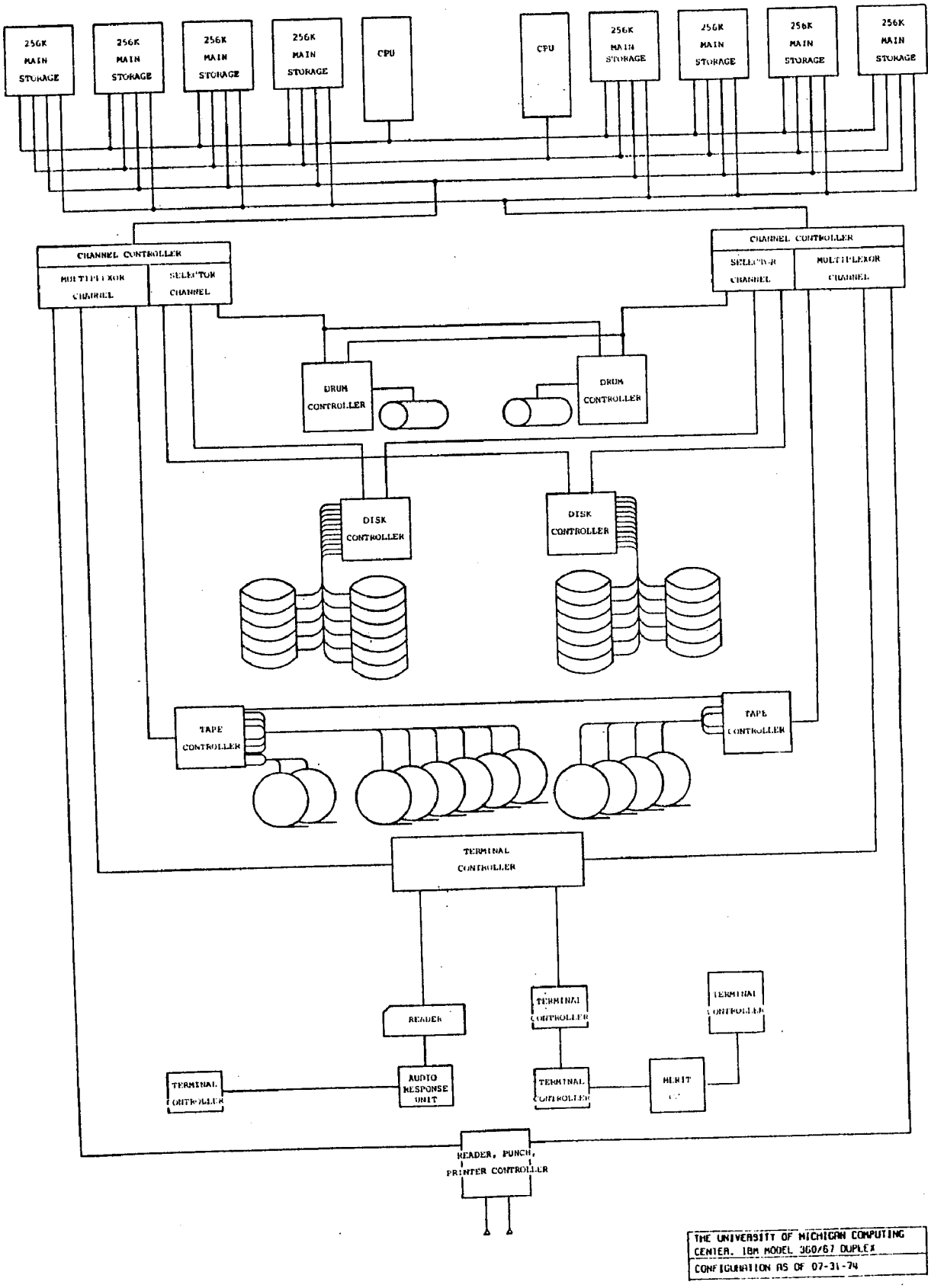


Fig. 2.7 University of Michigan Computing Center Machine Configuration

software is written to exploit the modularity of the hardware, various peripherals, storage boxes, and even CPU's may be partitioned in or out of the system.

One of the benefits of constructing a model for computer systems is that a well-constructed model yields a viewpoint from which to view a complex system. In particular, it enables us to put aside vast quantities of data that may be generated by the system and concentrate on those data which are relevant to the model. Data relevant to the single host model are those which would define arrival and service rates for batch and terminal jobs, batch queue lengths, and mean waiting times for batch jobs. There are presently two sources of such data in the MTS system. First, there are global records of the system state written into a file approximately every 20 seconds, as mentioned above. For our purposes, these records provide the record of actual CPU idle time and the batch queue length distribution over time. Second, the system accounting records, kept for billing purposes, provide statistics on the times of arrival and departure of jobs and the actual CPU time charged to a given job. Unfortunately, the arrival time of a batch job is the time at which it begins MTS execution, so there is no record of its wait in the HASP queues. Consequently, estimates of the mean waiting time for batch jobs are not possible. Since the CPU was the limiting resource in the system as it was configured when these measurements were made, CPU time will

be used as synonymous with service time in the model. In studying a different system, other functions might be used in the determination of the service rate. For example, if the system had a bottleneck at the channel or a shortage of memory, these factors could be included in characterizing the demand that a job placed on the system. For the case under consideration, however, CPU time required by a job will be used as an approximation of the job's total system demand.

Since both the job accounting and system status data are collected automatically at all times and saved on magnetic tape, the measurements themselves did not impose any additional unusual load on the system. Further, since monthly summaries of some statistics are generated routinely (Figures 2.8 and 2.9), it was possible to look over the monthly summary and pick out periods in which the system appeared to be in a relatively stable state. The average CPU utilization and batch queue length were used as indicators of the system state for this purpose. From the summaries, the system appears to exist in three roughly definable operating regions:

1. Lightly loaded: $0\% < \overline{CPU} \leq 60\%$; $\overline{\text{Batch Queue}} = 0$
2. Moderately loaded: $60\% < \overline{CPU} < 90\%$; $0 < \overline{\text{Batch Queue}} \leq 5$
3. Heavily loaded: $90\% \leq \overline{CPU}$; $\overline{\text{Batch Queue}} > 5$

In order to minimize transient effects and startup and shutdown problems, measurement periods were required to be of at least two hours duration, during which time all major

CPU UTILIZATION BY HOUR OF DAY (%) October 1973

DATE	09	10	11	12	13	14	15	16	17	18	19	20	21	22	23
01 MON	86	97	71	98	85	98	97	93	96	70	57	53	49	62	74
02 TUE	81	91	95	100	100	100	100	100	100	97	81	98	99	96	89
03 WED	85	91	92	64	88	92	96	99	96	53	54	61	57	48	46
04 THU	90	93	98	84	92	97	97	95	98	74	48	70	85	81	32
05 FRI	76	92	93	85	89	96	98	96	81	40	20	41	30	32	28
06 SAT	10	28	44	30	33	37	37	27	27	21	24	27	26	0	0
07 SUN	0	0	0	0	0	21	74	90	51	46	29	28	36	32	27
08 MON	73	76	97	72	89	99	100	99	86	50	49	68	38	89	76
09 TUE	79	93	100	98	96	100	96	99	86	39	49	78	58	39	46
10 WED	83	98	99	66	82	87	74	63	57	33	39	52	56	57	30
11 THU	75	98	92	80	98	100	98	100	97	47	0	0	40	21	0
12 FRI	96	100	99	100	100	100	98	97	68	35	31	32	28	27	22
13 SAT	15	27	39	0	0	0	18	31	31	17	23	32	24	10	6
14 SUN	0	0	0	0	0	24	46	44	34	28	24	32	33	29	29
15 MON	82	89	95	96	86	92	94	95	66	36	30	44	73	54	32
16 TUE	83	86	99	70	91	85	97	86	53	45	47	47	50	35	26
17 WED	64	75	96	58	81	100	98	98	58	35	45	38	56	44	32
18 THU	100	88	93	72	79	96	94	85	56	51	38	45	54	41	31
19 FRI	99	100	96	72	73	92	90	85	59	25	20	32	42	25	29
20 SAT	10	15	27	55	23	21	32	20	24	12	8	13	23	6	16
21 SUN	0	0	0	0	0	14	32	57	42	21	33	66	57	29	33
22 MON	51	77	63	62	73	85	94	94	58	43	54	59	64	55	55
23 TUE	84	88	92	64	89	100	94	99	76	31	50	69	65	67	51
24 WED	71	79	91	56	53	89	93	90	62	29	36	47	75	72	55
25 THU	69	90	91	61	78	99	96	89	49	42	40	76	77	45	39
26 FRI	75	84	92	79	66	85	99	98	61	32	18	26	28	26	26
27 SAT	6	13	28	50	47	50	30	39	16	5	0	31	15	41	0
28 SUN	0	0	0	0	0	0	0	0	66	72	57	63	58	51	31
29 MON	96	22	99	79	99	100	100	99	61	39	45	67	71	54	43
30 TUE	97	99	84	85	88	98	96	71	76	35	56	57	56	72	56
31 WED	88	90	66	85	48	99	91	98	79	48	60	94	91	98	83

Fig. 2.8 Mean CPU Utilization by Hour

HASP BATCH QUEUE BY HOUR OF DAY
AVERAGE LANGTH FOR EACH HOUR October 1973

DATE	08	09	10	11	12	13	14	15	16	17	18	19	20	21	22	23
01 MON	13	9	13	50	12	13	47	65	104	41	4	3	4	4	4	3
02 TUE	0	0	3	26	30	40	65	89	108	63	4	3	5	5	4	3
03 WED	0	0	4	12	3	16	71	68	71	24	2	2	2	2	3	3
04 THU	1	27	89	90	120	93	63	96	117	57	6	1	1	2	5	6
05 FRI	1	1	3	4	3	6	19	27	17	2	1	1	1	2	2	2
06 SAT	0	0	0	0	0	1	1	1	1	1	2	2	1	1	0	0
07 SUN	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1
08 MON	0	0	0	1	1	3	26	15	67	11	3	4	4	50	15	5
09 TUE	0	0	7	12	2	11	38	31	23	12	7	7	7	7	7	7
10 WED	1	0	9	2	0	3	3	3	10	8	5	5	5	5	5	6
11 THU	0	1	16	15	41	16	65	42	27	15	8	0	0	3	16	0
12 FRI	0	5	15	14	24	12	11	9	3	3	3	3	3	3	3	3
13 SAT	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
14 SUN	0	0	0	0	0	0	9	2	0	0	0	0	0	0	1	0
15 MON	7	0	1	3	2	3	8	1	10	1	1	1	1	1	1	1
16 TUE	0	1	0	11	3	3	3	6	8	3	3	3	3	3	3	3
17 WED	0	0	3	4	2	7	17	8	2	2	2	2	2	2	2	2
18 THU	2	18	1	3	2	3	3	4	4	4	4	4	4	4	5	5
19 FRI	4	8	7	8	1	1	3	3	3	4	4	4	4	4	4	4
20 SAT	0	0	0	0	2	0	1	1	0	0	0	0	0	0	0	0
21 SUN	0	0	0	0	0	0	0	0	0	0	0	1	2	0	0	0
22 MON	0	0	1	2	2	18	56	11	7	4	4	4	4	4	5	5
23 TUE	0	0	4	5	0	20	16	9	10	6	3	3	3	3	5	3
24 WED	1	3	3	3	2	3	7	3	4	4	4	4	4	4	4	5
25 THU	0	0	0	1	1	3	24	26	4	1	1	2	3	6	2	2
26 FRI	0	0	0	6	2	3	5	17	10	1	1	1	1	1	1	1
27 SAT	0	0	0	1	0	0	0	0	0	0	8	0	0	0	0	0
28 SUN	0	0	0	0	0	0	0	0	0	0	0	0	0	0	9	16
29 MON	5	8	18	27	51	6	64	106	63	6	4	4	4	4	4	4
30 TUE	0	1	4	9	1	2	4	7	2	3	2	2	2	2	2	2
31 WED	0	0	2	17	8	59	119	197	158	9	3	3	3	3	3	3

Fig. 2.9 Mean Batch Queue Length by Hour

hardware (CPU's, storage, and channel controllers) was operating and the system was in a stable state. The periods in October 1973 actually chosen for measurement are underlined in Figures 2.8 and 2.9.

Programs were written to extract the desired statistics from the records of the chosen periods. The distributions for the CPU time used by batch and terminal jobs were measured from the accounting statistics, while the batch queue length distribution and total CPU usage were taken from the system status data. (Total CPU usage is not simply the sum of the individual job CPU totals because system overhead for paging and some other functions is not charged to users). These statistics are the fundamental ones used in the following section to validate the model.

A problem arose in the data for the batch queue length distribution. HASP maintains all jobs awaiting execution on a single queue, and these jobs are chosen to be executed according to these priorities, as described above. The lowest priority jobs in the system, however, are never allowed to run unless the time is between midnight and eight a.m. (even if the batch queue is empty except for these low priority jobs). Unfortunately, the system statistics only reveal the number of batch jobs in the system; the priorities of the jobs are not recorded. Consequently, distributions such as those shown in Figures 2.10 and 2.11

```
STATISTICS FROM JOBS PROGRAM FOR 10-12-73, FROM 19: 0 TO 23: 0.

                                        MEAN        VARIANCE      STD DEV

NUMBER OF BATCH JOBS IN SYSTEM          3.405         0.396        0.630

NUMBER OF TERMINAL JOBS IN SYSTEM      17.824         6.169        2.484

PER CENT CPU IDLE                      69.979       447.230       21.148

TOTAL CPU TIME AVAILABLE = 28788.73 SEC.

TOTAL IDLE CPU TIME      = 20289.45 SEC.

FRACTION OF CPUS UNUSED  =    70.48 PER CENT


  HISTOGRAM FOR NUMBER OF BATCH JOBS IN SYSTEM

ONE "X" =       191 SECONDS OR 1.3269%

  #      CUM          CUM SEC        %         # SEC
  0       0.0            0.0        0.0         0.0|
  1       0.0            0.0        0.0         0.0|
  2       0.0            0.0        0.0         0.0|
  3     66.3250       9547.059     66.3250    9547.1|XXXX
```

STATISTICS FROM JOBS PROGRAM FOR 10-10-73, FROM 13: 0 TO 16: 0.

	MEAN	VARIANCE	STD DEV
NUMBER OF BATCH JOBS IN SYSTEM	4.795	3.095	1.759
NUMBER OF TERMINAL JOBS IN SYSTEM	63.337	53.728	7.330
PER CENT CPU IDLE	19.093	546.219	23.371

TOTAL CPU TIME AVAILABLE = 21607.48 SEC.

TOTAL IDLE CPU TIME = 4212.72 SEC.

FRACTION OF CPUS UNUSED = 19.50 PER CENT

HISTOGRAM FOR NUMBER OF BATCH JOBS IN SYSTEM

ONE "X" = 57 SECONDS OR 0.5276%

#	CUM	CUM SEC	%	# SEC	
0	0.0	0.0	0.0	0.0	
1	0.0	0.0	0.0	0.0	
2	0.0	0.0	0.0	0.0	
3	25.9470	2803.252	25.9470	2803.3	XXXX

occur. In these relatively lightly loaded periods it is not difficult to compensate for the low priority jobs by shifting the distribution over to zero. When the system is heavily loaded, the problem is more difficult, since the queue length distribution as a whole tends to move away from zero. The number of low priority jobs becomes less clear, but in all cases we have attempted to compensate for this problem as accurately as possible.

In order to present an overall picture of the batch queue length distribution in different system states, the histograms from the observation periods have been summed to form Figures 2.12, 2.13, and 2.14. The light and moderate period distributions have the same general shape, while the heavy period histogram shows an unusual sort of oscillation. There is no apparent reason for the multiple peaks in the scheduling algorithm. Part of the explanation may be in the low priority job phenomenon: If a low priority job arrives in the middle of a measurement period, it will tend to displace the histogram for only half the period, thus "smearing" the entire distribution.

The interarrival time and CPU usage distributions were checked visually to be sure that they did not differ violently from the exponential assumption. Some typical examples have been included in Figures 2.15 - 2.18. The shapes of these histograms would probably be most closely

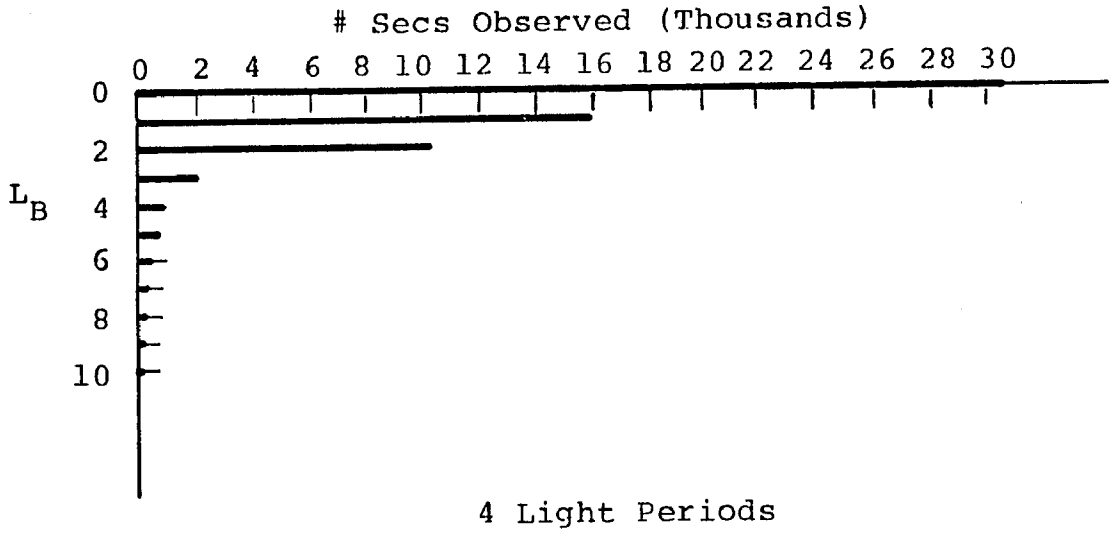


Fig. 2.12 Batch Queue Length Distribution-Light Periods

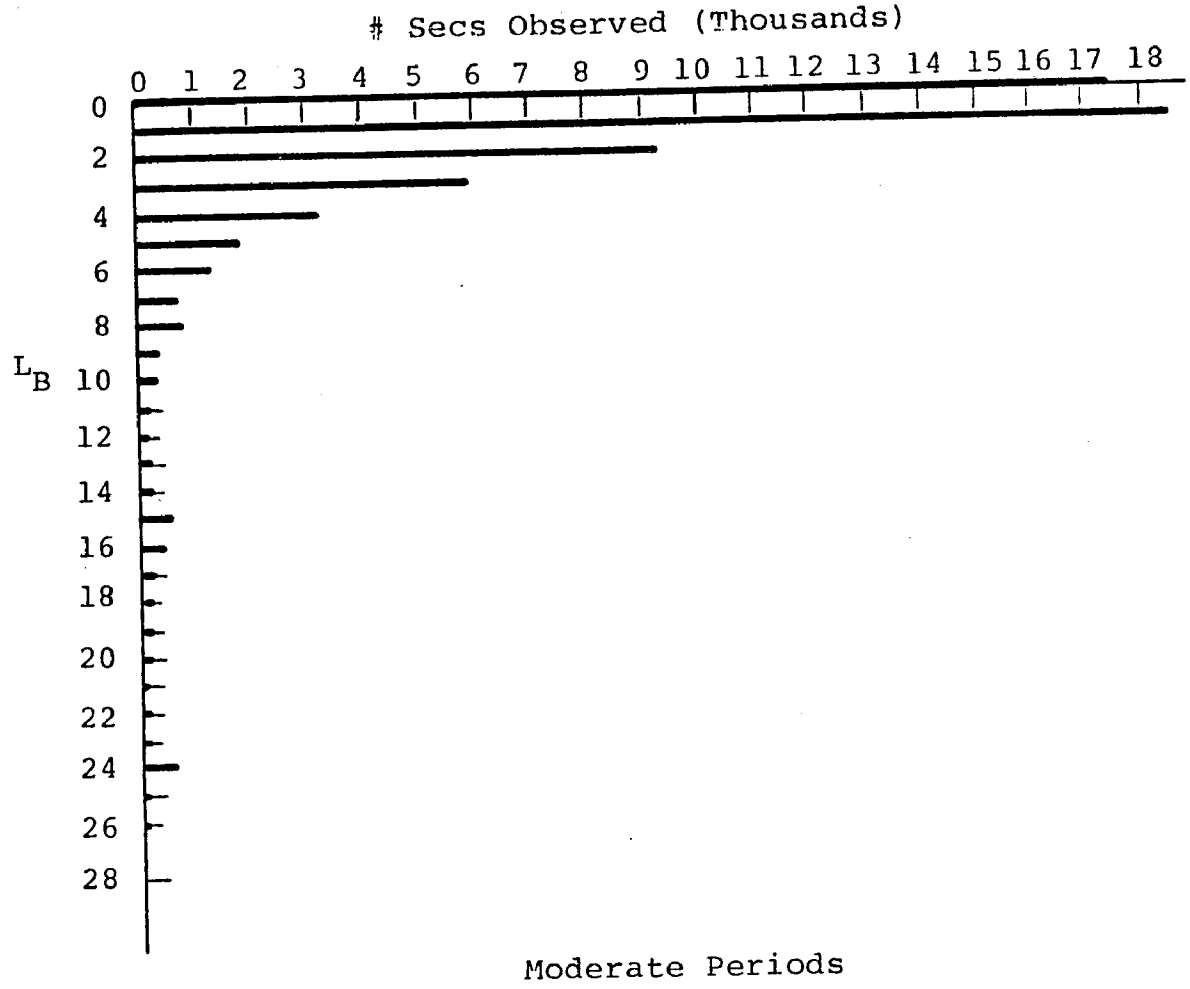


Fig. 2.13 Batch Queue Length Distribution-Moderate Periods

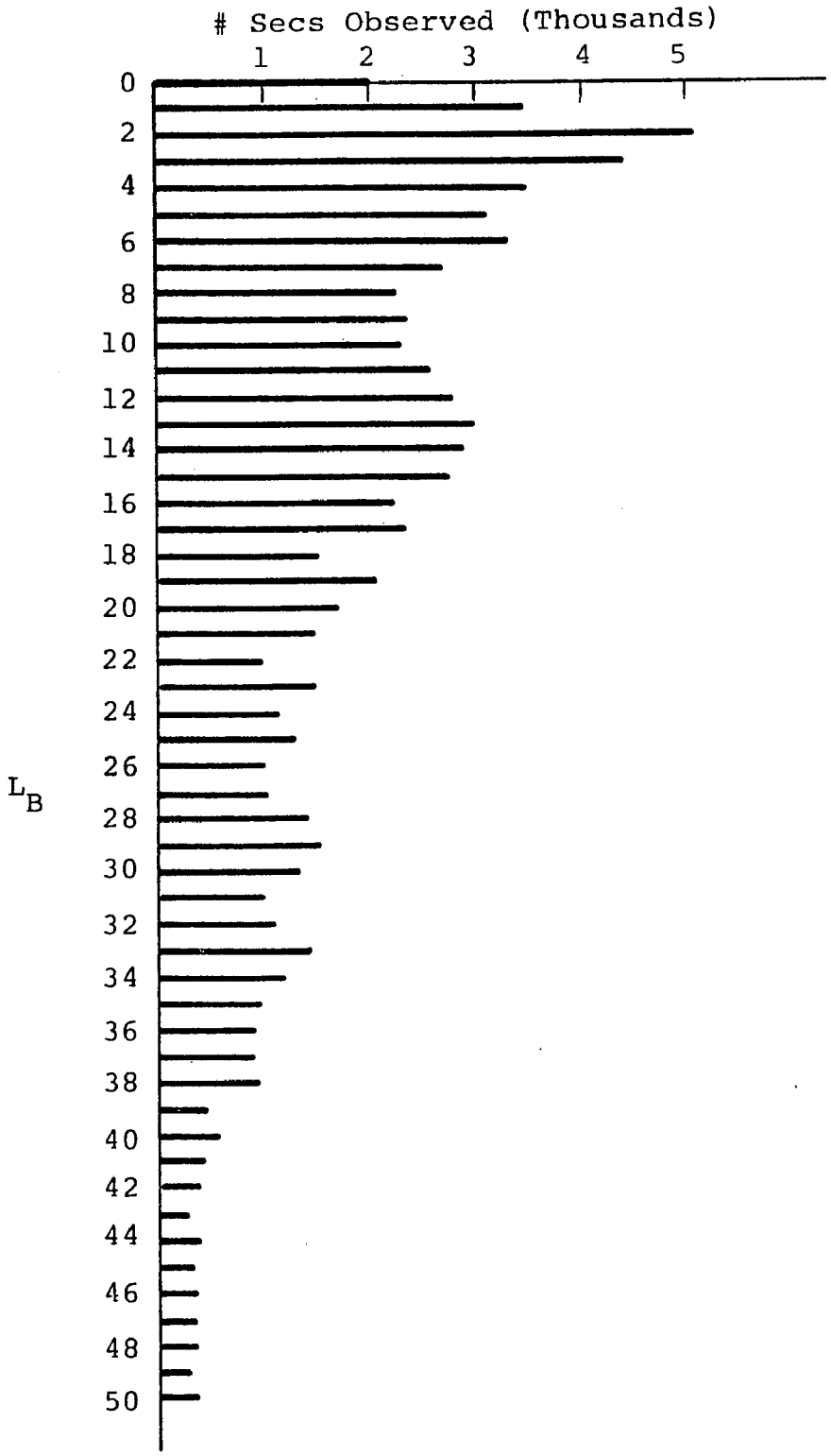


Fig. 2.14 Batch Queue Length Distribution-
Heavy Periods

```
HISTOGRAM/FREQUENCIES
MIDPOINT     COUNT FOR ON.DT.T      (EACH X = 15)
 0.           589 +XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
 25.759       467 +XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
 51.517       150 +XXXXXXXXXX
 77.276        60 +XXXX
 103.03        17 +XX
 128.79        10 +X
 154.55         4 +X
 180.31         2 +X
 206.07         2 +X
 231.83         2 +X
 257.59         1 +X
 283.34         0 +
 309.10         0 +
 334.86         0 +
 360.62         0 +                          Terminals
 386.38         0 +                          OV213.10-18
 412.14         1 +X                         Heavy
 437.90         0 +                          (Period 14)
 436.66         1 +X
 489.41         0 +
 515.17         1 +X
 540.93         0 +
 566.69         2 +X
 592.45         0 +
 618.21         0 +
 643.97         0 +
 669.72         0 +
 695.48         0 +
 721.24         0 +
 747.00         1 +X
 MISSING     1044
 TOTAL       2354  ( 25.759    = INTERVAL WIDTH)
```

Fig. 2.15 Terminal Interarrival Distribution -- Period 14

```
HISTOGRAM/FREQUENCIES
MIDPOINT     COUNT FOR ON.DT.B      (EACH X = 4)
 0.            34 +XXXXXXXXX
 12.944       146 +XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
 25.889        70 +XXXXXXXXXXXXXXXXXX
 38.833        30 +XXXXXXXX
 51.778        17 +XXXXX
 64.722         8 +XX
 77.667         9 +XXX                       Batch
 90.611         8 +XX                        CT103.13-16 Moderate
 103.56         5 +XX
 116.50         2 +X
 129.44         1 +X
 142.39         1 +X
 155.33         0 +
 168.28         1 +X
 181.22         1 +X
 194.17         0 +
 207.11         1 +X
 220.06         0 +
 233.00         1 +X
 MISSING      533
 TOTAL        868  ( 12.944    = INTERVAL WIDTH)
```

Fig. 2.16 Batch Interarrival Distribution -- Period 8

```
HISTOGRAM/FREQUENCIES

MIDPOINT     COUNT FOR CPU(SEC)     (EACH X = 2)

0.             0 +
1.0000        61 +XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
2.0000        60 +XXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
3.0000        35 +XXXXXXXXXXXXXXXXX
4.0000        19 +XXXXXXXXXX
5.0000        28 +XXXXXXXXXXXXX
6.0000        21 +XXXXXXXXXX
7.0000        11 +XXXXX
8.0000         9 +XXXX
9.0000         8 +XXXX
10.000         2 +X
11.000         4 +XX
12.000         3 +XX
13.000         6 +XXX
14.000         6 +XXX
15.000         1 +X
16.000         1 +X
17.000         2 +X
18.000         1 +X
19.000         6 +XXX
20.000         1 +X
21.000         0 +
22.000         1 +X
23.000         0 +
24.000         0 +
25.000         0 +
26.000         0 +
27.000         0 +
28.000         0 +
29.000         0 +
30.000         0 +
31.000         0 +
32.000         0 +
33.000         1 +X
34.000         0 +
35.000         0 +
36.000         1 +X
37.000         0 +
38.000         0 +
39.000         0 +
40.000         1 +X
41.000         0 +
42.000         0 +
43.000         0 +
44.000         0 +
45.000         0 +
46.000         2 +X
47.000         0 +
48.000         0 +
49.000         0 +

BATCH
CT123 19-23 - light
Period 2
```

Fig. 2.17 Batch CPU Service Distribution -- Period 2

```
HISTOGRAM/FREQUENCIES

MIDPOINT     COUNT FOR CPU(SEC)     (EACH X = 6)

0.              93 +XXXXXXXXXXXXXXX
2.0000         208 +XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
4.0000         126 +XXXXXXXXXXXXXXXXXXXX
6.0000          77 +XXXXXXXXXXXXX
8.0000          41 +XXXXXXX
10.000          33 +XXXXXX
12.000          29 +XXXXX
14.000          15 +XXX
16.000          27 +XXXXX
18.000          13 +XXX
20.000          11 +XX
22.000          16 +XXX
24.000           6 +X
26.000           8 +XX
28.000           8 +XX
30.000           4 +X
32.000           6 +X
34.000           9 +XX
36.000           7 +XX
38.000           4 +X
40.000           2 +X
42.000           4 +X                          Terminals
44.000           5 +X                          CT08.13-18  Heavy
46.000           3 +X
48.000           2 +X
50.000           4 +X
52.000           0 +
54.000           1 +X
56.000           3 +X
58.000           1 +X
60.000           2 +X
62.000           0 +
64.000           2 +X
66.000           2 +X
68.000           2 +X
70.000           0 +
72.000           3 +X
74.000           2 +X
76.000           1 +X
78.000           1 +X
80.000           2 +X
82.000           0 +
84.000           0 +
86.000           1 +X
88.000           1 +X
90.000           3 +X
92.000           0 +
94.000           2 +X
96.000           1 +X
98.000           0 +
100.00           1 +X

                21 > 100.00
TOTAL          813 ( 2.0000    = INTERVAL WIDTH)
```

Fig. 2.18 Terminal CPU Service Distribution -- Period 13

matched by hyperexponential distributions, since they tend to show an initial rise and a relatively high variance, but the exponential assumption simplifies the analysis considerably and does not appear to be greatly in error.

