Yale University
Department of Computer Science

Adaptive Load Sharing in the Presence of Delays

Ravi Mirchandaney

YALEU/DCS/TR-597
January 1988

This work has been supported in part by the National Science Foundation under grant ECS-8406402 and by RADC under contract RI-44896X.

Abstract

ADAPTIVE LOAD SHARING IN THE PRESENCE OF DELAYS

September 1987

Ravi Mirchandaney

B.E., University of Bombay, India
M.S., Ph.D., University of Massachusetts

Directed by: Professor John A. Stankovic

In this dissertation, we study the problem of load sharing in distributed computer systems. The underlying system consists of a number of autonomous host computers (nodes) interconnected by a communication network. Jobs arrive at each node according to some arrival process. The jobs may either be processed locally or at a remote node, after being transferred through the communication network. Significant delays are encountered during job transfers and state updates. While load balancing has been an active research area for some time, there has been very little work done that specifically addresses the problem of delays. Two important consequences of these delays are the following: The first is that the delays in job transfers can potentially increase the average system response time. The second important effect of delays pertains to the quality of remote state information. This information can become out of date because of delays, resulting in incorrect decisions and hence poor performance.

We consider a class of load sharing policies that use thresholds at each node in order to make job transfer decisions. These policies are dynamic because they gather state information at decision time, and decentralized because each node is equal to every other node and there exists no central scheduler. Initially, the study is restricted to homogeneous systems, i.e., the nodes are identical as regards job arrival rates and processing speeds. We formulate analytical models of the load sharing algorithms under the above conditions. These models are solved using the Matrix-Geometric solution technique. The queueing models are then extended to study load sharing in heterogeneous systems where the arrival rates and/or processing speeds of the nodes may differ. The analytical solutions are valid over a very large range of system parameters. From these solutions, we have been able to study the effects of various important parameters on load sharing.

Next, we study the applicability Entropy Minimax, an Information Theoretic estimation technique to determine the operating threshold of the policies as a function of job transfer delay and load. Finally, we consider the problem of load sharing in systems where the arrival rates of jobs may change over time. We study the applicability of simple techniques to estimate the values of the changed parameters (e.g., arrival rate, load, etc), and develop simple algorithms to adapt the control policy in response to these changes.

Table of Contents

List of Tables

List of Figures

Chapter 1

INTRODUCTION

Here byggynneth the Book of the tales of Canterbury
Chaucer

1.1 Problem Statement

Job and task scheduling are integral functions of any distributed operating system. The boundary between what is called a job versus what is a task is not always very precise. However, the general rule of thumb is that a job is the initial entity of work designated by the user, for example, the 3-D interpretation of an object by an image understanding system. A task is a smaller entity of work, for example, recognition of the object by the image understanding system may involve several tasks like pattern recognition, feature extraction, semantic interpretation and so on. Each of these tasks could, in turn, consist of several subtasks. Actual implementations of the scheduling function normally consist of the job and task schedulers at each node in the system cooperating to improve the performance of the distributed system as a whole.

In this study, we focus upon one aspect of distributed operating system design, namely network-wide job scheduling. This is often referred to as load balancing or load sharing. The system under consideration is assumed to be comprised of a number of autonomous nodes (host computers) interconnected by a communication medium. We make no specific assumptions regarding the nature of this medium, except that each node is able to communicate with every other node in the system within a finite time interval. The actual implementation of the network could be in the form of a local area network or a store-and-forward network.

It is assumed that local jobs arrive at each node according to some arrival process. A job can be executed either locally, i.e., at its point of arrival in the system, or it can be transferred to some other node in the system via the communication network. Thus, nodes are assumed to be identical in their functional capabilities. However, nodes may not necessarily possess identical processing speeds. If a job is transferred to a remote node for processing, it is assumed that it incurs a delay during the transfer. This delay is due to the processing involved in the protocols at the sender and destination nodes, as well as the delay in physical transmission of the messages over the network. The remote execution of a job normally requires transmission of the results of the computation back to the original node. In practice, delays will also occur in this process.

The reason that jobs need to be transferred to remote nodes for execution is the following: Under certain assumptions regarding arrival and departure processes, it can be shown that the instantaneous load at a node undergoes tremendous fluctuations. One major consequence of this fact is that even if the system is homogeneous (i.e., the time averaged arrival rate of local jobs is identical at each node and the nodes have the same processing speeds), there exists a very high probability that some nodes will be idle while others have waiting jobs at the same time [LIVN82].

It is the function of the job scheduler at each node to recognize such conditions (which are obviously detrimental to performance) and try to rectify them by transferring jobs from very busy nodes to less busy ones, in the hope of reducing the average waiting time of jobs. While at times we refer to the algorithm used by the scheduler as the load balancing policy, we are less concerned with the specific act of load balancing than we are with load sharing. The distinction between these two concepts is not always very clear but we choose to make the following point: Traditionally, *load balancing* has been concerned with balancing the load between the nodes over an interval of time. *Load sharing* however, is concerned with smoothing out the instantaneous variations in load, with no particular emphasis placed upon balancing the load between the nodes over any interval of time.

As is common with this type of study, the main metric by which the performance of any algorithm will be judged is the average response time of jobs for the entire system. Thus, it behooves any good scheduling policy to be fair and not improve the performance of jobs at a particular node by subjecting the rest of the network to harmful decisions, if by that policy, the overall system response time is degraded as a consequence.

There are various schemes used to classify load sharing policies. A reasonable method is based upon the kind of state information used by the policy; namely, is the policy *static* as opposed to being *dynamic.* Static load sharing policies generally utilize information about the average characteristics of the system, e.g., the utilization of the nodes over a certain window of time. Dynamic load sharing policies are generally designed to react to the current state of the system, for example the queue lengths of the nodes at any instant of time. Various studies have shown that dynamic policies are, in general, able to achieve better performance over a large range of system parameters, than are static policies. However, because most of these policies very actively gather state information, they tend to generate higher system overheads than do static policies. Further, because of the very nature of distributed systems, the state information that is so critical to dynamic policies can be out of date by the time the decisions are effected. Thus, the system designer needs to keep these tradeoffs in mind before selecting a particular policy for implementation.

Load sharing policies can also be separated on the basis of the type of control strategy utilized; namely is the control policy *centralized* as opposed to being *decentralized.* In a centralized policy, a designated node collects relevant state information about the entire network and on that basis, it schedules jobs to particular nodes. Decentralized policies, on the other hand, do not possess the concept of a central scheduler for the entire system. In such policies, the scheduler at each node is free to take any action for jobs that arrive at that node. Each of the above schemes has its advantages and disadvantages: For example, centralized schemes are inherently unreliable (although their reliability can be enhanced considerably by designating backup nodes in the event that the scheduler node fails). Further, a central node places a very heavy burden on some links in the network, causing large communication delays. Decentralized policies, while being immune to the above shortcomings, have their own problems. For instance, it is not always easy to orchestrate a group of schedulers to work in harmony towards solving a particular problem. This is particularly relevant when all or some of the schedulers may possess different views of the state of the system, very easily resulting in anarchic behavior by the schedulers.

1.2 Motivation and Goals of this Research

Distributed systems are different from their centralized counterparts in many obvious ways. One of the consequences of physical distribution of nodes over large distances is the time delay involved in sending messages over the interconnecting network. While it is true that delays are also experienced by messages in a shared-memory shared-bus type of architecture, these delays are typically very small and there exist simple and relatively inexpensive strategies for nodes to maintain globally consistent and exact system states, as may be exemplified by the various cache-coherence algorithms described in the literature.

In relation to load sharing in distributed systems, the two primary contributors to delays are the following: Packetizing and depacketizing of jobs for load sharing is a time consuming task, particularly given the sizes of jobs that we already see these days and what can be expected in the future. Aside from the processing delays at the sender and destination nodes, there is also the delay incurred by the messages during transmission over the network. However, it has been observed by practitioners [LANT85] that the first contribution is the more significant of the two, because network transmission rates are high but the network controllers which perform most of the processing are slow, in comparison.

The two main consequences of transmission delays are the following: Jobs that are transferred over the network experience an additional waiting time due to the delays involved in the transfer process. Thus, there exists the possibility that a sub-optimal policy may actually cause degradation of performance as a consequence of load sharing, by transferring jobs when delays are very high. Further, delays can degrade the quality of remote state information. Because the policies we are interested in studying actively acquire state information, the load sharing decisions could very easily become sub-optimal, if they are based upon an incorrect view of the system state.

One of the goals of this dissertation is to examine simple load sharing policies that are able to perform adequately in the presence of significant delays. In particular, we are interested in studying a class of policies which utilize thresholds at each node in order to make decisions about job transfers. These policies are dynamic in the sense that every time an action is warranted, a node requests the state of a subset of the nodes in the system. For our study, this state comprises of the queue length of the nodes in question. Further, the policies are decentralized because each node is able to make decisions about its own jobs, i.e., whether to execute a job locally or to transfer it to a remote node. As a matter of fact, there need only be consensus between schedulers at the the sender and destination nodes regarding the conditions of the transfer.

The policies we study can be divided into three groups, depending upon which node initiates the job transfer. For instance, in *sender-initiated* policies, the node which possesses a spare job initiates the process of transfer. In *receiver-initiated* policies, the node which requires a job initiates the transfer process and in *symmetric policies*, the process works in both directions, i.e., a node may initiate a transfer to or from itself, depending upon the state in which it finds itself.

The class of load sharing policies under consideration have internal parameters (e.g., thresholds) that must be tuned in order to obtain good performance of the system. In this dissertation, we study two methods to achieve adequate (and in some cases optimal) parameters of the policies. The first method is based upon Entropy Minimax [CHRI85] [CHRI81], which is a non-parametric estimation technique based upon Information Theory [GALL68]. While this method is quite elegant in its formulation of the problem, the utility of this method is limited. For example, we have determined that this method has applicability for homogeneous systems (i.e., where the arrival rates at the nodes are identical and the nodes possess identical processing speeds and capabilities), and for a limited range of parameter values. Extending this particular aspect of our study to include heterogeneous systems did not seem very feasible. Another limiting factor was that this study was based upon simulations which are computationally very expensive. In any case, this part of our study provides valuable insight into the dynamics of the load sharing process, as it particularly relates to delays and imperfect state information.

For a more general solution of the problem, we develop approximate queueing models which represent the behavior of the load sharing policies in the presence of delays associated with job and probe transfers. These models are then solved using the Matrix-Geometric Solution technique. Using these models, we have been able to study the load sharing problem over a vast range of system parameter values and produce many interesting results. Because the assumption of homogeneity may sometimes be restrictive (although a large fraction of the work in the literature does make this assumption), we have extended our models to include systems where the arrival rates and/or processing speeds of the nodes may not all be the same; in other words, for heterogeneous systems. It has been possible to analyze the behavior of the load sharing policies in such systems, providing us a great deal of insight into the feasibility of load sharing in the presence of delays in heterogeneous systems. In this study, we have been able to formulate solutions of two different types of heterogeneous architectures.

In the research outlined above, we have assumed that the arrival process at each node is time-invariant. Although the time between arrivals is a random variable, the mean and higher moments of this random variable are assumed to be fixed. We know from experience that many systems do not exhibit such behavior over reasonable windows of observation. For instance, averaged over an entire week, the mean interarrival time between jobs at an installation may be the same as any other week. However, observed over the period of a day, there may be significant changes in arrival patterns. In this connection, we consider the efficacy of load sharing in systems where the arrival rates can vary over time. We study several simple methods to track the changes in arrivals and formulate high-level policies which supervise the changes in the internal parameters of the policy in operation. In the extreme, the changes in the arrival rates may warrant a change of the load sharing policy itself. The studies conducted by us on time-invariant systems provide the values of the internal parameters of the policies (or even a change of the policy itself) for effective control in time-varying systems, once a stable estimate of the new arrival rate is produced.

1.3 Contributions of the Dissertation

The main contributions of this dissertation lie in providing an understanding of the various important issues that arise when delays are encountered in job transfers as well as in acquiring remote state information, for the purposes of distributed job scheduling. The specifics of the contributions, as regards to particular aspects of our research will be deferred to the relevant chapters. This is because many of the interesting phenomenon we have observed, do not lend themselves to easy explanation unless the background of the experiments and the various important terminology has been introduced. Nevertheless, we would like to provide a brief summary of some of the important contributions of this dissertation.

We have developed analytical models and have solved these models using the Matrix-Geometric solution technique, for the following load sharing policies:

- A sender-initiated policy called *Forward.*
- A receiver-initiated policy called *Reverse.*

- A combination policy called *Symmetric*.

The analytical solutions are valid over a very large range of system parameters. From these solutions, we have been able to study the effects of various important parameters on load sharing, particularly in relation to delays. In order to simplify the above analytical models, we had made certain assumptions which we believed were reasonable. For example, we had assumed that probes were transferred in zero time, in spite of large delays during job transfers, that K, the maximum number of pending remote jobs was one, that the probing and probed nodes have the same thresholds and so on. To address the validity of these assumptions, we have developed analytical models and solved these, using the Matrix-Geometric solution technique, for the following receiver-initiated load sharing policies:

- Policy R_K, where K is a parameter representing the maximum number of allowable pending remote jobs.
- Policy R_{K_T}, which is a threshold probing variation of R_{K_T}.
- Policy R_{T^2}, where the probing and probed nodes may have different operating thresholds.
- Policy R_D, where probes take non-zero times.
- Policy R_{D_T}, which is a threshold probing variation of R_D.

From the results of the studies conducted on the above algorithms, we concluded that our assumptions were in fact, quite reasonable, in the first instance and that our intuition was by and large, correct. Thus far, our study had assumed that the underlying system was homogeneous. We know that in practice, this is a restrictive assumption and that many systems are comprised of heterogeneous nodes. Consequently, we have extended our analytical models and determined solutions for policies operating in such systems. These are:

- A sender-initiated policy called *Forward*
- A receiver-initiated policy called *Reverse*

From the solutions of the above models, we have observed several interesting phenomena. For instance, load sharing in heterogeneous systems is effective for much higher delays than for homogeneous systems, especially when the degree of imbalance in the loads is large. The performance of the policies is more or less sensitive to the probe limit (the maximum number of nodes that may be queried by a node in search of a spare task or to find a placement for its spare task), depending upon the degree of imbalance between the nodes, and so on.

In order to be able to determine the threshold on-line, based upon the average job transfer delay, we conducted simulation studies and summarize the salient points of this part of our study as follows:

- To determine the internal parameters of the load sharing policies (e.g., thresholds), one of the methods we have utilized is based upon Entropy Minimax, an Information theoretic estimation technique. This method has the advantage of being non-parametric (which is particularly relevant in the context of load sharing), is computationally inexpensive and has adaptive capability.
- From our experiments, we have seen that simple policies perform quite well and the performance of these policies using the thresholds generated by Entropy Minimax, is found to either be optimal or very near optimal, in most instances.
- Further, this part of our study provides valuable understanding about how to deal with the uncertainty inherently present in distributed systems, as for example by designing state classification techniques which reduce the impact of this uncertainty.

Finally, we consider systems in which the arrival rates at the nodes may be time varying. We have developed several *simple* strategies to recognize these variations and adapt the parameters of the policies and in some cases, change the policy itself, as a means of providing effective control in such situations. The reason we emphasize the word simple is because we have observed that the policies are quite stable in relation to small errors in the values of the estimated parameters. We conjecture that sophisticated techniques to solve the problem will in all likelihood, provide little or no advantage over simple techniques.

1.4 Overview of the Dissertation

Load Sharing has been an active research area for some time. In Chapter 2, we briefly describe some of the interesting contributions made in this field of study. Further, in Chapter 2, we also point out the lack of emphasis placed by these studies, on the problem of delays. In Chapter 3, we develop queueing models for three probing policies to study the effects of delays on load sharing, over a very large range of system parameters. For the study in Chapter 3, we felt that it was necessary to make certain assumptions, in order to simplify the Markov processes resulting from the policies. It is our belief that these assumptions are valid, over most reasonable values of system parameters. In Chapter 4, we specifically address these assumptions by relaxing them, one at a time. The setting for this study is the class of receiver-initiated load sharing policies. It was seen from the results of this chapter that the assumptions were, in fact, quite reasonable in the first instance. In Chapter 5, we study the problem of load sharing in the presence of delays as it relates to heterogeneous systems. We study heterogeneity in the form of classes of nodes with both different arrival rates and/or processing speeds. This chapter is in some sense a generalization of Chapter 3. In Chapter 6, we study the problem of load sharing in the presence of delays in homogeneous distributed systems where each node has the same processing power and the same external workload. We study the performance of four threshold based load sharing policies, three based on probing and one being a random assignment policy. The threshold is used at each node in order to make decisions for job transfer. These thresholds are obtained on-line, using Entropy Minimax, an Information-Theoretic estimation technique. We study the performance of the policies that use the thresholds as estimated by Entropy Minimax, and compare this performance to the no load balancing or the $M/M/1$ system and the perfect load balancing $M/M/K$ system. Chapter 7 addresses the problem of adaptation of load sharing policies and parameters in time-varying systems. The main thrust of Chapter 7 is to determine strategies that are able to react to changes in the arrival patterns and determine the load sharing parameters on-line in such a way that near-optimal performance may be achieved as a consequence. Finally, in Chapter 8, we summarize the main contributions of this dissertation and discuss suggestions for future research.

Chapter 2

LITERATURE SURVEY

Those who cannot remember the past are condemned to repeat it.
Santayana

There have been numerous studies concerning job scheduling in distributed systems. Some researchers have chosen to perform simulation studies of their algorithms, while others have tried to formulate their problems in a mathematical framework, generally performing some kind of queueing analysis of the system under consideration. In the past few years, there has been a proliferation of distributed systems being built and the designers of some of these have actually implemented load sharing algorithms. Most of the research done this in field, particularly the analytical studies, has tended to ignore the effects of delays. In some models, the authors have contended that delays will be small and hence can be neglected. While this assumption may be true in some cases, it is our belief that in many instances, delays cannot be wished away. Some of the simulation studies, particularly [MIRC86] [STAN85a] [STAN84] have peripherally studied the impact of delay as one parameter among many others. In Theimer et al., [THEI85], the authors report their concerns with task transfer delays and develop a heuristic to reduce the impact of these delays. However, these studies have not specifically addressed many of the interesting issues that arise as a consequence of transfer delays. We now present a brief summary of some of the interesting works in the field of distributed job scheduling.

Simulation studies provide tremendous flexibility in choosing system models and enable us to evaluate load balancing algorithms which may be very hard to study analytically. However, simulations are extremely time consuming and computationally expensive and consequently should be used in very limiting circumstances.