Table 2.1 contains a summary of the basic data collected for the 15 periods chosen for measurement. The periods have been ordered by CPU utilization; the first four are from lightly loaded periods, the next four from moderately loaded periods, and the last seven from heavily loaded intervals. More studies were made of heavily loaded cases since there was a considerably greater variation in system behavior in periods of heavy demand.

A number of trends may be observed in the data. As the CPU utilization increases, the proportion of batch jobs in the mix tends to decrease, and the mean CPU time used by a batch job tends to increase. Conversely, the mean CPU time used by interactive jobs generally decreases as system load increases. This last effect may be due to a human factors consideration: people will only sit at a terminal for a fixed length of time, and if the system is heavily loaded, each user gets less actual CPU time during that period. Also, proportionately more of the non-idle CPU time is spent performing overhead operations (primarily paging, see Section 2.4) when the system is heavily loaded. Again, this observation is to be expected. Finally, the adjusted mean

Table 2.1 Summary of MTS Measurements

Period	Date & Time	Length (hours)	#Batch / #I.A.	CPU Util.	% of Util. CPU not Charged	Mean CPU use Batch	Mean CPU use I.A.	Adj Mean Batch Q	Std. Dev.	
1	10/26 8pm	4	2.32	26.31	29.82	5.77	21.18	.374	.65	
2	10/12 7pm	4	2.03	29.52	24.02	7.34	29.29	.405	.63	light
3	10/27 12noon	3	2.01	48.90	34.07	9.00	24.72	.823	1.23	
4	10/24 8pm	4	1.53	60.33	22.16	7.43	29.00	2.08	3.68	
5	10/22 7pm	5	1.90	57.36	28.48	8.78	14.40	1.65	1.88	
6	10/23 8pm	3	2.07	66.91	28.81	6.93	14.83	2.27	3.51	moderate
7	10/26 9am	5	.952	79.1	31.5	10.72	18.33	3.39	5.64	
8	10/10 1pm	3	.63	80.51	37.72	7.21	15.81	1.80	1.76	
9	10/26 2pm	3	.938	91.95	38.7	7.81	18.77	12.2	9.46	
10	10/17 1pm	4	.792	93.95	37.17	14.43	13.00	10.34	8.84	
11	10/25 1pm	3	.751	90.02	44.3	8.28	15.74	19.4	16.1	heavy
12	10/9 1pm	5	.974	95.32	26.48	10.27	18.59	25.8	14.4	
13	10/8 1pm	5	.835	94.67	38.35	11.23	16.46	27.3	26.0	
14	11/21 10am	8	.796	98.30	44.55	10.17	15.86	50.1	37.8	
15	10/12 10am	5	.776	99.43	35.7	11.41	19.28	13.8	9.6	

batch queue length also increases as the CPU utilization increases.

In the following section, these data will be used to determine parameters to use in the model and to evaluate the correspondence between the analytic predictions and the real world.

2.4 Validation

The single node model, as described in Section 2.1, has four parameters which are used to characterize the load on the system: The arrival and service rates for batch and terminal jobs. The dimensions of the model as a whole are determined by M and L -- the maximum number of terminals allowed in the system and the number of server states. The vector of breakpoints $(b_0,\ldots,b_{L-1})$ determines the priority the system places on terminal response and batch throughput. Given these parameters, the analysis provides the equilibrium probability distribution of the system state, the load factor, and the mean system residence time for batch jobs. In order to compare the behavior of the model with that of MTS, we will use values for the arrival and service rates taken from the system data and check the analytic results for batch queue length against the observed batch queue distribution. The size of the model will be kept

as small as possible without sacrificing the behavior of the model, since computational costs of solution increase rapidly with increasing block size.

Determining the appropriate arrival rates from the data is relatively easy. Assuming that arrivals are Poisson-distributed, the rate is simply the number of jobs that arrived divided by the length of the measurement period. Computation of the service rates is more complex. Since CPU time is being used to approximate service, the CPU times billed to batch and terminal jobs are related to the service rates for the two job classes. However, because time spent by the paging tasks and various HASP tasks is not billed to users, the service rates indicated by billed CPU time alone will be too high. The amount of time spent in performing overhead functions can be determined by subtracting the CPU time billed to all jobs from the total non-idle CPU time during the period.

The question then arises as to how this overhead time should be distributed between batch and terminal services. A number of rough measures were made of the composition of this overhead time. Permanent statistics are not kept of the ratios of paging overhead to HASP overhead, but the samples indicated that typically 60% of the overhead time is consumed by the paging drum processor, while HASP components make up the remaining 40%. Since paging is done for all

tasks, that fraction of the overhead is distributed equally among batch and interactive jobs. HASP overhead is due primarily to batch processing functions and is, therefore, charged to the batch service rate.

The actual construction of the service rate then proceeds as follows:

Define $\overline{CPU}_B$ = mean CPU time for batch jobs

$\overline{CPU}_T$ = mean CPU time for terminal jobs

BCT = CPU seconds charged to all batch jobs

TCT = total CPU time charged to all batch and terminal jobs

TT = total non-idle CPU time

OHT = overhead time = TT - TCT

Paging overhead factor: $OHF_P = \frac{.60 \times OHT}{TCT}$

HASP overhead factor: $OHF_H = \frac{.40 \times OHT}{BCT}$

then

Terminal service rate = $\mu_T = 2 \times \frac{1}{\overline{CPU}_T + OHF_P \cdot \overline{CPU}_T}$

Batch service rate = $\mu_B = 2 \times \frac{1}{\overline{CPU}_B + OHF_P \cdot \overline{CPU}_B + OHF_H \cdot \overline{CPU}_B}$

The factor of two enters each service rate because there are two CPU's in the system, so there are two CPU seconds available per real second.

Several initial studies were performed to determine reasonable values for M,L, and the breakpoint vectors. In order to restrict computational costs, ML+1 (the block size) was restricted to be at most 41. The most appropriate balance of M and L for heavily loaded conditions appeared to be M=10 and L=4. When dealing with lightly loaded cases, costs were cut considerably with little, if any, change in results by cutting M to five. A breakpoint vector of $(1,2,3,\infty)$ was used in all of the final studies. The system did respond to different breakpoint values as anticipated, but the magnitude of the change in, for example, the mean batch queue length when the breakpoints were varied was relatively small -- in the range of fine tuning rather than gross adjustment.

Analytic studies were performed for all 15 observation periods, using parameters constructed as described above. Histograms of the marginal distributions for the number of batch and terminal jobs in the system appear in Figures 2.19, 2.20, and 2.21. The oscillations observed in the MTS data for heavily loaded periods do not show up in the analytic results, but otherwise the curves do have similar

```
  BATCH JOBS
N    PROB(N)   EACH X = 0.0274000

 0 0.824054 |XXXXXXXXXXXXXXXXXXXXXXXXXXXXX
 1 0.135870 |XXXX
 2 0.028675 |X
 3 0.007672 |
 4 0.002414 |
 5 0.000833 |
 6 0.000302 |
 7 0.000113 |
 8 0.000044 |
 9 0.000017 |
10 0.000007 |
11 0.000003 |
12 0.000001 |
13 0.000000 |

  TERMINAL JOBS
N     PROB(N)   EACH X = 0.0270000

 0 0.812312 |XXXXXXXXXXXXXXXXXXXXXXXXXXXXX
 1 0.153589 |XXXXX
 2 0.028025 |X
 3 0.005025 |
 4 0.000896 |
 5 0.000159 |
```

	NUMBER IN SYSTEM		
	TERMINALS	BATCH	TOTAL
MEAN #	0.22834	0.23239	0.46074
VARIANCE	0.27331	0.34275	2.71840
STD DEVIATION	0.522794	0.585452	1.648759
MEDIAN #	0	0	0
MODE	0	0	0
SUM	1.0000019	1.0000010	1.0000019

Period 2

Fig. 2.19 Analytic Results -- Period 2

```
  BATCH JOBS
N    PROB(N)   EACH X = 0.0139000

 0 0.419771 |XXXXXXXXXXXXXXXXXXXXXXXXXXXXX
 1 0.220837 |XXXXXXXXXXXXXX
 2 0.126186 |XXXXXXXXX
 3 0.077903 |XXXXX
 4 0.050515 |XXX
 5 0.033628 |XX
 6 0.022682 |X
 7 0.015401 |X
 8 0.010491 |
 9 0.007159 |
10 0.004889 |
11 0.003340 |
12 0.002283 |
13 0.001560 |
14 0.001066 |
15 0.000729 |
16 0.000498 |
17 0.000341 |
18 0.000233 |
19 0.000159 |
20 0.000109 |
21 0.000074 |
22 0.000051 |
23 0.000035 |
24 0.000024 |
25 0.000016 |
26 0.000011 |
27 0.000008 |
28 0.000005 |
29 0.000004 |
30 0.000002 |
31 0.000002 |
32 0.000001 |
33 0.000001 |
34 0.000001 |
35 0.000000 |
```

	TERMINALS	BATCH	TOTAL
MEAN #	1.48196	1.65694	3.13912
VARIANCE	2.10226	5.54102	17.43073
STD DEVIATION	1.449918	2.353936	4.175012
MEDIAN #	1	1	2
MODE	0	0	0
SUM	1.0000038	1.0000076	1.0000086

```
  TERMINAL JOBS
N    PROB(N)   EACH X = 0.0109000

 0 0.328497 |XXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
 1 0.255235 |XXXXXXXXXXXXXXXXXXXXXX
 2 0.181880 |XXXXXXXXXXXXXXX
 3 0.117878 |XXXXXXXXX
 4 0.073118 |XXXXXX
 5 0.043397 |XXX
```

Fig. 2.20 Analytic Results -- Period 8

BATCH JOBS

N	PROB(N)	EACH X = 0.0034700
0	0.104268	XXXXXXXXXXXXXXXXXXXXXXXXXXXXX
1	0.079671	XXXXXXXXXXXXXXXXXXXXX
2	0.064015	XXXXXXXXXXXXXXXXX
3	0.054331	XXXXXXXXXXXXXX
4	0.047901	XXXXXXXXXXXX
5	0.043229	XXXXXXXXXXX
6	0.039545	XXXXXXXXXX
7	0.036482	XXXXXXXXX
8	0.033819	XXXXXXXX
9	0.031450	XXXXXXXX
10	0.029311	XXXXXXX
11	0.027360	XXXXXX
12	0.025570	XXXXXX
13	0.023921	XXXXX
14	0.022393	XXXXX
15	0.020974	XXXXX
16	0.019651	XXXX
17	0.018417	XXXX
18	0.017264	XXX
19	0.016185	XXX
20	0.015176	XXX
21	0.014230	XXX
22	0.013345	XX
23	0.012515	XX
24	0.011737	XX
25	0.011007	XX
26	0.010323	XX
27	0.009682	XX
28	0.009081	XX
29	0.008517	XX
30	0.007988	XX
31	0.007492	XX
32	0.007027	XX
33	0.006591	X
34	0.006181	X
35	0.005798	X
36	0.005438	X
37	0.005100	X
38	0.004784	X
39	0.004487	X
40	0.004208	X
41	0.003947	X
42	0.003702	X
43	0.003472	X
44	0.003256	
45	0.003054	
46	0.002865	
47	0.002687	
48	0.002520	
49	0.002364	
50	0.002217	

TERMINAL JOBS

N	PROB(N)	EACH X = 0.0076900
0	0.209855	XXXXXXXXXXXXXXXXXXXXXXXXXXX
1	0.230853	XXXXXXXXXXXXXXXXXXXXXXXXXXXXX
2	0.190672	XXXXXXXXXXXXXXXXXXXXXX
3	0.132349	XXXXXXXXXXXXXXX
4	0.087301	XXXXXXXXXX
5	0.056743	XXXXXX
6	0.036723	XXXX
7	0.023912	XXX
8	0.015482	XX
9	0.009797	X
10	0.005990	

NUMBER IN SYSTEM

	TERMINALS	BATCH	TOTAL
MEAN #	2.30082	13.43949	15.74192
VARIANCE	4.58348	228.96072	277.07617
STD DEVIATION	2.140905	15.131448	16.645599
MEDIAN #	2	8	11
MODE	1	0	0
SUM	0.9996773	0.9997051	0.9996848

Period 13

Fig. 2.21 Analytic Results -- Period 13

profiles. The means of the distributions thus form a reasonable basis for comparison between system data and analytic results.

Tables 2.2 and 2.3 present a summary of the analytic studies and a comparison with the observed mean batch queue lengths and CPU utilizations. The agreement between the model and the data for CPU utilization is quite close; this primarily indicates that the restriction that the maximum length of the terminal queue in the model be finite has not caused any serious problems. The mean batch queue length predicted by the analysis shows the same trend as the observed mean batch queue length, but tends to underestimate the observed values. The graphs of Figures 2.22, 2.23, and 2.24 make the correspondence between predictions and observations clear. In Figure 2.22, arrows point from predicted values to observed values. So long as the CPU utilization is less than 90%, the absolute error is not large, although there is a consistent underestimate in the predictions. In the lightly loaded periods, this underestimation seems to be due, in part, to the service rate derived from the single server assumption -- if the system is processing a single job, it has only half the service rate of the model.

Table 2.2 Summary of Parameter STudies

Data From Period	Max #Term (M)	λ_T	λ_B	μ_T	μ_B	P	$E(W_B)$	% non-idle	$E(L_B)$	Std. Dev.
1	5	.01063	.02465	.07524	.20454	.21	8.4	26.2	.206	.53
2	5	.01014	.02063	.05739	.1741	.23	11.3	29.6	.233	.59
3	5	.01509	.03028	.06717	.13116	.41	18.2	45.5	.552	1.02
4	5	.02319	.03542	.05890	.17083	.46	30.8	59.8	1.09	1.95
5	5	.02633	.04994	.11210	.14831	.56	18.8	57.1	.938	1.47
6	5	.03269	.06769	.1085	.18359	.63	23.9	66.9	1.62	2.39
7	5	.0370	.03522	.08522	.10421	.76	55.4	76.1	1.95	2.72
	10						64.5	77.4	2.27	3.26
8	5	.04926	.03111	.09279	.11338	.80	53.4	78.2	1.66	2.35
9	5	.04324	.04056	.07724	.11227	.89	145.2	89.1	5.89	7.09
	10						251.5	91.8	10.2	12.3
10	5	.04833	.03826	.11354	.07448	.90	184.8	91.8	7.07	7.94
	10						290.1	93.9	11.1	12.2
11	10	.04565	.03426	.08602	.09284	.90	180.7	89.7	6.19	7.73
12	10	.04906	.04778	.08846	.11962	.93	368.4	95.1	17.6	18.8
13	10	.04517	.03772	.08848	.08650	.95	280.5	94.4	13.4	15.1
14	10	.04552	.03622	.08509	.08081	.98	911.1	97.8	33.0	32.0
15	10	.04544	.03528	.07779	.08594	.99	1173.5	98.6	41.4	38.9

For all cases
L = 4
Breakpoints = 1,2,3,∞

Parameter Studies

Single Node Model

Table 2.3 Analytic Model-System Data Comparison

Collection Period	P (model)	% CPU Non-idle		Mean Batch Q		Std. Dev. Batch Q	
		Data	Model	Data	Model	Data	Model
1	.21	26.31	26.2	.374	.206	.65	.53
2	.23	29.52	29.6	.405	.233	.63	.59
3	.41	48.90	45.5	.823	.552	1.23	1.02
4	.46	60.33	59.8	2.08	1.09	3.68	1.95
5	.56	57.36	57.1	1.65	.938	1.88	1.47
6	.63	66.91	66.9	2.27	1.62	3.51	2.39
7	.76	79.1	77.4	3.39	2.27	5.64	3.26
8	.80	80.51	78.2	1.80	1.66	1.76	2.35
9	.89	91.95	91.8	12.2	10.2	9.46	12.3
10	.90	93.95	93.9	10.34	11.1	8.84	12.2
11	.90	90.02	89.7	19.4	6.19	16.1	7.73
12	.93	95.32	95.1	25.8	17.6	14.4	18.8
13	.95	94.67	94.4	27.3	13.4	26.0	15.1
14	.98	98.30	97.8	50.1	33.0	37.8	32.0
15	.99	99.43	98.6	13.8	41.4	9.6	38.9

Analytic Model - System Data Comparison

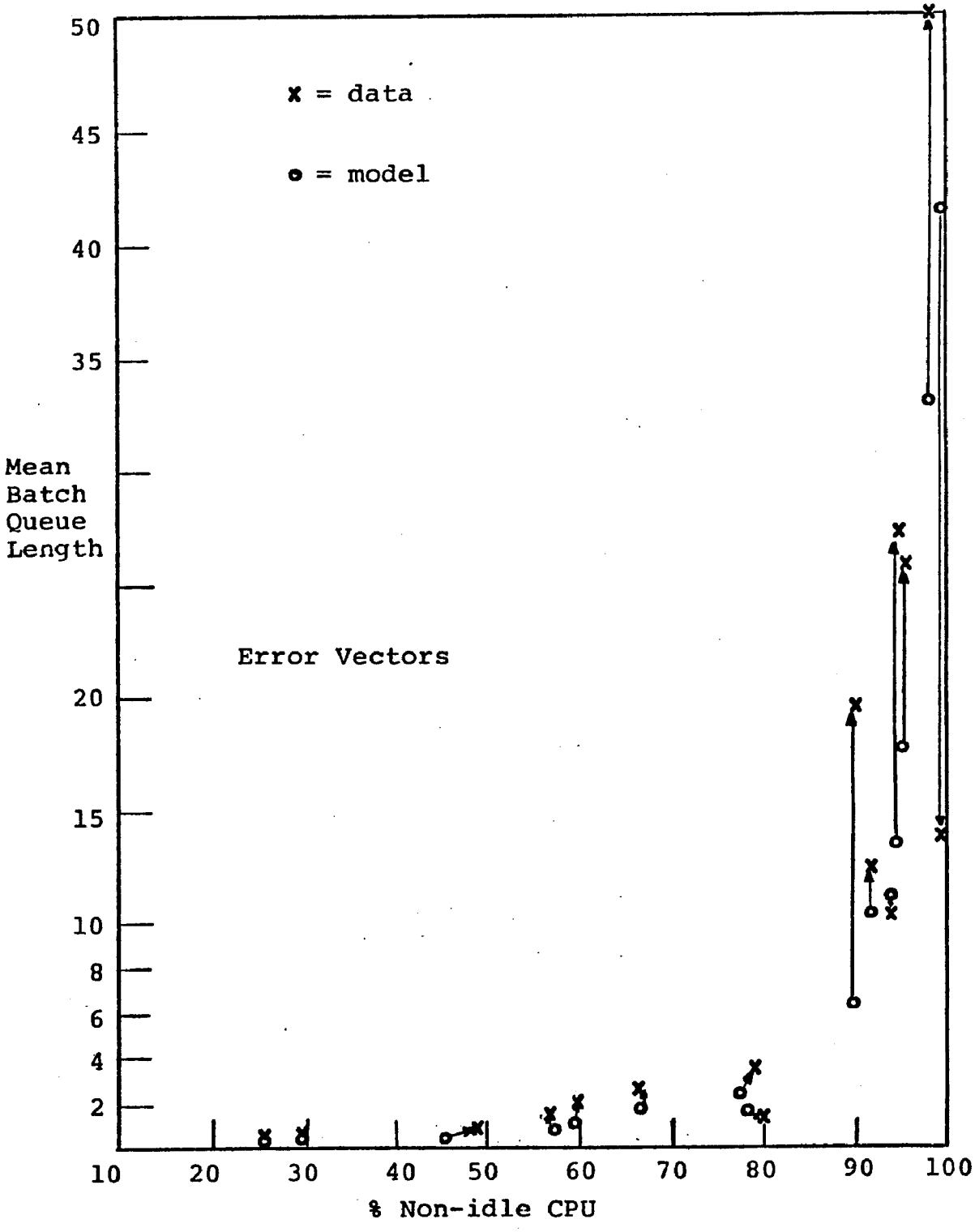


Fig. 2.22 Model vs. Data - Error Vectors

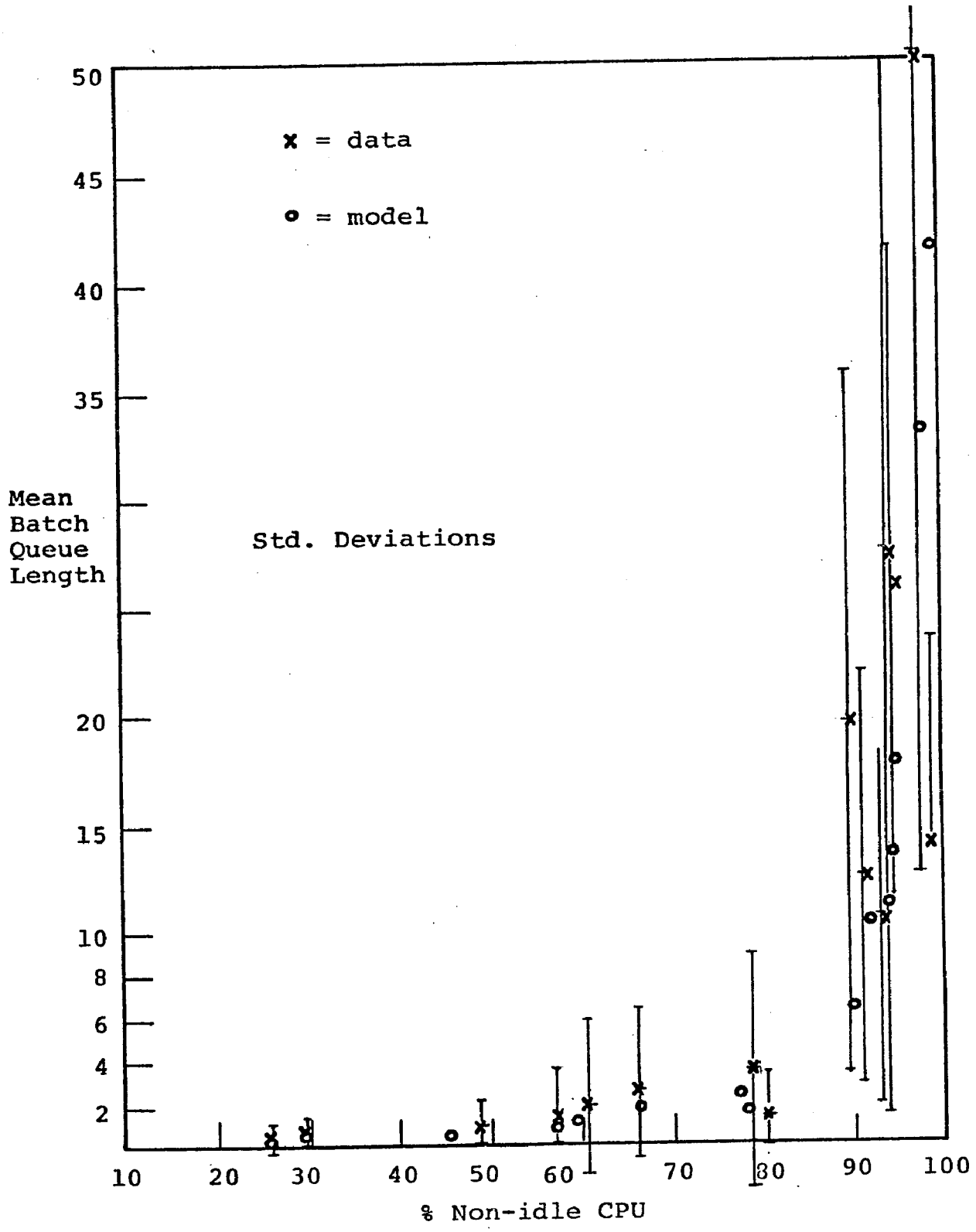


Fig. 2.23 Model vs. Data - Std. Deviations

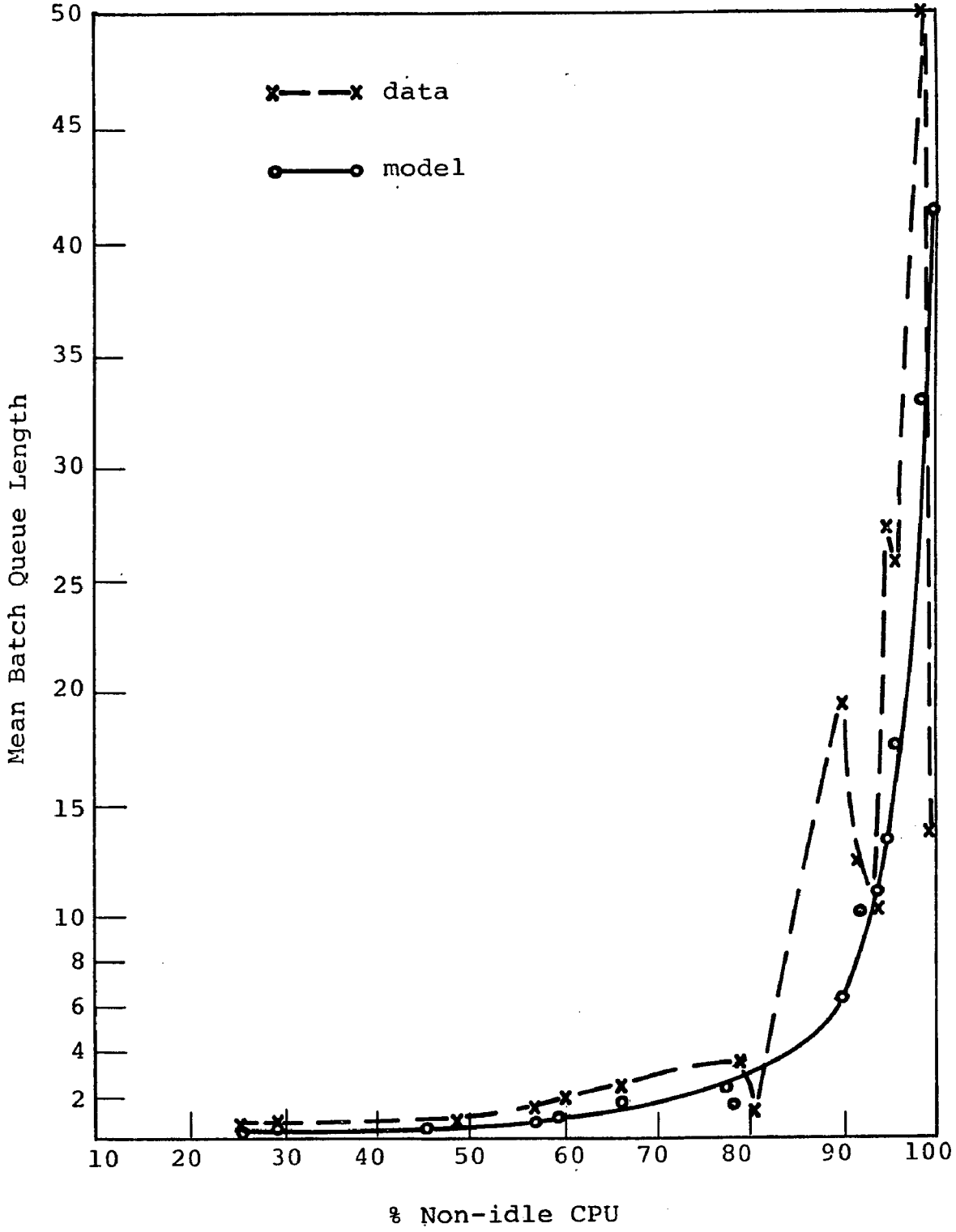


Fig. 2.24 Model vs Data

When the CPU utilization is above 90%, the observed data become much harder to predict, and the model's predictions vary in both directions from the observed statistics. Figures 2.23 and 2.24 are helpful in assessing the difficulties in the region. Figure 2.23 shows the data points with error bars indicating distances of one standard deviation in each direction from the mean of the observed distributions. From this plot it is clear that all of the predicted means fall within a standard deviation of the observed means. It is also apparent that the standard deviations in the measured data are quite large, especially in the heavily loaded region. Two explanations are possible for these observations. First, the mean batch queue length may not be describable as a simple function of CPU utilization. Particularly when the system load is heavy, other bottlenecks may appear in the system. In this case, the mean batch queue length may increase while the CPU utilization stays fixed. Additionally, when the system is heavily loaded, the basic assumption of the existence of a steady state distribution is called into question. A look at the actual structure of MTS heavy periods indicates that they are often characterized by rising terminal demand for several hours, during which the batch queue tends to grow in length, followed by a decrease in terminal use (around 6 p.m., for example). As terminal arrivals decline, the batch queue is processed more quickly by the system, so the CPU utilization remains high until the batch queue is depleted.

Thus, when the load is heavy a number of transient effects may dominate the process.

Finally, Figure 2.24 contrasts the smooth curve that fits the predicted values with the jagged line produced by connecting the data points. Since even the model predictions are a function of four parameters (the arrival and service rates, since the other parameters are held constant), and not simply the percent CPU utilization, the primary purpose of this graph is to contrast the smoothness and predictability of the model with the relatively unpredictable real system. Despite the differences, the model does reflect the dynamics of the system in an appropriate way.

2.5 Additional Parameter Studies

To apply the single host model to a network of systems, some further information about the model is required. In a network of systems which can transship batch jobs but not interactive jobs, the only model parameter which varies as a function of transshipment is the batch arrival rate. Consequently, it is of interest to see how the model behaves as the batch arrival rate is changed. The statistic of interest will be the mean waiting time for batch jobs, since the primary motivation for transshipping jobs is to decrease

user waiting time.

Data were chosen from seven of the observation periods, including two light, three moderate, and two heavy periods. The batch arrival rates were varied by small amounts, and the analytical results for the mean batch waiting times were then plotted as a function of the batch arrival rates, and curves were fitted to the data points (Figure 2.25). The circled points indicate the results when the original system data was used to determine the model parameters.

The most striking characteristic of this graph is the sharp division of the system into two apparent regions of operation. The first region, illustrated by the three steeply rising curves, corresponds to periods in which $E(W_B)$ is very sensitive to changes in the batch arrival rate. This sensitivity is due to the relatively heavy interactive load on the system in these cases; if the batch arrival rate is increased, the queue length (and hence waiting time) grow rapidly. Conversely, in the four cases in which $E(W_B)$ is relatively insensitive to the batch arrival rate, the interactive load is light and the system as a whole is underloaded. Additional batch arrivals can thus be handled with only a small increase in $E(W_B)$. It is also noteworthy that the moderate periods fall in both the heavily and

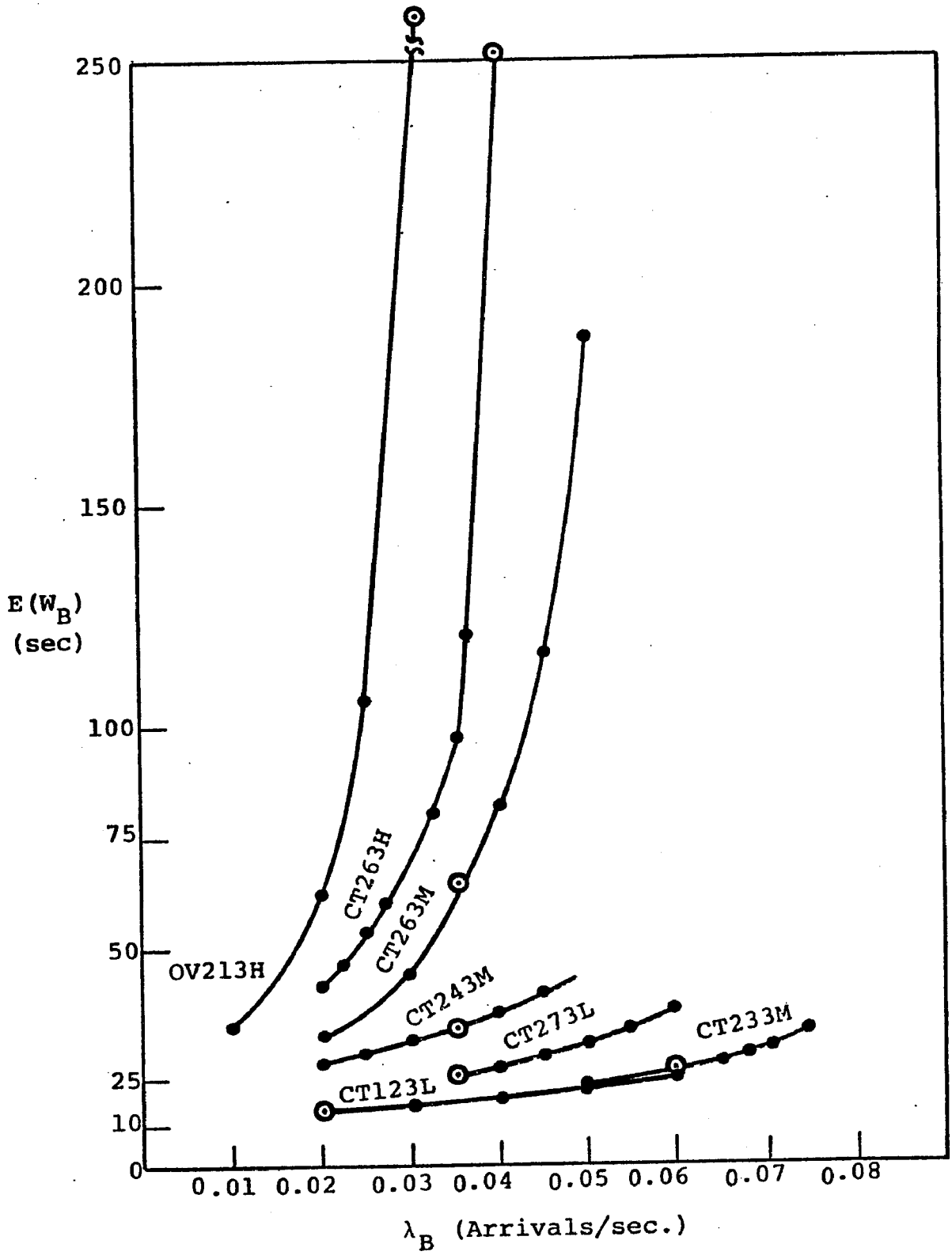


Fig. 2.25 Parameter Studies-Mean Batch Wait vs. Batch Arrival Rate

lightly loaded regions, indicating that the initial division of system states into three categories is finer than required. This division of the operating region of the system into "good turnaround" and "bad turnaround" regions has considerable intuitive appeal, since MTS batch users often observe that turnaround is either very fast (a few minutes) or else relatively long (an hour or more).

The bifurcation of the operating region suggested that if similar studies were run, varying the terminal arrival rate and observing $E(W_B)$, a knee should appear in the curve. The knee would correspond to the point at which terminal services begin to saturate the system; because of the priority placed on terminal service, batch waiting times rise rapidly beyond that point. Two such studies were made; one corresponding to a lightly loaded period in which the terminal arrival rate was gradually increased, and the other using data from a heavily loaded period, successively decreasing the terminal arrival rate. The results of these parameter studies are shown in Figure 2.26. In both cases, the curve for $E(W_B)$ begins increasing sharply as the arrival rate passes .030 jobs per second. This behavior seems realistic, as mentioned above, and was not designed into the model in any way beyond the initial specification that a priority be placed on interactive service.

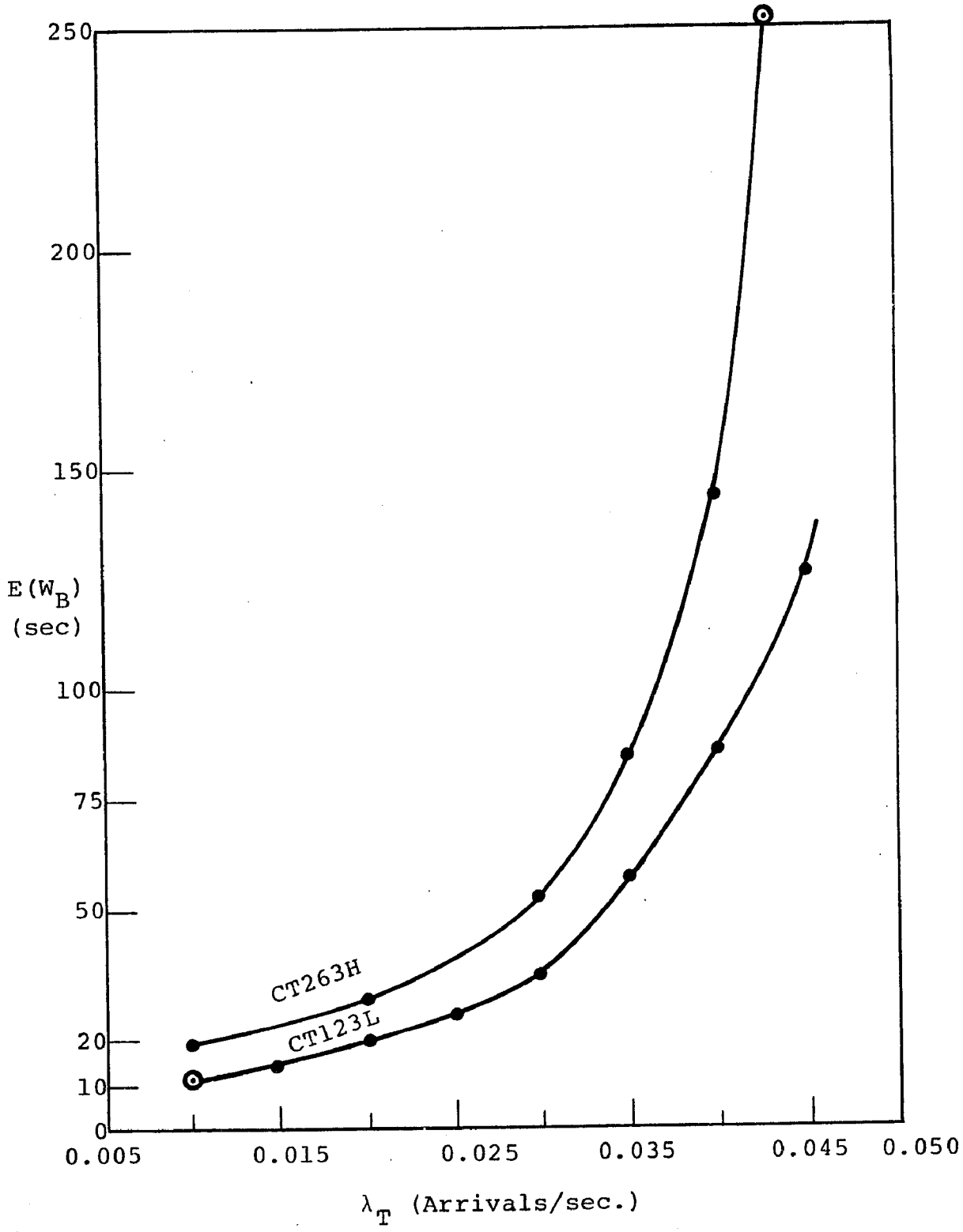


Fig. 2.26 Parameter Studies - Mean Batch Wait vs Terminal Arrival Rate

Finally, the absolute magnitudes of the mean waiting times in these studies should be examined. The longest mean wait observed in any of the studies was approximately 20 minutes. This reflects the fact that, most of the time, batch turnaround is fairly rapid in the system being studied. However, the high standard deviations of these distributions should be kept in mind. Although the mean may be low, there will still be jobs with relatively long waits. Thus, there may be some benefits to transshipping jobs even when the system is performing well on the average.

Chapter 3 - Network Considerations

After investigating in detail the model for a single host in Chapter 2, it is appropriate to consider the requirements for a model of a network of two or more hosts. A copy of the single node model may be used to represent each host, but the question of how to interconnect these models to represent the network structure remains to be answered. The physical and logical structure of the interconnections among hosts may take many forms. These forms are important to this research, but a vantage point must be established from which this complexity may be viewed. In this chapter, we will take the position of the decision makers -- those components in the network which make the transshipping decisions. From this prospect we shall examine the physical and logical structure of the communications subnet, the composition of the jobs to be transshipped, the characteristics of the users, and the structure of the decision process in the model. The models constructed in this study will then be used in the construction of the decision makers in the two-node and multi-node models (Chapters 4 and 5).

3.1 Network Structure

To enable two computers which do not share storage or

peripherals to communicate with each other, a transmission medium of some sort is required. Telephone lines, radio transmitters, microwave and satellite links have been used in different networks (A1, K3). To the decision maker, the method of transmission is of only secondary importance. The primary factors are to whom signals may be sent and how long the transmission will take. The investigation of these questions leads to the issues of channel speed and capacity and connectivity of the network.

The collection of hardware used to interconnect the various hosts is referred to by ARPA (H3) as the communications subnet, and we will adopt their terminology. Typically, the communications subnet consists of telephone lines or other electronic transmission media; interfaces for accessing the medium, such as data sets and line adapters; and switching and processing components such as ARPA's Interface Message Processors (IMPs) and Terminal Interface Processors (TIPs), and MERIT's Communications Computers. The speed of data transmission is ultimately limited by the physical bandwidth of the particular medium chosen. Delays may also occur in the switching processors due to message queueing and routing decisions. Published ARPANET data (K3) indicates that the network loading has not reached levels for which message congestion would significantly increase delays. The network delay model presented in (K3) includes terms for the node and line delays, but the values used for

these terms are, unfortunately, omitted.

The logical structure of the communications subnet is important to the extent that it affects the connectivity of the host systems and the delays for host to host messages. ARPANET accepts messages from hosts of up to 8063 bits. These messages are then broken into packets of up to 1008 bit each which are independently routed to the destination host. The IMP at the destination host reassembles the packets in the proper order and transmits the message to the host. The packet switching and dynamic routing schemes used in ARPANET are among the most complex structures currently in use; however, the data presented in (K3) indicate that over 95 percent of all messages transmitted are short enough to fit into a single packet. MERIT switches messages instead of packets, but the message size is similar to the size of an ARPANET packet. Both ARPANET and MERIT employ both telephonic and host-host protocols to ensure reliable transmission and process synchronization. The MERIT host-host batch transmission protocol (M3) requires considerably less overhead than the interactive protocol (M2), however, since the snychronization problem is greatly simplified.

Again, a wealth of detail confronts the modeller. The network under consideration here differs from ARPANET and MERIT in that its central purpose is the transshipment of batch jobs. Since all systems on the network must understand

a common job control language, it is possible that the systems would be quite homogeneous. In this case, minicomputers for message switching might not be required; all message switching functions could be included in a host system program which would be duplicated at all locations (an approach similar to that of the TSS network, which employs a Computer Access Method). Thus it seems preferable that the model not explicitly include such elements, but rather attempt to characterize the message delays the decision maker sees in a simpler way. Since the host model is a queueing model, each physical path open to a decision maker might be characterized by a queue connecting the two decision makers. (Note that if each logical path were modelled instead of each physical path, then every net that is simply connected would be modelled by a fully connected network of queues). The rate of service to this queue would then depend on the hardware and the protocols to be employed over this path. Such an approach is appealing, but adding even one queue per physical path would complicate the system greatly. Consequently, the topology of the network will be indicated by connections without queues between decision makers. The delays involved in actually transmitting jobs will then be modelled indirectly through the cost function (see Section 3.4).

3.2 Job Model

Ideally, a decision maker could know the demand which a job will place on a system and send the job to that host in the network which could provide the job with the fastest service. In addition, it could attempt to balance the loads on different systems, sending I/O bound jobs to systems with busy CPUs and CPU bound jobs to systems with excess processor time available. A recent paper (S7) indicates it may be possible for users to estimate the types of demand jobs impose on a system, but, in practice, such a scheduling policy would be unrealizeable, since system loads can fluctuate widely in short intervals, and users will make mistakes in their estimates.

In terms of the model, if a decision maker can discriminate between jobs (i.e., it is aware of different types of job demands) then the system state transitions become a function of the type of arrival that occurs. An arrival rate must then be defined for each arrival type at each host, and service times could depend on the type of job being processed. Even if there were only a few types of arrivals and a few hosts, this classification of jobs into types would greatly increase the complexity of the model to be analyzed. Consequently, we shall make the simplifying assumption that all jobs are drawn from a single demand distribution. Thus, the decision makers can only use

information about the current state of the system in determining where to ship a given job. The demand of a job may be considered as a host-independent measure of its resource requirements; different hardware at different host sites may alter the actual service time required to complete a given job.

In addition to the demand a job places on a host, it will require some network resources if it is to be transshipped. This demand can be characterized by the amount of data to be sent (i.e., the number and length of the cards in the job deck) and could be available to a decision maker, but the size of the output returned could, at best, be supplied by a user estimate. In a model concerned with choosing routing strategies which best utilize channel capacity, this information would be important. This research, however, is intended to assess the overall feasibility of load sharing, particularly from the user's viewpoint. A job's demand on the network is then of interest primarily in that the transmission of the job delays its processing. Again, classes of jobs with different quantities of data for transshipping could be introduced, but, for the reasons cited above, this would complicate the model considerably and increase its realism only slightly. So again, we will make the assumption that all jobs are drawn from the same distribution of network data shipping demand, and a decision maker will not be able to discriminate among

jobs on the basis of their requirements.

3.3 User Model

Since this study proposes to take the user's viewpoint on the question of what conditions make load sharing advantageous, it is necessary that a decision maker have some idea of what a user's interest is. Users of computing services are typically concerned about the quality of the services rendered, the speed with which services are delivered and the cost of the service. The "quality of service" is a term encompassing a number of difficult-to-define system parameters such as the availability of service, reliability of the system, quantity and variety of software available, and so on. These considerations cannot be used by our decision maker in deciding where a job is to be executed, since the alternative hosts can all provide the services required at the time of the decision.

The costs of executing the same job in different systems, on the other hand, may differ, and the decision maker can make use of this information. Cost of execution in this case refers to the usual charges for resources (CPU time, storage, I/O activity, etc.) assessed by a computing center on users so that the overhead costs of the center (staff salaries, equipment rentals, and supplies) may be

met. A decision maker considers users to be people who wish to minimize such costs. Since the job model developed above restricts all jobs to belong to one class in terms of the demand they exert on a system, the question "Which system will execute the job for the least resource cost?" reduces to "Which system has the lowest resource charge for an average job?"

If no further consideration were given to the user model, a decision maker would send all jobs to the host with the lowest charging rate. Because users are also interested in the speed with which the service is given, however, a balance must be struck between minimizing cost and simultaneously minimizing turnaround time, since these are conflicting goals. The precise value a user puts on his waiting time is difficult to assess. A computing center manager does, nevertheless, make decisions on the basis of his perception of what a tolerable turnaround or response time is whenever he orders new equipment to better handle the computing load. Bowden and Barr (B3) suggest a relatively complex function for the value of a job as a function of its completion time, including primary and secondary deadlines, and Streeter (S5) determines several job value curves by sampling users' opinions. The shape of such a curve need not be defined at this time; it is sufficient that the user model specifies that such a curve exists. For any particular situation, an appropriate curve

can be formulated, and a decision maker can use it to define the trade-off between turnaround time and resource cost. Note again that, as only one class of jobs is permitted, only one class of users is permitted; thus, there is a single waiting time cost curve used to represent the average of all of the individual user curves.

3.4 Decision Maker Model

The decision makers require information about the structure of the network, the jobs, and the users; consequently, the models for these elements have been presented before the decision maker model. Since the network to be studied may be distributed rather than centralized, it is appropriate to decentralize the decision making process as well. Thus, each host will contain a component that examines incoming batch jobs, considers a number of alternative hosts where the job might be sent for execution, and decides either to keep the job or to transship it to a different system, where the process may be repeated. For purposes of illustration, the process of making a decision will be specified mathematically as a function which has as its arguments the states at the alternative hosts. The value of the function will indicate the host to which the job would be transshipped. For example: Given three host systems L, M, and N,

$$D_L (S(H_L), S(H_M), S(H_N)) = M$$

would indicate that a job arriving at host L would be sent to host M for execution, given that the states of the three hosts are $S(H_L)$, $S(H_M)$, and $S(H_N)$ at the time of the job's arrival. The benefit of this formalism is that it points out the crucial elements of the decision: The scope of the function (the alternatives under consideration) and the action of the function (the indication of the host to which the job should be sent). The number of alternative hosts considered could include all the hosts in the network, in which case the decision maker presumably chooses the global optimum, or it could be some subset of the hosts. If the network is very large, the amount of information the decision maker would require to choose among all hosts would be considerable, and problems could arise in keeping this information up to date. Consequently, we shall assume that the decision maker only considers alternative hosts to which it is directly connected. Thus, the "neighborhood of a host" is defined as the set of systems that have direct connections to the host, and a decision maker only considers systems in the neighborhood of its host. This procedure does not restrict a job from passing through several hosts, since each host it enters may apply its decision maker to the job.

Several parameters have been subsumed under the notation $S(H_i)$. In addition to the current queue lengths and

server state in a host, the average transshipping delay between the two hosts, the charging rates, and the service rates of the hosts may be included in the decision process. The consideration of transshipment delay is the indirect modelling of shipping delays mentioned in Section 3.1. This delay also includes elements from the job model, since the transmission time is a function of the transmission rate and the amount of data to be sent.

In addition to the scope of the decision function, the actual decision algorithm must be defined. If a cost function is defined, and the cost for each alternative can be computed, the decision function can simply choose the least cost alternative. There is no loss of generality in assuming this structure for the decision maker, since arbitrary cost functions can be specified. The remainder of this section is devoted to developing the particular cost functions used in the parameter studies in Chapters 4 and 5.

Costs, as suggested in the user model, can be partitioned into the explicit cost of resources and the implicit cost of waiting. The cost function proposed here attempts to resolve these costs in a simple way. Resources consumed by jobs in the network may include host resources (CPU, storage, etc.) and network resources (channel capacity). Typically, rates for resource use at a center are designed so that the total operating costs may be recovered

from users, and also so that users are charged for particular resources in proportion to their use of those resources. Since the construction of a network involves costs for programming and channel capacity, it might be suggested that these costs should be parcelled among users of the network. A moment's reflection, however, indicates that in a load sharing network, the users whose jobs are not actually transshipped may still benefit from the network, since their jobs will not have to wait for the transshipped jobs to be executed. Consequently, a more just scheme would be to include the cost of lines and programming support for the network in the operations overhead for the center. Costs would then be recovered through charges for local resources.

For the decision maker, absolute resource costs are not as important as the relative costs, since the problem is always to choose the least expensive alternative. As explained in the job model, the network is not aware of the precise demand of a job in advance, so the prediction of resource cost must be based on the resource cost for an average job. One host in the network may be chosen as the standard, for which the resource cost per job will be normalized to 1.0. By comparing the rates at other hosts, the relative resource costs for processing the average job can be defined. Additional overhead for network lines must then be taken into account. This overhead could be distributed among centers in proportion to their total

computing budgets. Rates at each center would then rise by the percentage increase in the center's cost. For example, if the standard center charged a normalized rate of 1.0 with operating expenses of $.5 million/year without the network, and if the cost added by the network were $12,000/year, the resource cost per job would rise to 1.024 to account for a 2.4% rise in operating costs.

In order to construct parameter studies, rates at two MTS installations were obtained and are presented in Figure 3.1. Since CPU service has already been chosen to represent overall service to a job, the ratio of CPU charges has been used to approximate the overall resource costs. Approximate rates for the line rentals in three speed ranges were obtained from Michigan Bell Telephone (S1) and are shown in Figure 3.2. The distances used to determine the rates were for the approximate mileage between Ann Arbor and Detroit. This distance forms one link in the MERIT Network; costs would differ correspondingly for longer or shorter links. Finally, the impact of line costs in terms of percent increase in resource costs was computed for several different size computing center budgets (Figure 3.3).

University/Government
Charging Rates

	University of Michigan		Wayne State University
	Standard Priority	Low Priority	
CPU time (per hour)	$272	$222	$325
Virtual Storage (per page per CPU hour)	$2.72	$2.22	$3.25
Disk Storage (per page per day)	$.003	$.003	$.008

Industrial
Charging Rates

	University of Michigan		Wayne State University
	Standard Priority	Low Priority	
CPU	$316	$258	$390
Virtual Storage	$3.16	$2.58	$3.90
Disk Storage	$.004	$.004	$.0096

Fig. 3.1 Charging Rates at Two MTS Installations, March 1974

Line Speed	Charges	
2KB	Data sets (2)	$140/month
	Local phone	$5.75/month
	Line charge	$90/month
		$237/month
		approx. $2800/year
10.8KB	Data sets (2)	$400/month
	Line charge	$171/month
	C4 condition	$80/month
		$651/month
		approx. $7800/year
50KB	Total charges	$1000/month
		approx. $12,000/year

Fig. 3.2 Estimated Service Charges for Michigan Bell Telephone Service, Ann Arbor-Detroit (36 Miles) April 1974

Line costs as percent of computing center budget

Total Annual Budget (without lines)	Line Speed	% Budget Increase
$1 million	50 KB	+1.2
	10.8 KB	+ .78
	2 KB	+ .28
$0.5 million	50 KB	+2.4
	10.8 KB	+ .78
	2 KB	+ .28
$0.2 million	50 KB	+6.0
	10.8 KB	+3.4
	2 KB	+1.4

Fig. 3.3 Line Costs as Percent of Computing Center Budget

The cost of waiting has already been considered in the user model. The task of defining a particular waiting time cost function and a method of assessing the expected waiting time for a job remains. Streeter constructs several different curves for the value of a job as a function of turnaround time. Figure 3.4 reproduces two of the curves Streeter defined, one giving the value assigned by users for express jobs (such as debugging runs), and the other indicating the same measure for production runs. Since we are modelling a university computing environment in the following parameter studies, the express curve seems intuitively more reasonable than the production curve. The university computing load is typically dominated by a large number of student debugging runs with a small proportion of what could be called production. Note that even Streeter's express curve is truncated at a turnaround time of 15 minutes -- that is, all turnaround times of 15 minutes or less are equally desirable. For the parameter studies in the following chapters, this curve has been extrapolated to a truncation at five minute turnaround and then re-scaled as shown in Figure 3.5. The model will, of course, work with any arbitrary curve for this function; the curve of Figure 3.5 was simply chosen as more realistic for the case to be studied.

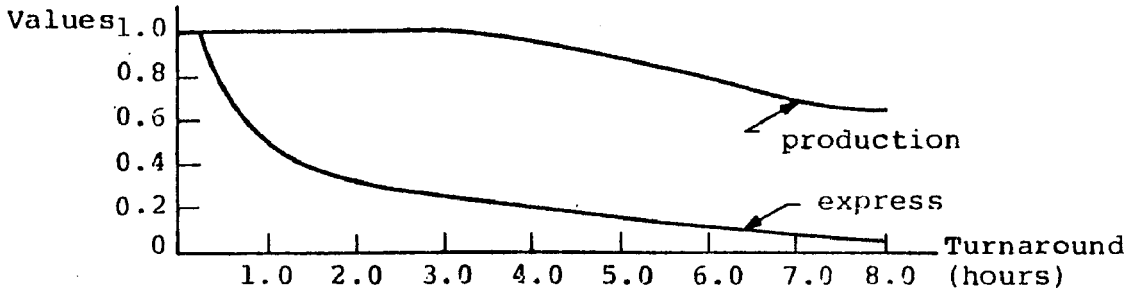


Fig. 3.4 Streeter's Curve

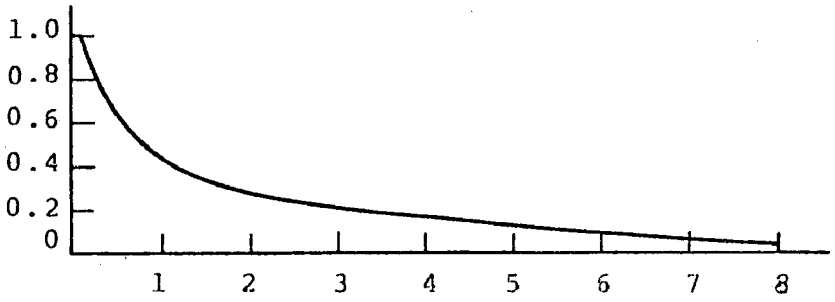


Fig. 3.5 Modified Express Curve

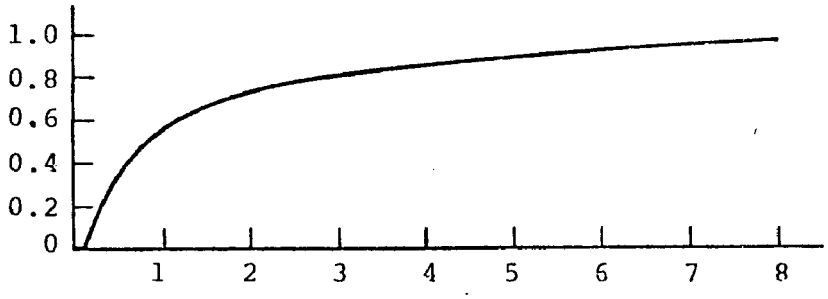


Fig. 3.6 Turnaround Factor Curve

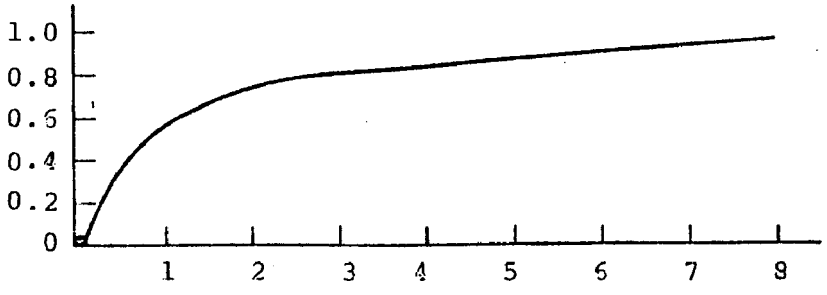


Fig. 3.7 Modified Turnaround Factor Curve

The method of assessing the projected waiting time for a job differs slightly in the two-node model (Chapter 4) and the multi-node model (Chapter 5). The total system residence time for a job consists of the time spent in the network (transshipping delay) and the time spent waiting and in execution at a particular host. In both models, the transshipping delay between two systems is estimated by a fixed value which depends on the speed of the lines and the average size of a job (number of cards) and its results (number of lines). Some rough measures of the Michigan system indicated a typical batch job consists of about 100 80-byte cards of input and approximately 500 60-byte lines of output, for a total of 38,000 bytes or 304,000 bits per job. The decision maker in the two-node model can estimate the expected waiting time in each host system on the basis of the number of jobs in the queues at the time of the job's arrival. The detailed method for making this estimate will be defined in Chapter 4. The decision makers of the multi-node model are unaware of the number of jobs in the queues of different systems at the time of a job's arrival; only the long run expected waiting time at each host is known. Estimates for the total delay are then made on the basis of the overall mean waiting times at different hosts.