Some of the interesting works using the simulation approach are the following: Stankovic [STAN84] simulated three load balancing algorithms under a large range of parameters. It was seen that simple algorithms, using very rudimentary state information, were able to generate very good performance. In another study, Stankovic [STAN85a] studied the load balancing problem in the presence of imperfect state information. The author utilized Bayesian decision theory in the decision making process. In this approach, determination of a proper utility function was determined to be crucial to the performance of the system.

Wang and Morris [WANG85] studied load balancing problems in a very general setting. They proposed a taxonomy of load balancing policies that draws a basic dichotomy between *source-initiative* and *server-initiative* approaches. Under these classifications they compared a variety of different algorithms using either an exact analysis or by simulation. Livny and Melman [LIVN82] studied simple load sharing algorithms by means of simulations. They showed that, over a very large range of system parameters, the probability of some nodes being idle while others have waiting jobs at the same time, is very high. Thus, load sharing is likely to provide significant performance improvements in these instances. Bryant and Finkel [BRYA81] performed simulation studies on a load balancing algorithm that was seen to be stable. The authors have designed a very elaborate scheme for pairing spare jobs with under-utilized processors. A node that wishes to transfer a job queries the other nodes. If a queried node is in the "idle" state, it responds to the query, forming a pair. The job is sent to the node which has the potential for the best response time for the transferred job. The authors have used several alternative methods to estimate the current load on the system. One is based upon the memoryless assumption, another on what they call "pastrepeats", and one on utilizing distributions of service times. The method based upon distributions is the most accurate (as might be expected), but is computationally expensive, so the study concludes that the method based upon "pastrepeats" appears to be the best compromise. Ni et al. [NI85b] utilized what they call a drafting approach to perform load balancing. In this technique, a node that enters a Low state "Drafts" a task from a set of High nodes. This approach has the flavor of reverse bidding. The authors determine that their algorithm performs better than one that is based upon bidding, for the parameter values tested. Further, they conclude that load information need not be very accurate, in order to achieve good performance.

Mirchandaney and Stankovic [MIRC86] determined the feasibility of using a job scheduler based upon an enhanced Stochastic Learning Automaton. It was shown that in systems where the state information possesses a great deal of uncertainty, a learning approach to acquiring job scheduling information is workable, for certain ranges of system parameters. Recently, several studies have been conducted in the area of load sharing where the tasks have hard real-time constraints. Stankovic et al. [STAN85b] considered the load sharing problem in the framework of distributed hard real-time systems. Several heuristics are evaluated by simulation studies. Zhao et al. [ZHAO85] used simulations to evaluate hard real-time scheduling based upon bidding and focussed addressing. Such techniques have also been shown to be applicable in non hard real-time systems.

The sheer number of analytical studies conducted on the load sharing problem makes it impossible to list them all. We will try to discuss the more relevant contributions. Many of the studies [AGRA82] [BUZE74] [CHOW77] [CHOW79] [NI86] [NI85a] have been concerned with a system consisting of two or more (heterogeneous) processors with one common job arrival stream. Jobs arrive in this stream at a single controller which then makes a decision on the assignment of jobs to processors. Obviously, this assumption seems to be quite restrictive for distributed systems. Chow and Kohler [CHOW77] [CHOW79] developed queueing models for heterogeneous multiprocessor systems where one job stream is distributed to a number of processors according to some state-dependent job routing policies. The authors used an approximate numerical method to analyze two-processor heterogeneous models, and showed that a deterministic strategy that maximizes the expected throughput during the next interarrival period gives the best performance among a class of policies.

Agrawala, Tripathi and Ricart [AGRA82] studied the problem of how to route a job stream to servers of different speeds. A routing strategy was devised, called virtual waiting time technique that minimizes the average completion time for a job. Ni and Hwang [NI86] [NI85a] studied the problem of probabilistically balancing loads among processors in a heterogeneous multiple processor system with many job classes. They developed an algorithm which allocates workloads among processors.

Another solution approach to study the load balancing problem is to formulate an optimization problem for which there might exist a standard optimization technique or an efficient, approximate algorithm may be devised [BANN83] [BUZE74] [dSeS84] [DUTT82] [INDU86] [KRAT80] [KURO86a] [LEE87] [STON78a] [STON78b] [TANT84] [TANT85] [TOWS86]. These works use different system models and/or focus on different aspects of load balancing problems. Buzen and Chen [BUZE74] studied the problem of determining optimal loading factors in memory hierarchies and devised an optimization algorithm for its solution.

Stone [STON78a] [STON78b] solved the problem of allocating program modules to processors in multiprocessor systems with the help of network flow algorithms. The interprocessor communication cost due to interacting tasks is explicitly accounted for and the minimum delay placement of tasks is determined. Most of the work on task placement assumes complete a priori knowledge about the execution times (in case of stochastic scheduling, the mean execution times) of tasks as well as the communication between pairs of tasks. Towsley [TOWS86] devised two polynomial time assignment algorithms for allocating program modules to processors. Two optimization problems are formulated, one called Restricted and the other, Unrestricted, in which the author utilizes a graph-theoretic computational model of distributed programs containing loops and branches. Kratzer and Hammerstrom [KRAT80] formulated the problem of allocating jobs to processors in multiprocessors and distributed computer systems as an assignment problem. It is shown in the paper that the general problem is NP-complete. An algorithm was devised that yields optimal solutions in restricted instances of the problem and another heuristic bidding algorithm was used for general problems. Dutta, Koehler and Whinston [DUTT82] formulated a quadratic assignment problem for task allocation in a distributed processing environment. For uncapacitated problems (no capacity constraint at each processor), they used a probabilistic branch and bound technique for its solution. For capacitated problems, heuristic algorithms are developed. Bannister and Trivedi [BANN83] considered the problem of allocating tasks in fault-tolerant distributed systems. They formulated a constrained sum of squares minimization problem for which an efficient approximate algorithm is developed. Silva and Gerla [dSeS84] studied the load balancing problem in distributed systems that can be modeled by product-form queueing networks. It is assumed that there exist multiple classes of jobs in the system with site constraints for execution of each class of jobs. The optimal solution is found by a downhill search method in which the steepest descent direction can be easily calculated using mean value analysis(MVA).

Tantawi and Towsley [TANT85] formulated a nonlinear optimization problem for a probabilistic static load balancing in distributed systems that can be modeled by a product-form queueing network. An efficient algorithm is developed for its solution, based on sorting the nodes according to their incremental delays. This technique is also applied to obtain a static load balancing strategy for star-configured systems [TANT84]. Kurose and Singh [KURO86a] mapped the static load balancing problem into the problem of planning in the field of mathematical economics, from which they derived a distributed optimization algorithm. Similar distributed optimization algorithms have been developed for the minimum delay routing problem in communication networks [GALL77]. The load balancing problem in soft real-time distributed systems is also considered, in which a job arriving at a node must begin execution within a specified amount of time after its initial arrival in the system [KURO86b]. It is shown that a simple approach may perform equally as well as more sophisticated ones. Recently, Lee [LEE87] studied load balancing in homogeneous and heterogeneous systems. The author developed efficient heuristic algorithms based upon distributed integer optimization where each node computes its own load balancing parameters. Indurkhya et al., [INDU86] utilize a Random-Graph model of distributed programs and explore the important dichotomy between the parallel execution of the modules (tasks) of a program which tends to distribute the tasks evenly among the processors, and the communication overhead incurred between modules executing at different nodes, which tends to keep the tasks on a few processors. A simple example they show consists of two processors; with low communication overheads, the optimal solution divides the tasks equally between the two processors. When the overhead increases above a certain threshold, all the tasks are either placed on one or the other processor, with no other states besides these two.

Eager, Lazowska and Zahorjan [EAGE86a] [EAGE86b] studied a class of threshold load balancing policies in homogeneous distributed systems. In [EAGE86a], they studied the effect of state information used in load balancing policies to the performance of the system. It is shown that extremely simple policies which use primitive system state information, are very effective. In [EAGE86b], load balancing policies are classified into two categories depending upon which node initiates the process of job transfer; namely, *sender-initiated* policies and *receiver-initiated* policies. It is shown that sender-initiated policies outperform receiver-initiated policies at light to moderate system loads, and that receiver-initiated policies are preferable only if the costs of job transfer under the two strategies are comparable. The class of policies considered in this dissertation is similar to those in [EAGE86a] and [EAGE86b], except that we also utilize *symmetric* policies, which are a combination of sender and receiver-initiated policies and our work is quite closely related to theirs. The main difference between our system model and that of Eager et al. is that in their model, the transfer overheads appear as additional service times on the CPU, while for the most part we assume that the overheads are either small or even negligible (this assumption is actually relaxed in

Chapter 6) but that the delays are prominent. This is because of the general trend towards intelligent network controllers which would offload most of the scheduling related functions from the CPU.

Finally, the proliferation of distributed systems has given rise to actual implementations of simple load sharing algorithms, as for example [AGRA85] [AYAC82] [BUTT84] [KIRR84] [NEED82] [POWE83] [THEI85] [VANT84]. Theimer et al. [THEI85] have implemented a simple load sharing scheme for their network of workstations. A node that is overloaded (i.e., the response time of tasks at that node appears to be high), polls a subset of the remaining nodes and transfers a task to the first one that is lightly loaded. In their implementation, a transfer of a 2 Mb program space takes on the average about 6 seconds to complete. Because the workstations are diskless, no files need to be transferred along with the task image and the authors acknowledge that in systems which require the movement of all the associated files along with the task (which may be a reasonable practice), the delays could become very high, and possibly unbounded. The results from this experimental system give credence to the importance of studying the effects of large delays in distributed systems. Locus [BUTT84] provides preemptable remote execution across a network of multiuser machines. Several of the systems that have implemented load sharing algorithms have been very application specific, for instance, in the field of real-time process and industrial control, as in [AYAC82] [KIRR84]. Van-Tilborg and Wittie [VANT84] describe a task scheduling heuristic based upon wave scheduling of task forces (parallel programs). This algorithm is conjectured to be applicable for a large class of homogeneous multicomputer systems like CM*, MICRONET and others and was designed as a scheduling algorithm for the MICROS distributed operating system.

2.1 Summary

In this chapter, we have provided a brief survey of the studies conducted in the field of distributed job scheduling. We have classified these studies into three groups, i.e., simulation studies, analytical studies and implementations of job scheduling algorithms for distributed systems. While there has been considerable research in this field, very little emphasis has been placed upon studying the impact of job transfer delays and decisions based on imperfect state information, in spite of several experimental studies indicating the possible importance of delays. In the following chapter, we formulate analytical models of simple load sharing algorithms and determine the performance characteristics of these algorithms, when significant delays are encountered in job transfers.

Chapter 3

LOAD SHARING IN HOMOGENEOUS SYSTEMS

3.1 Introduction

In this chapter, we study the performance characteristics of simple load sharing algorithms for distributed systems. In the system under consideration, it is assumed that non-negligible delays are encountered in transferring jobs from one node to another and in gathering remote state information. Because of these delays, the state information gathered by the load sharing algorithms is out of date by the time the load sharing decisions are taken. This chapter analyzes the effects of these delays on the performance of three algorithms that we call Forward, Reverse and Symmetric. We formulate queueing theoretic models for each of the algorithms operating in a homogeneous system under the assumption that the job arrival process at each node is Poisson and the service times and job transfer times are exponentially distributed. Each of the models is solved using the Matrix-Geometric solution technique and the important performance metrics are derived and studied.

In this connection, we have developed analytical models that help us better understand the above issues. Various relevant performance metrics are derived from these models and the load sharing algorithms are compared on the basis of these metrics. By studying the results obtained from the model solution, we are able to determine the exact effects of delays and out of date state information on load sharing in general. Furthermore, we are able to determine the range of delays and loads over which state information is worth gathering and useful load sharing can be performed.

The remainder of this chapter is organized as follows: In Section 3.2, we provide a brief description of the system architecture and the load sharing algorithms.

Section 3.3 comprises the description of the Markov process corresponding to the Symmetric algorithm and its Matrix Geometric solution. The analysis corresponding to the Forward and Reverse probing algorithms will only be described in brief. This is because the analysis of Symmetric subsumes that of Forward and Reverse. In Section 3.4, we describe the important results of this research and we summarize our work in Section 3.5. Appendix A describes the internals of the matrices involved with the solution of the Markov processes.

3.2 System Architecture and Load Sharing Algorithms

3.2.1 System Architecture and Motivation

Processing and transmission of communication messages for state updates (probes) and for jobs can potentially generate considerable overhead at the nodes. Different system architectures can impose very different costs for these overheads. At one end of the spectrum, nodes can have dedicated processors to handle communication overheads, supported by a very high bandwidth fiber-optic bus communication. On the other end of the spectrum, nodes can be multiplexed between application jobs and communication packet processing.

We have made the following assumptions about the system that we will be considering. The architecture of the individual nodes includes a powerful Bus Interface Unit(BIU), which is used to process most of the overhead generated by job and probe movement. For instance, the BIU will have a DMA capability to access main memory without much interference to the CPU.

While the bulk of the overhead processing for job transfer is transferred to the BIU, delays will nevertheless occur during this processing. There will also be network delays in the transmission of probes and jobs. We are interested in studying the combined effects of these delays. Furthermore, we believe that it is reasonable to assume that the relative sizes of jobs and probes will be quite different. The physical transfer of a job may require tens of communication packets, while a probe or a response to one would in all likelihood need at most one packet. Thus, it is reasonable to imagine a ratio of 50:1 or more in the relative sizes of jobs vs. probes. Consequently, it appears that the delays incurred by jobs in the BIU's and the network will be significantly larger than those incurred by the probes. In our analysis, the delays incurred by probes will be assumed to be negligible when compared with those incurred by job transfers.

3.2.2 Load Sharing Algorithms

The three algorithms that we have studied in the context of this research are called Forward, Reverse and Symmetric. Each algorithm is provided with a threshold T. The algorithms are described in the following few paragraphs.

- **Forward:** The algorithm is activated each time a local job arrives at the node. If the number of jobs at this node (including the job currently being executed) is greater than $T + 1$, an attempt is made to transfer the newly arrived job to another node. A finite number, L_p, of nodes (usually $L_p = 2$ or 3 is adequate) is probed at random to determine a placement for the job. A probed node responds positively if the number of jobs it possesses is less than $T + 1$ and it is not already waiting for some other remote job. If more than one node responds positively, the sender node transfers the job to one of these respondents, picked at random. If none of the probed nodes responds positively, i.e., this probe was unsuccessful, the node waits for another local arrival before it can probe again.

- **Reverse:** This algorithm is activated every time a job completes at a node and the total number of jobs at the node is less than $T+1$ and the node is not already waiting for a remote job to arrive. If so, the node probes a subset of size L_p remote nodes at random to try and acquire a remote job. Only nodes that possess more than $T + 1$ jobs, (including the currently executing one) can respond positively. If more than one node can transfer a job, the probing node chooses one of these at random from which it requests a job.

- **Symmetric:** This algorithm combines the two schemes of Forward and Reverse. Thus, if a node goes above $T + 1$ upon the arrival of a local job, it attempts to transfer a job and if it drops below $T + 1$ upon a job completion, it attempts to acquire a remote job.

In all the algorithms described above, it is assumed that probing takes zero time. This is based upon the initial assumption that probes are much smaller entities than are jobs. Thus, the overhead for processing a probe at the BIU is much smaller than for jobs. Further, probes occupy much less of the communication bandwidth than jobs. Thus, the entire delay is assumed to occur during actual job transfer. Furthermore, we have seen in separate studies (not described in this chapter) that as long as the ratio of job transfer times to probe transfer times is sufficiently large (≥ 20), the system essentially behaves as if the probes actually take zero time. We are currently investigating this phenomenon in greater detail.

3.3 Mathematical Analysis

It is assumed that the job arrival process at each node is Poisson, with parameter λ. Also, the service times and job transfer times are assumed to be exponentially distributed, with means $1/\mu$ and $1/\gamma$, respectively. The job transfer time includes the time between the initiation of a transfer from a node and the successful reception of the job at the destination node. The nodes are assumed to be homogeneous, i.e., the nodes have identical processing power and the arrival process at each node is the same. Jobs are assumed to be executed on a First-Come-First-Served (FCFS) basis at each node.

Let $N_t^{(i)}$ be the number of jobs at node i at time t and $J_t^{(i)}$ be the probe state of node i, at time t. The probe state indicates whether the node is probing or being probed, etc. For example, in a system of M nodes, the instantaneous state of the network can be represented by the 2M-tuple

$$(N_t^{(1)}, N_t^{(2)}, ..., N_t^{(M)}; J_t^{(1)}, J_t^{(2)}, ...J_t^{(M)})$$

If the probe state $J_t^{(i)}$ is defined appropriately then, due to the Poisson arrival assumption and the exponential service and job transfer times, the process corresponding to the above state description is Markovian.

It is clear that the model has a very large state space and is difficult to solve, even for moderately sized systems. Consequently, we decompose the model such that the model for each node can be solved independently of the others [EAGE86a]. The interactions between the nodes which result in job transfers for the purpose of load sharing in the distributed system, are modelled by means of modifications to the arrival and/or departure process at each node. These interactions will be described in detail further in this subsection.

We conjecture that the method of decomposition is asymptotically exact as the number of nodes tends to infinity. Actual experimental results indicate that there exists very good agreement between the model and simulations even when the systems are of relatively small size (= 10 nodes). Thus, the approximation is likely to be even better for larger systems. These analytical results have been validated through simulation for networks of at least 10 nodes.

The analysis of the algorithms is performed using the Matrix-geometric solution technique [NEUT81], which yields an exact solution of the model for each node. The model for the Symmetric probing algorithm will be described in detail. However, the analysis of the Forward and Reverse algorithms will only be described in brief, with a presentation of the main performance metrics.

The material in this chapter involves several Jacobi matrices, whose detailed definitions will be provided as in Latouche [LATO81]. A matrix such as

$$\begin{bmatrix} b_0 & c_0 & 0 & 0 & & & \dots & \\ a_1 & b_1 & c_1 & 0 & & & \dots & \\ 0 & a_2 & b_2 & c_2 & & & & \\ & & & & & & & \\ \dots & \dots & & & a_{m-2} & b_{m-2} & c_{m-2} & 0 \\ \dots & \dots & & & 0 & a_{m-1} & b_{m-1} & c_{m-1} \\ \dots & \dots & & & 0 & 0 & a_m & b_m \end{bmatrix}$$

will be displayed as

$$\begin{Vmatrix} & c_0 & c_1 & \dots & c_{m-3} & c_{m-2} & c_{m-1} \\ b_0 & b_1 & b_2 & \dots & b_{m-2} & b_{m-1} & b_m \\ a_1 & a_2 & a_3 & \dots & a_{m-1} & a_m & \end{Vmatrix}$$

Figure 3.1 represents the state diagram for the Symmetric algorithm operating at a single node using an arbitrary threshold T. The state of the node is represented by a tuple (N_t, J_t), where N_t is the number of jobs at a node and J_t is the probe state that indicates if the node is either probing, being probed, neither of the above, or both. The probe states have the following codes:

- 0 : if not probing and not being probed,
- 1 : if reverse probing,
- 2 : if being forward probed,
- 3 : if reverse probing and being forward probed.