Given an estimate of the delay a job may encounter, some function is required to convert this delay to a cost. Streeter's curves relate the value of a job to the

turnaround delay. As the value of the job decreases, the cost of waiting for the job must increase, and it is this cost we wish to measure. The simplest technique to convert value, normalized between zero and one, to cost is to simply plot the cost as one minus the value (Figure 3.6). The final question in the construction of the cost function then is how to combine the waiting time cost and the resource cost. The two costs could simply be added, but this technique would indicate that, for a job with high resource costs, waiting time costs would become insignificant. It seems more appropriate to use the waiting time cost as a multiplier for the resource cost. Schemes similar to this are employed in systems that offer express services at a higher unit cost or low priority service at a reduced rate. In these cases, the user's resource costs are multiplied by a factor greater or less than one if the service given is faster or slower, respectively, than the default service offered. Such a scheme does lead to one difficulty in relation to the modified cost function of Figure 3.6. If the waiting time is expected to be less than five minutes, the total cost will be zero, and so the decision maker would be insensitive to differences in resource costs. To circumvent this problem, the turnaround factor curve of Figure 3.7 will be used. This curve is truncated to a value of .005 at zero expected waiting time, and a slight slope is introduced so that the turnaround cost factor at five minutes is .006. This small slope is introduced so that there will always be a slight

preference for shorter mean waiting times. Again, other curves are possible; this curve was chosen as representative for the parameter studies which follow.

In summary, the cost to process at host j a job which arrives at host i is given by:

$$C_{ij} = TF(TD_{ij}) \cdot RC_j$$

$TF(x)$ maps a delay x to a cost multiplier which is called the turnaround factor

TD_{ij} is the total time required for a job submitted at host i to be sent to host j, to be executed, and have the results returned to host i (thus it is the total delay incurred by the job)

RC_j is the resource cost to process a job at host j

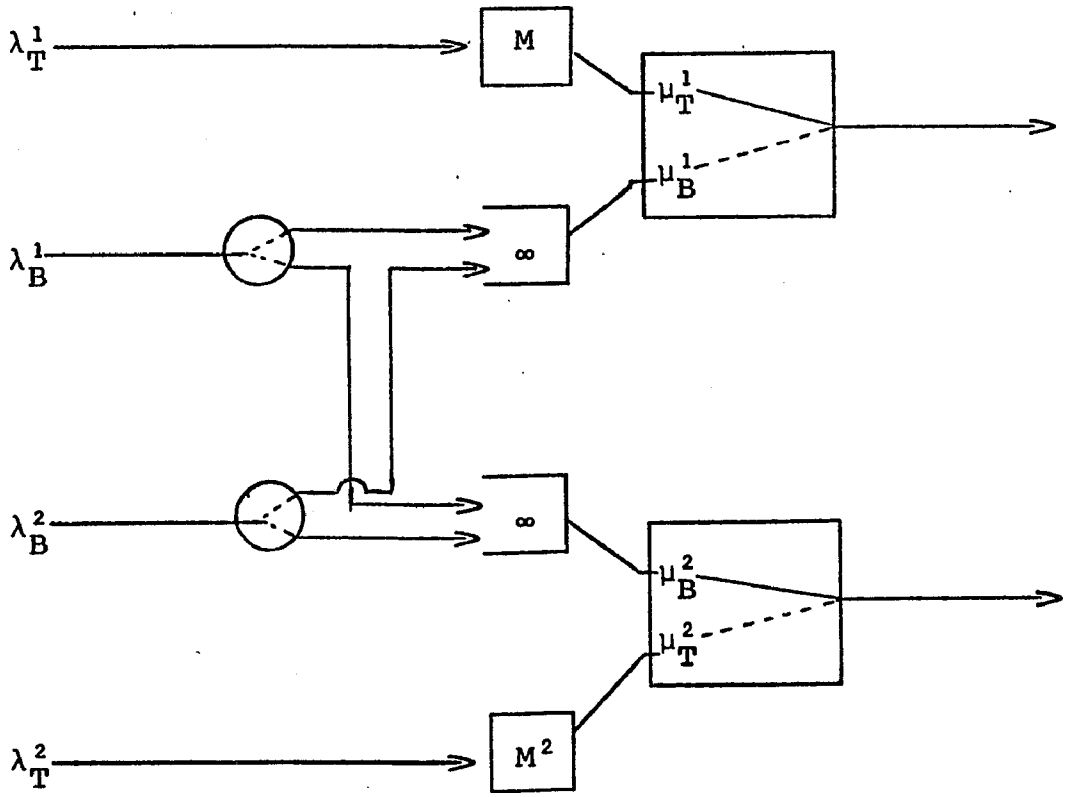
Note that if i=j, then the transshipment delay is zero. Thus, we have defined a particular cost function which a decision maker may use to assess the total cost to the user of executing a job submitted at host i on host j. The decision maker at host i can perform this computation for all hosts j in its neighborhood and choose the least predicted cost alternative.

Chapter 4 - Two-node Model

Building outward from the single node model, the next step is to model a network of two nodes -- the smallest possible network. While the immediate construction of a general network model is appealing, the fixed, small size of a two node network will allow a more detailed model to be formulated than is possible in the general case. This chapter will present a formulation of a two-node model, a modified version of the model which compensates for certain inadequacies in the previously described analytic technique of Wallace, analytic arguments concerning the model, and finally a group of parameter studies.

4.1 Description and Discussion of the Model

Two copies of the single node model can be linked relatively easily as shown in Figure 4.1. The structure of each host subsystem is identical with the single host model: there is an infinite capacity batch queue, finite capacity terminal queue, and a single server with two service rates and a breakpoint priority discipline. Superimposed on this model are two decision makers -- one at the entry to each batch queue. When a batch job arrives to either subsystem, a decision maker determines whether to accept this job for



Each node same as single node model, except batch queue has two possible input streams.

Decision Mechanism  is unspecified.

System State specified by $(T^1, B^1, S^1, T^2, B^2, S^2)$

System parameters: $\lambda_T^1, \lambda_B^1, \mu_T^1, \mu_B^1, L^1, M^1, Bk^1(s)$
$\lambda_T^2, \lambda_B^2, \mu_T^2, \mu_B^2, L^2, M^2, Bk^2(s)$
Decision algorithm for sharing

Assumptions: Hosts may have different service rates, but all jobs are drawn from same demand distribution.

Two Node Network

Fig. 4.1 Two Node Network

execution locally or to transship the job to the other subsystem's batch queue. Some constraints must be placed on the decision-making process in order to keep the model analytic.

To be analyzed by conventional means, the model must have the Markov property. If certain restrictions are placed on the arrival process, the decision makers, and the transshipping delays, the model will be Markovian, with a state space given by a six-tuple: $(T_1, B_1, S_1, T_2, B_2, S_2)$ which is simply the concatenation of the subsystems' state spaces. The following restrictions are sufficient:

1. Arrivals to the terminal queues and to the decision makers must be Poisson-distributed.

2. Service times must be exponentially distributed.

3. Transshipping delays must be zero.

4. Decisions must be made in zero time and may depend only on the system state.

Conditions three and four ensure that arrivals to the two batch queues will have the same distribution as arrivals to the decision makers (thus they will also be Poisson-distributed). As indicated in Chapter 3, the shipping delays will be modelled indirectly through a cost function. The

restriction that decisions must depend only on the system state requires that all jobs be treated uniformly; this rule is in accordance with the job model developed in Chapter 3. Given the system state, then, the arrival rate at batch queue i must be either zero, λ_{B_i}, or $\lambda_{B_1}+\lambda_{B_2}$. (λ_{B_i} denotes the local batch arrival rate to host i). Network topology for the two-node system is trivial (only one configuration is possible) and is modelled by the linkage of batch queues. Terminal jobs may not be shared, so there is no interconnection of the terminal queues.

Unfortunately, the model as constructed cannot be analyzed by Wallace's techniques, as described in Chapter 2. The intuitive argument that the technique fails is straightforward. In this model there are two potentially infinite queues. For Wallace's method to work, the transition intensity matrix must exhibit a block tri-diagonal structure, with the block reaching a limiting size and internal structure. If there is only one potentially infinite queue in the system, the length of that queue provides the infinite component in the state space, and the blocks of the transition intensity matrix have a repeating structure as the length of that queue increases. If there are two potentially infinite queues, however, the blocks never reach a limiting size but continue to grow without bound. The following section gives a proof of this fact.

Proof that Wallace's Technique is Insufficient to Analyze the Two-node Model

Wallace's method may be used to find the steady state probability distribution of an infinite state continuous parameter Markov chain. For the method to succeed, the Markov chain must satisfy certain conditions (W2, p. 11):

A. It must be possible to order the state space into rows of states, the first row containing m states and all other rows containing m_0 states, where $m_0 \geq m$.

B. Transitions of non-zero intensity are allowed only among states in the same row or in adjacent rows.

C. The states within a row are ordered, and if there is a transition of intensity u between the state in row i, column j, and row k, column ℓ, then there must be a similar transition of intensity u between the states (i+1,j) and (k+1,ℓ). ($u \geq 0$).

If we view the transition intensity matrix of the Markov chain as the adjacency matrix of a graph, we can reformulate Wallace's conditions as restrictions on the structure of the graph as follows:

A. There must exist a simple ordered partition of the states (nodes) such that the first block contains m_0

states and all other blocks contain m states.

B. States within a block may have non-zero transition intensities (arcs) to other states within the same block or to states in the preceding and succeeding blocks only.

C. (Since this condition is not required for the proof, it will not be restated.)

In the case under consideration, the state space consists of 6-tuples of integers: $(T_1, B_1, S_1, T_2, B_2, S_2)$ where

T_i = number of jobs in the terminal queue at subsystem i $(0 \leq T_i \leq M_i)$

B_i = number of jobs in the batch queue at subsystem i $(0 \leq B_i < \infty)$

S_i = server state at subsystem i $(0 \leq S_i \leq L_i - 1)$

We also define

$n_i(s)$ = total number of jobs in subsystem i for state s, and

$I(s) = n_1(s) + n_2(s)$ -- called the index of s.

There is only one state for which $I(s) = 0$ and this state is the idle state -- the state in which there are no jobs in either subsystem. Note that given any two states s_i and s_j $(i \neq j)$, if there is a non-zero intensity of transition between them, then s_i and s_j are in the same or adjacent rows, and $|I(s_i) - I(s_j)| = 1$. This relation holds since any arrival or departure to the system changes the number of jobs in the system by one, and also changes the system state.

The proof will be by contradiction: we assume that a partition satisfying conditions A and B exists and show that the number of states in some block of such a partition will exceed any finite value, thus violating condition A. The key to the proof is the demonstration that all states with index less than k must be included in the first k blocks of the partition. The number of states for which $I(s)=k$ is then shown to increase faster than linearly with k so that no finite value for the block size is sufficient.

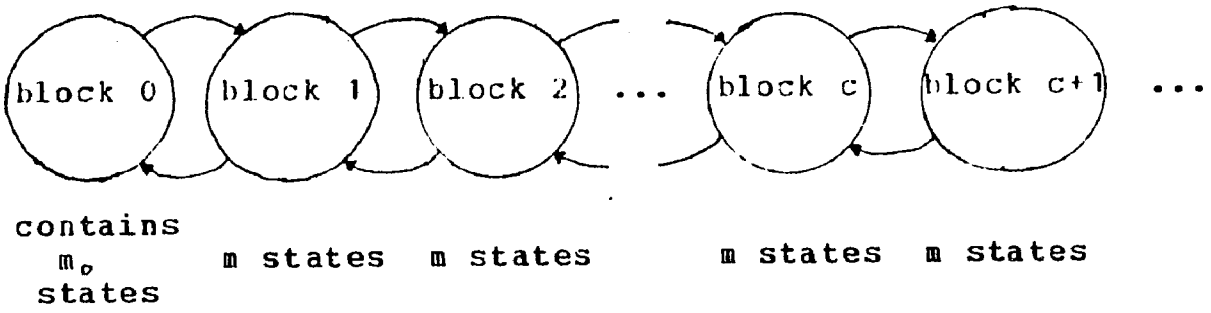
Proceeding with the proof, then, suppose there is an ordered partition which satisfies the conditons A and B. The first block must contain m_0 nodes of the graph and all other blocks must contain m nodes. Since m_0 and m are finite, there must be a state or states in each block for which I(s) is minimal. Denote this state in block j as Q_j. (If there are several minimal states, the choice is arbitrary among them.)

Examine Q_0. If $I(Q_0)\neq 0$, there must be an arc leading from Q_0 to a state s in block 1 for which $I(s)=I(Q_0)-1$. (This arc represents a job departure occurring when the system is in state Q_0. The arc representing this departure must enter block 1 since Q_0 is minimal and condition A must hold.) Follow the same procedure for block 1, block 2, and so on until the state s for which $I(s)=0$ is located. The

search must terminate since at each step the index of minimal state in block j+1 must be at least one less than the index of the minimal state in block j.

Denote the one block in which $I(Q_j)=0$ as block c (c may be zero). We now wish to show that for all blocks j such that $j \geq c$, $I(Q_{j+1})-I(Q_j) \geq 1$. That is, the blocks are ordered by increasing Q . If $j=c$, $I(Q_j)=0$ and since $I(s)=0$ for only one state and $I(s) \geq 0$ for all states, the relation holds. Now consider Q_j for $j>c$. Since $j>c, I(Q_j)>0$. Since Q_j is not the zero state, there must be an arc, representing a departure, connecting Q_j to a state s not in block j. (If s is in block j, then Q_j is not minimal.) Thus, by condition B, s must be in block j+1 or j-1. If s is in block j-1, the desired condition holds. If s is in block j+1, we can repeat the above argument: since s cannot be the zero state, it must have an arc, representing a departure transition, connected to some other state s', and $I(s')=I(s)-1$. If we continue to trace the departure arcs, we must ultimately return to the zero state. By condition B, we must then pass through all blocks between j+1 and c, and, in particular, we must pass through block j. To do this, however, there must be a state s" in block j for which $I(s")<I(s')<I(s)<I(Q)$, violating the assumption that Q is the minimal state in block j. Thus $I(Q_{j+1})-I(Q_j) \geq 1$ for all $j \geq c$.

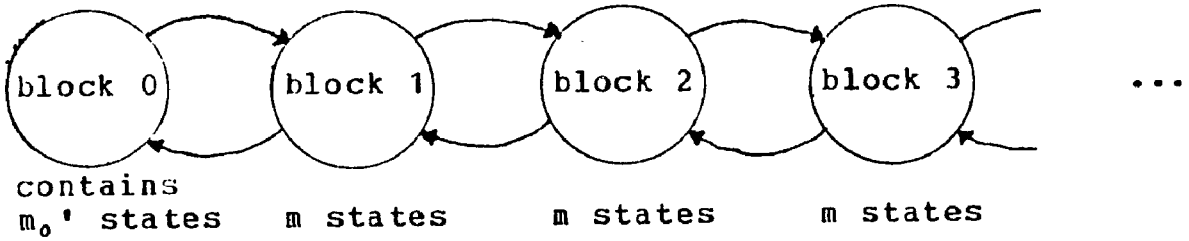
We now have the following grouping of states:



If the original partition satisfies conditons A and B, then a regrouping which puts all the blocks up to and including block c into one block also satisfies the conditions. The new value of m is then

$$m_o' = m_o + cm$$

and the new system is (renumbering the blocks from zero):



Observe that for $j \geq 0$, no state with $I(s) \leq (Q_j)$ can occur to the right of block j and $I(Q_j) \geq j$ and $I(Q_{j+1}) - I(Q_j) \geq 1$. Compute the number of states in blocks zero through j:

$$N_j = m_o' + jm$$

Now examine the number of states s for which $I(s)=k$. If there are k jobs in the system, the number in subsystem 1 may be from 0 to k. The number of jobs in subsystem 2 is simply k minus the number of jobs in subsystem 1.

For example:

	n_1	# states possible for subsystem 1	# states possible for subsystem 2	total # possible states
k=0	0	1	1	1
	0	1	L_2	
	1	L_1 1	1^2	$L_1 + L_2$
k=2	0	1	$L_2 \cdot 2$	
	1	L_1	L_2	
	2	$L_1 \cdot 2$	1	$2(L_1+L_2)+L_1L_2$
k=3	0	1	$3L_2$	
	1	L_1	$2L_2$	
	2	$2L_1$	L_2^2	$3(L_1+L_2)+4L_1L_2$
	3	$3L_1$	1	

Now look at a case for large k ($k>M_1+M_2$).

$k=K>M_1+M_2$	0	1	$L_2(M_2-1)+1$
	1	L_1	$L_2(M_2-1)+1$
	2	$2L_1$	$L_2(M_2-1)+1$
	.	.	.
	.	.	.
	.	.	.
	.	.	.
	M_1-1	$L_1(M_1-1)$	$L_2(M_2-1)+1$
	M_1	$L_1(M_1-1)+1$	$L_2(M_2-1)+1$
	.	.	.
	.	.	.
	.	.	.
	$K-(M_2-1)$	$L_1(M_1-1)+1$	$L_2(M_2-1)+1$
	$K-(M_2-2)$	$L_1(M_1-1)+1$	$L_2(M_2-2)$
	.	.	.
	.	.	.
	.	.	.
	$K-1$	$L_1(M_1-1)+1$	L_2
	K	$L_1(M_1-1)+1$	1

Instead of attempting to sum all of these expressions to get the total number of states s for which $I(s)=K$, simply extend the above table for the case $k=K+1$. It should be clear that the only change is that another line is added to the middle section, and this line will have the same value as the other lines in that section:

$$= (L_1(M_1-1)+1)(L_2(M_2-1)+1)$$

Denote the number of states s for which $I(s)=k$ as N_k. Then there are $\sum_{i=0}^{k} N_i$ states for which $I(s)\leq k$. If $k>M_1+M_2$, then

$$N_{k+1} = N_k + (L_1(M_1-1)+1)(L_2(M_2-1)+1)$$

Since all states s for which $I(s)\leq j$ must be in blocks zero through j (see (I) above), blocks zero through j must contain at least $\sum_{i=0}^{j} N_i$ states. From above, the actual number of states in blocks zero through j is $m_0' + mj$. If $\sum_{i=0}^{j} N_i > m_0' + mj$ for any j, then the partition fails. So, let

$$x_j = (m_0'-mj) - \sum_{i=0}^{j} N_i \text{ and choose } j>M_1+M_2.$$

Now find

$$\frac{x_{j+1}-x_j}{(j+1)-j} = \frac{[m_0' + m(j+1)) - \sum_{i=0}^{j} N_i] - [(m_0'+mj) - \sum_{i=0}^{j} N_i]}{(j+1)-j}$$

$$= \frac{-N_{j+1} + m}{1}$$

To find the second order differences, note that for all $j>M_1+M_2$,

$$N_{j+1} = N_j + (L_1(M_1-1)+1)(L_2(M_2-1)+1)$$

So,

$$\frac{\Delta^2 x}{\Delta j^2} = (m - N_{j+2}) - (m - N_{j+1})$$
$$= N_{j+1} - N_{j+2}$$
$$= -(L_1 (M_1 - 1) + 1)(L_2 (M_2 - 1) + 1 < 0$$

Clearly all higher order differences are zero, so the second order differences are constant. Thus, since the second order difference of x with respect to j is always negative, x must ultimately assume values less than zero, and the partition which was assumed to be correct must fail. Thus, a partition satisfying conditions A and B cannot be constructed, and the proof is complete.

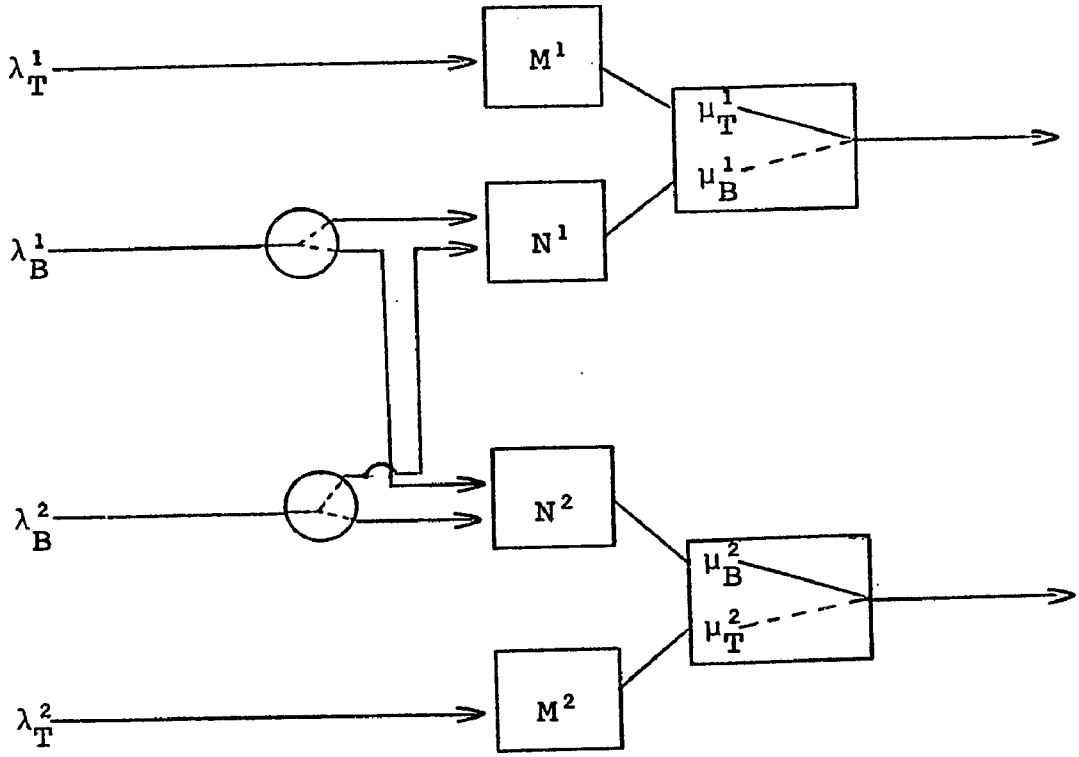
The Two-node Finite State Model

It would be possible to pursue the analysis of this model by attempting to generalize Wallace's work or by attempting to apply queueing theory tools in new ways. It is not, however, the principal goal of this work to advance the state of the art of queueing theory, and a simpler path is available. It is possible to maintain the model structure, but replace the potentially infinite batch queues with finite capacity queues. With this restriction, the state space of the model becomes finite and can be analyzed by conventional means. Some generality is lost in making this restriction: jobs which arrive when batch queues are full will be lost to the system, causing the mean batch queue length to be underestimated. We shall show later that the proportion of jobs rejected in this way can be kept small

(by allowing large capacities or by restricting studies to lightly loaded conditions), and therefore the error introduced by the restriction can be minimized.

A diagram of the restricted model is shown in Figure 4.2. The decision maker model need not be altered, and the only change to the host subsystems is that batch arrivals occurring when the batch queue is full will be rejected. If a batch arrival occurs to a full subsystem and the other subsystem is not full, the decision maker could send the arrival to the non-full subsystem. This algorithm was rejected on the grounds that traffic generated from overflows of this sort is really artificial and occurs only because of the finiteness of the queue. In the following work, arrivals to full subsystems are always rejected and are never transshipped to the other subsystem.

One final difficulty arises in the algorithm for the decision maker. The expected waiting time for a job, given the current state of a subsystem, is not known precisely, since execution times of batch jobs in front of the current arrival may vary. Also, depending on the number of terminal arrivals that occur while the job under consideration is enqueued, different numbers of terminals may be serviced because of the breakpoint priority scheme. Thus, a method of estimating the potential waiting time in each subsystem is required. The most precise estimate would be found by generating an analytical result for the expected waiting



Each node same as single node model, except batch queue is finite capacity and has two possible input streams.

Decision Mechanism  is unspecified.

System State: $(T^1, B^1, S^1, T^2, B^2, S^2)$

System parameters: $\lambda_T^i, \lambda_B^i, \mu_T^i, \mu_B^i, L^i, M^i, N^i, Bk^i(s)$

Decision Algorithm

All jobs assumed to be drawn from same demand distribution.

Finite State
Two Node Model

Fig. 4.2 Finite State Two Node Model

time for a batch job, given the system state. Again, such an analysis becomes exceedingly complex, and so a simple estimate has been used instead. Each batch job already in the system is estimated to require the mean batch service time, and each terminal already enqueued is assumed to be serviced before the new batch arrival. Terminal services are also assumed to require the mean terminal service time. With these assumptions, the decision makers can compute the expected delay in each subsystem and add the transshipping delay (if any) to give the total delay for a job. The turnaround factor for this delay is then multiplied by the resource cost for the subsystem under consideration, and the least cost subsystem is chosen to execute the job.

4.2 Analysis

Steady State Probability Distribution

Finding the steady state probability distribution for a finite state Markov model is straightforward. One simply writes down the Kolmogorov equations which define the rates of transition between states and then solves them as a group of simultaneous linear equations. This type of approach is outlined in standard queueing texts (P1, T1, C9).

In practice, the state spaces involved may be quite

large, so that solving the equations directly, even with the aid of a computer, becomes a considerable task. The recursive queue analyzer (RQA) of Wallace and Rosenberg (W3) provides a computational technique for finding numerical solutions to the steady state distributions of finite state Markov chains. The technique is described in detail in (W3); a brief description of the approach is all that is required here. A vector p^n is used to store the nth approximation of the steady state probability distribution. The true equilibrium p, when multiplied by the transition intensity matrix U, must yield

$$p \cdot U = \underline{0}$$

Since U is a transition intensity matrix, all its rows must sum to zero, and only elements on the diagonal can be less than zero. The iterative technique used by RQA involves transforming U into a stochastic matrix (all the rows sum to one and no elements are less than zero) by multiplying U by a scalar Δ and adding an identity matrix to it:

$$Q = \Delta U + I$$

(Δ is required to be less than 1/R, where R=max $| u_{ii} |$)

The iteration used to produce p^{n+1} from p^n is then given by

$$p^{n+1} = p^n (\Delta U + I)$$

If $p^{n+1} = p^n$ for some n, then

$$p^{n+1} = p^n = p^n (\Delta U + I)$$

$$= p^n \Delta U + p$$

thus, $\Delta p^n \cdot U = \underline{0}$

and p^n is the solution sought. This technique is also shown

to be applicable to discrete time Markov chains; in that case, the transition matrix is stochastic to begin with, so no transformation is required. All the models examined in this work are continuous time Markov chains. Wallace and Rosenberg also prove that the iterations are convergent for both continuous and discrete Markov chains, and an error estimate is obtained.

In addition to this theoretical work, Wallace and Rosenberg developed a computer program to implement their algorithm on an IBM 7090 computer. This program has since been modified to run on the IBM /360 series (H4). These implementations employ a data structure for the representation of sparse matrices, since a Markov chain with, for example, 1000 states would require a 1000 x 1000 matrix to represent its transition intensity matrix in the conventional way. Usually, most of the elements in the matrix are zero, so it is more economical to store only the non-zero elements. Wallace and Rosenberg observe that transition intensity matrices often exhibit considerable structure; in particular the models in which they are interested tend to have the same transition intensities repeated periodically along diagonals of the matrix. Consequently, they define elements in terms of quadruples and a periodicity factor, D, which is constant for the matrix. Each quadruple consists of (1) the transition intensity, (2) the row subscript for the transition, (3) the

column subscript, and (4) the last column subscript to which the transition applies. This last component acts as the upper bound in a DO - loop which uses D as its increment. Thus, a quadruple

$$(\lambda, i, j, L)$$

represents transitions of intensity from state i to state j, state i+D to state j+D, state i+2D to j+2D, etc., up to a transition from state i+rD to state j+rD, where

$$j+rD=L$$

The blocks of the transition intensity matrix for the two-node model, unfortunately, are not of fixed size (see Figure 4.3) and so this representation is not economical - considerable CPU overhead is incurred in attempting to find periodic state transitions and few are found. Consequently, RQA was modified so that the fourth element of each quadruple is unused. (This alteration is also appropriate because the /360 offers virtual storage not available on the 7090, so the cost of storage relative to CPU is decreased and an extremely compact data structure is less important.)