The actual representation of this process takes the form of an infinite cylinder. However, for ease of description, we have chosen to open out this cylinder and consequently, the row corresponding to probe state 3 is duplicated, once as the top row and again as the bottom row in Figure 3.1. In 3-Space, the top and the bottom rows would be merged together.

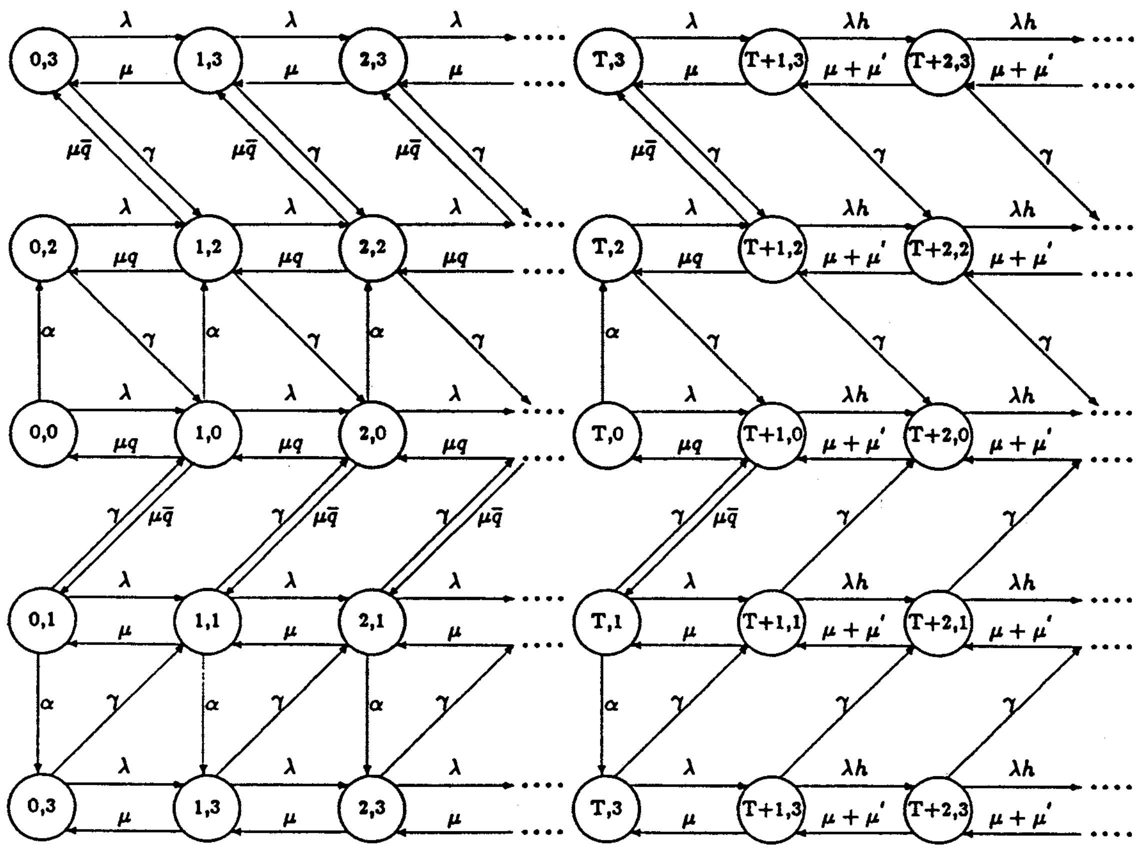

Figure 3.1: State Diagram of Symmetric Algorithm

We define

$$y(n,j) = \lim_{t\to\infty} P(N_t = n,\ J_t = j),\ 0 \le n,\ 0 \le j \le 3,$$

$$\mathbf{p}_n = (y(n,0), y(n,1), y(n,2), y(n,3)),\ 0 \le n,$$

$$\vec{\mathbf{p}} = (\mathbf{p}_0, \mathbf{p}_1, \mathbf{p}_2, ...\mathbf{p}_i, ...).$$

If the Markov process (N_t, J_t) is ergodic then $\vec{\mathbf{p}}$ is its steady state probability vector satisfying $\vec{\mathbf{p}}Q = 0$, where Q is the infinitesimal generator of this Markov process. Q_S, the infinitesimal generator for the Symmetric algorithm, has the structure of a block-tridiagonal matrix of the form

$$Q_S = \left\| \begin{array}{ccccccc} & B_{01} & ... & B_{01} & B_{01} & A_0 & A_0 & ... \\ B_{00} & B_{11} & ... & B_{11} & B_{21} & A_1 & A_1 & ... \\ B_{10} & B_{10} & ... & B_{10} & A_2 & A_2 & A_2 & ... \end{array} \right\|$$

where we define the matrices $B_{00}, B_{01}, B_{10}, B_{11}, B_{21}, A_2, A_1$ and A_0 in Appendix A at the end of the dissertation.

In the subsequent discussion, h is the probability of failure in finding an assignment for a spare job in response to a set of forward probes. Thus, $\bar{h} = 1 - h$, is the probability that at least one of the probed nodes will accept a remote job. q is the probability of failure in finding a remote job for a set of reverse probes, and $\bar{q} = 1 - q$.

The effect of a node sending a forward probe when it goes above $T + 1$ is represented by the transition λh. When the node makes a transition anywhere below $T + 1$ on the completion of a job, it sends out reverse probes in order to get a remote job. A successful transition is represented by $\mu\bar{q}$ and an unsuccessful set of reverse probes is represented by the transition μq.

Thus, on the completion of a job when the node goes below $T + 1$, it sends out reverse probes, if it is not already waiting for a remote job to arrive in response to an earlier reverse probe. A transition of this type is represented from $(n, 0)$ to $(n - 1, 1)$ or $(n, 2)$ to $(n - 1, 3)$, where $0 < n \leq T + 1$. When a remote node sends a forward probe into this node, it makes the transition from $(n, 0)$ to $(n, 2)$ or $(n, 1)$ to $(n, 3)$, where $0 \leq n \leq T$. This means that the remote node is going to transfer a job to this node, on the basis of a successful probe. The rate of receiving forward probes is denoted by α. The rate at which this node sends out jobs in response to nodes that asked it for jobs is μ'. Thus, the rate at which a node makes the transition (n, j) to $(n - 1, j)$, for $n \geq T + 2$ equals $\mu + \mu'$.

As can be seen from the generator Q_S, the Markov process has a regular structure comprised of the A_0, A_1 and A_2 matrices, preceded however by the irregular boundary conditions. The size of the irregular portion of the matrix depends upon the threshold at which the process is operating. There will be exactly $T - 1$ columns of the matrices (B_{01}, B_{11}, B_{10}).

Neuts [NEUT81] examined Markov processes with such generators and determined the conditions for positive recurrence when the infinitesimal generator $A = A_0 + A_1 + A_2$, corresponding to the geometric part of the Markov Process, is irreducible. However, for our problem, A is lower triangular and reducible. In such cases, the stability criterion has to be determined explicitly.

Consider the non-linear matrix equation

$$A_0 + RA_1 + R^2 A_2 = 0$$

such that R is its minimal non-negative solution. It can be shown that R is lower triangular, given the structure of A_0, A_1 and A_2 [NEUT81]. Furthermore, $R = [r_{i,j}]$, where

$$\begin{aligned} r_{i,j} &= 0, \forall i < j \\ r_{1,1} &= \frac{\delta - (\delta^2 - 4(\mu + \mu')\lambda h)^{1/2}}{2(\mu + \mu')} \end{aligned}$$

$$\begin{aligned}
r_{2,2} &= \frac{\delta+\gamma-((\delta+\gamma)^2-4(\mu+\mu')\lambda h)^{1/2}}{2(\mu+\mu')} \\
r_{3,3} &= r_{2,2} \\
r_{4,4} &= \frac{\delta+2\gamma-((\delta+2\gamma)^2-4(\mu+\mu')\lambda h)^{1/2}}{2(\mu+\mu')} \\
r_{2,1} &= \frac{\gamma}{\delta-(r_{1,1}+r_{2,2})(\mu+\mu')} \\
r_{3,1} &= r_{2,1} \\
r_{3,2} &= 0 \\
r_{4,1} &= \frac{(r_{4,2}r_{2,1}+r_{4,3}r_{3,1})(\mu+\mu')}{\delta-(r_{1,1}+r_{4,4})(\mu+\mu')} \\
r_{4,2} &= \frac{\gamma}{\delta+\gamma-(r_{2,2}+r_{4,4})(\mu+\mu')} \\
r_{4,3} &= r_{4,2}
\end{aligned}$$

where $\delta = (\lambda h + \mu + \mu')$.

Thus, the diagonal elements of R can be written explicitly in terms of the parameters of the Markov process. Once the diagonal elements are determined, the elements below the diagonal are computed recursively from the solution of the diagonal elements.

By adapting Theorem 1.5.1 from Neuts [NEUT81], the Markov process Q_S is positive recurrent if and only if $sp(R) < 1$ and that the matrix M (defined below) possesses a positive left invariant probability vector. Because R is lower triangular, its eigenvalues are its diagonal elements. One can show that $sp(R) < 1$ if

$$\lambda h < \mu + \mu'$$

The matrix M, given by

$$\left\| \begin{array}{ccccc} & B_{01} & \dots & B_{01} & B_{01} \\ B_{00} & B_{11} & \dots & B_{11} & B_{21}+RA_2 \\ B_{10} & B_{10} & \dots & B_{10} & \end{array} \right\|$$

is an irreducible, aperiodic matrix. The second condition holds because of the irreducibility of M. The vector $(\mathbf{p}_0, \mathbf{p}_1, ..., \mathbf{p}_{T+1})$ is the left eigenvector of M.

Intuitively, the stability condition means that the rate of processing jobs (including the ones that are sent out of this node) is greater than the total arrival rate of jobs into this node. Thus, on the average, whenever there are more than $T+1$ jobs at a node, the process drifts towards the boundary specified by the threshold T. Similar analysis may be carried out for the Forward and Reverse probing algorithms, with the appropriate substitution of parameters.

We now assume that all the values of all the parameters are known. First, the boundary conditions are determined, by solving a system of linear equations. Thus, for an arbitrary threshold T, we have

$$(\mathbf{p}_0, \mathbf{p}_1, ..., \mathbf{p}_{T+1}) \left\| \begin{matrix} & B_{01} & \dots & B_{01} & B_{01} \\ B_{00} & B_{11} & \dots & B_{11} & B_{21} + RA_2 \\ B_{10} & B_{10} & \dots & B_{10} & \end{matrix} \right\| = 0$$

where the number of columns in the matrix is exactly $T+1$. We know from Neuts [NEUT81] that

$$\mathbf{p}_i = \mathbf{p}_{T+1} R^{T+1-i}, \forall i \geq T+1$$

Thus,

$$\sum_{i \geq T+1} \mathbf{p}_i = \mathbf{p}_{T+1}(I - R)^{-1}$$

Also,

$$[\sum_{i=0}^{T} \mathbf{p}_i + \mathbf{p}_{T+1}(I - R)^{-1}]e = 1.$$

$E[N]$, the expected number of jobs at a node, and $E[D]$, the expected response time of a job, are given by the following expressions:

$$E[N] \;=\; \sum_{i \geq 1} i\, \mathbf{p}_i\, e$$

$$= \mathbf{p}_{T+1}(I-R)^{-2}e + T * [\mathbf{p}_{T+1}(I-R)^{-1}e] + \sum_{i=1}^{T} i\mathbf{p}_i e$$

$$E[D] = \frac{(E[N] + \frac{(Total-Flow-In)}{\gamma})}{\lambda}$$

where $Total - Flow - In$ is the flow into a node of remote jobs due to forward and reverse probes. In the next subsection, we derive the equations required to determine the values of the unknowns h, q, μ' and α and describe the iterative algorithm used to solve the resulting model.

3.3.1 Computational Procedure

Initially, it is assumed that the values for h, q, μ' and α are known and the model is solved using these values. In a typical step, a model solution is used to derive new values for h, q, μ' and α, and a new solution is computed. The iteration procedure that we use is described in a step-wise form, after the following definitions.

- $FFRO$: Flow rate out of jobs, as a result of forward probes made by this node.
- $FFRI$: Flow rate in of jobs, as a result of forward probes made to this node by other nodes.
- $RFRO$: Flow rate out of jobs, as a result of reverse probes made by other nodes to this node.
- $RFRI$: Flow rate in of jobs, as a result of reverse probes made by this node.

Let i denote the iteration count. Thus, $h^{(i)}, q^{(i)}, \mu'^{(i)}, \alpha^{(i)}, FFRO^{(i)}, RFRO^{(i)}$ denote the value of the variables after the i-th iteration.

Iteration Procedure

1. Let $i = 0$; choose values for $h^{(0)}, q^{(0)}, \mu'^{(0)}, \alpha^{(0)}, FFRO^{(0)}, RFRO^{(0)}$

2. Determine $Q^{(i)}$ from $h^{(i)}, q^{(i)}, \mu'^{(i)}, \alpha^{(i)}$

3. Determine $R^{(i)}$

4. Solve the linear system corresponding to the boundary conditions

5. Determine $FFRO^{(i+1)}$ and $RFRO^{(i+1)}$ from the model solution

6. If $ABS(FFRO^{(i+1)} - FFRO^{(i)}) \leq \epsilon$ and $ABS(RFRO^{(i+1)} - RFRO^{(i)}) \leq \epsilon$, where ϵ is an arbitrary small number, stop, else

7. Let $i = i + 1$. Go to 2

We have observed from experiments that the solution was insensitive to the initial values chosen for the unknown quantities. Consequently, we conjecture that there exists a unique solution to the model. Further, the number of iterations necessary for convergence was usually small, ranging between 10 and 30.

Because of the assumption of homogeneity and because of the symmetric nature of the algorithm

$FFRO = FFRI$ and, $RFRO = RFRI$.

To determine α, we use the relation $FFRO = FFRI$, where

$$\begin{aligned} FFRO &= \lambda \bar{h} \sum_{i>T} \mathbf{p}_i\, e, \\ FFRI &= \alpha \sum_{i\leq T} \mathbf{p}_i\, [1100]^T, \end{aligned}$$

Here, h can be represented as $h = x^{L_p}$ where, L_p is the number of nodes that are probed and x is the probability that a particular node will respond negatively to a forward probe. This is given as

$$x = \sum_{i\leq T} \mathbf{p}_i\, [0011]^T + \sum_{i>T} \mathbf{p}_i e.$$

Also, $\bar{x} = 1 - x$ and $\bar{h} = 1 - h$.

Thus,

$$\alpha = \frac{FFRO}{\sum_{i \leq T} \mathbf{p}_i \, [1100]^T}$$

To determine μ', we use the relation $RFRI = RFRO$, as follows:

$$RFRI = \sum_{i \geq 0} \mathbf{p}_i \, [0101]^T \gamma,$$

where $1/\gamma$ is the mean delay in receiving a remote job. Thus, $RFRI$ denotes the total flow in due to reverse probes made by this node.

$$RFRO = \mu' \sum_{i > T+1} \mathbf{p}_i \, e$$

Thus,

$$\mu' = \frac{RFRI}{\sum_{i > T+1} \mathbf{p}_i \, e}$$

To determine q, the probability that a set of reverse probes result in failure, we use the following procedure:

Let

$$y = \sum_{i \leq T+1} \mathbf{p}_i \, e$$

If the node probes L_p nodes to receive a remote job, then the probability that all of them will be unsuccessful is denoted by: $q = y^{L_p}$, and $\bar{q} = 1 - q$ is the probability that at least one of the reverse probes is successful.

3.3.2 Forward and Reverse

As mentioned in Section 3.2, we will only briefly describe the analysis for the Forward and Reverse probing algorithms, because these algorithms are in some sense subsumed by the Symmetric algorithm. Figure 3.2 represents the state diagram for the Forward probing algorithm operating at a single node using an arbitrary threshold T. The state of the node is represented by a tuple (N_t, J_t), where N_t is the number of jobs at a node and J_t is the probe state that indicates if the node is being forward probed or not. The probe states have the following codes:

- 0 : if not being probed,
- 1 : if being forward probed,

The infinitesimal generator matrix corresponding to this process is:

$$Q_F = \left\| \begin{array}{ccccccc} & B_{01} & \ldots & B_{01} & B_{01} & A_0 & A_0 & \ldots \\ B_{00} & B_{11} & \ldots & B_{11} & A_1 & A_1 & A_1 & \ldots \\ A_2 & A_2 & \ldots & A_2 & A_2 & A_2 & A_2 & \ldots \end{array} \right\|$$

with exactly $T-1$ columns of (B_{01}, B_{11}, A_2).

Figure 3.3 represents the state diagram for the Reverse probing algorithm operating at a single node using an arbitrary threshold T. The state of the node is represented by a tuple (N_t, J_t), where N_t is the number of jobs at a node and J_t is the probe state that indicates if the node is either probing or not. The probe states have the following codes:

- 0 : if not probing,
- 1 : if reverse probing,

The infinitesimal generator matrix for this process is:

$$Q_R = \left\| \begin{array}{ccccccc} & A_0 & \ldots & A_0 & A_0 & A_0 & A_0 & \ldots \\ B_{00} & B_{11} & \ldots & B_{11} & B_{11} & A_1 & A_1 & \ldots \\ B_{10} & B_{10} & \ldots & B_{10} & A_2 & A_2 & A_2 & \ldots \end{array} \right\|$$

with exactly $T-1$ columns of (A_0, B_{11}, B_{10}).

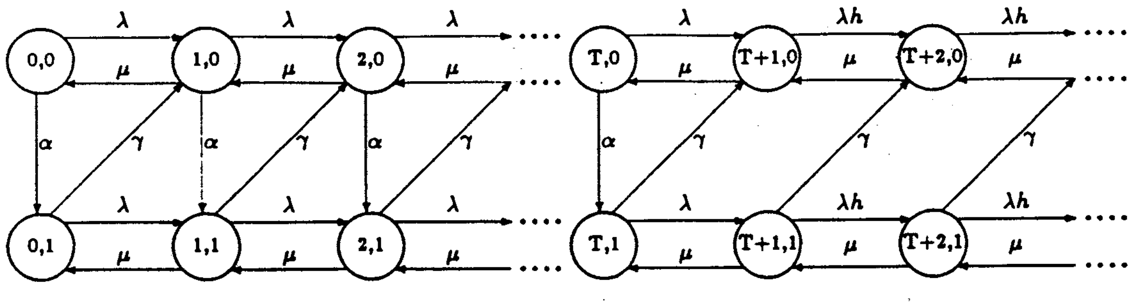

Figure 3.2: State Diagram of Forward Algorithm

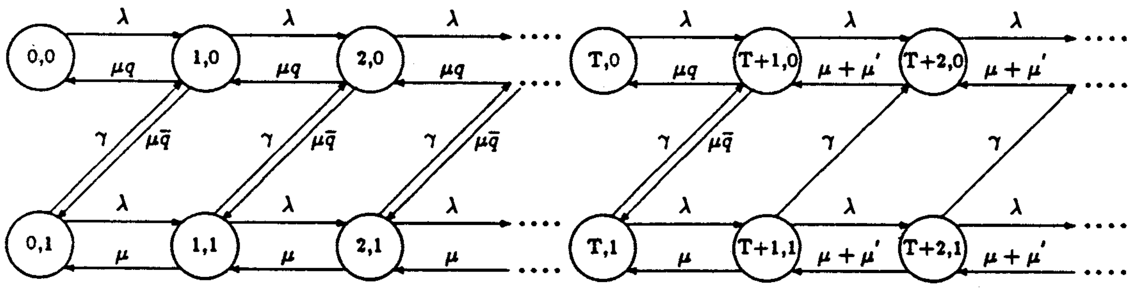

Figure 3.3: State Diagram of Reverse Algorithm

All the parameters for the Forward and Reverse probing algorithms have the same meanings as their counterparts in the Symmetric algorithm. For instance, μ' in Reverse probing is the rate at which a node sends out jobs in response to reverse probes made by other nodes, as in the case of Symmetric probing.

The computational procedure for both these algorithms is very similar to that for the Symmetric probing algorithm, which was described earlier in this section in detail. Iteration is used to solve the system in both these cases. The unknown parameters in the case of Forward are α and h and in the case of Reverse, the unknowns are μ' and q. The internals of the matrices of Forward and Reverse are described in Appendix A.

In both these cases, initial values of the unknown parameters are used to solve the model. Based upon this solution, new values of the parameters are determined. The iteration continues until the stopping criterion has been satisfied. It was seen that the iteration was insensitive to the initial values chosen for the unknown parameters. Further, the number of iterations was usually small, between 10 and 20.

3.4 Performance Comparisons

In this section, the performance of the three load sharing algorithms will be compared to each other and to two bounds, represented by the no-load-balancing $M/M/1$ model (also referred to as NLB) for K nodes and the perfect load sharing with zero costs, i.e., the $M/M/K$ model. Wherever relevant, we will also compare the algorithms against a Random assignment algorithm, which transfers jobs based only upon local state information. This algorithm is similar to Forward in the sense that a node that goes above $T+1$ transfers a job. However, the node does not send any probes. Instead, it picks a destination node at random and transfers a job to this node. The key performance metric for comparison is the mean response time of jobs.

A large number of parameters such as the service time, the threshold T, the probe limit L_p, the communication delay $1/\gamma$, the number of nodes in the network etc., can affect the performance of load sharing algorithms. In this connection, we will try to present the results that we believe are the most relevant. The presentation will be in the following sequence:

- Validation of the analytical results with simulations.
- Nominal comparisons between the algorithms.
- Relation between delays and thresholds.
- Optimal response times as a function of delays.
- Optimal thresholds as a function of delays.

Unless specifically mentioned otherwise, $L_p = 2$ in all the runs. Also, $S = 1/\mu$ and $C = 1/\gamma$ are the means of the service time and job transfer delay, respectively. Further, it will be assumed that $S = 1$ unit and all measurements of response times will be in terms of this unit.

Validation with Simulations

We mentioned in Section 3.2 that the decomposition used in this chapter is only an approximate solution which is conjectured to be exact for infinitely large systems. Thus, it is important to determine how well this approximation compares to simulations of finite sized systems. The simulation model consisted of 10 nodes in all cases except when $\rho = 0.9$, where the model consisted of 20 nodes. Figure 3.4 depicts a representative set of curves regarding this study.

Because the simulation results were almost identical to the analytical model, we have chosen not to depict the actual sample means of the response times from the simulations. Instead, the 95% confidence intervals of the simulation results are presented, as computed by the Student-t tests. On the average, the confidence interval for the response time is about $\pm 3\%$ about the sample mean. The only

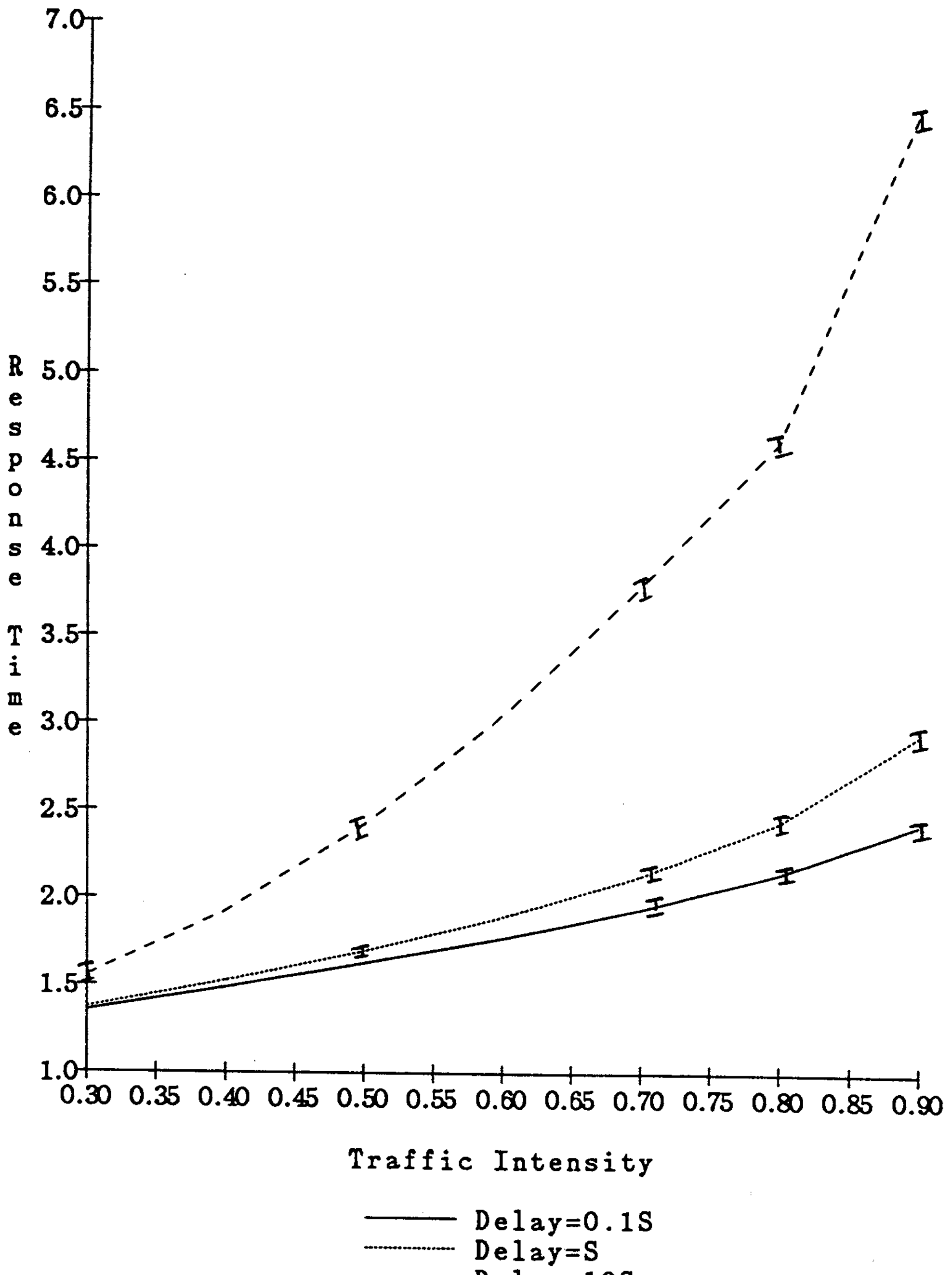

Figure 3.4: Comparison with Simulations

exception to this is at $\rho = 0.9$, when the confidence interval is about $\pm 6\%$ about the sample mean.

We have observed (results not presented here) that in most of the cases, the variation between the simulation results and the analytical models is less than 2%. Furthermore, the model is almost invariably optimistic, compared to the simulation results. The maximum variation that we observed was about 15%, and such numbers were very infrequent and were seen to occur at low communication delays and high loads ($\rho \geq 0.9$). As the delays increase however, the model tends to become more accurate. In any case, for loads ≤ 0.8, the model is a very good approximation, even for reasonably small systems. In cases where the variation was more than 2%, it was seen that by increasing the size of the simulation system to 20 nodes, the results generated better agreement with those of the analytical model. For instance, the variation at $\rho = 0.9, C = 0.1S$, which was about 15% when the simulation system comprised of 10 nodes, decreased to about 5% for a system of 20 nodes.

Comparison of the Algorithms

In an earlier study by Wang and Morris [WANG85], it was postulated that at low loads, Forward probing is likely to perform best, while at high loads, Reverse would be more suitable. However, it was not known exactly where one policy became better than the others, especially when there are significant communication delays involved.

Another factor that takes on a degree of importance in this comparison between algorithms, is that of probe overhead. While we have assumed that probes take zero time, there is the potential for the probes to interfere with other messages, especially if they are generated in large enough numbers. It has been shown in [MIRC87a] that the Symmetric algorithm generates probes at a higher rate than do Forward and Reverse. While we have not included the effects of such overhead in our model thus far, this aspect of the study is currently under progress.

Figure 3.5 shows the performance curves of the algorithms for $C = 0.1S$ and

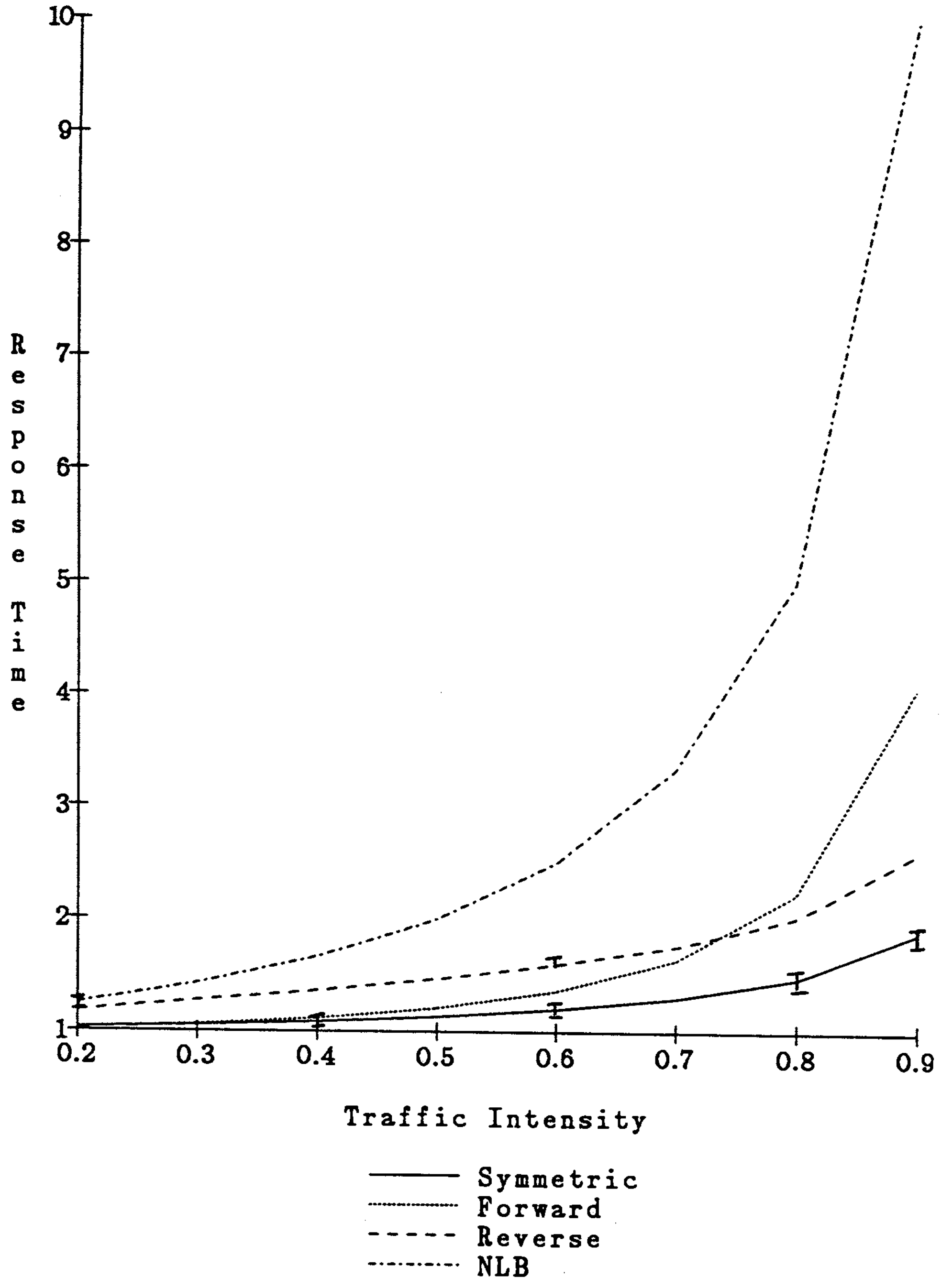

Figure 3.5: Comparison of Algorithms, Delay = $0.1S$

$T = 0$. From this figure, we can make the following observations:

- At low delays and low loads ($\rho \leq 0.5$), Forward performs essentially like Symmetric but Reverse is worse by as much as 30%. This can be explained by the fact that in most cases, Reverse is ineffective in load sharing as most nodes will not have a spare job. Thus, the Reverse component of Symmetric does not improve its performance over Forward.

- At moderate loads, Symmetric performs much better than both Forward and Reverse, by as much as 20%, while Forward and Reverse are about the same.

- At high loads ($\rho > 0.9$), it is seen that Reverse is better than Forward by a substantial margin of about 25% while Symmetric is still the best overall, being better than Reverse by about 25%.

- At all the loads tested, there appears to be a substantial gain in load sharing as opposed to NLB. This is true for all three algorithms. However, the improvement is much more pronounced as the load increases. For instance, at $\rho = 0.9$, the response time for Symmetric is about 2 units whereas the NLB response time is 10 units, a significant difference.

- As may be expected, the algorithms perform worse than the exact $M/M/K$ model. However, Symmetric generates close performance to the $M/M/K$ model. For instance, at $\rho = 0.9$, $M/M/K$ results in a response time of 1.3 units while Symmetric generates 2 units.

Figure 3.6 shows the performance curves of the algorithms for $C = 2S$ and ($T = 2$). From Figure 3.6, we can reach the following conclusions:

- For moderate communication delays and low to moderate loads ($\rho \leq 0.7$), the behavior of the three algorithms is virtually the same. It would appear that the delay overhead predominates at these loads.

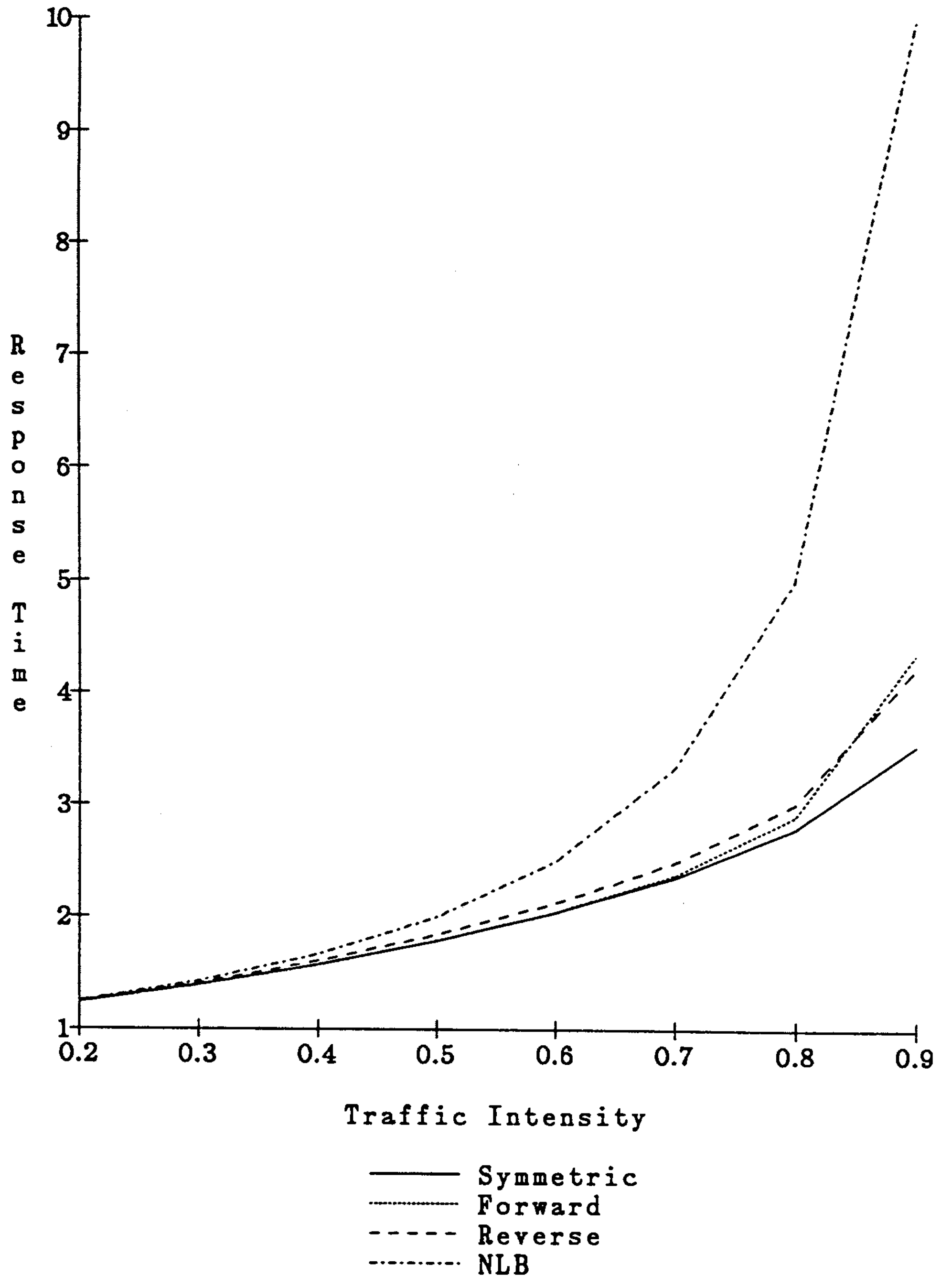

Figure 3.6: Comparison of Algorithms, Delay=$2S$

- At moderate loads, ($\rho = 0.8$), Symmetric is about 10% better than Reverse but almost identical to Forward.
- Only at very high loads ($\rho \geq 0.9$) does Symmetric actually perform significantly better than both Forward and Reverse.
- In comparison with NLB, it is seen that at low loads ($\rho \leq 0.5$), there is little or no improvement by load sharing. However, as the load increases, load sharing becomes more viable. At $\rho = 0.9$, Symmetric generates a response time of 3.5 units as opposed to 10 units for NLB.
- The comparison against the $M/M/K$ model is not very flattering at high delays, as might be expected. For instance, Symmetric at $\rho = 0.9$ is about 2.5 times worse than the $M/M/K$ value of about 1.3 units.