The automatic generation of the state transitions in itself is quite complicated. The breakpoints of both systems affect the state transitions due to departures, and the decision maker mechanism affects transitions due to arrivals. To decrease overhead in the state space generation

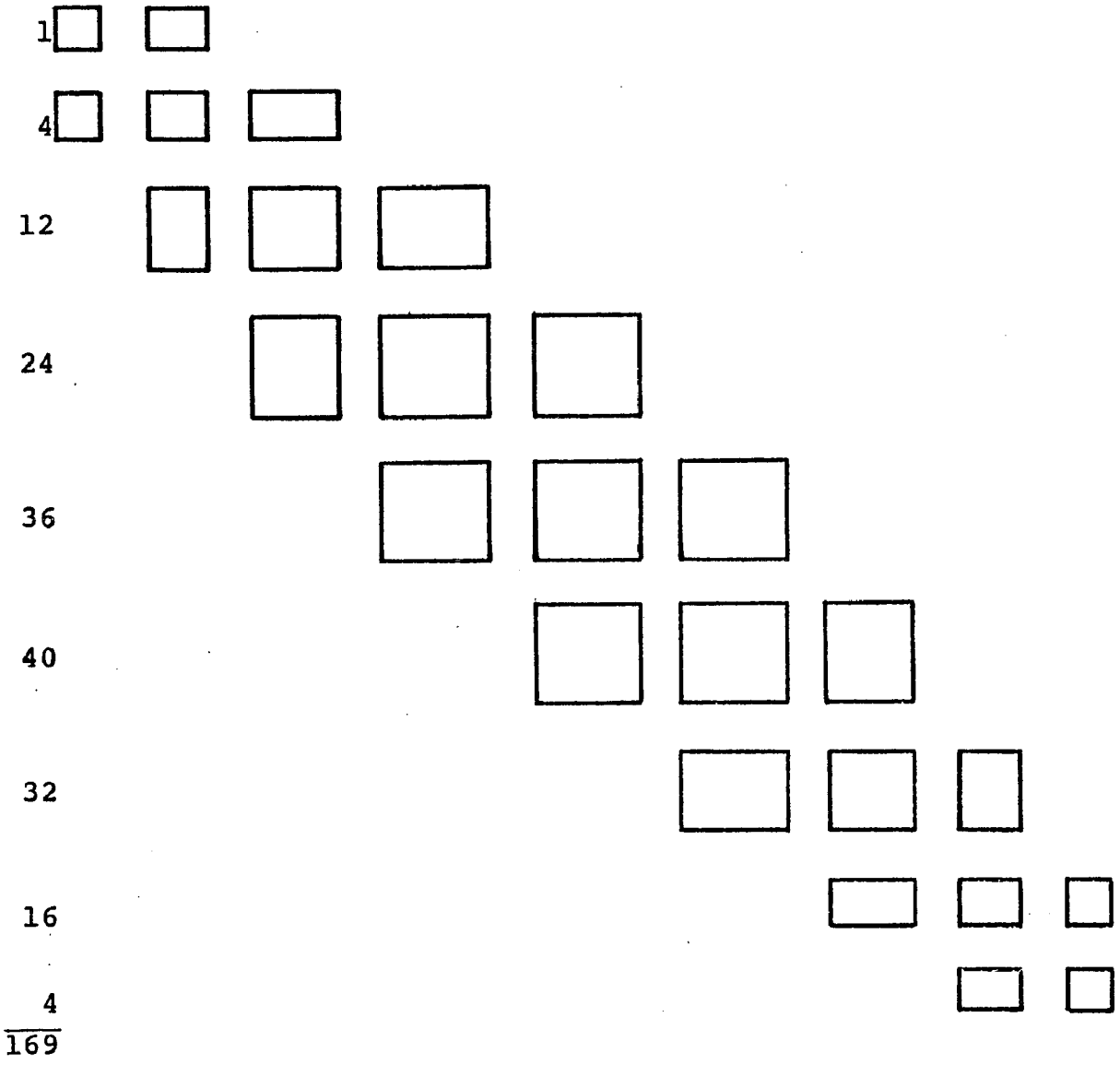


Fig. 4.3 Finite State Two Node Model - Block Structure

a hashing algorithm was used to provide a mapping from the six-tuple $(T_1, B_1, S_1, T_2, B_2, S_2)$ to the final single number used as an index to represent that system state.

Expected Waiting Time

As in the single node model, it will be useful to have a measure for the overall expected waiting time for the two-node model, in addition to the steady state probability distribution. The following argument will show that the mean waiting time for batch jobs in the unmodified model (with potentially infinite batch queues) again follows Little's Theorem. This result will be used to approximate the mean waiting time in the finite capacity batch queue model; so long as the system is not overloaded, the error in the approximation will be tolerable. The primary use for the measure will be in the comparison of results rather than as an absolute estimate of waiting time.

Definitions:

$E(N)$ = Mean number of batch jobs in the system (over time)

$E(N_A)$ = Mean number of batch jobs in the system observed at batch arrivals

$E(N_D)$ = Mean number of batch jobs in the system observed at batch departures

For this proof, we take the view that the system is a black box with a batch arrival rate of $\lambda_{B_1}+\lambda_{B_2}$. The

relation

$$E(N) = E(N_A)$$

must hold since arrivals are Poisson-distributed. Now consider the number of batch jobs in the system, measured at batch departures. So long as the model forms an irreducible, positive recurrent Markov chain, the mean number of batch jobs in the system observed at batch departures must be the same as the mean observed at batch arrival points. That is,

$$E(N_A) = E(N_D)$$

Next, define

N_{D_i} = Number of batch jobs left behind by ith departure

W_i = System residence time for ith departure

A_i = Number of batch arrivals during system residence time of ith departure.

f_i = Number of batch jobs that both arrive and depart during system residence time of ith departure

s_i = Number of batch jobs that were in the system when the ith departure arrived and were still in the system at the departure of the ith job

$R_i = s_i - f_i$ = the net remainder for the ith departure

Then it must be true that

$$N_{D_i} = A_i + s_i - f_i$$

Taking the expected value for both sides,

$$E(N_D) = E(A + s - f)$$

$$= E(A) + E(s-f) = E(A) + E(R)$$

Now observe that if a job j_1 passes up another job j_2 in the stream, then j_1 both arrives and departs during j_2's tenure in the system, adding one to f_i for j_2. At the same time, j_2

is in the system at j_1's arrival and also at j_2's departure, so one is added to s_i for j_1. Thus the effect of one job's passing another in the stream is cancelled out in the long run, as long as all jobs are processed eventually.

That is,

$$E(R) = E(s-f) = 0$$

and
$$E(N_D) = E(A) + 0$$
$$= E(A)$$

Intuitively, this last equation must be true; if $E(R) \neq 0$ then jobs must, on the average leave behind a greater (or lesser) number of jobs than arrived during their system residence times. While such a condition can hold for a particular job, if it is true on the average then the mean queue length must either grow without bound or achieve negative lengths.

Finally, we argue that, since arrivals are Poisson-distributed, the number of jobs that arrive during the system residence time for a job, must, on the average, be equal to the arrival rate times the system residence time. That is,

$$E(A) = (\lambda_{B_1} + \lambda_{B_2}) E(W) \ ;$$

therefore $E(N) = E(N_D) = E(A) = (\lambda_{B_1} + \lambda_{B_2}) E(W)$

and Little's theorem is demonstrated to hold for this model.

4.3 Parameter Studies

Determination of Parameters

To test the utility of the two-node model, several parameter studies were constructed. Parameters to be determined for each subsystem include:

1. Maximum number of terminals allowed in subsystem
2. Maximum number of batch jobs allowed in subsystem
3. Number of server states
4. Breakpoints for priority scheme
5. Batch and terminal arrival rates
6. Batch and terminal service rates
7. Resource cost per job
8. Round trip transshipping delay to the other system

In addition, the algorithm for the decision makers (described in section 4.1) requires specification of a waiting time versus turnaround factor curve.

Preliminary studies indicated that the size of the state space had a strong effect on the cost of computing the analytic results. Consequently, the maximum number of jobs allowed in terminal and batch queues and the number of server states have been limited as much as possible without distorting the system behavior. This requirement of short limits on the queue sizes has restricted parameter studies from investigating system behavior in cases where both subsystems are heavily loaded. Some examples of system

Table 4.1 State Space Size and Cost of Analysis

State Space Limits $(T_1,B_1,S_1,T_2,B_2,S_2)$	Number of States	Typical Cost For Analysis
(2, 2, 2, 2, 2, 2)	169	$2
(2, 3, 2, 2, 3, 2)	324	$3
(2, 4, 2, 2, 4, 2)	529	$5.75
(2, 2, 2, 3, 4, 3)	611	$6.75
(3, 3, 2, 3, 3, 2)	625	$7
(3, 2, 3, 3, 2, 3)	729	$8.25
(2, 3, 2, 4, 3, 3)	864	$10
(2, 2, 2, 4, 5, 3)	962	$14.50

Table 4.2 Traffic Parameters

Lightly Loaded System	Heavily Loaded System
.01509	.04324
.03028	.04056
.06717	.07724
.13116	.11227

units=jobs/sec

Table 4.3 Transshipping Parameters

Line Speed	Resource Cost/Job	Round Trip Transshipping Delay
no lines	1.0	
2000 bits/sec	1.0028	152.0 sec
10.8 Kilobits/sec	1.0078	28.1 sec
50 Kilobits/sec	1.012	6.1 sec

parameters with corresponding state space sizes are given in Table 4.1. The costs of computation shown in the table are approximate, since the precise cost of the analysis also depends on how heavily the model systems are loaded. Nevertheless, it is clear that the cost increases faster than linearly with the state space size. The time required per iteration of the algorithm will, of course, increase as the state space size increases, as will the amount of storage required; thus the non-linear increase in cost is to be expected.

The necessity of remaining within relatively small state space sizes did not prove to be too great a difficulty in examining cases where both systems were lightly loaded or only one system was heavily loaded. The problem of what arrival and service rates to use to define a heavily loaded or lightly loaded system was met by choosing representative periods for the cases from the curves for $E(W_B)$ vs. λ_B (Figure 2.25, reproduced here as Figure 4.4). The values derived from the original data point on the curve labelled CT273L were used to define the parameters for a lightly loaded system, and values from the original data point on curve CT263H were used to define parameters for a heavily loaded system. The actual values used for arrival and service rates are given in Table 4.2. The breakpoints used for the priority mechanism were again the same as in the single node runs, although fewer breakpoints were required

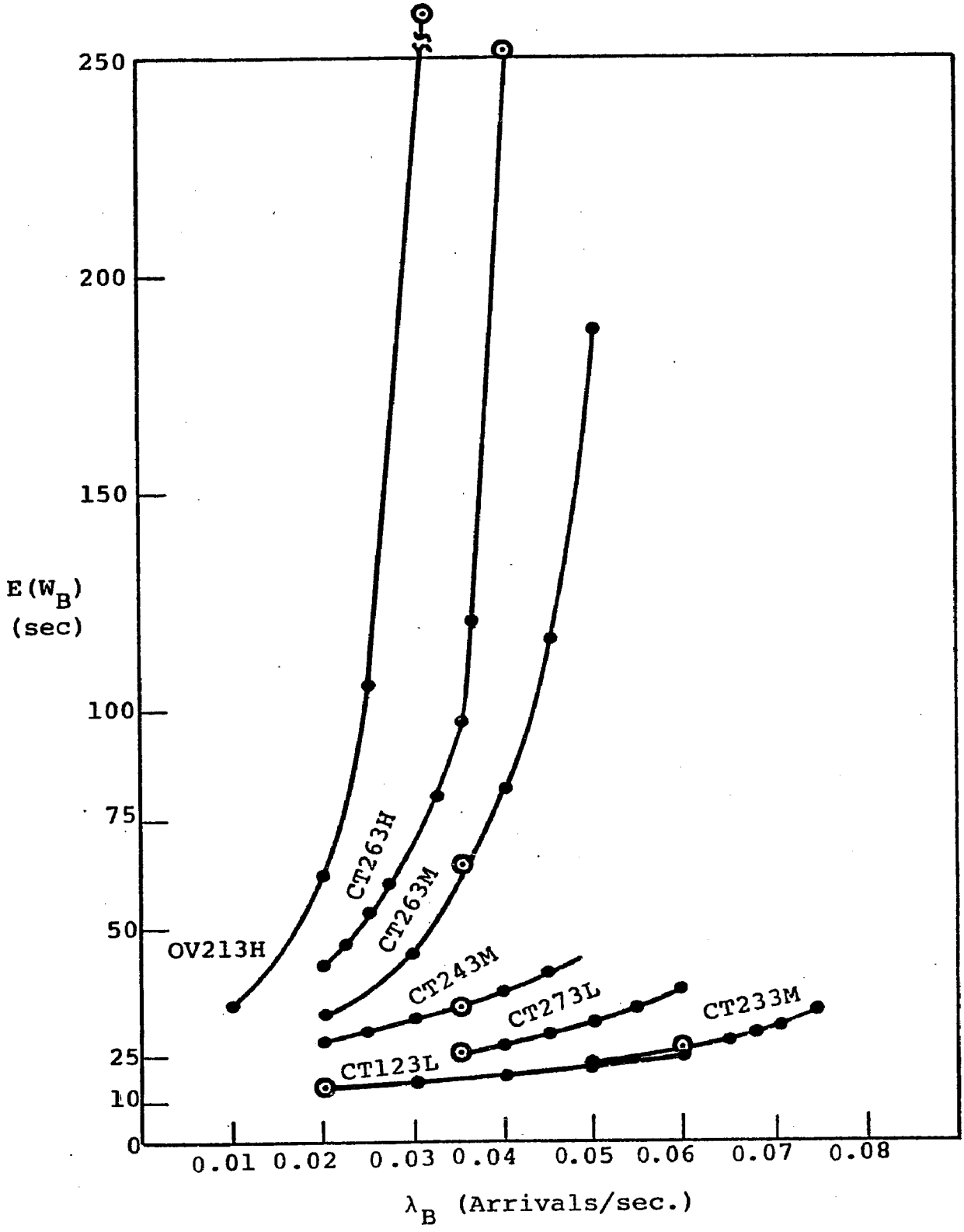


Fig. 4.4 Parameter Studies -- Mean Batch Wait vs. Batch Arrival Rate

BATCH JOBS IN SYSTEM (222;222) 50KB

ONE X = 0.02313

NODE 1			NODE 2		
0	0.69380	XXXXXXXXXXXXXXXXXXXXXXXXXXXXXX	0	0.69378	XXXXXXXXXXXXXXXXXXXXXXXXXXXX
1	0.27379	XXXXXXXXXXX	1	0.27380	XXXXXXXXXXX
2	0.03241	X	2	0.03241	X

BATCH JOBS IN SYSTEM (323;323) 50KB

ONE X = o.02290

NODE 1			NODE 2		
0	0.68695	XXXXXXXXXXXXXXXXXXXXXXXXXXXXX	0	0.68694	XXXXXXXXXXXXXXXXXXXXXXXXXXXX
1	0.27811	XXXXXXXXXXX	1	0.27812	XXXXXXXXXXX
2	0.03495	X	2	0.03495	X

BATCH JOBS IN SYSTEM (332;332) 50KB

ONE X = 0.02273

NODE 1			NODE 2		
0	0.68188	XXXXXXXXXXXXXXXXXXXXXXXXXXXXX	0	0.68182	XXXXXXXXXXXXXXXXXXXXXXXXXXXX
1	0.21280	XXXXXXXXX	1	0.21281	XXXXXXXXX
2	0.07579	XXX	2	0.07582	XXX
3	0.02953	X	3	0.02955	X

BATCH JOBS IN SYSTEM (332;332) No Lines

ONE X = 0.02276

NODE 1			NODE 2		
0	0.68275	XXXXXXXXXXXXXXXXXXXXXXXXXXXXX	0	0.68275	XXXXXXXXXXXXXXXXXXXXXXXXXXXX
1	0.27579	XXXXXXXXXXX	1	0.27580	XXXXXXXXXXX
2	0.03492	X	2	0.03492	X
3	0.00654	X	3	0.00654	X

Fig. 4.5 Light vs Light State Space Size Studies (Batch)

TERMINALS IN SYSTEM (222;222) 50KB

ONE X = 0.02505

NODE 1			NODE 2		
0	0.75159	XXXXXXXXXXXXXXXXXXXXXXXXXXXXXX	0	0.75156	XXXXXXXXXXXXXXXXXXXXXXXXXXXXX
1	0.19991	XXXXXXX	1	0.19993	XXXXXXX
2	0.04850	X	2	0.04851	X

TERMINALS IN SYSTEM (323;323) 50KB

ONE X = 0.02475

NODE 1			NODE 2		
0	0.74240	XXXXXXXXXXXXXXXXXXXXXXXXXXXXX	0	0.74237	XXXXXXXXXXXXXXXXXXXXXXXXXXXXX
1	0.19775	XXXXXXX	1	0.19777	XXXXXXX
2	0.04849	X	2	0.04851	X
3	0.01137	X	3	0.01137	X

TERMINALS IN SYSTEM (332;332) 50KB

ONE X = 0.02466

NODE 1			NODE 2		
0	0.73973	XXXXXXXXXXXXXXXXXXXXXXXXXXXXX	0	0.73968	XXXXXXXXXXXXXXXXXXXXXXXXXXXXX
1	0.19684	XXXXXXX	1	0.19685	XXXXXXX
2	0.05115	XX	2	0.05117	XX
3	0.01229	X	3	0.01230	X

TERMINALS IN SYSTEM (332;332) No Lines

ONE X = 0.02468

NODE 1			NODE 2		
0	0.74051	XXXXXXXXXXXXXXXXXXXXXXXXXXXXX	0	0.74048	XXXXXXXXXXXXXXXXXXXXXXXXXXXXX
1	0.19656	XXXXXXX	1	0.19657	XXXXXXX
2	0.05077	XX	2	0.05078	XX
3	0.01217	X	3	0.01218	X

Fig. 4.6 Light vs Light State Space Size Studies (Terminal)

```
 # BATCH JOBS IN SYSTEM                                          (2,2,2;4,5,3) No Lines

ONE X = 0.02339
NODE 1                                                  NODE 2
   0   0.70161  |XXXXXXXXXXXXXXXXXXXXXXXXXXXXXX        0   0.30083  |XXXXXXXXXXXX
   1   0.22086  |XXXXXXXXX                               1   0.20426  |XXXXXXXX
   2   0.07754  |XXX                                     2   0.15058  |XXXXXX
                                                        3   0.12312  |XXXXX
                                                        4   0.12072  |XXXXX
                                                        5   0.10048  |XXXX
```

```
 # BATCH JOBS IN SYSTEM                                          (232;433) 10.8KB

ONE X = 0.02088
NODE 1                                                  NODE 2
   0   0.62640  |XXXXXXXXXXXXXXXXXXXXXXXXXXXXXX        0   0.38303  |XXXXXXXXXXXXXXXXX
   1   0.24328  |XXXXXXXXXXX                            1   0.30962  |XXXXXXXXXXXXXX
   2   0.09749  |XXXX                                   2   0.25360  |XXXXXXXXXXXX
   3   0.03283  |X                                       3   0.05376  |XX
```

```
 # BATCH JOBS IN SYSTEM                                          (232;433) 50KB

ONE X = 0.01955
NODE 1                                                  NODE 2
   0   0.58658  |XXXXXXXXXXXXXXXXXXXXXXXXXXXXXX        0   0.47217  |XXXXXXXXXXXXXXXXXXXXXXX
   1   0.28641  |XXXXXXXXXXXXXX                         1   0.42600  |XXXXXXXXXXXXXXXXXXXX
   2   0.09300  |XXXX                                   2   0.08387  |XXXX
   3   0.03401  |X                                       3   0.01797  |X
```

Fig. 4.7 Light vs. Heavy State Space Size Studies (Batch)

since the number of server states allowed was reduced.

To be considered valid, a parameter study was required to have a probability of rejecting a batch job (due to a full system) of no more than .10 at either host. In nearly all studies, this probability was kept below .05. Figures 4.5 and 4.6 show histograms for numbers of batch and terminal jobs in each subsystem for two lightly loaded systems with varying state space sizes. These figures indicate a relatively small change in results as the state space size is varied. Figure 4.7 illustrates the most extreme test for the limitation on number of batch jobs in the system. The first subsystem in this example is lightly loaded, while the second is heavily loaded. When there are no communications lines, the probability of rejecting a batch job from the heavily loaded subsystem reaches .10, but as lines are introduced, the sharing of loads reduces the probability to .054 even when a capacity of three batch jobs instead of five is imposed.

The resource costs per job were determined on the basis of computing center budgets of $.5 million per center. For studies in which no lines existed (used as a baseline -- in this case no transshipping of jobs is possible and no line costs are incurred) resource cost per job was assumed to be 1.0. If lines existed in a study, the cost of the lines was divided equally between the two hosts, and the respective

resource costs per job were increased in proportion to the percentage budget increases. Line costs were estimated as shown in Figure 3.2 (p. 108). Transshipping delays incurred by jobs were estimated on the basis of the job sizes and line speeds as discussed in Chapter 3. Table 4.3 displays the actual values used for resource costs and transshipping delays. The sole exception to these figures is the series of studies in which University of Michigan and Wayne State University charges were used as a basis for comparing centers with different resource costs. The ratio of normal priority university/government CPU rates was used to define the relative resource costs; this ratio was 1.195 (Wayne State) to 1.0 (Michigan).

Finally, the turnaround factor versus waiting time curve developed in Chapter 3 was used in the decision makers' algorithm. The curve was digitized as shown in Table 4.4 and a linear interpolation method was used between the points defined in the table.

Table 4.4 Turnaround Factor Function

Total Waiting Time (sec)	Turnaround Factor
0.	.005
120	.006
300	.008
450	.040
900	.140
1800	.360
2700	.480
3600	.550
5400	.660
7200	.730

Case Studies

Figures 4.8-4.11 summarize the results of the two-node model parameter studies. In each figure, the top bar graph gives the mean waiting time for batch jobs in the system. The middle graph indicates the probability that a random arrival will be transshipped to the other host for execution. Note that if arrival rates are the same to both systems, even if all jobs from one host are shipped to the other host, the overall probability of transshipping an arbitrary job is only .5. The bottom graph shows the expected cost per job. This cost is computed in the same way

the decision maker computes expected cost for a given arrival:

mean total cost/job = $\overline{TC}$ = $\overline{RC}$*TF($\overline{TD}$)

where $\overline{RC}$ = mean resource cost per job

TF(x) = Turnaround factor function for delay x

$\overline{TD}$ = mean total delay per job

= mean transshipping delay

+ mean system residence delay

$$\text{mean system residence delay} = \frac{E(\text{number in system})}{\lambda_{B_1} + \lambda_{B_2}}$$

A vertical slice from any figure then corresponds to a study with a given set of parameters.

The first set of studies, documented in Figure 4.8, investigates load sharing between two hosts which are both lightly loaded. The parameters for both systems are thus exactly identical. The three studies at the right side of the figure are all for a line speed of 50KB, but with varying state space sizes. Because of the relatively slight variation in results for these studies, the smallest of the state spaces was chosen for the studies at different line speeds. The study on the far left (no lines) uses a larger state space in order to compensate for the truncation of the queue length distributions when no sharing takes place. Although the cost per job appears lowest for the 2KB case,

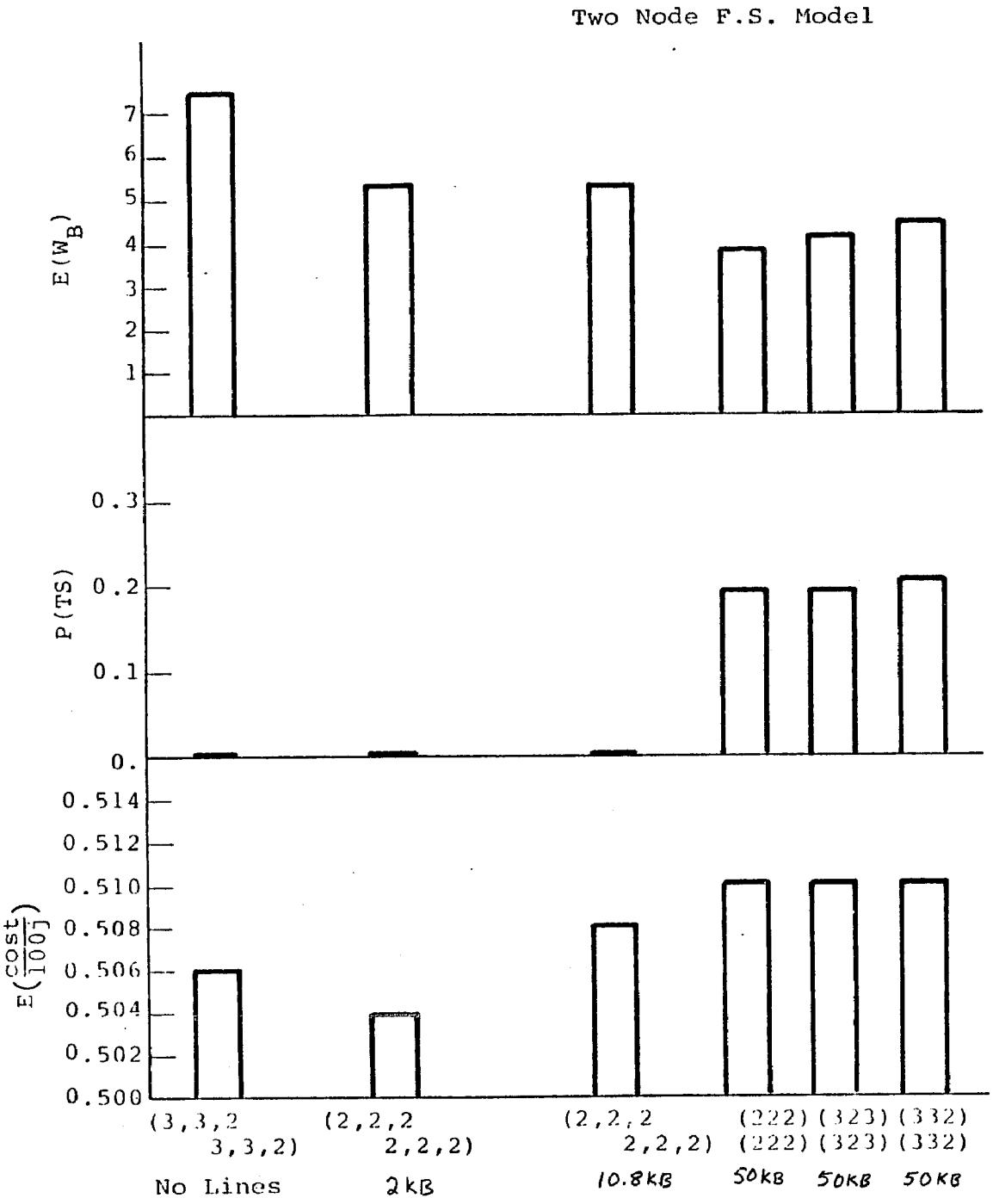


Fig. 4.8 Two Lightly Loaded Systems Identical Parameters

this result is an artifact of the smaller state space size. Since there is no transshipping of jobs in the 2KB and 10.8KB cases, the cost per job must be higher than if there are no lines. The larger state space used in the no lines study, however, allows longer batch queues to form, leading to a longer mean queue length and therefore a longer mean waiting time and higher cost per job. In the 50KB case, sharing does become feasible; the probability of transshipping a job is approximately .19, and the expected waiting time is less than in the 2KB and 10.8KB cases. The cost per job is still higher than the no lines case. This cost per job reflects the values built into the turnaround factor curve: the value of the decrease in waiting time brought about by the introduction of 50KB lines is not as great as the cost of the lines. Thus, given the turnaround factor curve, the line rates, job sizes, and the fact that both systems are lightly loaded, the model indicates that introducing a load sharing network would not benefit users enough to compensate them for the additional resource cost.