Thus, one can conclude that at moderately high delays, the performance of the three algorithms is virtually identical. A surprising result though is that Symmetric is significantly better at very high loads.

All the subsequent discussion is based on the results obtained from the Symmetric algorithm. Unless explicitly mentioned otherwise, the conclusions reached are also applicable to Forward and Reverse. In cases where the performance of these algorithms is markedly different from that of Symmetric, a separate discussion will be provided.

Delays vs Thresholds

Figures 3.7, 3.8 and 3.9 show the response times for the Symmetric algorithm tested over a wide range of communication delays and thresholds, for the loads of 0.5, 0.7 and 0.9. It can be seen from Figure 3.7, that at low delays ($C = 0.1S$), the optimal threshold is 0 and the performance is a monotonically increasing function of the threshold. Also, the response time generated at $T = 0$ is only about 20% worse than the exact $M/M/K$ value for moderate loads ($\rho \leq 0.7$). For example, at $\rho = 0.7$, the Symmetric response time is about 1.3 units whereas the exact $M/M/K$

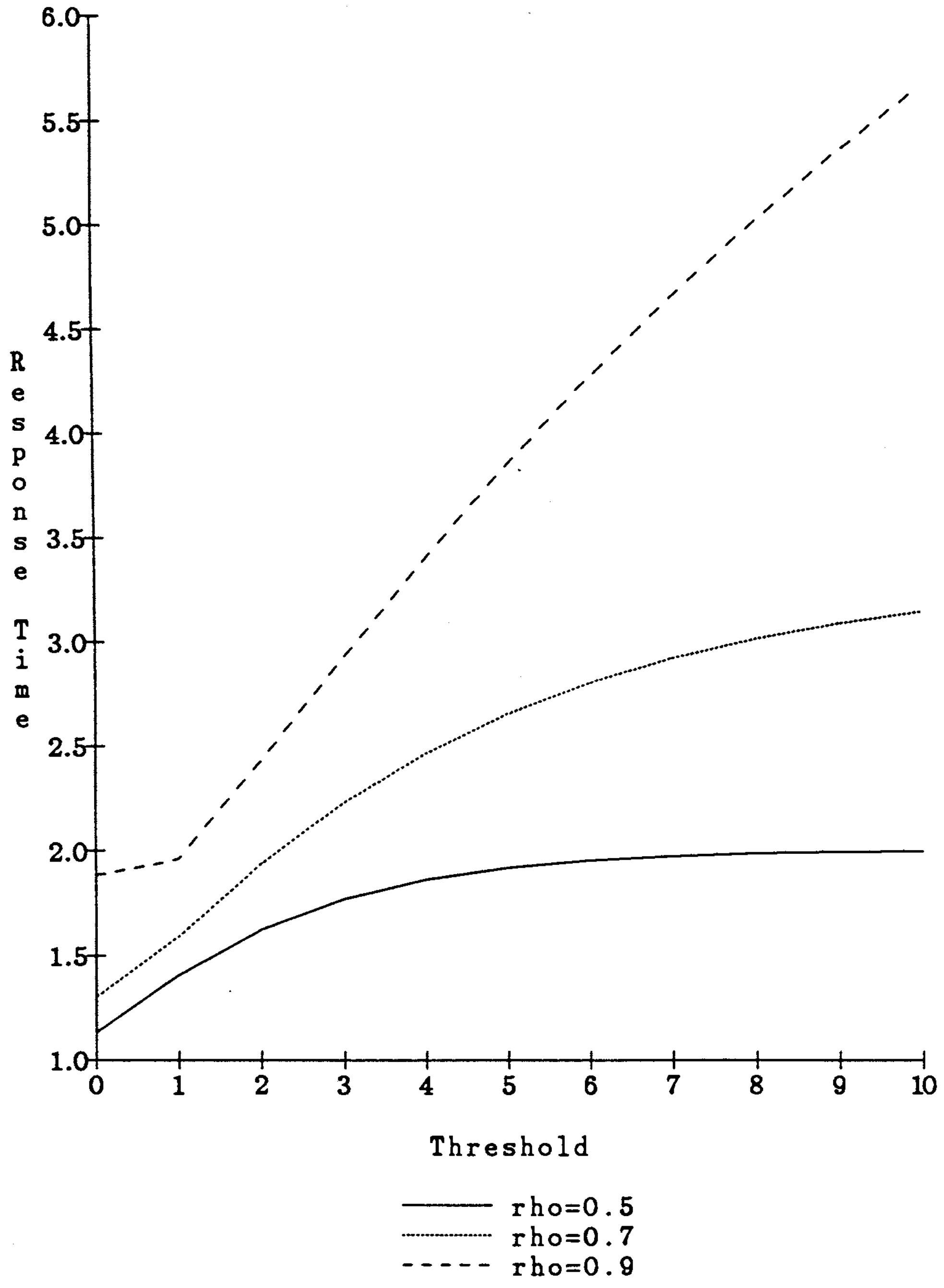

Figure 3.7: Variation of Thresholds, Delay=0.1S

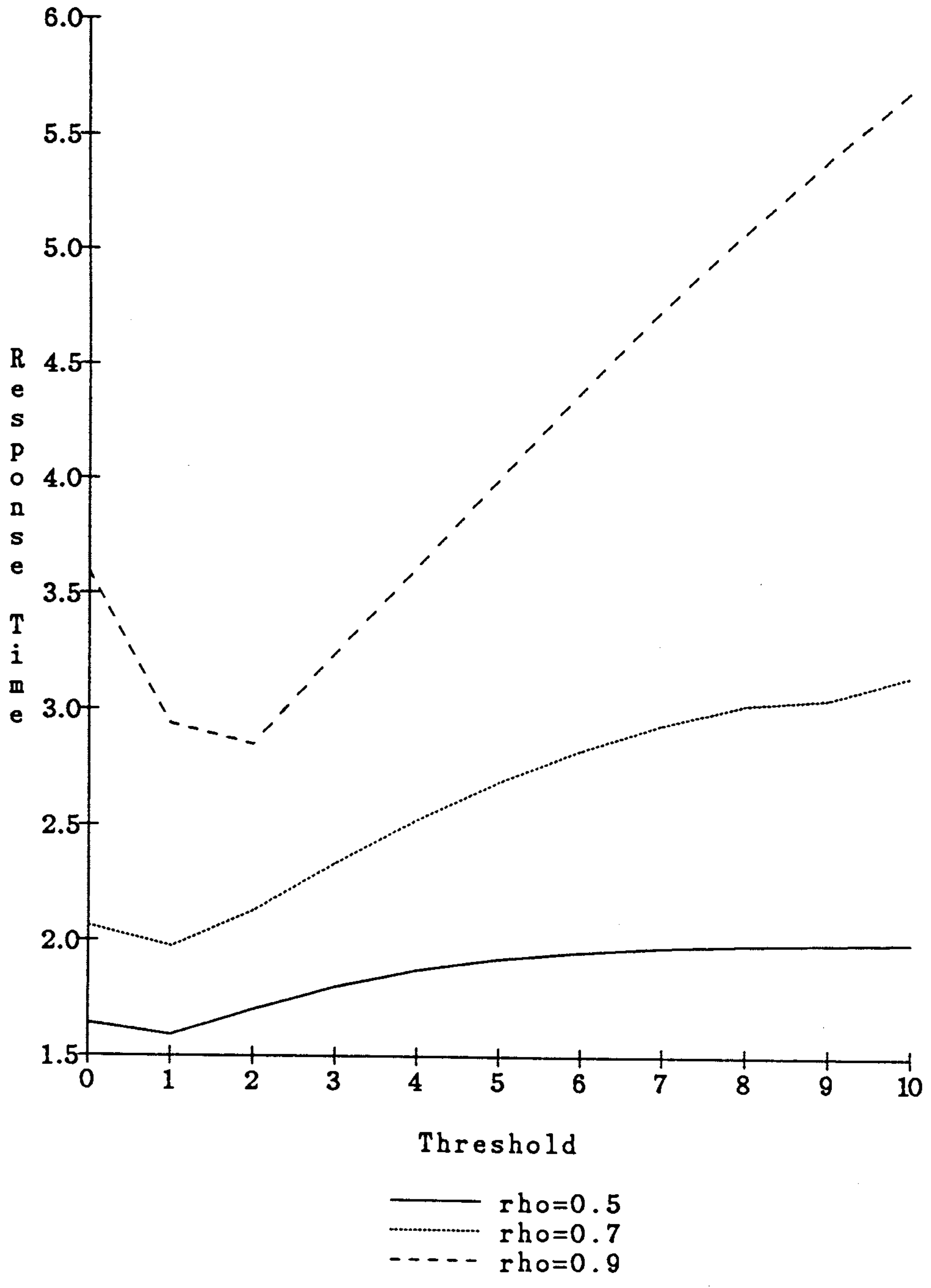

Figure 3.8: Variation of Thresholds, Delay=S

value is approximately 1.04 units. Further, the NLB response time for this load is 3.3 units, which is much worse than the performance of the Symmetric algorithm. The performance improvement due to load sharing in this case can be explained by the following arguments:

- At low delays, the cost of transferring a job is much lower than the potential improvement due to the effect of load sharing. Thus, $T = 0$ permits very active load sharing.

- Because the delays are small, much greater certainty exists in the knowledge that an idle node will continue to remain idle during the time it takes to transfer a job to it. Thus, in some sense, $T = 0$ ensures that all job transfers are useful in that a remote job arrives at the node soon after it becomes idle.

For moderate delays ($C = S$, Figure 3.8), the behavior is as follows: Even at $\rho = 0.5$, there is a gain of about 22% from load sharing. For instance, the best response time at this load is about 1.56 units while the corresponding NLB performance is 2 units. The improvements over NLB by load sharing at higher loads are even more substantial, being as high as about 73% for $\rho = 0.9$. The NLB response time in this case is 10 units whereas the optimal Symmetric value is about 2.7 units, as can be seen from Figure 3.8. Further, $T = 1$ for $\rho = 0.5$ and 0.7, while $T = 2$ for $\rho = 0.9$, are the *optimal* thresholds.

When the communication delays increase to the order of $10S$ (Figure 3.9), it is seen that the best that can be achieved for $\rho = 0.5$ is the NLB performance which is 2.0 units response time. Thus, it would be appropriate to turn off load sharing here. For $\rho = 0.7$, a small gain of about 5% is seen, at $T = 5$. This improvement is small enough that if the interference of probes could be accounted for, the best strategy might very likely be to turn off load sharing. However, at $\rho = 0.9$, the reduction in response time from the NLB is about 40% and this occurs at $T = 6$, where the Symmetric response time is about 6.0 units. In any case, the response times at high delays are significantly worse than the $M/M/K$ values as might be

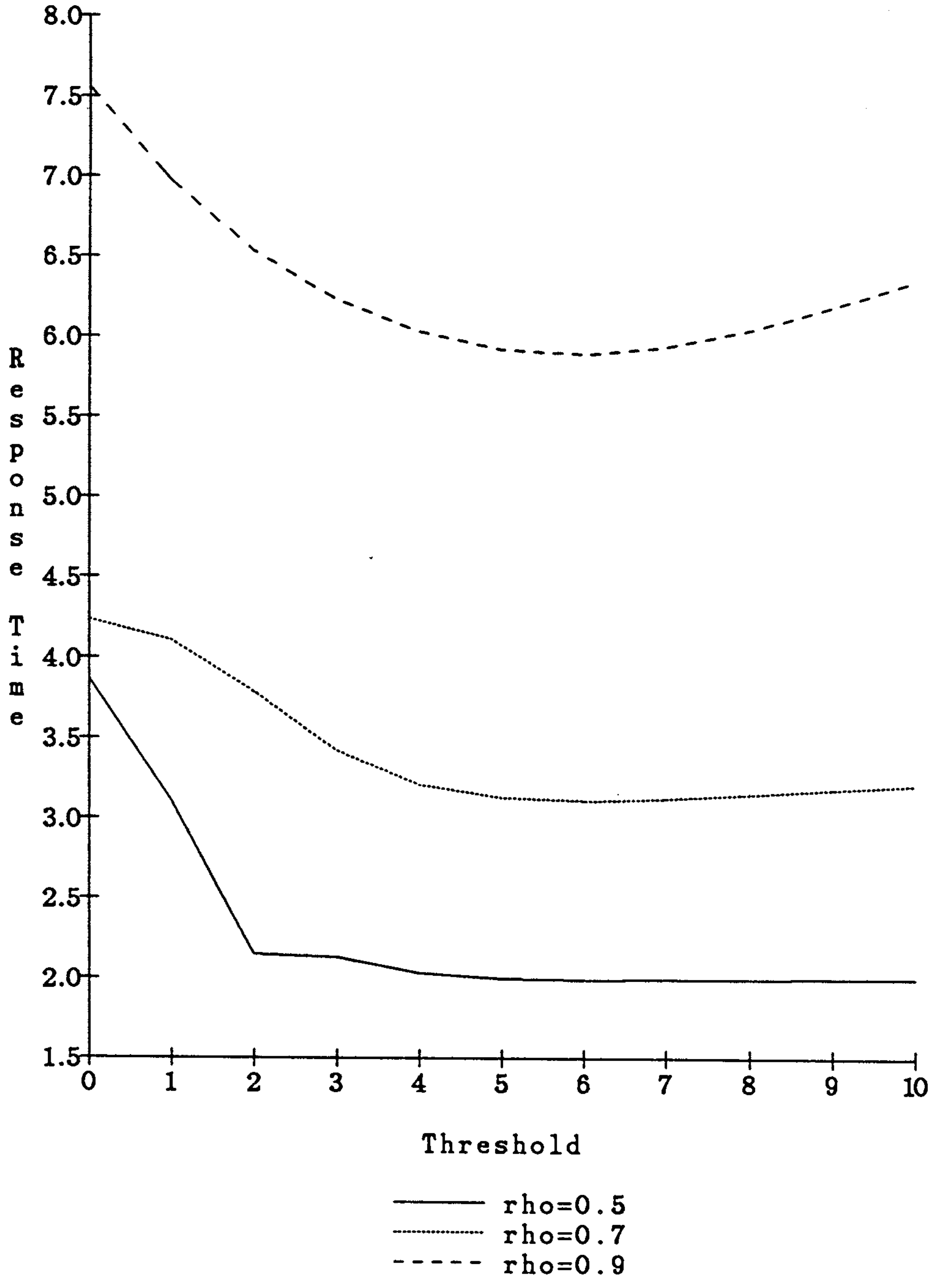

Figure 3.9: Variation of Thresholds, Delay=10S

expected. For instance, at $\rho = 0.7$, the $M/M/K$ response time is 1.04 units whereas the best load sharing value is about 3.1 units.

Optimal Response Times

The purpose of this set of tests is to determine the best possible performance of the algorithms under a very large range of transfer delays, ranging from as small as $1/100\,S$, to as large as $100\,S$. Thus, in this study, one can assume very fast local area networks will form one end of the spectrum and slow, long-haul networks the other end. Figure 3.10 shows the results of the tests for the algorithm.

The response time in each case is normalized by the $M/M/1$ response times. Thus, a lower ratio indicates greater improvements as a consequence of load sharing. Corresponding to each curve representing a particular load, there is a curve for the performance of the Random assignment algorithm, to be used as a baseline. From the figure, one can see that at low delays ($\leq 1/2\,S$), the gain from load sharing is quite substantial, at all loads considered. Further, the gains are greater for higher loads. At loads of 0.9, the response times are 0.25 times those for the no load sharing case.

As can be seen from the curves representing the performance of the random assignment, there is a definite advantage in probing. However, as the delays increase, ($> 1\,S$), this advantage of probing seems to disappear. Random with a suitable threshold is able to perform as well as any probing policy, giving the impression that the state information due to probing is so out of date as to not really be useful. Also, the best that can be achieved in lower loads (≤ 0.5) is no better than the $M/M/1$ response time at these delays. However, there is still a marked improvement in the performance of load sharing at higher loads, for example at 0.8 and 0.9. The remarkable fact that should be noticed here is that even at delays as high as $100\,S$, there is about an 8% improvement over no load sharing for load 0.9. We postulate that at higher loads, this effect will be even more prominent.

Optimal thresholds

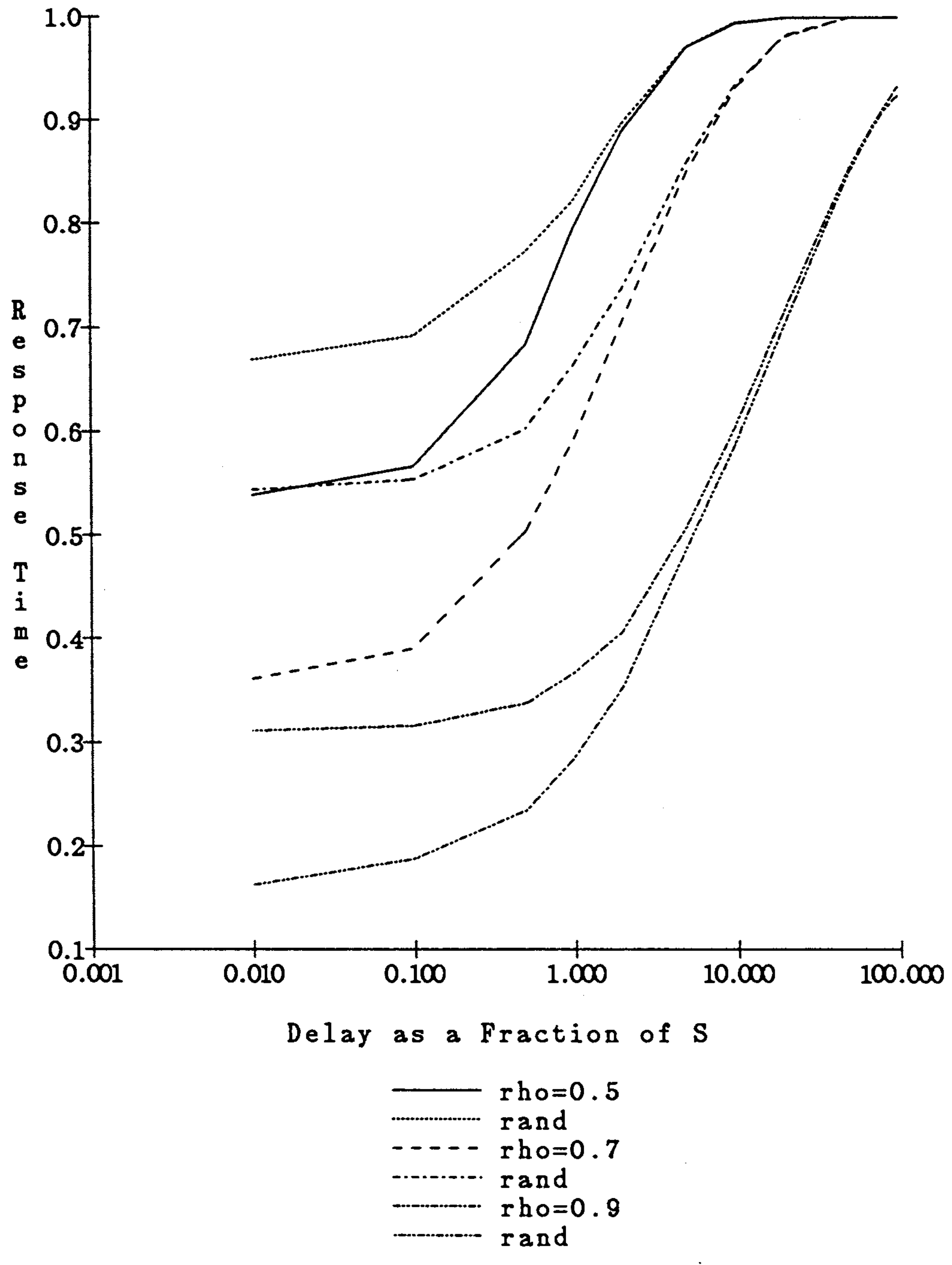

Figure 3.10: Optimal Normalized Response Times

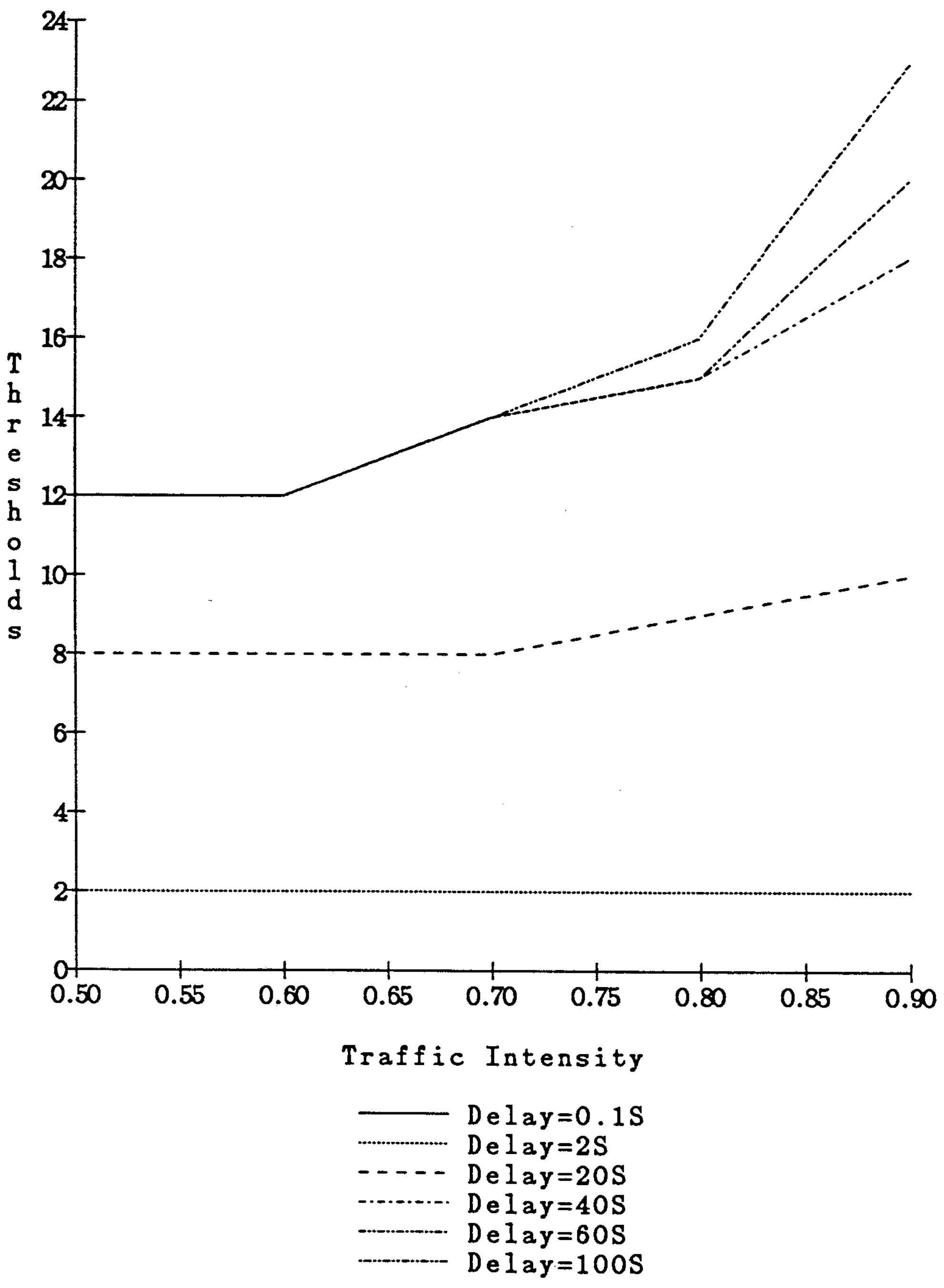

Figure 3.11: Optimal Thresholds

Figure 3.11 indicates the variation of the *optimal* thresholds corresponding to the *optimal* response times indicated in Figure 3.10. Note that the thresholds are low at lower delays and get higher as the delays increase. Further, this effect is seen to be more prominent at higher loads. At $\rho = 0.9$, the optimal threshold varies between 0 when the delay is $1/10\,S$ and 25, when the delay is $100\,S$. The variation is significantly lower at low loads.

3.5 Summary and Conclusions

This study was concerned with the performance analysis of simple load sharing algorithms in the presence of significant job transfer delays. The three algorithms that we tested were called Forward, Reverse and Symmetric. The analysis of the algorithms was carried out using the Matrix-Geometric solution technique.

The Markov process of the entire network appeared to be computationally intractable. Thus, we employed a decomposition technique to solve the Markov process. While this resulted in an approximate solution of the original system, it was seen by means of simulation studies, that the variation between the exact and approximate solutions was minimal for systems of 10-20 nodes. Consequently, the analytical solution is likely to be more accurate for larger systems. This leads us to hypothesize that the decomposition is an exact solution of the system in the limit as the number of nodes tends to infinity.

The three load sharing algorithms were tested over a large range of parameter values. Some of the salient observations that we made were as follows:

- There is considerable difference between the performance of the three algorithms at low to moderate delays ($\leq S$), with Symmetric providing the best results. As delays increase, the algorithms tend to provide almost identical performance, especially when ($D \geq 10S$). Further, at such delays, Random assignment performs as well as any probing scheme, leading us to believe that at moderate to high delays, probing is wasted effort.

- At high delays ($\geq 10S$), the optimal response times are no better than those for the NLB case, leading us to believe that load sharing is not useful in such situations, for low to moderate loads. However, at high loads ($\rho \geq 0.9$), substantial benefits accrue from load sharing even at these delays.

- Reverse probing is outperformed by Forward over most of the range of loads tested, except when $\rho \geq 0.9$. While Symmetric is the best of the three algorithms tested, it does have the potential for generating high probing overheads. Given these observations, Forward would appear to have even greater applicability if realistic overhead costs might be assigned to probes.

- The benefits of load sharing are more pronounced at high loads ($\rho \geq 0.8$). This is evidenced by the fact that the percentage reduction in response times in these cases is greatest over the corresponding NLB values.

- At extremely high loads $\rho = 0.9$, it is seen that about 8% reduction is achieved over the corresponding NLB response time, even when the delays are as high as $100S$.

- The optimal threshold was seen to be a function of the load and the job transfer delay. At low delays, the optimal threshold was 0 for all the loads tested. However, as the delays increased, the optimal threshold increased correspondingly, becoming about 24 for $\rho = 0.9$ and delay $= 100S$.

Chapter 4

ANALYSIS OF RECEIVER-INITIATED LOAD SHARING

4.1 Introduction

In Chapter 3, we had determined the effects of delays upon three algorithms called Forward, Reverse and Symmetric. To provide a large breadth of study, we had made some simplifying assumptions in the analysis. For instance, K was always equal to 1 and probing times were assumed to be zero. In this chapter, we relax the simplifying assumptions made in our previous work and study Receiver-Initiated load sharing algorithms in greater depth and in a more general setting. By studying the results obtained from the model solutions, we are able to determine the exact effects of delays and out of date state information on Receiver-Initiated load sharing policies. Furthermore, we are able to determine the range of delays and loads over which state information is worth gathering and useful load sharing can be performed.

Various relevant performance metrics are derived from these models and the load sharing algorithms are compared on the basis of these metrics. Some of the interesting observations that we have made are as follows: The analytical models are shown be very good approximations of the underlying system. It is seen that the algorithms are insensitive to the parameter K and the effects of probing delays are determined to be negligible, under reasonable assumptions regarding probe sizes.

The remainder of this chapter is organized as follows: In Section 4.2, we provide a brief description of the system architecture and the load sharing algorithms called R_K, R_{K_T}, R_D, R_{D_T} and R_{T^2}. Section 4.3 comprises the description of the Markov process corresponding to the R_K algorithm and its Matrix Geometric solution. The analysis corresponding to the R_{K_T}, R_D, R_{D_T} and R_{T^2} probing algorithms will only be described in brief since it is similar to the solution for R_K. In Section 4.4, we study the performance characteristics of the algorithms. The analytical models are validated against simulation studies. We summarize our work in Section 4.5. Appendix B at the end of the dissertation describes the internals of the matrices involved with the solution of the Markov processes.

4.2 System Architecture and Load Sharing Algorithms

4.2.1 System Architecture and Motivation

In this research, we assume a network of nodes that contain the algorithms and mechanisms necessary for distributed load sharing. As discussed in the earlier chapters, we assume that delays occur during the act of load sharing.

4.2.2 Load Sharing Algorithms

In this subsection, we briefly describe the reverse probing algorithms that we study in this chapter. Each algorithm utilizes a threshold T.

- Algorithm R_K : This reverse probing algorithm with parameter K is activated every time a job completes at a node and the total number of jobs at the node is $\leq T$. If so, the node probes a small subset of remote nodes at random to try and acquire a remote job. Only nodes that possess more than $T + 1$ jobs, (including the currently executing one) can respond positively. If more than one node respond positively, the probing node chooses one of these nodes at random from which it requests a job. Because there is a delay in acquiring a remote job, a node can request another remote job in the meantime, if a local job completes and the node is still $\leq T$. However, a node may only have a maximum of K remote jobs pending at any time. An important aspect of this study is to determine the impact of varying the parameter K.

- Algorithm R_{K_T} : This algorithm is similar to R_K, except that the node sends out reverse probes only when a job completes and the number of remaining jobs is exactly T. In this case too, a node may only have a maximum of K remote jobs pending. The reason that we are interested in studying this algorithm is because R_K has the potential to generate a large number of probes, especially if the threshold is high. In many instances, these probes are not likely to result in job transfers. Consequently, we postulate that in some situations, the extra probing of R_K may not provide any significant performance improvements over R_{K_T}.

- Algorithm R_{T^2} : Threshold based load sharing algorithms have been generally designed with one threshold. Thus, the threshold below which a node sends out reverse probes is the same as the one the probed node needs to be above, to provide a spare job. In general, this may not always be the optimal strategy, especially at high delays, where the sending threshold may have to be higher. Thus, we are interested in determining the conditions under which a dual threshold algorithm, in which the probing node utilizes a threshold T_p and the probed node uses a higher threshold T_t, may be useful.

In the above algorithms R_K, R_{K_T} and R_{T^2}, it is assumed that probes take zero time. Thus, a probing node has instantaneous knowledge about the status of the probed nodes. In general, this may not be a realistic assumption, although probes on the average experience much smaller delays than do jobs. The reason for making this assumption at this point is the resulting simplicity achieved in the analysis, because we believe that the issues of interest in the above algorithms are orthogonal to the effects of probing time. This assumption is subsequently relaxed.

To test the effect of the assumption of zero probing times, we study two algorithms R_D and R_{D_T}, corresponding to R_K and R_{K_T} respectively, except that probes experience a delay. Thus, if the node sends probes which do not result in a job transfer, this fact is made known to the probing node after an exponentially distributed time interval, with mean $1/\alpha$. Further, in R_D and R_{D_T}, we restrict the maximum number of pending remote jobs, K to exactly one, because of our results concerning the effects of parameter K (results presented in this chapter). In summary,

- Algorithm R_D corresponds to R_K in the sense that upon completion of a job, a node sends out reverse probes if the number of remaining jobs at the node is $\leq T$, but negative replies to probes experience a non-zero delay.
- Algorithm R_{D_T} corresponds to the R_{K_T} algorithm in the sense that upon completion of a job, a node sends out reverse probes if and only if the number of remaining jobs at the node is exactly T, but negative replies to probes experience a non-zero delay.

4.3 Mathematical Analysis

In this section, we develop the analytical model for R_K. It is assumed that the job arrival process at each node is Poisson, with parameter λ. Also, the service times and job transfer times are assumed to be exponentially distributed, with means $1/\mu$ and $1/\gamma$, respectively. The job transfer time includes the time between the initiation of a transfer from a node and a successful reception of the job at the destination node. Further, we assume that the job transfer times are independent of the origin and destination of the jobs and the load placed on the network. The nodes are assumed to be homogeneous, i.e., the nodes have identical processing power and the arrival process at each node is the same. Jobs will be assumed to be executed on a First-Come-First-Served (FCFS) basis at each node.

Let $N_t^{(i)}$ be the number of jobs at node i at time t and $J_t^{(i)}$ be the probe state of node i, at time t and i be the condition code that indicates whether the node is not probing, or if it is probing, then how many remote jobs are pending. The codes are as follows:

- 0 : if not probing,

- k : if reverse probing and waiting for k jobs.

For example, in a system of M nodes, the instantaneous state of the network can be represented by the 2M-tuple

$$(N_t^{(1)}, N_t^{(2)}, ..., N_t^{(M)}; J_t^{(1)}, J_t^{(2)}, ...J_t^{(M)})$$

Due to the Poisson arrival assumption and the exponential service and job transfer times, the process corresponding to the above state description is Markovian.

It is clear that the model has a very large state space and would become difficult to solve, even for moderately sized systems. Consequently, we decompose the model such that each node can be solved independently of the others [EAGE86a]. The interactions between the nodes which result in job transfers for the purpose of load sharing in the distributed system, are modelled by means of modifications to the arrival and/or departure process at each node.

We conjecture that the method of decomposition is asymptotically exact as the number of nodes tends to infinity. Actual experimental results indicate that there exists very good agreement between the model and simulations even when the systems are of relatively small size (= 10 nodes). Thus, the approximation is likely to be even better for larger systems.

The analysis of the algorithms is performed using the Matrix-geometric solution technique [NEUT81], which yields an exact solution of the model for each node. The model for the R_K algorithm will be described in detail. However, the analysis of the R_{K_T}, R_D and R_{D_T} algorithms will only be described in brief, with a presentation of the main performance metrics.

The material in this chapter involves several Jacobi matrices, whose detailed definitions will be provided as described in Latouche [LATO81]. A matrix such as

$$\begin{bmatrix} b_0 & c_0 & 0 & 0 & & & \dots & \\ a_1 & b_1 & c_1 & 0 & & & \dots & \\ 0 & a_2 & b_2 & c_2 & & & & \\ \dots & \dots & & & a_{m-2} & b_{m-2} & c_{m-2} & 0 \\ \dots & \dots & & & 0 & a_{m-1} & b_{m-1} & c_{m-1} \\ \dots & \dots & & & 0 & 0 & a_m & b_m \end{bmatrix}$$

will be displayed as

$$\begin{Vmatrix} & c_0 & c_1 & \dots & c_{m-3} & c_{m-2} & c_{m-1} \\ b_0 & b_1 & b_2 & \dots & b_{m-2} & b_{m-1} & b_m \\ a_1 & a_2 & a_3 & \dots & a_{m-1} & a_m & \end{Vmatrix}$$

Figure 4.1 represents the state diagram for the R_K algorithm using an arbitrary threshold T. We define

$$y(n,j) = \lim_{t\to\infty} P(N_t = n,\ J_t = j),\ 0 \le n,\ 0 \le j \le K,$$

$$\mathbf{p}_n = (y(n,0), y(n,1), y(n,2), \dots\dots y(n,K)),\ 0 \le n,$$

$$\vec{\mathbf{p}} = (\mathbf{p}_0, \mathbf{p}_1, \mathbf{p}_2, \dots\dots \mathbf{p}_i, \dots\dots).$$

If the Markov process (N_t, J_t) is ergodic then $\vec{\mathbf{p}}$ is its steady state probability vector satisfying $\vec{\mathbf{p}}Q_K = 0$, where Q_K is the infinitesimal generator for this Markov process. Q_K has the structure of a block-tridiagonal matrix of the form

$$Q_K = \begin{Vmatrix} & A_0 & \dots & A_0 & A_0 & A_0 & A_0 & \dots \\ B_{00} & B_{11} & \dots & B_{11} & B_{11} & A_1 & A_1 & \dots \\ B_{10} & B_{10} & \dots & B_{10} & A_2 & A_2 & A_2 & \dots \end{Vmatrix}$$

where we define the matrices $B_{00}, B_{10}, B_{11}, A_2, A_1$ and A_0 in Appendix B.