Figure 4.9 illustrates results for a very similar case. The parameters used in these studies are again taken from the lightly loaded system data, but the batch input rates to both systems have been redistributed. The studies were done in pairs; the left hand member of each pair is for the identical parameters as the right hand member, except that

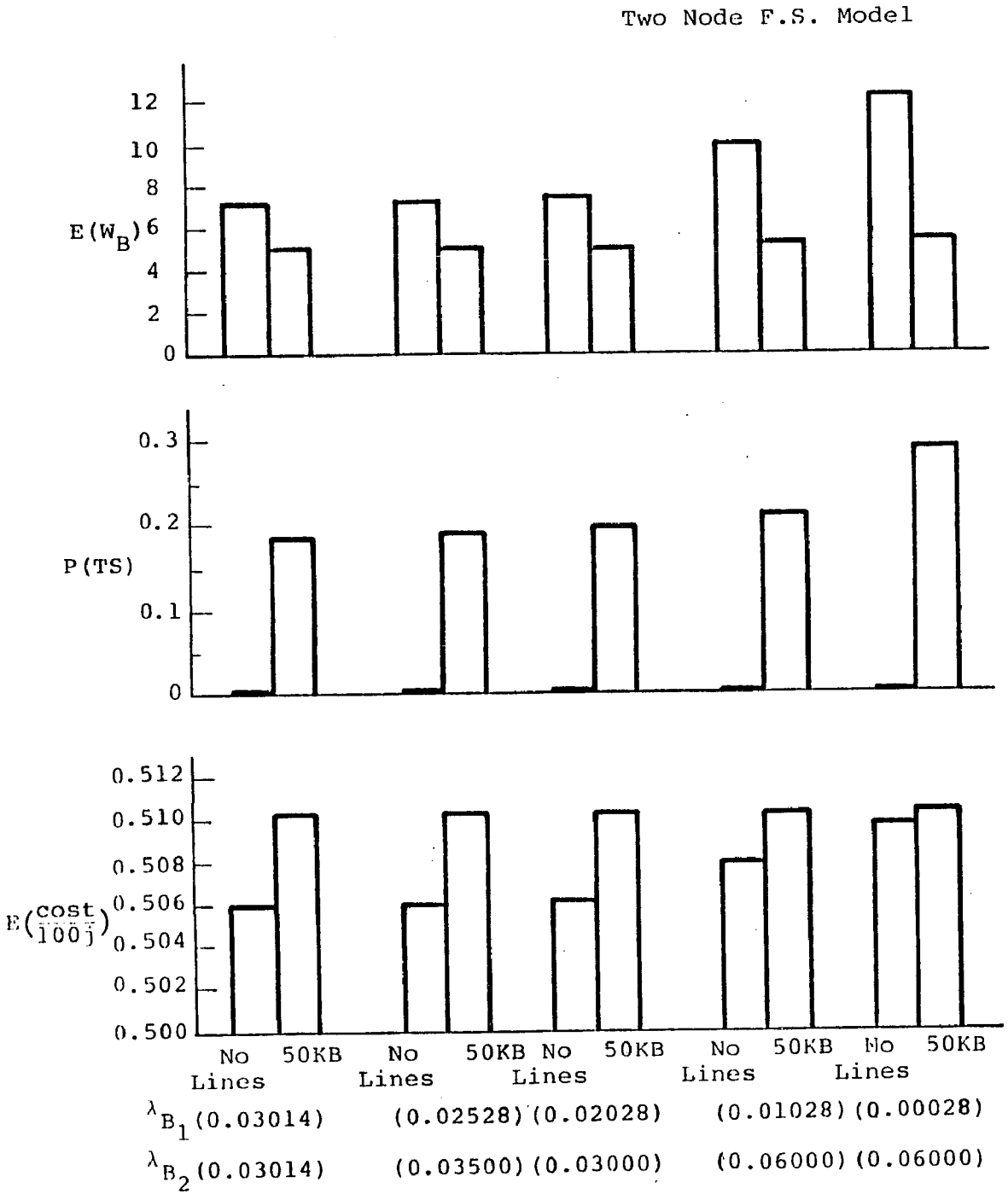


Fig. 4.9 Two Lighly Loaded Systems Varying Batch Arrival Rates

there are no lines in the left hand case and there are 50KB lines in the right hand case. The no lines cases were again run with slightly larger state spaces in order to avoid truncation effects on the queue length distributions. When there is a network and transshipping occurs, the mean queue lengths of both systems are diminished, and so the finite capacities of the queues cause less of a problem.

The leftmost pair of studies corresponds to identical loads on both systems, and the load is gradually shifted so that, for the rightmost study more than 99% of the batch arrivals occur to only one of the subsystems. The most noteworthy observation to be made here is the relative constancy of the expected waiting time in the 50KB line cases as the load is redistributed. The no-line studies show a marked increase in mean waiting time as the load is shifted more and more onto one of the systems, while the system is able to transship jobs in the 50KB cases so that the mean waiting time is kept nearly constant. This decrease in the variance of waiting times has often been put forth as one of the potential benefits of load sharing. In spite of this behavior, however, the mean cost per job remains lower in the no line cases throughout these studies. Again, the decrease in mean waiting time is not of sufficient magnitude to compensate for the line costs.

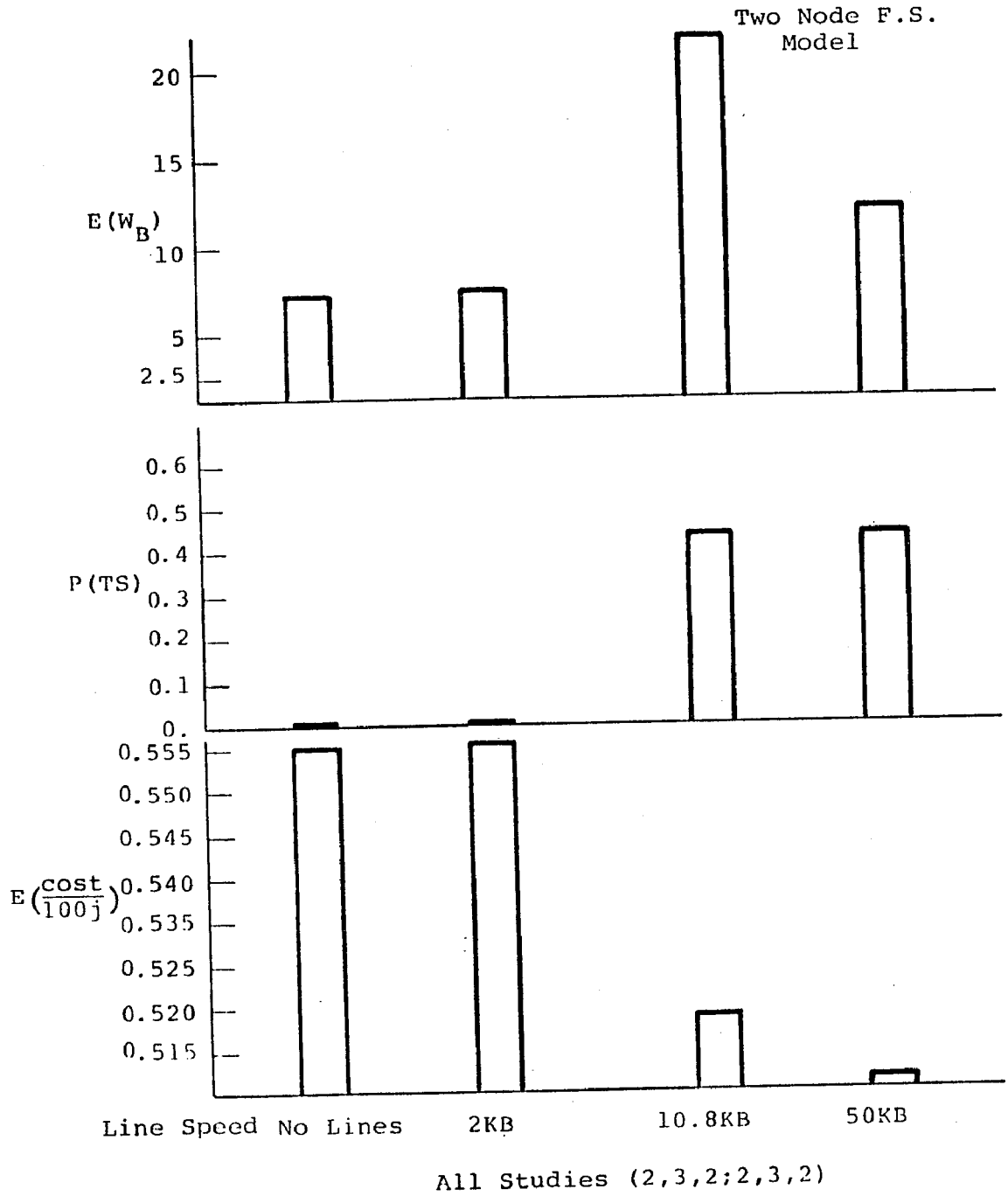


Fig. 4.10 Two Lightly Loaded Systems
Different Resource Costs

The studies of Figure 4.10 show a case in which the cost per job is decreased dramatically by the introduction of a network. In this case, the resource costs have been made to reflect the ratio observed between the University of Michigan and Wayne State University CPU rates. The no lines case and the 2KB lines cases are virtually identical; transshipping of jobs is not profitable so long as the line speed is low. Once the line speed is increased to 10.8KB, however, virtually all the jobs from the more expensive system are transshipped to the other host. (Probability of transshipping is more than .43.) The expected waiting time in this case increases, but the increase in waiting time cost is more than offset by the reduction in resource cost. Increasing the line speed to 50KB cuts the waiting time (and therefore the cost per job) down somewhat, since virtually all jobs from the expensive host were being transshipped even in the 10.8KB case.

The final set of parameter studies is summarized in Figure 4.11. In these studies, one system uses the parameters for a heavily loaded system, while the other system reflects lightly loaded conditions. As in the first studies, three different state spaces were used in the 50KB lines case to compare their results. The symmetric state space on the far right was tested first, but because of the asymmetric nature of the terminal loads in the two systems, asymmetric state spaces proved to be more appropriate. For

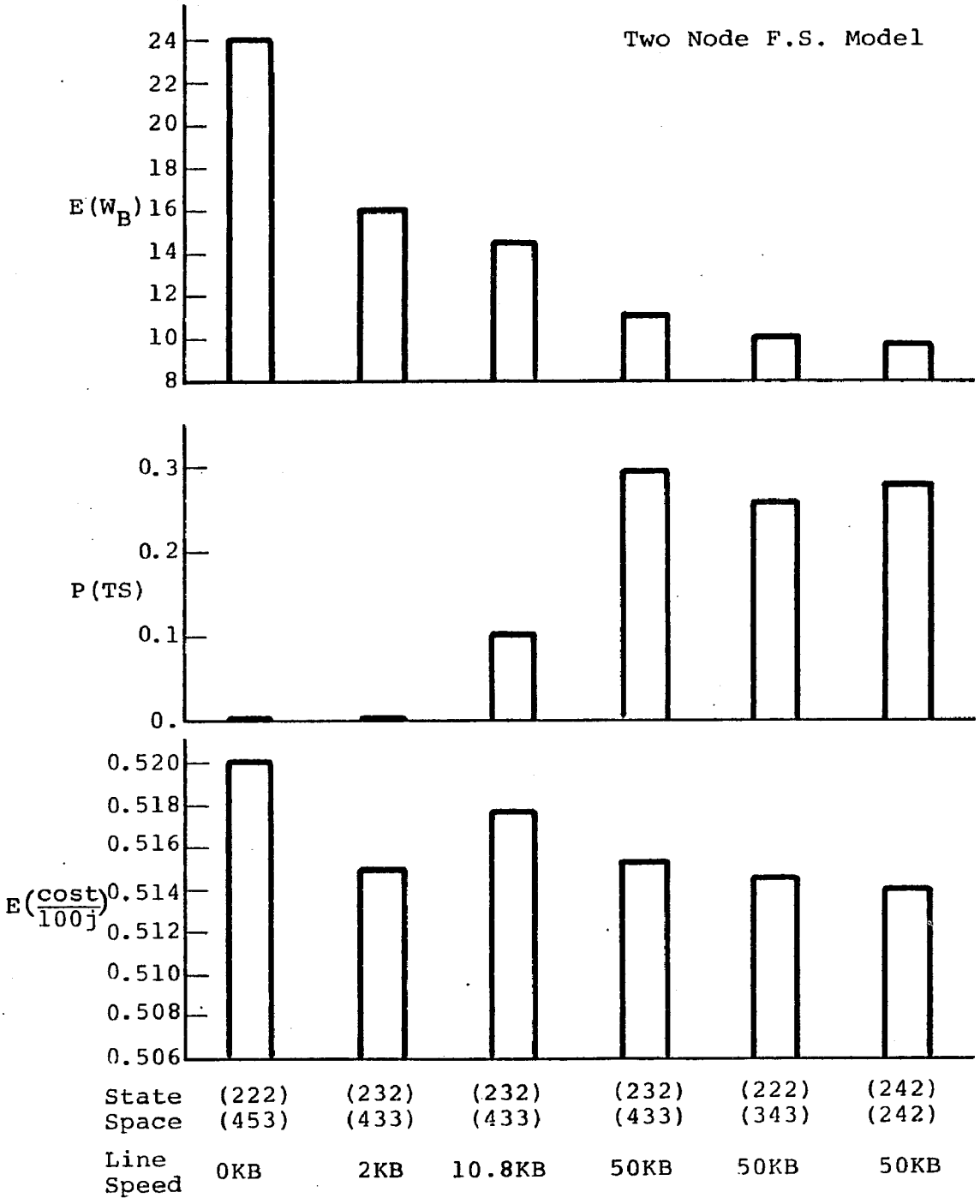


Fig. 4.11 One Lightly Loaded System
One Heavily Loaded System

the no lines study, a larger state space for the heavily loaded system was required in order to keep the probability of rejecting arrivals under 10%. Like the studies of Figure 4.8, the 2KB case and the no lines case differ only because of state space size; there is no transshipping of jobs in either case. In these studies, however, the mean cost per job and the mean waiting time are reduced below the no lines level when 10.8KB lines are used, and they are reduced again when the line speed is increased to 50KB. Predictably, the probability of transshipping increases with the line speed.

In summary, the model predicts that load sharing is a reasonable endeavor if one system is lightly loaded, the other is heavily loaded, and the line speeds are 10.8KB or faster; or if both systems are lightly loaded, line speeds are 10.8KB or faster, and resources costs differ by 20%. These predictions are necessarily dependent on the particular parameters used: the arrival and service rates, the estimates of job size and line delays, and the turnaround factor function. If the user of the model can specify these parameters and functions, the model can generate reasonable predictions about the behavior and cost effectiveness of the resulting network.

Chapter 5 - Multi-node Model

In order to represent networks of more than two nodes, a more general model is required than the one presented in Chapter 4. We could attempt to extend the two-node model directly; however a three-node model of similar construction would require a nine-tuple to describe its state space. Thus, solutions for three-node or larger models, while theoretically no more complex, could not in practice be obtained because of the large numbers of states involved. Consequently, a simpler model has been developed to provide information about load sharing in larger networks. This chapter will describe the model, provide some insights into its analytical structure and convergence properties, and finally, present some parameter studies which illustrate its behavior.

5.1 Description

Several competing goals had to be resolved in the construction of the multi-node network model. Because of the effort involved in constructing and validating the single-node model, there was a strong incentive to retain it as a model for each network node. Maintaining the ability of the decision maker to send jobs to different systems on the

basis of their current states was also a desirable goal. The overriding factors considered, however, were the need to have an extendable model to reflect relatively large networks and the need to maintain a model which would yield analytical results. The knowledge available to decision makers of the instantaneous states of other subsystems finally was sacrificed so that the other goals could be achieved.

The multi-node model, like the two-node model, includes models for each host system and for decision makers which can transship batch jobs to the neighbors of a given host. The original single node model with its two queues and breakpoint priority scheme has been retained to model each node of the network. Because of certain properties of the decision maker model (described below) it was not necessary to restrict the single node model to have a finite capacity batch queue as we did in Chapter 4. Thus, each host model in the system (and so the entire network) has an infinite state space.

The model for the decision makers is more restricted than the one employed in Chapter 4. The decision maker of the multi-node model is prevented from knowing anything about the system state at the time a transshipping decision is to be made. The decision on where to send a particular arrival is then made on a probabilistic basis. If a node i

has, for example, three neighbors j, k, and ℓ to whom it may transship jobs, there will be probabilities b_{ij}, b_{ik}, $b_{i\ell}$ assigned to the links (i,j), (i,k), (i,ℓ), and also a probability a that the job will not be transshipped, but be accepted by host i. These probabilities sum to one; i.e., when a job arrives, we can picture the decision maker as throwing a four-sided die which is loaded in proportion to the probabilities a_i, b_{ij}, b_{ik}, and $b_{i\ell}$. The job is then transshipped or accepted locally on the basis of the face that shows on the die. This scheme for assigning a job to a host may seem arbitrary and unrealistic -- and, in truth, the primary motivation for it is to maintain a model which is analytic (see Section 5.2). However, it may be a less unrealistic model than it appears at first glance. Several systems acting together could cause considerable message traffic merely in keeping each other informed of the precise queue lengths in their systems. In addition, instantaneous queue length is not necessarily a good measure of waiting time, since jobs do not all require the same processing time. Thus, simply assigning fixed probabilities for transshipping arriving jobs to particular hosts may have some merit if the probabilities are carefully assigned. To add flexibility to such a scheme, we might allow these probabilities to be altered as the states of different host systems change over time.

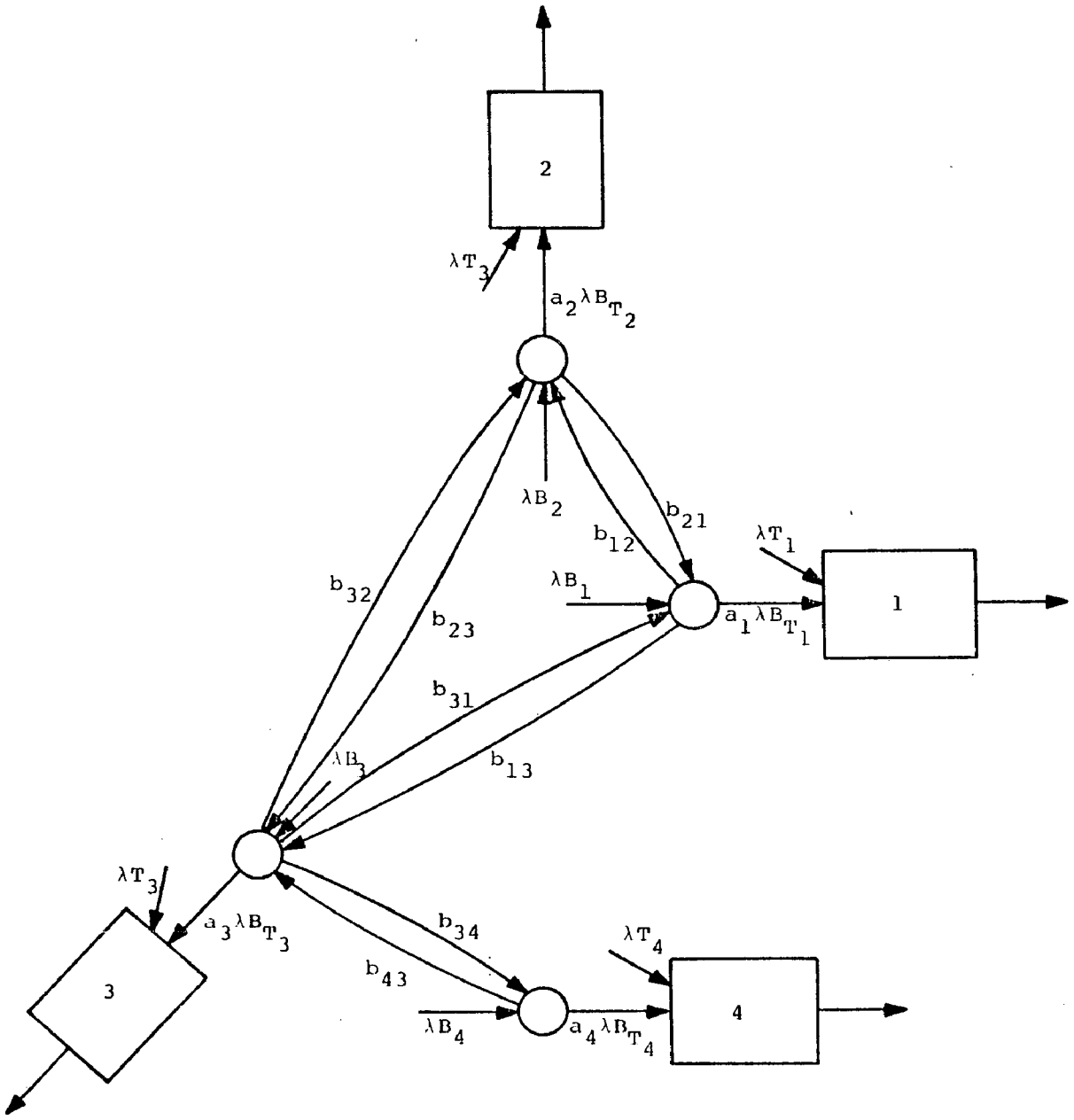


Fig. 5.1 Multi-Node Model

Another difference between the multi-node and two-node model decision maker is the precise point where a transshipped job enters the host system. In the two-node model, a transshipped job was sent directly to the batch queue of the other system -- no further decisions were made with respect to that job. In the multi-node model, a transshipped job enters the new host at the decision maker, not at the batch queue. Thus, each system that receives a job may apply its decision maker to that job, whether the job was submitted locally or arrived via the network. This approach yields a model which is slightly more complex, but more realistic than the two-node model, since in practice a job should be sent to the least cost alternative regardless of the job's origin. Because a given host is assumed to be connected with (and have knowledge about) only a subset of the hosts on the network, the host which receives a transshipped job may know of an alternative of which the original host is unaware. If the receiving host is allowed to transship arriving jobs, it can make use of this knowledge.

A drawing of a typical multi-node model as we have described it appears in Figure 5.1. The method for assigning probabilities a_i and b_{ij} will now be outlined. The technique used is an iterative one: initially all a_i's are set to one and all b_{ij}'s are set to zero. These values are then successively modified until a stable configuration is

reached. The precise method used employs the cost functions defined in Chapters 3 and 4:

Using the notation of Section 4.3:

let C_{ij} = cost of executing a job from node i at node j

$= RC_j * TF(TD_{ij})$

Given a matrix of probabilities

$$\begin{pmatrix} a_1 & b_{12} & \cdots & b_{1n} \\ b_{21} & a_2 & \cdots & b_{2n} \\ \vdots & & & \vdots \\ b_{n1} & b_{n2} & \cdots & a_n \end{pmatrix}$$

($b_{ij} = 0$ always if nodes i and j are non-adjacent)

Calculate the total batch input rate to each decision maker:

$$(\lambda_{B_{T_1}}, \lambda_{B_{T_2}}, \ldots, \lambda_{B_{T_n}})$$

The net batch input rate to the server at host i is then

$$\lambda_{B_{N_i}} = a_i \cdot \lambda_{B_{T_i}}$$

Using the single node model, each node can now be analyzed in isolation, and the mean system residence time for batch jobs in each system ($E(W_{B_i})$) can be found, yielding

$$(E(W_{B_1}), E(W_{B_2}), \ldots, E(W_{B_n}))$$

Since the transshipping delays are fixed, the TD 's can be found, and the matrix of costs

$$\begin{pmatrix} C_{11} & C_{12} & \cdots & C_{1n} \\ C_{21} & C_{22} & \cdots & C_{2n} \\ \vdots & & & \\ C_{n1} & C_{n2} & \cdots & C_{nn} \end{pmatrix}$$

($C_{ij} = \infty$ if i and j are nonadjacent)

can be determined. Now for each pair of nodes in the network, calculate a cost ratio

$$CR_{ij} = \frac{C_{ii} - C_{ij}}{C_{ii}}$$

The transshipping probabilities are then adjusted as follows:

Let x and y be chosen so that $0<x<1$ and $0<y<1/n$ (x determines the criterion for convergence and y determines the rate of change of the probabilities). Then

if $\|CR_{ij}\| \leq x$	$a_i \leftarrow a_i$
	$b_{ij} \leftarrow b_{ij}$
if $CR_{ij} > x$	$a_i \leftarrow a - y \cdot a_{ij}$
	$b_{ij} \leftarrow b_{ij} + y \cdot a_{ij}$
if $CR < -x$	$a_i \leftarrow a_i + y \cdot b_{ij}$
	$b_{ij} \leftarrow b_{ij} - y \cdot b_{ij}$

If no transshipping probabilities are changed by this procedure (i.e., $|CR_{ij}| \leq x$ for all i and j) then a stable set of probabilities has been reached.

An examination of the convergence properties of this algorithm and the overall analytic characteristics of the model will be made in the next section. Note, however, that the iterative method as outlined above depends on the separability of the network -- that is, it requires that each node can be analyzed in isolation. In the next section we shall show that the multi-node model has this property. Also, it should be clear that the network exhibits a stable behavior with any set of transshipping probabilities -- a

set of probabilities, together with the local batch input rates is sufficient to determine the traffic intensities throughout the network. So long as the system is separable, these traffic intensities can be used to determine the queue length distribution and expected waiting times at each host independently.

5.2 Analytic and Computational Aspects of the Multi-Node Model

In the previous section, the general structure and operation of the model were described. This section will examine the model in more detail, first looking at the model at a particular iteration and then investigating the properties of the model across iterations of the probability matrix.

At a particular iteration (that is, for a particular set of values in the transshipment probability matrix) the model can be solved analytically in the sense that equilibrium queue length and server state distributions can be obtained for all of the subsystems in the network. Because the Poisson process is preserved under random selection (P1, p. 47), a decision maker in this model will generate Poisson-distributed output streams if the sum of the input steams to the decision maker is Poisson-

distributed. Thus, since the sum of Poisson streams is Poisson-distributed and since the only external sources in the model (the local batch input streams) are Poisson-distributed, all flows in the network must be Poisson-distributed, with different parameters. The net flow of batch jobs into subsystem i, then, is simply a_i multiplied by the total flow into the decision maker for that subsystem. By the same argument presented for the single node model (pp.39-40), the model for each subsystem will have as its state space an irreducible Markov chain. The conditions for positive recurrence again will depend on the particular values for the arrival and service rates at a node. Since all of the other assumptions required of the single node model are satisfied by the subsystem model, we can again apply Wallace's method of analysis; if the iterations converge, the subsystem model must be positive recurrent. It is important to note that this gain in analytical simplicity over the two-node model is made at the expense of the knowledge the decision maker has about the states of different subsystems. If the decision to send a job to a particular site is influenced by the state of the various subsystems, the traffic distributions will no longer be Poisson-distributed and we could not analyze the network by decomposing it into independent subsystems. It is precisely the decision maker's ignorance of subsystem states that makes the subsystems independent.

The problem of calculating the traffic intensities for the model given the transshipment probability matrix has not been considered so far. In fact, for a general network this computation can become fairly involved. The complexity is primarily due to the fact that the same job may pass through several decision makers before finally being accepted by a particular host. Suppose, for example, the network has only two nodes. It is possible that all elements of the transshipment matrix are positive; that is, both systems transship some jobs to each other. (We might question the value of such a transshipment scheme or the value of a strategy that leads to such a matrix, but such matrices are within the scope of this model.) If we wish to find the net input rate to the two decision makers, we can write the following equations:

$$\lambda_{B_{T_1}} = \lambda_{B_1} + b_{21} * \lambda_{B_{T_2}}$$

$$\lambda_{B_{T_2}} = \lambda_{B_2} + b_{12} * \lambda_{B_{T_1}}$$

so,

$$\lambda_{B_{T_1}} = \lambda_{B_1} + b_{21} * (\lambda_{B_2} + b_{12} * \lambda_{B_{T_1}})$$

$$= \frac{\lambda_{B_1} + b_{21} * \lambda_{B_2}}{1 - b_{12} * b_{21}}$$

and

$$\lambda_{B_{T_2}} = \frac{\lambda_{B_2} + b_{21} * \lambda_{B_1}}{1 - b_{12} * b_{21}}$$

While such equations may be written down and solved for a particular network topology, if a general program is desired which does not require alterations for each particular network topology used, an alternative method must be found. For this research, a recursive subroutine was written making use of the fact that the total input to a decision maker

could be defined in terms of the local input rate and the total input rates of the neighbors of the decision maker under consideration:

Define n_i to be the number of neighbors of node i

λ_{B_i} = local input rate to decision maker at node i

$\lambda_{B_{T_i}}$ = total input rate to decision maker at node i

h_{ij} = host number of the jth neighbor of host i

then

$$\lambda_{B_{T_i}} = \lambda_{B_i} + \sum_{j=1}^{n_i} b_{h_{ij}i} * \lambda_{B_{T_{h_{ij}}}}$$

Each $\lambda_{B_{T_{h_{ij}}}}$ term is multiplied by the fraction of the load sent from host j to host i. As the recursion proceeds to the next level down, a new b factor will multiply all terms generated by lower level λ_{B_T}'s. Thus, a product of b-factors is built up as the recursion proceeds downward. So long as all b's are less than one, the multiplier for additional traffic by successive levels of the recursion will decrease, and an arbitrarily small threshold can be defined so that when the multiplier for the series falls below the threshold, the recursion is terminated. A brief example of a two-node network can illustrate this recursion:

let $b_{12} = .2$, $b_{21} = .3$, $\lambda_{B_1} = 10$, $\lambda_{B_2} = 8$, and

multiplier threshold = .01

Recursion level		Accumulated multiplier
0	$\lambda_{B_{T_1}} = 10 + (.3) * \lambda_{B_{T_2}}$	1
1	$\lambda_{B_{T_2}} = 8 + (.2) * \lambda_{B_{T_1}}$	.3
2	$\lambda_{B_{T_1}} = 10 + (.3) * \lambda_{B_{T_2}}$	.3*.2=.06
3	$\lambda_{B_{T_2}} = 8 + (.2) * \lambda_{B_{T_1}}$	.06*.3=.018
4	$\lambda_{B_{T_1}} = 10 + (.3) * \lambda_{B_{T_2}}$	.018*.2=.0036

When the multiplier falls below the specified threshold, the local input rate alone is returned as the value to the next level upwards. So, in this example, level 4 returns $\lambda_{B_{T_1}}=10$ to level 3:

Recursion level

4	$\lambda_{B_{T_1}} = 10$
3	$\lambda_{B_{T_2}} = 8 + .2*10 = 10$
2	$\lambda_{B_{T_1}} = 10 + .3*10 = 13$
1	$\lambda_{B_{T_2}} = 8 + .2*13 = 10.6$
0	$\lambda_{B_{T_1}} = 10 + .3*10.6 = 13.8$

According to the formula derived above, the true value for $\lambda_{B_{T_1}}$ in this case is 13.19. By lowering the multiplier threshold, any desired accuracy can be obtained.

One other approximation has been made in programming the multi-node model in order to decrease the cost of computation. As explained above, the equilibrium distributions for the queue lengths and server state can be obtained for each subsystem at each iteration of the model by using the techniques of Chapter 2. The iterative procedure for the multi-node model does not, however,

require these distributions; only the mean system residence time for the batch jobs in a subsystem is necessary. Consequently the parameter studies shown in Figure 4.4 (p. 140) were used to give an approximation for the mean system residence times, given the batch arrival rate. Since the higher derivatives of the curves which join the data points appear to be small, linear interpolation of the curves should provide adequate accuracy, and if unusual behavior is observed, an approximate solution can always be rechecked analytically. In utilizing these continuous curves, we assume that the intermediate values for the batch arrival rate will all generate positive recurrent Markov chains. For particular parameter studies with the multi-node model, then, the parameters of each subsystem will be approximated by a curve relating batch arrival rate to mean total delay for batch jobs. Given a new batch arrival rate for a subsystem, computation of the new mean delay for batch jobs is reduced to a simple interpolation.