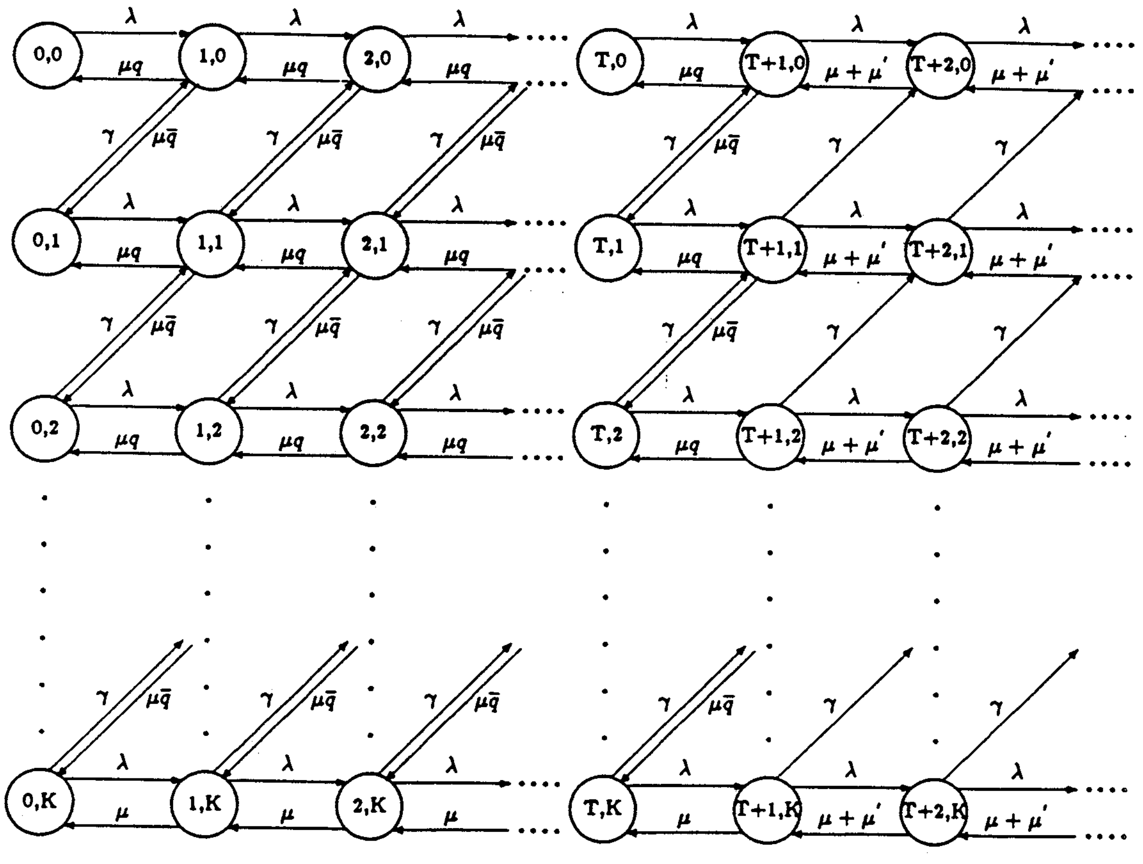

Figure 4.1: State Diagram for Algorithm R_K

In the subsequent discussion, q is the probability of failure in finding a remote job for a set of reverse probes, and $\overline{q} = 1 - q$.

When the node makes a transition below $T + 1$ on the completion of a job, it sends out reverse probes in order to get a remote job. A successful transition is represented by $\mu\overline{q}$ and an unsuccessful set of reverse probes is represented by the transition μq.

Thus, on the completion of a job when the node goes below $T+1$, it sends out reverse probes, if it is not already waiting for K remote jobs to arrive in response to earlier reverse probes. A transition of this type is represented from (n, j) to $(n-1, j+1)$, where $0 < n \leq T+1$ and $0 \leq j \leq K-1$. The rate at which this node sends out jobs in response to nodes that asked it for jobs is μ'. Thus, the rate at which a node makes the transition (n, j) to $(n-1, j)$, for $n \geq T+2$ equals $\mu + \mu'$.

As can be seen from the generator Q_K, the Markov process has a regular structure comprised of the A_0, A_1 and A_2 matrices, preceded however by the irregular boundary conditions. The size of the irregular portion of the matrix depends upon the threshold at which the process is operating. For example, there will be exactly $T - 1$ columns of the matrices (A_0, B_{11}, B_{10}).

Neuts [NEUT81] examined Markov processes with such generators and determined the conditions for positive recurrence when the infinitesimal generator $A = A_0 + A_1 + A_2$, corresponding to the geometric part of the Markov Process, is irreducible. However, for our problem, A is lower triangular and reducible. In such cases, the stability criterion has to be determined explicitly.

Consider the non-linear matrix equation

$$A_0 + RA_1 + R^2A_2 = 0$$

such that R is its minimal non-negative solution. It can be shown that R is lower triangular, given the structure of A_0, A_1 and A_2 [NEUT81]. Also, $R = [r_{i,j}]$ for arbitrary K can be determined by the following procedure. By solving K quadratic equations resulting from the matrix equation shown above, we can directly determine the values of the diagonal elements of R, as shown below.

$$\begin{aligned} r_{i,j} &= 0, \forall j > i \\ r_{1,1} &= \lambda/(\mu + \mu') \\ r_{i,i} &= \frac{\phi + \gamma - ((\phi + \gamma)^2 - 4(\mu + \mu')\lambda)^{1/2}}{2(\mu + \mu')}, \forall i > 1 \end{aligned}$$

where $\phi = \lambda + \mu + \mu'$.

Once the diagonal elements of R have been determined, the elements of the next lower diagonal can be determined from the solution of the main diagonal elements. This structure holds true for all subsequent diagonals as well (i.e., the elements of any diagonal depend upon the diagonals above that one) and this procedure is continued until all the diagonals are exhausted.

By adapting Theorem 1.5.1 from Neuts [NEUT81], the Markov process Q is positive recurrent if and only if $sp(R) < 1$ and that the matrix M (defined below) possesses a positive left invariant probability vector. Because R is lower triangular, its eigenvalues are its diagonal elements. One can show that $sp(R) < 1$ if

$$\lambda < \mu + \mu'$$

The matrix M, given by

$$\left\| \begin{array}{ccccc} & A_0 & \dots & A_0 & A_0 \\ B_{00} & B_{11} & \dots & B_{11} & B_{11} + RA_2 \\ B_{10} & B_{10} & \dots & B_{10} & \end{array} \right\|$$

is an irreducible, aperiodic matrix. The second condition holds because of the irreducibility of M. The vector $(\mathbf{p}_0, \mathbf{p}_1, ..., \mathbf{p}_{T+1})$ is the left eigenvector of M.

Intuitively, the stability condition means that the rate of processing jobs (including the ones that are sent out of this node) is greater than the total arrival rate of jobs into this node. Thus, on the average, whenever there are more than $T+1$ jobs at a node, the process drifts towards the boundary specified by the threshold T. Similar analysis may be carried out for the Forward and Reverse probing algorithms, with the appropriate substitution of parameters.

Intuitively, this means that the rate of processing jobs (including the ones that are sent out of this node) is greater than the total arrival rate of jobs into this node. Thus, on the average, whenever there are more than $T+1$ jobs at a node, the process drifts towards the boundary specified by the threshold T.

We now assume that the values of all the parameters are known. First, the boundary conditions are determined, by solving a system of linear equations. Thus, we have

$$(\mathbf{p}_0, \mathbf{p}_1, ..., \mathbf{p}_{T+1}) \left\| \begin{array}{ccccc} & A_0 & ... & A_0 & A_0 \\ B_{00} & B_{11} & ... & B_{11} & B_{11} + RA_2 \\ B_{10} & B_{10} & ... & B_{10} & \end{array} \right\| = 0$$

where the number of columns in the matrix is exactly $T+1$. We know from Neuts [NEUT81] that

$$\mathbf{p}_i = \mathbf{p}_{T+1} R^{T+1-i}, \forall i \geq T+1$$

Thus,

$$\sum_{i \geq T+1} \mathbf{p}_i = \mathbf{p}_{T+1}(I - R)^{-1}$$

Also,

$$[\sum_{i=0}^{T} \mathbf{p}_i + \mathbf{p}_{T+1}(I - R)^{-1}]e = 1$$

where R is the minimal solution of

$$A_0 + RA_1 + R^2 A_2 = 0$$

with $R \geq 0$ and the spectral radius of $R, sp(R) < 1$, and I is the identity matrix.

$E[N]$, the expected number of jobs at a node, and $E[D]$, the expected response time of a job, are given by the following expressions:

$$\begin{aligned} E[N] &= \sum_{i \geq 1} i\, \mathbf{p}_i\, e \\ &= \mathbf{p}_{T+1}(I-R)^{-2}e + T * [\mathbf{p}_{T+1}(I-R)^{-1}e] + \sum_{i=1}^{T} i\mathbf{p}_i e \\ E[D] &= \frac{(E[N] + \frac{(Flow-In)}{\gamma})}{\lambda} \end{aligned}$$

where $Flow - In$ is the flow into a node of remote jobs due to reverse probes. In the next subsection, we derive the equations required to determine the values of the unknowns q, and μ' and describe the iterative algorithm used to solve the resulting model.

4.3.1 Computational Procedure

Initially, it is assumed that the values for q and μ' are known and the model is solved using these values. In a typical step, a model solution is used to derive new values for q, and μ', and a new solution is computed. The iteration procedure that we use is described in a step-wise form, after the following definitions.

- $RFRO$: Flow rate out of jobs, as a result of reverse probes made by other nodes to this node.
- $RFRI$: Flow rate in of jobs, as a result of reverse probes made by this node.

Iteration Procedure

In the iteration procedure described below, i denotes the iteration count. $q^{(0)}, \mu'^{(0)}$ and $RFRO^{(0)}$ represent the initial values selected for the unknowns.

1. Let $i = 0$; choose values for $q^{(0)}, \mu'^{(0)}, RFRO^{(0)}$

2. Determine $Q_K^{(i)}$ from $q^{(i)}, \mu'^{(i)}$

3. Determine $R^{(i)}$

4. Solve the linear system corresponding to the boundary conditions

5. Determine $RFRO^{(i+1)}$ from the model solution

6. If $ABS(RFRO^{(i+1)} - RFRO^{(i)}) \leq \epsilon$, where ϵ is an arbitrary small number, stop, else

7. Let $i = i + 1$. Go to 2

We have observed from experiments, that the solution was insensitive to the initial values chosen for the unknown quantities. Consequently, we conjecture that there exists a unique solution to the model. Further, the number of iterations was usually small, ranging between 10 and 30.

Because of the assumption of homogeneity and because of the principle of equivalence of flow:

$$RFRO = RFRI.$$

$$RFRI = \sum_{i \geq 0} \mathbf{p}_i \, [011...1]^T \, \gamma,$$

where $1/\gamma$ is the mean delay in receiving a remote job. Thus, $RFRI$ denotes the total flow in due to reverse probes made by this node.

$$RFRO = \mu' \sum_{i > T+1} \mathbf{p}_i \, e$$

Thus,

$$\mu' = \frac{RFRI}{\sum_{i > T+1} \mathbf{p}_i \, e}$$

To determine q, the probability of a set of reverse probes resulting in failure, we use the following procedure:

Let

$$y = \sum_{i \leq T+1} \mathbf{p}_i e$$

If the node probes L_p nodes to receive a remote job, the the probability that all of them will be unsuccessful is denoted by: $q = y^{L_p}$, and $\bar{q} = 1 - q$ is the probability that at least one of the reverse probes is successful.

Algorithm R_{K_T}

Figure 4.2 depicts the birth-death process for the R_{K_T} algorithm, corresponding to a threshold T. Let $N_t^{(i)}$ be the number of jobs at node i at time t and $J_t^{(i)}$ be the probe state of node i, at time t and i be the condition code that indicates whether the node is not probing, or if it is probing, then how many remote jobs are pending. The codes are as follows:

- 0 : if not probing,
- k : if reverse probing and waiting for k jobs.

As can be seen from Figure 4.2, the only time the node sends out reverse probes in when a transition is made from $(T+1, i)$ to $(T, i+1)$ on the completion of a job, $\forall i < K$. The infinitesimal generator for the Markov process corresponding to R_{K_T}, $\forall T \geq 1$, has the following form:

$$Q_{K_T} = \left\| \begin{array}{cccccccc} & A_0 & \dots & A_0 & A_0 & A_0 & A_0 & A_0 & \dots \\ B_{00} & B_{11} & \dots & B_{11} & B_{11} & B_{11} & A_1 & A_1 & \dots \\ B_{10} & B_{10} & \dots & B_{10} & B_{20} & A_2 & A_2 & A_2 & \dots \end{array} \right\|$$

with exactly $T-1$ columns of (A_0, B_{11}, B_{10}). For $T = 0$, the generator takes the following form:

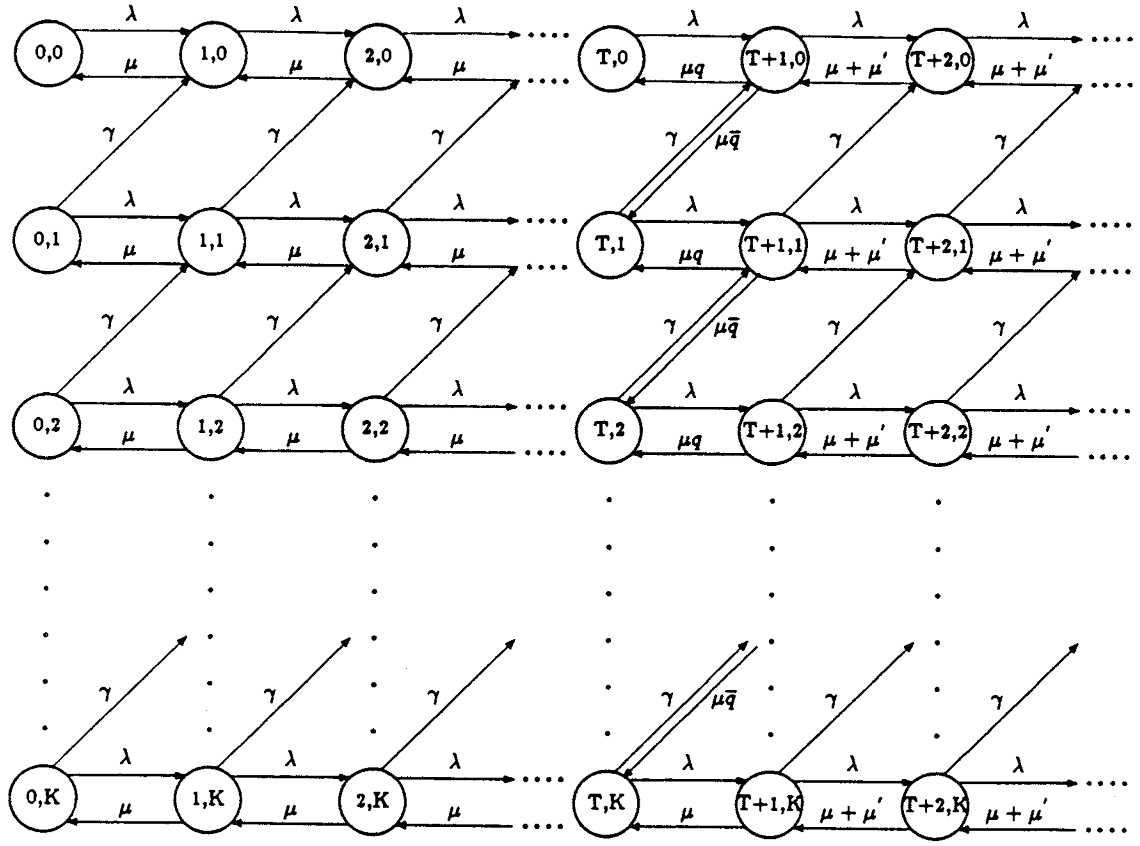

Figure 4.2: State Diagram for Algorithm R_{K_T}

$$Q_{K_T} = \left\| \begin{array}{cccccc} & A_0 & A_0 & A_0 & \dots \\ B_{00} & B_{11} & A_1 & A_1 & \dots \\ B_{20} & A_2 & A_2 & A_2 & \dots \end{array} \right\|$$

Algorithm R_{T^2}

Figure 4.3 depicts the birth-death process for the R_{T^2} algorithm. As can be seen from the figure, the process has two thresholds, T_p and T_t, which represent the threshold at which probing can be performed and the threshold above which a node can transfer a job, respectively. Thus, when a job completes at a node and the remaining number of jobs is $\leq T_p$ and the node is not already waiting for a job, it sends out reverse probes. Upon receiving a probe, a node may agree to transfer a job if it possesses at least $T_t + 2$ jobs.

The infinitesimal generator for the Markov process corresponding to R_{T^2} has the following form:

$$Q_{T^2} = \left\| \begin{array}{cccccccccc} & A_0 & \dots & A_0 & A_0 & \dots & A_0 & A_0 & A_0 & A_0 & \dots \\ B_{00} & B_{11} & \dots & B_{11} & B_{11} & \dots & B_{11} & B_{11} & A_1 & A_1 & \dots \\ B_{10} & B_{10} & \dots & B_{10} & B_{20} & \dots & B_{20} & A_2 & A_2 & A_2 & \dots \end{array} \right\|$$

with $T_p - 1$ columns of A_0, B_{11}, B_{10} and $T_t - T_p$ columns of A_0, B_{11}, B_{20}. The computation procedure for this algorithm is identical to that for algorithm R_K.