The convergence of the series of iterations to a stable transshipment probability matrix depends on the values of y and x chosen. For example, if $x > \max_{ij} |CR_{ij}|$ initially, the system converges at once. If y is chosen to be large (close to 1/n) while x is small, then loads may oscillate back and forth between systems so that the convergence criterion is never met. An example of such oscillation will be discussed in Section 5.3. Thus it will not be possible to prove that

the solution to an arbitrary network converges independent of y and x. A proof of the convergence of the algorithm dependent on x and y has not been obtained. However, for all of the parameter studies in the following section the algorithm led to stable solutions (with the exception of those which were intentionally made to oscillate).

5.3 Parameter Studies

The number of parameters required for the specification of a subsystem in the multi-node model is smaller than the number needed in the two-node case, since the behavior of the subsystem is summarized by the $E(W_B)$ vs. λ_B curves. This one curve takes into account λ_T, μ_B, μ_T, the maximum number of terminals allowed in the subsystem, and the breakpoints for the priority scheme. Each subsystem thus requires the following parameters:

1. λ_B vs. $E(W_B)$ curve
2. Local batch arrival rate (λ_B)
3. Resource cost per job
4. Number of hosts adjacent to this host and their indices
5. Round trip shipping delay to each adjacent host

In addition, the system as a whole requires the specification of the turnaround factor vs. delay curve.

The values used for these parameters were determined in the same way as the parameters for the two-node model studies. In particular, the same turnaround factor curve was used (see Figure 3.7, p.111), and λ_B vs. $E(W_B)$ curves were taken from the curves of Figure 4.4 (p.140), with curve CT273L used to represent heavily loaded systems. Local batch input rates were varied in some of the studies; when fixed rates were used throughout a study, the rates were taken from the original data points on the curves cited. The same assumptions concerning job sizes and line speeds were also employed, and all lines were assumed to be symmetric (that is, if node i could ship to node j, then node j could also ship to node i and the transshipping delays would be identical in both cases). In the studies with two-node networks, resource costs used were the same as in the two-node model (see Table 4.3, p. 138), the three-node network studies required alterations to these values. The three-node networks studies were all fully connected and all three links were assumed to be of identical line speeds. Each center's budget was then assumed to increase by the cost of one of the paths. Center budgets were again assumed to be \$.5 million year, so the resource costs shown in Table 5.1 were used.

Table 5.1 Resource Costs for Fully Connected Three Node Networks

Line Speed (all links)	Resource Cost
50 KB	1.024
10.8 KB	1.0156
2 KB	1.0056

Budget per node = $.5 million

Initial runs of the multi-node model converged to a stable transshipping probability matrix within less than 100 iterations. Investigating the convergence properties of the algorithm was not the primary goal of the parameter studies; however, one particular study of a three-node network was run with varying values of y. With the value for x fixed at .01, it was found that for $y \geq .2835$, the algorithm cycles indefinitely through a series of solutions; for $y \leq .2830$, a stable solution was rapidly attained. Details of these runs are given in Figure 5.2. Charges for computations of a few example runs are included in Tables 5.2-5.4; the use of the $E(W_B)$ vs. λ_B curves to approximate the analytical results of the single-node models clearly yields significant savings overall.

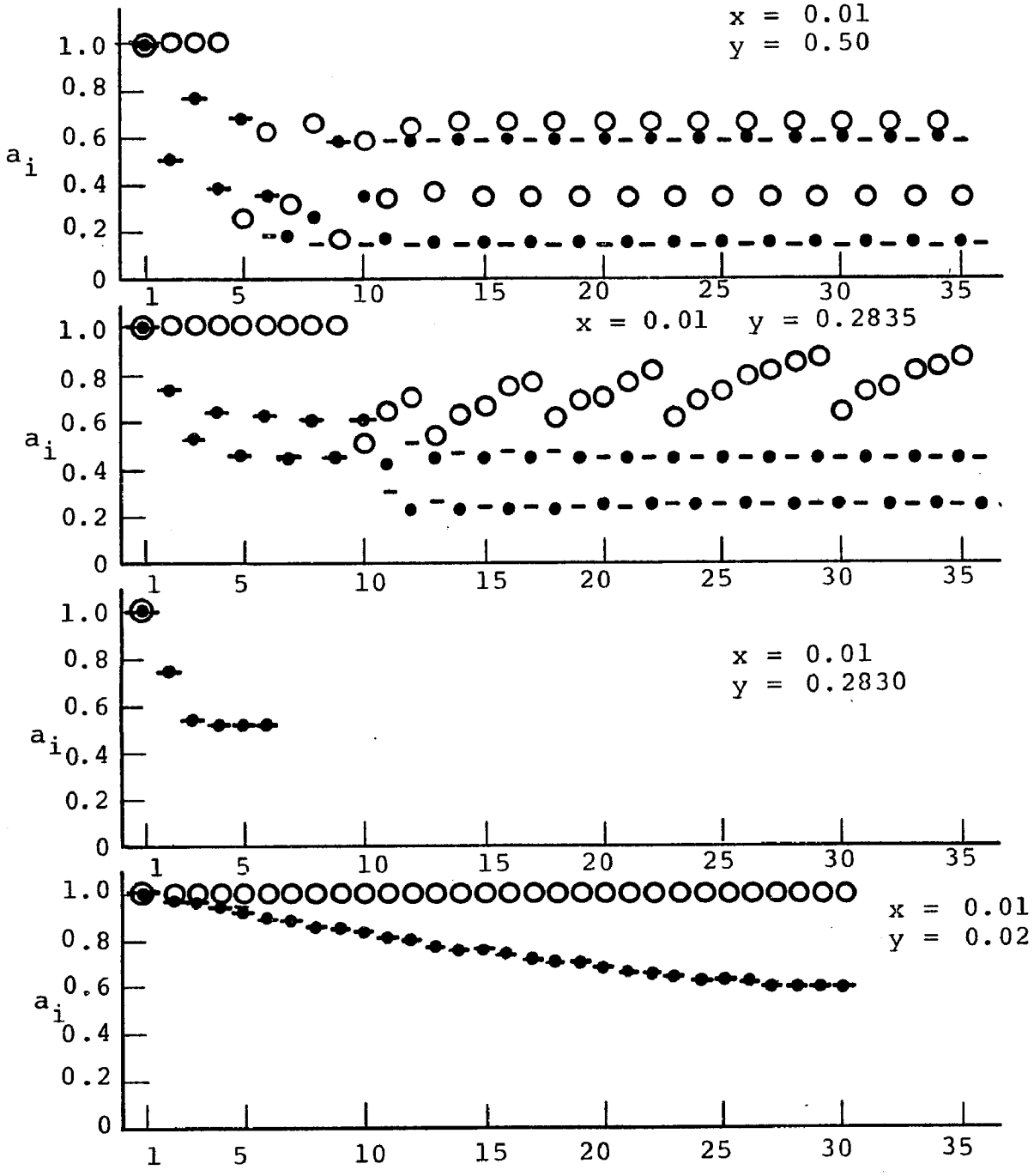


Fig. 5.2 Multi-node Model
Convergence Studies

Table 5.2 Multi-Node Studies -- Group 1

		Node 1	Node 2	Node 3	Total
50KB lines N=28 C=$2	$E(W_B)$	36.89	48.99	48.99	43.92
	P(Trans)	0	.409	.409	.298
	$E(\frac{cost}{100\ jobs})$	.546	.554	.554	.549
10.8KB lines N=19 C=$1.20	$E(W_B)$	29.34	63.64	63.64	53.01
	P(Trans)	0	.291	.291	.212
	$E(\frac{cost}{100\ jobs})$	.543	.562	.562	.553
2KB lines N=12	$E(W_B)$	20.17	166.56	166.56	127.05
	P(Trans)	0	.0688	.0688	.0251
	$E(\frac{cost}{100\ jobs})$	.540	.655	.655	.618
No lines N=2 C=$.60	$E(W_B)$	17.97	252.72	252.72	188.55
	P(Trans)	0	0	0	0
	$E(\frac{cost}{100\ jobs})$	.515	.747	.747	.684

Data for Multi-node Model Studies of 1 Lightly Loaded System and Two Heavily Loaded Systems

Node 1 = lightly loaded

Nodes 2 and 3 = heavily loaded

x=.01, y=.02 in all studies

N= number of iterations before convergence

C= approximate computing cost

Table 5.3 Multi-Node Studies -- Group 2

		Node 1	Node 2	Node 3	Total
50KB lines N=23	$E(W_B)$	22.71	22.60	34.16	26.03
	P(Trans)	0	0	.288/.283	.229
	$E(\frac{cost}{100\ jobs})$	.533	.533	.541	.534
10.8KB lines N=13	$E(W_B)$	20.87	20.81	54.45	33.53
	P(Trans)	0	0	.181/.178	.144
	$E(\frac{cost}{100\ jobs})$	.530	.530	.554	.554
2KB lines N=9	$E(W_B)$	18.54	18.53	164.81	77.36
	P(Trans)	0	0	.0354/.0347	.028
	$E(\frac{cost}{100\ jobs})$	.524	.524	.653	.572
No lines N=2	$E(W_B)$	17.97	17.97	252.72	111.93
	P(Trans)	0	0	0	0
	$E(\frac{cost}{100\ jobs})$	.515	.515	.747	.608

Data for Multi-node Model Studies of 2 Lightly Loaded Systems and one Heavily Loaded System

Nodes 1 and 2 = lightly loaded

Node 3 = heavily loaded

P(Trans) = a/b means P(Trans from 3 to 1)=a and
P(Trans from 3 to 2)=b

x=.02, y=.02 for all studies

N=number of iterations before convergence

Total cost of computing for the four studies was approximately $3.50

Table 5.4 Multi-Node Studies -- Group 3

		Node 1	Node 2	Node 3	Total
λ_{B_2}=.03056					
λ_{B_3}=.05056	$E(W_B)$	37.19	47.77	49.23	42.32
50KB lines	P(Trans)	0	.393	.362/.161	.345
N=38	$E(\frac{cost}{100\ jobs})$	.547	.554	.554	.550
	$E(W_B)$	17.97	70.35	559.33	278.04
No lines	P(Trans)	0	0	0	0
N=2	$E(\frac{cost}{100\ jobs})$	.515	.559	6.43	3.21
λ_{B_2}=.02056					
λ_{B_3}=.06056	$E(W_B)$	37.37	47.70	48.79	44.73
50KB lines	P(Trans)	0	.359	.356/.302	.395
N=40	$E(\frac{cost}{100\ jobs})$	.547	.555	.554	.550
	$E(W_B)$	17.97	41.03	866.44	483.48
No lines	P(Trans)	0	0	0	0
N=2	$E(\frac{cost}{100\ jobs})$	.515	.534	13.25	7.44
λ_{B_2}=.01056					
λ_{B_3}=.07056	$E(W_B)$	37.43	46.96	49.35	45.01
50 KB lines	P(Trans)	0	.272	.356/.302	.445
N=42	$E(\frac{cost}{100\ jobs})$	.547	.556	.554	.550
	$E(W_B)$	17.97	19.55	1173.56	750.06
No lines	P(Trans)	0	0	0	0
N=2	$E(\frac{cost}{100\ jobs})$	.515	.516	20.69	13.29

Data for Multi-node Model Studies of 1 Lightly Loaded System and Two Heavily Loaded Systems with Varying Batch Input Rates

x=.01, y=.015 for all studies

Notation as in Table 5.3, total computing cost approximately $4.60

Studies using the same model parameters for different size state spaces are not necessary for the multi-node models. All of the studies correspond to infinite state spaces -- the state space of each subsystem is infinite, since it corresponds to the single node model. The curves used for $E(W_B)$ vs. λ_B are, of course, generated by single-node model runs which have particular breakpoints and maximum numbers of terminals and server states. Details of these curves were presented in Chapter 2.

The investigations performed fall into two groups: studies of two-node networks and studies of three-node networks. For purposes of comparison, the two-node network studies were patterned after the analyses performed with the two-node model. Although studies of larger networks are easily within the range of the model, three-node networks were chosen as the final group to be studied, since we wished to proceed incrementally in size. These two groups of studies indicate the utility of the model; studies of larger networks would be useful only insofar as the particular network modelled is of interest.

Case Studies

Data from the two-node studies is presented in Figures 5.3-5.5; these studies parallel those of Figures 4.9-

4.11(pp. 150, 152, and 154). As described above, the first set of two-node model studies (Figure 4.8) involved systems with exactly identical parameters. These studies cannot be replicated using the multi-node model because of its more restricted model for the decision maker. Since the two-node model decision maker is aware of the instantaneous queue lengths of the two subsystems, it can decide to which subsystem to send an arrival on the basis of the queue lengths which exist when the job arrives. Thus, even if both subsystems have identical parameters, their instantaneous queue lengths will vary stochastically, and this variance acts as an incentive for the decision maker to transship jobs. In the multi-node system, the knowledge of instantaneous queue lengths is withheld from the decision maker; the transshipping decisions may not depend on the states of the subsystems. Thus, if two systems have the same parameters in the multi-node model, they will also have identical mean waiting times for batch jobs and there will be no incentive to transship jobs, since there can be no decrease in mean waiting time.

The studies presented in Figure 5.3 correspond to the two-node model studies shown in Figure 4.9. The statistics are in the identical format, with the predicted mean system residence for batch jobs shown in the top bar graph, transshipment probabilities in the middle graph, and mean cost per job in the bottom graph. The leftmost study of

Figure 4.9 has not been replicated with the multi-node model since it corresponds to two subsystems with identical parameters. In both sets of studies, the trends are identical: the probability of transshipping jobs, mean delay, and cost per job all increase as the difference in the input rates to the subsystems increases. The differences in the absolute results for the no-communication-lines cases point out a basic difference in the multi-node and two-node models. The multi-node model generally predicts higher mean waiting times and therefore higher costs per job in the no-lines studies. The reason that the two-node model predicts lower mean waiting times is its assumption of a finite capacity batch queue. The state spaces used in the studies of Figure 4.9 were all (2,2,2;2,2,2); in the no-lines cases and particularly when one system is heavily loaded, the finite length batch queue restriction clearly affects the results obtained.

On the other hand, in the studies with 50KB lines, we note that the two-node model predicts a significant level of transshipping even when the two system loads are only slightly different (.02528,.03500), while the multi-node

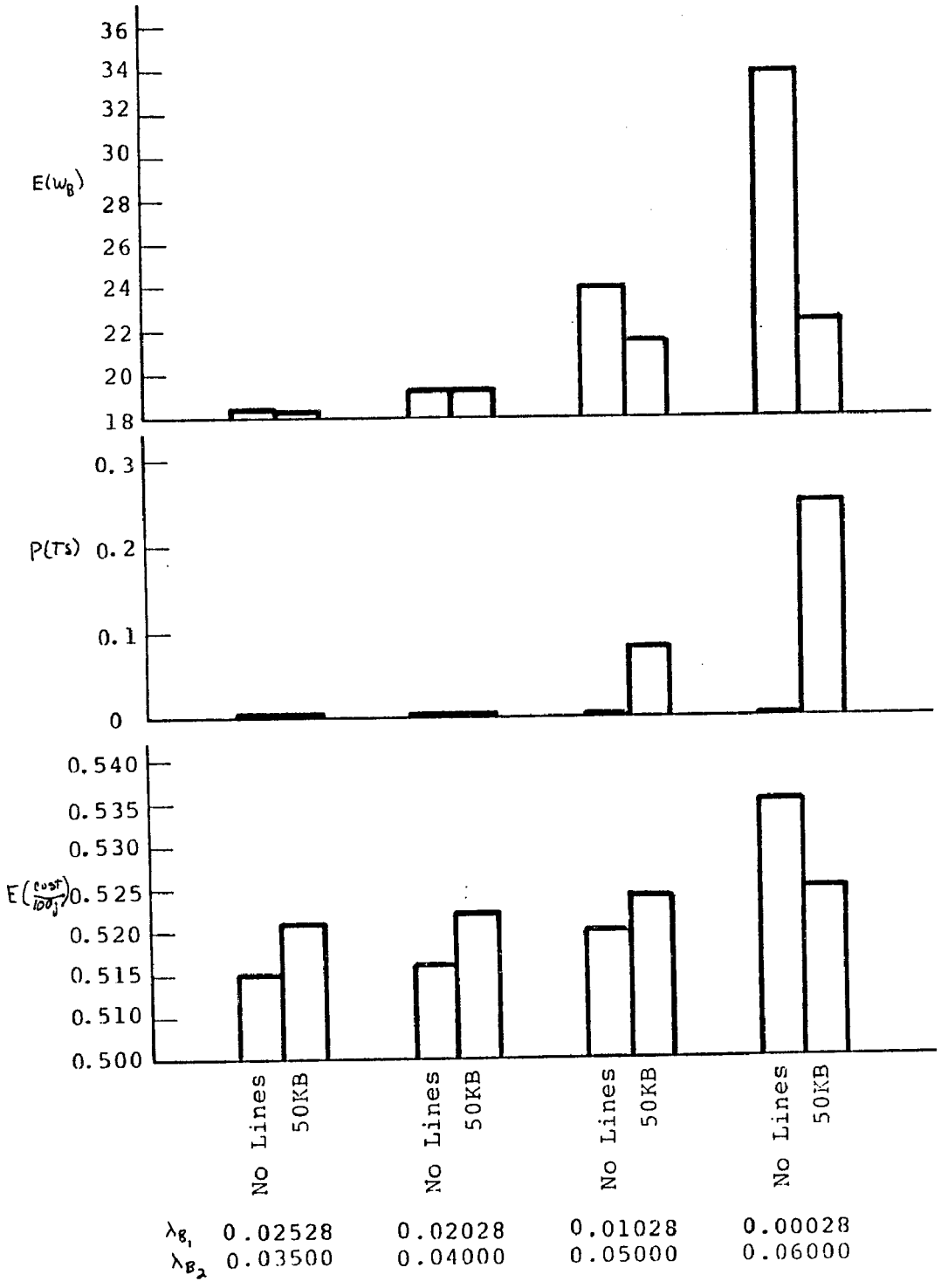


Fig. 5.3 Multi-node Model - Two Lightly Loaded Systems Varying Batch Input Rates

model does not predict any transshipping until the loads are distributed (.01028,.0500). The difference in the decision maker models accounts for this result. As noted above, the decision maker of the two-node model can detect cases where transshipping is profitable even if the systems are identical. The multi-node model produces more conservative results in this respect; it can only predict transshipping when a net flow from one system to the other will result. One final point to be noted in these studies is that the expected cost per job in the 50KB lines vs. no-lines case actually favors the implementation of a network in the multi-node model case where the loads are distributed (.00028,.0600). The reason a similar result is not obtained in the two-node model is again the truncation of the batch queue in the no-sharing case, which leads to a shorter mean queue length and thus shorter waiting-time and lower cost-per-job estimates than the multi-node model predicts.

Different resource costs at the two hosts were assumed in the studies of Figure 5.4. As in the two-node model studies of Figure 4.10, both systems were assumed to be lightly loaded, and resource costs of 1.0 at one subsystem and 1.195 at the other were used. Studies were then conducted for different line speeds. Again, the multi-node model predicts longer absolute delays and thus higher

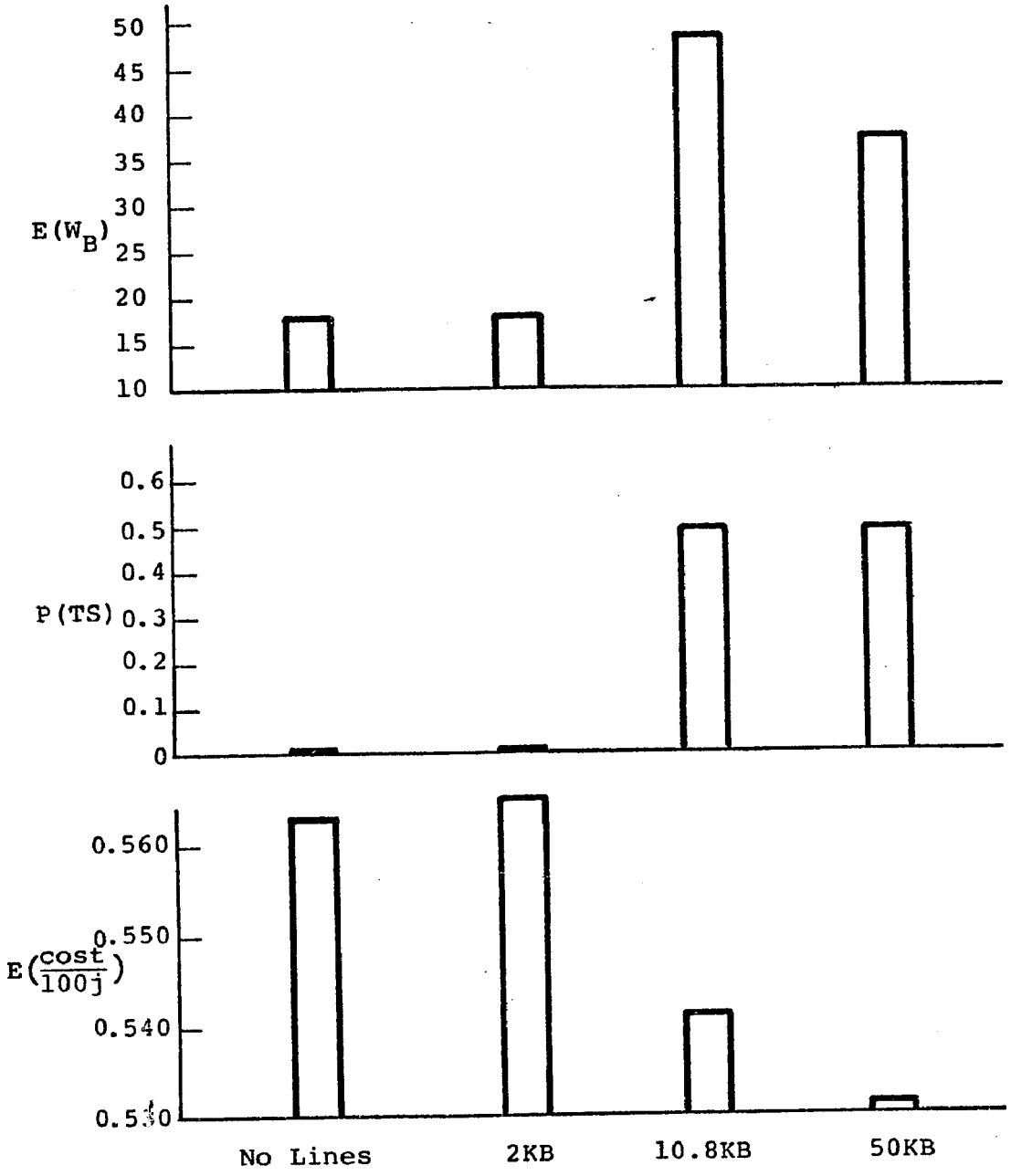


Fig. 5.4 Multi-node Model - Two Lightly Loaded Systems Different Resource Costs

absolute costs per job; however, the transshipment probabilities are quite close in both cases. The reason that both models give the same prediction is that, because of the considerable difference in resource costs, the two-node model will in fact ship virtually all jobs from the more expensive subsystem to the cheaper one, regardless of the queue lengths involved. Thus its behavior is quite similar to the simpler multi-node model decision maker and the analysis produces similar results. In both sets of studies, the 2KB lines introduce too much delay for transshipping to be profitable. The 10.8KB and 50KB lines both generate maximum transshipping, and the reduction in delay with 50KB lines more than offsets their higher cost.

The finite batch queue length assumption in the two-node model also shows its effect in the studies shown in Figure 4.11, which correspond to those of Figure 5.6. In these runs, one subsystem was assumed to be lightly loaded and the other to be heavily loaded. Again, the trends in expected waiting time and transshipment probabilities are the same. Note, however, that the multi-node model predicts some transshipping (and hence some reduction of waiting time and expected cost per job) in the 2KB line test, while the two-node model does not. Again, it appears that the

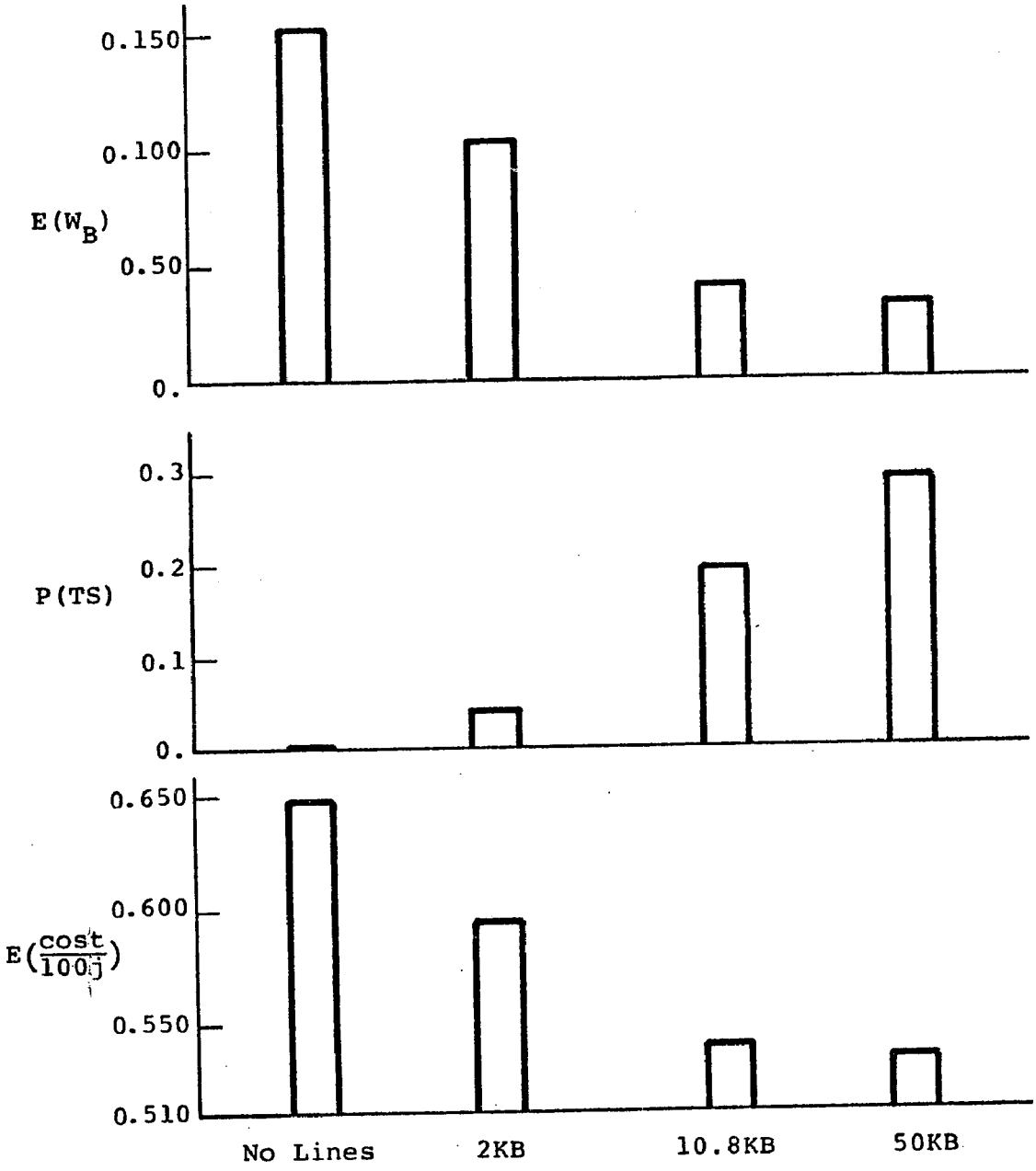


Fig. 5.5 Multi-node Model - Lightly Loaded and Heavily Loaded System

restriction on the batch queue length in the two-node model eliminated some cases in which transshipping of jobs would be advantageous. The drop in expected cost between the no-lines and 2KB cases was mentioned in Chapter 4 as the result of the difference in the state space sizes used; the three rightmost studies in Figure 4.11 are represented by the single 50KB multi-node model study, since they vary only in state space size. Both models predict a drop in cost per job if 50KB lines are used instead of 10.8KB lines and the 50KB case in fact generates the lowest cost per job in both models if the state space size of the 2KB two-node result is taken into account.

The match between the relative predictions of the two models, then, seems quite good. The multi-node model generally predicts longer absolute mean waiting times than the two-node model, because of the finite batch queue assumption in the two-node model and because the multi-node model generally predicts lower transshipment probabilities. These lower predictions are due to the restricted knowledge of the multi-node model decision maker about the state of the subsystems. Thus the absolute differences can be explained by the different assumptions made by the two models; the multi-node model assumptions are more realistic in terms of the batch queue length, but less realistic in terms of the decision maker's knowledge.

In order to investigate the properties of the multi-node model in larger networks, three sets of studies were performed using a three-node network. The network was assumed to be fully connected, with all links having the same speeds. These assumptions were made in order to limit the number of parameters to be varied in successive experiments. Since there is no data available to confirm the results of these experiments, they must be viewed as feasibility studies for a projected network rather than as direct models for existing situations. The validity of the model behavior can only be judged by its overall consistency and reasonableness.

Figures 5.6 and 5.7 illustrate studies in which the line speeds throughout the network are varied while the other parameters of the model are held constant. In Figure 5.6, two of the three nodes are assumed to be heavily loaded while the third node is lightly loaded; Figure 5.7 shows results for the converse -- two lightly loaded systems and one heavily loaded system. The results are quite similar in both cases: even with 2KB lines, the cost per job is reduced from the no-lines case, and successive increases in line speeds show increases in transshipping probabilities and

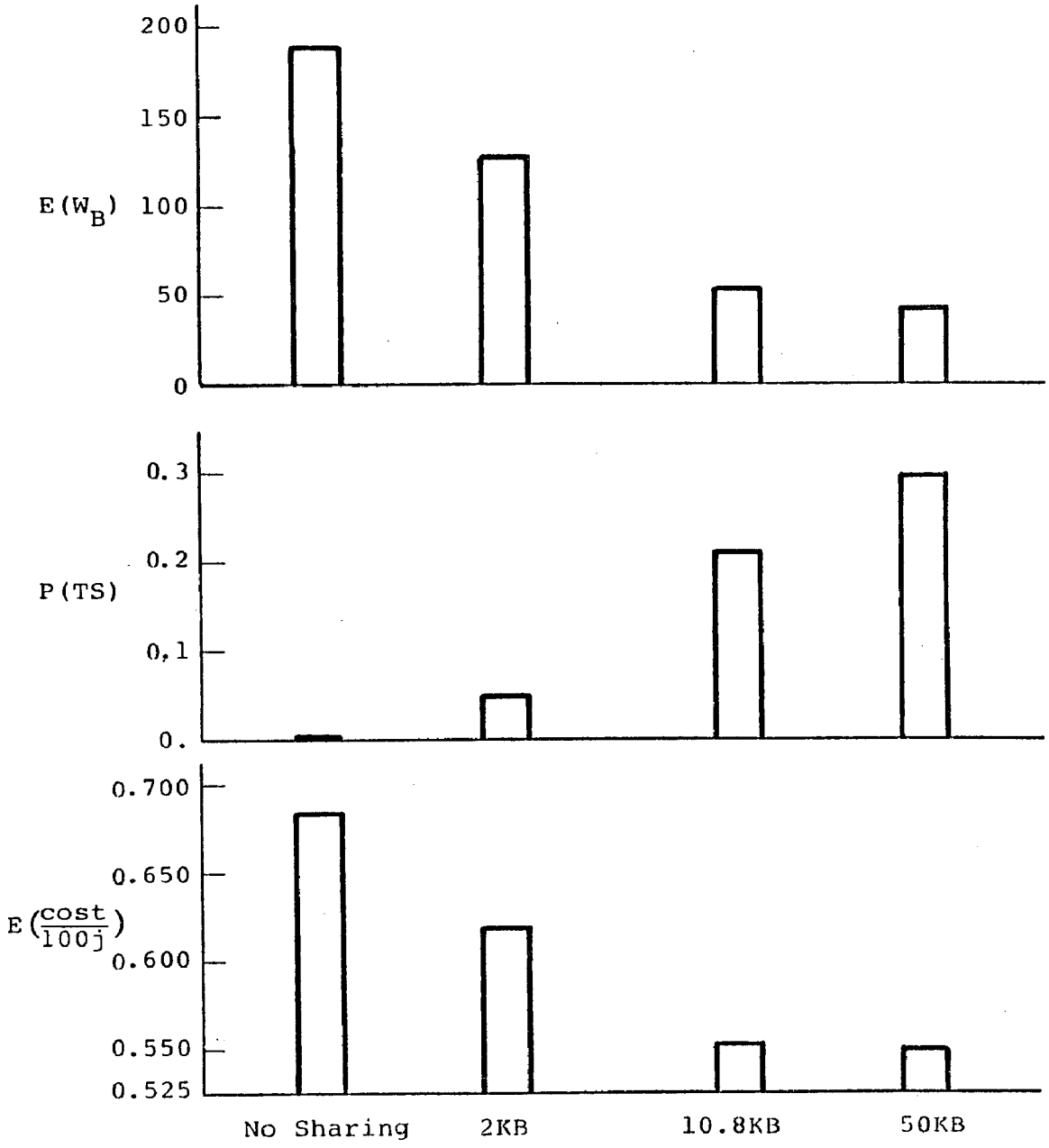


Fig. 5.6 Multi-node Model - 1 Lightly Loaded and 2 Heavily Loaded Systems

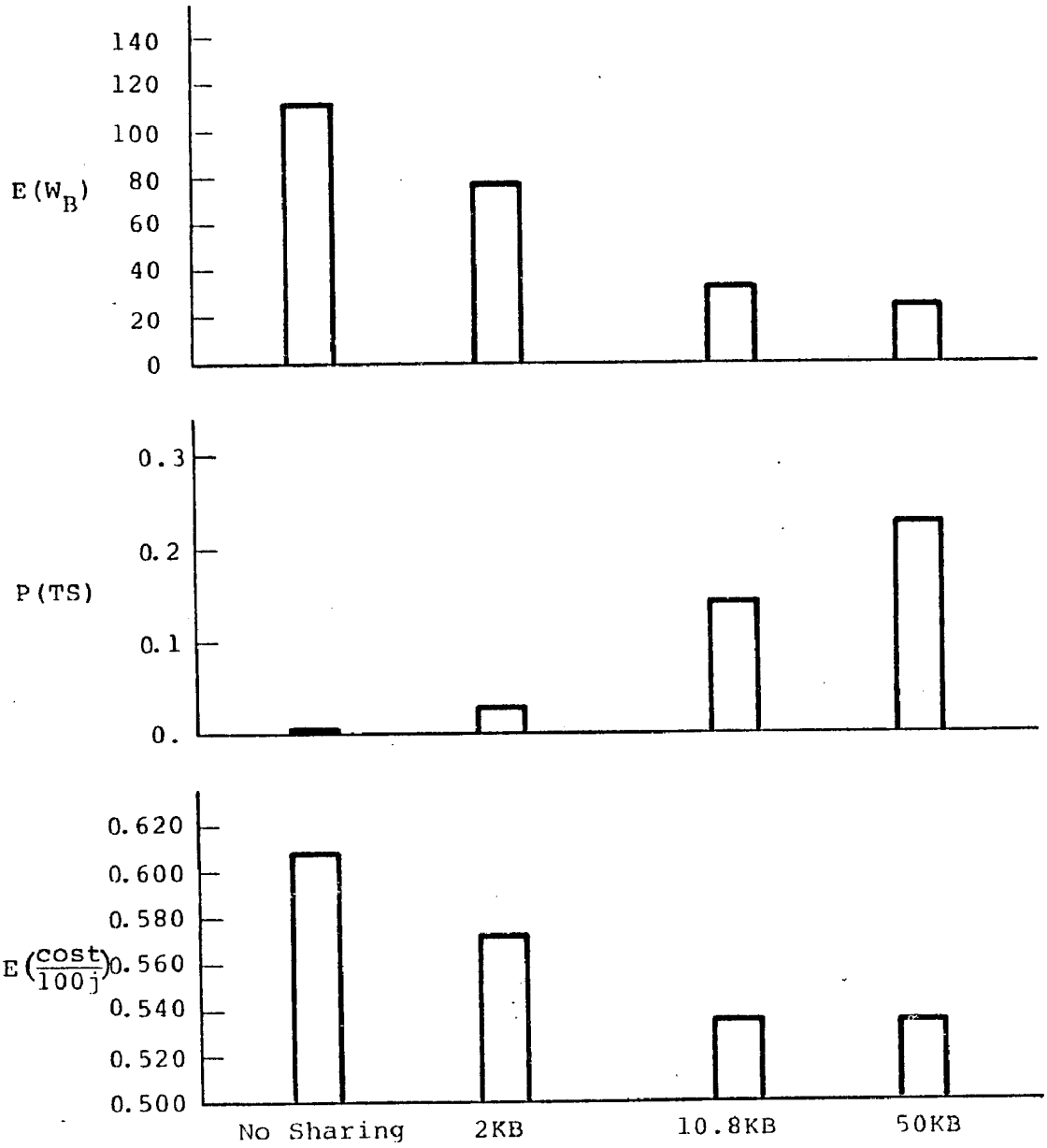


Fig. 5.7 Multi-node Model - 1 Lightly Loaded and 2 Heavily Loaded Systems

decreases in expected delay and cost per job. The decrease in cost per job between 10.8KB and 50KB lines, however, is minimal. In light of the results shown in Figure 5.6 for a lightly loaded and heavily loaded system, these results are to be expected. Tables 5.2 and 5.3 show the final statistics for each subsystem in these two sets of studies.

The final set of studies involving a three-node network is shown in Figure 5.8. In these runs, one system was lightly loaded and two were heavily loaded, but the batch input rates of the two heavily loaded systems were summed and then re-apportioned between the two systems in varying ratios. Three different ratios were used, and each study was conducted for the no-lines case and the case of 50KB lines throughout the network. The results are similar to those seen in the two-node experiments of Figure 5.3: in the no-lines cases, as the difference in the local batch input rates to the heavily loaded systems increases, so do the expected delay and mean cost per job. In the 50KB lines cases the same effects are observed, but the increases are much smaller; only the transshipping probability increases noticeably. Again, this result shows the sort of load leveling we would expect from a network. The detailed statistics for these runs are given in Table 5.4.

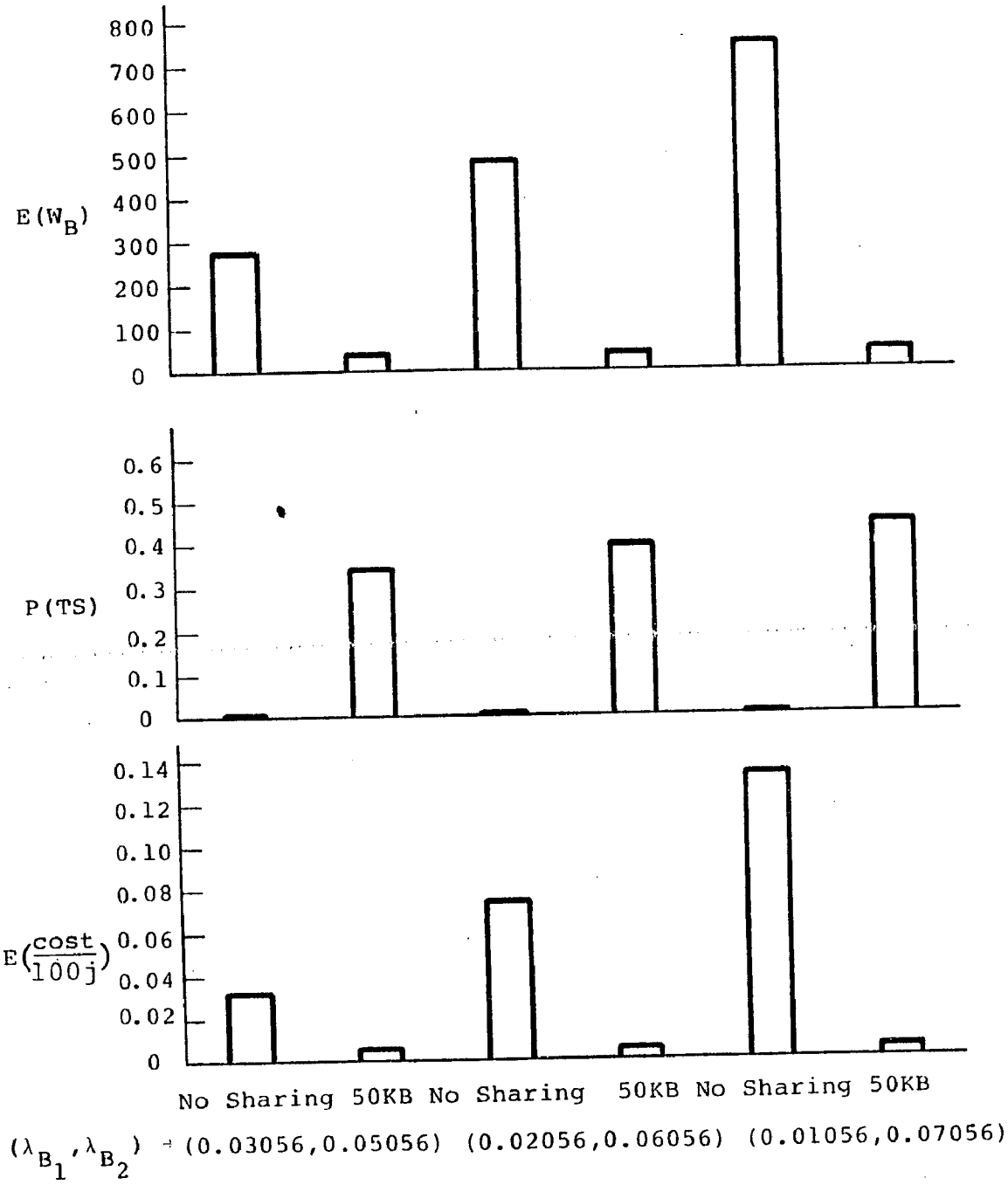


Fig. 5.8 Multi-node Model - 1 Lightly Loaded and 2 Heavily Loaded Systems with Varying Input Rates

The results of the three-node studies thus contain no surprises and seem to follow directly from the earlier parameter studies of two-node networks. One cannot, of course, generalize to a network of n nodes from these examples; nevertheless, the multi-node model has been shown to function reasonably in these cases and no alteration of its structure or algorithm is required to investigate larger networks.

Chapter 6 - Conclusions

6.1 Conclusions

The purpose of this research has been to investigate under what conditions load sharing of batch jobs is feasible in a computer network. To this end, two different network models have been developed and analyzed. Parameter studies of these models indicate that load sharing can be a useful technique under conditions which exist in current computer systems. The greatest incentive for load sharing is a significant difference in resource costs at different hosts in the network. Secondarily, if two subsystems operate under different load parameters (one is near saturation and the other is not) load sharing will be feasible. Finally, even if two subsystems are both lightly loaded, load sharing can reduce the average total cost per job if the hosts have widely different arrival rates for batch jobs.

An additional benefit of load sharing has been demonstrated in the parameter studies. That is, the average waiting time for a job remains relatively constant as arrival rates vary. This load levelling effect has been projected as one of the benefits of load sharing.

These results have been derived from the studies of network models presented in Chapters 4 and 5. The first model developed (Chapter 4) was a finite state Markov model and was used to study a network of two nodes. Each subsystem was modelled with finite capacity batch and terminal queues and a single server using an endogenous priority discipline. A cost function, including the effects of host and network resource consumption and projected waiting time costs (Chapter 3) was used to identify the least cost system for executing a job. Projected waiting time was a function of system state. Transshipping decisions were made on the basis of these costs. Computational considerations limited the queue sizes to small finite values; consequently parameter studies were restricted to cases in which at least one subsystem was lightly loaded. Estimates of transshipping delays and resource costs were based on data presented in Chapter 3.

A more comprehensive but less detailed network model was presented in Chapter 5. The subsystem model used was an infinite state Markov model, similar to that of Chapter 4, but infinite length batch queues were permitted. The cost function of Chapter 4 was altered to make projected waiting time independent of system state, and jobs were routed randomly according to a probability matrix. This matrix was modified iteratively by an algorithm which attempted to reduce the mean cost of executing jobs arriving at each

subsystem. With the use of some computational approximations, this model was used to analyze networks (at a lower cost than the model of Chapter 4).

When operated with similar parameters on two-node networks, the two network models yielded similar predictions, as shown in Chapter 5. Studies of three-node networks generated results comparable to the two-node cases. The longer absolute waiting times predicted by the second model are attributable to its allowance of potentially infinite batch queues. This feature also accounts for the fact that the second model predicted that load sharing would reduce the mean cost per job in a few cases where the first model predicted an increase.

The subsystem models used in both network models were derived from the single host model presented in Chapter 2. This model included a finite capacity terminal queue, infinite capacity batch queue, and a server operating under an endogenous priority discipline designed to give priority to interactive service while maintaining a minimum level of batch throughput. A technique due to Wallace was used to analyze the model, and the mean waiting time for batch jobs was shown to obey Little's theorem. In Chapter 4, Wallace's technique was shown to be insufficient to analyze a network consisting of two copies of the single node model.

6.2 Future Research

The models investigated in this dissertation have consisted of networks of queues. Queueing networks arise in many applications as well as computer networks and will undoubtedly be a source of continuing research efforts. Models for decision processes in such networks also deserve further investigation.

One of the central assumptions in the queueing analyses presented above is that the system studied exists in a steady state. This assumption may not be satisfied by a heavily loaded system, as indicated by the data in Chapter 2. It is possible to make studies with the network models under the assumption that a subsystem of the network exists in a given state for some time and then changes to a different state. (A different subsystem state might be represented by changes in arrival and service rates.) Estimations of network behavior could then be made on the basis of this cycling through different "steady" states.

The stability of the models developed in this work has not been a major concern since all models appeared to be stable within the regions of interest. Explicit conditions for the positive recurrence of the Markov chains used in the models could be derived. The convergence of the algorithm to modify the transshipping probabilities in the multi-node

model could also be examined in detail. With special considerations for the discrete nature of the flows and the finite size of alterations in probabilities, network flow theory may be applied to this problem.

In a more general way, this work indicates a need for research on systems as they appear to the user. Much of the current modelling of computer systems is done at the microscopic level. This emphasis is justifiable in that concrete results may be easier to obtain by modelling a small portion of a complex system instead of viewing the entire system at once. At the same time, however, a global view of system behavior is necessary in order to better understand the ultimate value of the system to users.

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 Sys. J.	IBM Systems Journal
IEEE	Institute of Electrical and Electronics Engineers
JACM	Journal of the Association for Computing Machinery
NCC	National Computer Conference
ORSA	Operations Research Society of America
Proc.	Proceedings
SJCC	AFIPS Spring Joint Computer Conference
Trans.	Transactions

A1 Abramson, N. The ALOHA System -- Another Alternative for Computer Communications. AFIPS Conf. Proc. 37 (FJCC 1970), 281-285.

A2 Adiri, I. A Dynamic Time Sharing Priority Queue. JACM 18, 4 (October 1971), 603-610.

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

A4 Adiri, I., Hofri, M., and Yadin, M. A Multiprogramming Queue. JACM 20, 4 (October 1973), 589-603.

A5 Alexander, M. T. Time-Sharing System Supervisor Programs. In Advanced Topics in Systems Programming, University of Michigan Engineering Summer Conferences 7016 (June 1970).

A6 Arden, B. and Boettner, D. Measurement and Performance of a Multiprogramming System. Proc. 2nd ACM Symposium on Operating System Principles (October 1969) 130-146.

A7 Avi-Itzhak, B. and Heyman, D. P. Approximate Queueing Models for Multiprogramming Computer Systems. Technical Memorandum Mm-7k-1713, Bell Telephone Laboratories, Holmdel, New Jersey, 1971.

B1 Blatny, J., Clark, S. R., and Rourke, T. A. On the Optimization of Performance of Time Sharing Systems by Simulation. CACM 15, 6 (June 1972), 411-420.

B2 Bowden, E. K., Sr. Network Computer Analysis. Report UIUCDCS-R72-505, Computer Science Department, University of Illinois, December 1972.

B3 Bowden, E. K., Sr. and Barr, W. J. Cost Effective Priority Assignment in Network Computers. Report UIUCDCS-R72-509, Department of Computer Science, University of Illinois, March 1972.

B4 Buzen, J. P. Analysis of System Bottlenecks Using a Queueing Network Model. SIGOPS Workshop on System Performance Evaluation, [illegible] 1971, 82-103.

C1 Chandy, M., Herzog, [illegible], and Woo, L. Parametric Analysis of Queueing Network [illegible]s. IBM Research Report RC4730, Yorktown Heights, [illegible]ary 1974.

C2 Chang, W. Sin[illegible]er Queueing Processes in Computing Systems. IBM [illegible] 1 (1970), 36-71.

C3 Cocanower, A. [illegible], Fisher, W., Gerstenberger, W. S., Read, B. The Communications Computer Operating System - The Initial Design. MERIT Computer Network, Ann Arbor, Michigan, August 1970.

C4 Cole, G. D. Performance Measurements on the ARPA Computer Network. ACM/IEEE 2nd Symposium on Problems in the Optimization of Data Communication Systems, October 1971, 39-45.

C5 Coffman, E. G., Jr. and Denning, P. J. Operating Systems Theory. Prentice-Hall, New Jersey, 1973.

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

C7 Coffman, E. G., Muntz, R. R., and Trotter, H. Waiting Time Distribution for Processor Sharing Systems. JACM 17, 1 (January 1970), 123-130.

C8 Conway, R. W., Maxwell, W. L., and Miller, L. W. Theory of Scheduling. Addison-Wesley, Reading, Massachusetts, 1967.

C9 Cox, D. R. and Smith W. L. Queues. Wiley, New York, 1961.

C10 Crocker, S., Heafner, J., Metcalfe, J., and Postel, J.

Function Oriented Protocols for the ARPA Network. AFIPS Conf. Proc. 40 (SJCC 1972), 271-280.

D1 Davies, D. W. Control of Congestion in Packet Switching Networks. ACM/IEEE 2nd Symposium on Problems in the Optimization of Data Communications Systems, October 1971, 46-49.

E1 Estrin, G. and Kleinrock, L. Measures, Models, and Measurements for Time Shared Computer Utilities. Proc. ACM 1967 National Conf., 85-96.

F1 Farber, D. J. Networks: An Introduction. Datamation 18, 4 (April 1972), 36-39.

F2 Feller, W. An Introduction to Probability Theory and its Applications, Vol I. Wiley, New York, 1957.

F3 Ferdinand, A. E. An Analysis of the Machine Interference Model. IBM Sys J 10, (1971), 129-142.

F4 Fletcher, J. G. The Octopus Computer Network. Datamation 19, 4 (April 1973), 58-63.

F5 Frank, H., Frisch, I. T., and Chou, W. Topological Considerations in the Design of the ARPA Computer Network. AFIPS Conf. Proc. 36 (SJCC 1970), 581-587.

F6 Frank, H., Kahn, R. E., and Kleinrock, L. Computer Communication Network Design. AFIPS Conf. Proc. 40 (SJCC 1972), 255-270.

F7 Fuchs, E. and Jackson, P. E. Estimates of Distributions of Random Variables for Certain Computer Communications Models. CACM 13, 12 (December 1970), 752-757.

G1 Gaver, D. P. Probability Models for Multiprogramming Computer Systems. JACM 14, 3 (July 1967), 423-438.

G2 Gaver, D. P. and Shedler, G. S. Multiprogramming System Performance via Diffusion Approximations. IBM Research Report RJ938, San Jose, March 1972.

G3 Gerla, M. The Design of Store and Forward (S/F) Networks for Computer Communications. Report UCLA-ENG-7319, School of Engineering and Applied Science, UCLA, January 1973 (PhD Diss.).

G4 Gordon, W. J. and Newell, G. F. Closed Queueing Systems with Expontial Servers. Operations Research 15, 2 (March 1967), 254-265.

G5 Greenberger, M. The Priority Problem and Computer Time Sharing. Management Science 12, 11 (July 1966), 888-906.

H1 Hanssman, F., Kistler, W., and Schultz, H. Modelling for Computer Center Planning. IBM Sys J 10, 4 (1971), 305-324.

H2 Hayes, D. F. and Sherman, D. N. Traffic and Delay in a Circular Network. ACM/IEEE 2nd Symposium on Problems in the Optimization of Data Communication Systems, October 1971, 102-107.

H3 Heart, F. E., Kahn, R. E., Ornstein, S. M., Crowther, W. R., and Walden, D. C. The Interface Message Processor for the ARPA Computer Network. AFIPS Conf. Proc. 36 (SJCC 1970), 550-567.

H4 Hunter, B. D. Fortran Version of RQA-1, Memo to Project 011312 File, July 1972, CICE Dept., University of Michigan, Ann Arbor.

J1 Jaiswal, N. K. Priority Queues. Academic Press, New York, 1968.

J2 Jackson, J. R. Jobshop like Queueing Systems. Management Science 10 (1963), 131-142.

J3 Jackson, P.E. and Stubbs, C. D. A Study of Multi-Access Computer Communications. AFIPS Conf. Proc. 34 (SJCC 1969), 491-504.

K1 Kleinrock, L. Communications Nets: Stochastic Message Flow and Delay. McGraw Hill, New York, 1964.

K2 Kleinrock, L. Analytic and Simulation Methods in Computer Network Design. AFIPS Conf. Proc. 36 (SJCC 1970), 569-579.

K3 Kleinrock, L. and Naylor, W. E. On Measured Behavior of the ARPA Network. Proc. NCC 1974, 262-280.

K4 Kleinrock, L. Personal communication, November 16, 1973.

K5 Kleinrock, L. and Muntz, R. R. Processor Sharing Queueing Models of Mixed Scheduling Disciplines for Time Shared Systems. JACM 19, (July 1972), 464-482.

K6 Kobayashi, H. Application of the Diffusion Approximation to Queueing Networks I: Equilibrium Queue Distributions. JACM 21, 2 (April 1974), 316-328.

K7 Kobayashi, H. Application of the Diffusion Approximation to Queueing Networks II: Nonequilibrium Distributions and Applications to Computer Modelling. JACM 21, 3 (July 1974), 459-469.

K8 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. and Shedler, G. S. A Cyclic Queue Model of System Overhead in Multiprogrammed Operating Systems. JACM 18, 2 (April 1971), 199-220.

L2 Little, J. A Proof of the Queueing Formula L=λW. Operations Research 9, 3 (1961), 383-387.

M1 McKinney, J. M. A Survey of Analytical Time-Sharing Models. Computing Surveys 1, 2 (June 1969), 105-116.

M2 MERIT Memo ND-70-159. A Flow Model. MERIT Computer Network, Ann Arbor, Michigan, July 1970.

M3 MERIT Memo ND-72-239. Batch Job Submission. MERIT Computer Network, Ann Arbor, Michigan, July 1972.

M4 Moore, C. G. Network Models for Large Scale Time Sharing Systems. Technical Report No. 71-1, Dept. of Industrial Engineering, University of Michigan, April 1971.

M5 MTS: University of Michigan Terminal System. Vol. 1 to 13, University of Michigan Camputing Center.

O1 Oldehoeft, A. E. and Halstead, M. H. Maximum Computing Power and Cost Factors in the Centralization Process. CACM 15, 2 (February 1972), 94-96.

O2 Omahen, K. Analytic Models of Multiple Resource Systems. PhD Dissertation, University of Chicago, June 1973.

O3 Ornstein, S. M., Heart, F. E., Crowther, W. R., Rising, H. K., Russell, S. E., and Michel, A. .The Terminal IMP. AFIPS Conf. Proc. 40 (SJCC 1972), 243-254.

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

P2 Peterson, J. J. and Veit, S. Catalog of Network Features. MITRE Report WP9695, Washington, D. C., March 1971

P3 Prabhu, N. U. Queues and Inventories. Wiley, New York, 1965.

R1 Rettus, R. C. and Smith, R. A. Accounting Control of Data Processing. IBM Sys J 11, 1 (1972), 74-92.

R2 Roberts, L. G. and Wessler, B. D. Computer Network Development to Achieve Resource Sharing. AFIPS Conf.

Proc. 36, (SJCC 1970), 543-549.

R3 Rustin, R., Ed. Computer Networks. Courant Computer Science Symposium 3, Prentice Hall, New Jersey, 1972.

S1 Schrage, L. E. The Queue M/G/1 with Feedback to Lower Priority Queues. Management Science 13 (1967), 466-474.

S2 Schultz, G. D. A Stochastic Model for Message Assembly with a Comparison of Block Assignment Strategies. JACM 19, 3 (July 1972), 483-495.

S3 Shedler, G. S. A Queueing Model of a Multiprogrammed Computer with a Two-Level Storage System. CACM 16, 1 (January 1973), 3-10.

S4 Storey, R. Personal Communication, April 1974.

S5 Streeter, D. N. Cost-Benefit Evaluation of Scientific Computing Services. IBM Sys J 11, 3 (1972), 219-233.

S6 Streeter, D. N. Centralization or Dispersion of Computing Facilities. IBM Sys J 12, 3 (1973), 283-301.

S7 Swoyer, V. H. and Armstrong, M. F. Timing and Charging Computer Use in a Multiprogrammed Environment. Technical Report 2.1-69, University of Rochester Computing Center, 1969.

S8 Swoyer, V. H. and Armstrong, M. F. Analyzing the Operation of a Large-Scale Computer System. Presented at ORSA National Conference, New Orleans, 1972.

T1 Takacs, L. Introduction to the Theory of Queues. Oxford University Press, New York, 1962.

W1 Walden, D. C. A System for Inter-Process Communication in a Resource Sharing Network. CACM 15, 4 (April 1972), 221-230.

W2 Wallace, V. L. The Solution of Quasi Birth and Death Processes Arising from Multiple Access Computer Systems. PhD Dissertation, University of Michigan, 1969.

W3 Wallace, V. L. and Rosenberg, R. S. RQA-1, The Recursive Queue Analyzer. Technical Report No. 2, Systems Engineering Laboratory, Dept. of Electrical Engineering, University of Michigan, 1966.

W4 Wiorkowski, G. and Wiorkowski, J. A Cost Allocation Model. Datamation 19, 8 (August 1973), 60-65.

W5 Wyszewianski, R. J. Feedback Queues in the Modelling of Computer Systems: A Survey. Technical Report No. 74-1,

Dept. of Industrial and Operations Engineering, University of Michigan, May 1974.

Y1 Yule, A. Human Response Times in a Graphic Environment. Computer Bulletin 16, 6 (June 1972), 304-305.