4.3.2 Non-Zero Probing Times

Figures 4.4 and 4.5 depict the birth-death process for the R_D and R_{D_T} algorithms respectively, corresponding to a threshold T. It is assumed that the job arrival process at each node is Poisson, with parameter λ. Also, the service times and job transfer times are assumed to be exponentially distributed, with means $1/\mu$ and $1/\gamma$, respectively. In the previous algorithms R_K and R_{K_T}, negative probes took zero time. We relax this assumption in this model. The time to receive a negative

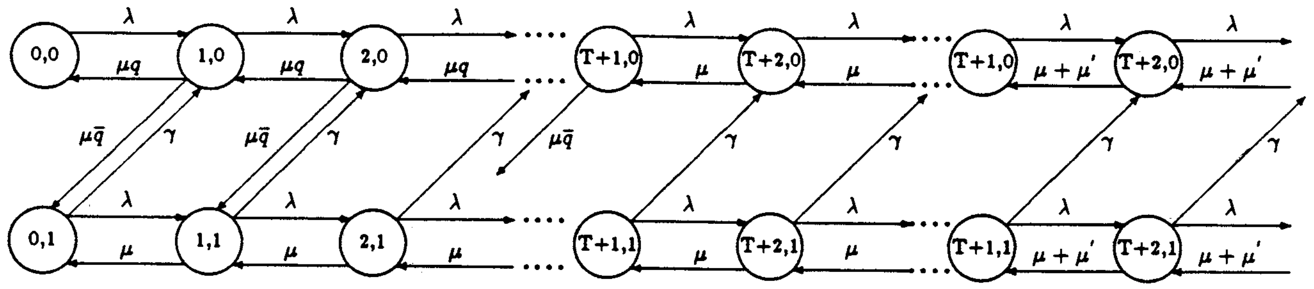

Figure 4.3: State Diagram for Algorithm R_{T^2}

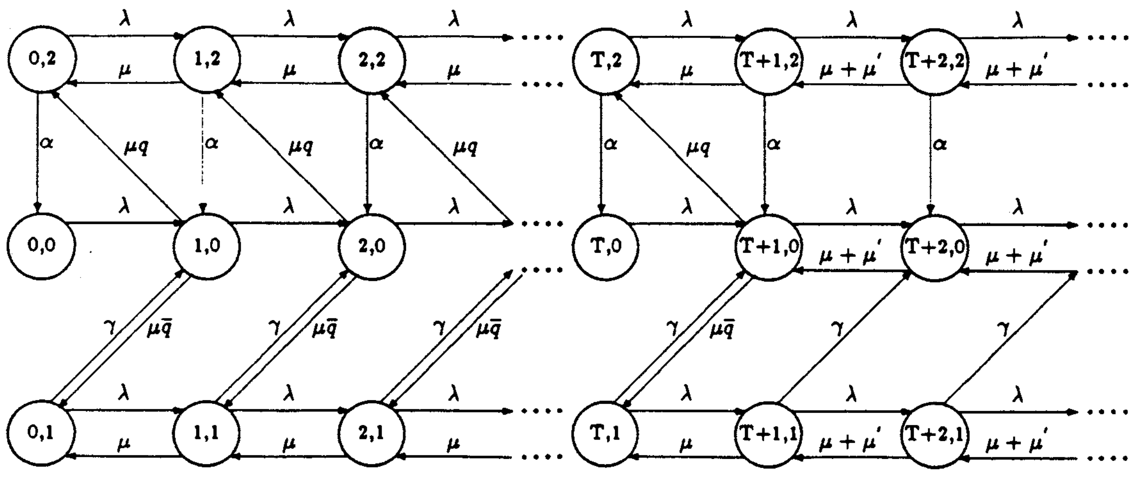

Figure 4.4: State Diagram for Algorithm R_D

reply to a set of reverse probes is exponentially distributed, with mean $1/\alpha$. It is assumed that α is independent of the number of probed nodes.

Let $N_t^{(i)}$ be the number of jobs at node i at time t and $J_t^{(i)}$ be the probe state of node i, at time t and i be the condition code that indicates whether the node is not probing, or if the node is probing and that the probe will result in a job transfer, or that the probe will not result in a job transfer. The codes are as follows:

- 0 : if not probing,
- 1 : if reverse probing and waiting for a job,
- 2 : if reverse probing and waiting for a negative reply.

The infinitesimal generator for the Markov process corresponding to the R_D algorithm has the following form:

$$Q_D = \left\| \begin{array}{cccccccc} & A_0 & \dots & A_0 & A_0 & A_0 & A_0 & \dots \\ B_{00} & B_{11} & \dots & B_{11} & B_{11} & A_1 & A_1 & \dots \\ B_{10} & B_{10} & \dots & B_{10} & A_2 & A_2 & A_2 & \dots \end{array} \right\|$$

The infinitesimal generator for the Markov process corresponding to R_{D_T}, $\forall T \geq 1$ has the following form:

$$Q_{D_T} = \left\| \begin{array}{cccccccc} & A_0 & \dots & A_0 & A_0 & A_0 & A_0 & \dots \\ B_{00} & B_{11} & \dots & B_{11} & B_{11} & A_1 & A_1 & \dots \\ B_{10} & B_{10} & \dots & B_{10} & A_2 & A_2 & A_2 & \dots \end{array} \right\|$$

with exactly $T-1$ columns of (A_0, B_{11}, B_{10}). The form for $T = 0$ is as follows:

$$Q_{D_T} = \left\| \begin{array}{ccccc} & A_0 & A_0 & A_0 & \dots \\ B_{00} & B_{11} & A_1 & A_1 & \dots \\ B_{20} & A_2 & A_2 & A_2 & \dots \end{array} \right\|$$

The computational procedure for both these algorithms is very similar to that for the R_K algorithm, which was described in detail in the previous subsection.

Also, the parameters of the Markov Processes corresponding to these algorithms have the same meanings as in R_K and R_{K_T}.

Initial values of the unknown parameters are assumed to solve the model. Based upon this solution, new values of the parameters are determined. The iteration continues until the stopping criterion has been satisfied. It was seen that the iteration was insensitive to the initial values unknown parameters. Further, the number of iterations was usually small, between 10 and 20.

4.4 Performance Comparisons

In this section, the performance of the Receiver-Initiated load sharing algorithms will be compared to each other and to two bounds, represented by the no load balancing $M/M/1$ model for K nodes (also referred to as NLB) and the perfect load sharing with zero costs, i.e., the $M/M/K$ model. Wherever relevant, we will also compare the algorithms against a Random assignment algorithm, which transfers jobs based only upon local state information. The key performance metric for comparison is the mean response time of jobs.

A large number of parameters such as the service time, the threshold T, the probe limit L_p, the job transfer delay $1/\gamma$, the number of nodes in the network etc., can affect the performance of load sharing algorithms. In this connection, we will try to present the results that we believe are the most relevant. The presentation will be in the following sequence:

- Validation of the analytical results with simulations.
- Selection of parameter K.
- Comparison between R_K and R_{K_T}.
- Effect of non-zero negative probe times.
- Relation between response time and thresholds.

- Effects of multiple thresholds
- Network traffic density

Unless specifically mentioned otherwise, $L_p = 2$ in all the runs. Also, $S = 1/\mu$, $D = 1/\gamma$ and $A = 1/\alpha$ are the means of the service time, the job transfer delay and negative probe delay, respectively. Further, it will be assumed that $S = 1$ unit and all measurements of time will be in terms of this unit.

Validation with Simulations

We mentioned in Section 4.2 that the decomposition used in this chapter is only an approximate solution which is conjectured to be exact for infinitely large systems. Thus, it is important to determine how well this approximation compares to simulations of finite sized systems. The simulation model consisted of 10 nodes in all cases except for $\rho = 0.9$, when the model consisted of 20 nodes. Figure 4.6 depicts a representative set of curves regarding this study.

The results presented here correspond to algorithm R_1 (i.e. $K = 1$), but the conclusions are generally representative of all five of the algorithms described in this chapter. Because the simulation results were almost identical to the results of the analytical model, we have chosen not to depict the actual sample means of the response times from the simulations. Instead, the 95% confidence intervals of the simulation results are presented, as computed by the Student-t tests. The curves correspond to the results obtained from the analytical model. On the average, the confidence interval for the response time is less than $\pm 3\%$ about the sample mean. The only exception to this is at $\rho = 0.9$, when the confidence interval is about $\pm 5\%$ about the sample mean.

We have observed (results not presented here) that in most of the cases, the variation between the simulation results and the analytical models is less than 2%. Furthermore, the model is invariably optimistic, compared to the simulation results. In any case, for loads ≤ 0.8, the model is a very good approximation, even for reasonably small systems. In cases where the variation is more than 2%, it was seen that by increasing the size of the simulation system to 20 nodes, the results generate better agreement with those of the analytical model. For instance, the variation at $\rho = 0.9, D = 0.1S$, which was about 10% (results not depicted in figure), decreases to less than 5% for a system of 20 nodes.

Selection of K

In this set of tests, we determine the effects of varying K, the maximum number of pending remote jobs at a node. We would like to determine when, if at all, there is any significant gain in sending out probes while waiting for one or more remote jobs to arrive. Thus, we have compared the performance of R_1, R_2, R_3 and R_4 under a variety of conditions.

Figure 4.7 depicts the effect of K on optimal response times generated by various values of K for delays $0.1S, S$ and $10S$. The load under consideration is 0.8. Also shown is the NLB response time. As can be seen, the effect of varying K is almost negligible. The best improvement over R_1 is less than 1% and this occurs at delay of S. In fact, we have observed very similar behavior for all loads less than 0.8. At $\rho = 0.9$ (results not depicted here), marginally better performance (about 4%) is exhibited by R_3 at delay $= S$. For delay $= 10S$, the improvement is about 2.5%. The conditional mean number of pending probes (conditioned on the event that at least one probe is pending), $E[P]$, was computed for all the tests.

The insensitivity to K may be explained by the following reasons:

- At low delays, remote jobs arrive much quicker than the node is able to complete a job (and send out another probe, as a result). For $\rho = 0.8$, and delay $= 0.1S$, it was seen that $E[P] = 1.0044$ $(K = 4)$, for the optimal threshold, which is zero.

- At moderate to high delays $E[P] = 1.242$ $(K = 4, \textit{Delay} = S)$ and $E[P] = 1.636$ $(K = 4, \textit{Delay} = 10S)$. However, the effect of the corresponding increase in load sharing is in all likelihood balanced out by the added costs due to high transfer delays.

Thus, we can conclude that the effect of K on optimal response times is negligible. In the following discussion, K will equal 1, unless mentioned otherwise.

Comparison between R_1 and R_{1_T}

The results of performance comparison between R_1 and R_{1_T} is depicted in Figures 4.8 and 4.9. The loads represented in the graphs are 0.6 and 0.9. The job transfer delay in Figure 4.8 is $0.1S$ while in Figure 4.9 it is $10S$. It is seen that at low loads, the performance of the two schemes is almost identical for the most part, with R_1 being marginally superior in some cases, especially at low delays. This can be explained by the fact that at low delays, R_{1_T} is unable to take as much advantage of the low cost of job transfer as is R_1.

For high loads, R_1 appears significantly better than R_{1_T}, for low as well as high delays. For low job transfer delays, the same argument applies as in the earlier paragraph. Thus, we will not study the performance of the threshold probing variations of the algorithms any further in this chapter.

Effect of non-zero negative probe times

Figures 4.10 and 4.11 depict the effect of non-zero negative probe times on R_D. The loads under consideration are 0.5 and 0.9. The job transfer delay corresponding to the results in Figure 4.10 is $10S$ and in Figure 4.11, the delay is $0.1S$. Also shown in the figures are the baseline results corresponding to zero negative probe times.

At high transfer delays and low loads ($\rho = 0.5$), it is seen that the effect of non-zero probe times are not significant. The dominant effects are due to the job transfer delays. As a matter of fact, the response times actually become slightly worse as the negative probes arrive at higher speeds. However, for high loads, ($\rho = 0.9$), there is about 15% improvement by increasing the rate of negative responses. However, in both cases, the performance of R_D approaches that of R_1 when $A \leq 1/10D$.

The effects of non-zero probe times appear more prominent at small job transfer delays $= 0.1S$, as depicted in Figure 4.11. This is especially evident at $\rho = 0.9$

where the slow negative probes cause a significant deterioration in performance. Here again, R_D approaches R_1 when $A \leq 1/10D$. Thus, it appears that as long as the average probing time is less than $1/10th$ the average job transfer time, the system essentially behaves like one with zero probing time. This brings us back to the argument that probes being much smaller entities than jobs (one packet of information as opposed to several hundreds or thousands in the case of jobs), it would not be unreasonable to believe that probes might take a fraction of the transfer time associated with jobs.

Response Time vs. Thresholds

Figures 4.12, 4.13 and 4.14 show the response times for the R_1 algorithm tested over a wide range of communication delays and thresholds, for the loads of 0.5, 0.7 and 0.9. It can be seen from Figure 4.12, that at low delays ($D = 0.1S$) and low to moderate loads (≤ 0.7), the optimal threshold is 0 and the performance is a monotonically increasing function of the threshold. Also, the response time generated at $T = 0$ is only about 50% better than the NLB value for moderate loads ($\rho \leq 0.7$). For example, at $\rho = 0.7$, the R_1 response time is about 1.7 units while the corresponding NLB is 3.33 units. However, at low to moderate loads, there is significant room for improvement as compared to the $M/M/K$ model, which produces a response time of about 1.04 units at $\rho = 0.7$. These observations can be explained by the following arguments:

- At low delays, the cost of transferring a job is much lower than the potential improvement due to the effect of load sharing. Thus, $T = 0$ permits very active load sharing.

- Because the delays are small, much greater certainty exists in the knowledge that an idle node will continue to remain idle during the time it takes to transfer a job to it. Thus, in some sense, $T = 0$ ensures that all job transfers are useful in that a remote job arrives at the node soon after it becomes idle.

For moderate delays ($D = S$, Figure 4.13), the behavior is as follows: Even at $\rho = 0.5$, there is a gain of about 15% over the corresponding NLB performance. The response time for the NLB is 2 units while the algorithm generates about 1.7 units. The improvements at higher loads are even more substantial, being as high as 66% for $\rho = 0.9$. The response time values for the algorithm and NLB in this case are 3.4 units and 10 units respectively. However, as might be expected the results do not compare very favorably against the $M/M/K$ model, which generates a response time of 1.3 units, for $\rho = 0.9$. Further, $T = 1$ for $\rho = 0.5$ and 0.7, while $T = 2$ for $\rho = 0.9$, are the *optimal* thresholds.

When the communication delays increase to the order of $10S$ (Figure 4.14), it is seen that the best that can be achieved for $\rho = 0.5$ is the NLB performance. Thus, it would be appropriate to turn off load sharing here. For $\rho = 0.7$, a small gain of about 5% is seen, at $T = 6$. This improvement is small enough that if the interference of probes could be accounted for, the best strategy might very likely be to turn off load sharing. However, for $\rho = 0.9$, the reduction in response time over the corresponding $M/M/1$ is about 35% (6.5 units as opposed to 10 units for NLB) and this occurs at $T = 8$.

Effects of Multiple Thresholds

To determine the usefulness of multiple thresholds, we studied the R_{T^2} algorithm over a wide range of loads and delays. The loads were varied between 0.5 and 0.9 and the delays ranged from $0.01S$ to $100S$. Initially, the optimal performance generated by algorithm R_1 (the single threshold counterpart of R_{T^2}) for the above loads and delays was recorded. The results for algorithm R_{T^2} indicated that the optimal threshold-pair (T_p, T_t) (corresponding to probing and job transfers) resided in the near neighborhood of the original threshold T (generated by R_1), as might be expected.

From the results of the model (details not presented here), it was seen that for low to moderate delays, the pair (T_p, T_t) was identical to T. This was seen to occur for all loads tested until delays of about $5S$. At delay $= 10S$, and $\rho = 0.7$, $T = 5$, the optimal performance was generated by $T_p = 5$ and $T_t = 6$. At

$\rho = 0.9$ and delay $= 10S$, $T = 7$ for R_1 whereas $T_p = 7$ and $T_t = 8$ generated the optimal performance. However, the response time improvements in both these cases were almost insignificant, being less than 0.25%. As the delays were increased, the pattern was very similar, with little or no improvement being noticed over the single threshold algorithm. Thus, we can conclude that there appears to be no benefit in utilizing dual thresholds for the types of load sharing policies we have studied.

Network Traffic Density

Figure 4.15 depicts the effects of transfer delays on the amount of network traffic generated by the nodes running algorithm R_1 for loads 0.5, 0.7 and 0.9. For each load, two curves are presented, one depicting the rate at which a node generates probes and other the rate at which jobs are transferred, which is the same as the flow into or out of a node. These results correspond to the optimal behavior generated by this algorithm under the delays indicated. For $\rho = 0.5$ and 0.7, we can see the following behavior: The job flow rate drops to zero at about $10S$ for $\rho = 0.5$ and at about $40S$ for $\rho = 0.7$. The probe curves for these loads follow a more or less increasing function until the delays corresponding to zero job transfer rate are reached. At this point, the curves stabilize at the value corresponding to the following equation:

$$\tau = L_p \mu \sum_{i=1}^{T+1} \mathbf{p}_i[10]^T,$$

where τ is the probe rate for a node at a particular load. The optimal thresholds for extremely high delays (about $40S$) and loads ≤ 0.7 are very high (in essence, they are infinite because no load sharing is being performed). Thus, every time a job completes, a node sends out reverse probes.

At first glance, the behavior for load of 0.9 appears to violate the above rules. One is inclined to believe that probe rates should increase with loads, for high delays. However, this is not borne out by the probe curve for $\rho = 0.9$. The reason for this behavior is the following: As the delays increase, the optimal performance produces very little load sharing. However, as long as the flow rate of jobs is greater than zero, a large fraction of the time is spent waiting for remote jobs to arrive and consequently the nodes do not send out probes upon completion of jobs (recall that for algorithm R_K, there can be at most K pending remote jobs. In this set of tests, we have set $K = 1$). From Figure 4.15, we can see that even at delay $= 100S$, the flow rate of jobs is about 0.01 units. From the above equation, we can predict that at even larger delays, when load sharing at $\rho = 0.9$ will cease completely, the probe rate will stabilize at 1.8 units. Thus, Receiver-Initiated load sharing algorithms appear to have the shortcoming that even though no load sharing is performed at very high delays, the nodes continue to generate probes at a very high rate. These probes could potentially interfere with other traffic on the network. Sender-Initiated load sharing algorithms do not possess this property [TOWS87].

4.5 Summary and Conclusions

This study was concerned with the performance analysis of Receiver-Initiated load sharing policies in the presence of job transfer and probing delays. The algorithms that we tested were called $R_K, R_{K_T}, R_D, R_{D_T}$ and R_{T^2}. All of the above algorithms were tested over a large range of parameter values. In addition to the job transfer delays, R_D and R_{D_T} were subjected to the effects of probing delays.

The analysis of the distributed load sharing algorithms was carried out using the Matrix-Geometric solution technique. Because the Markov process of the entire network appeared to be computationally intractable, the system was solved by decomposing the state of the Markov process. This decomposition resulted in an approximate solution for the original Markov process. However, comparisons with simulation studies of 10-20 node systems indicated that the decomposition was very accurate, with the variation being less than 5%.

Some of the key observations that we made from our studies were as follows:

- In Chapter 3, we had restricted the value of K (the maximum number of pending remote jobs at a node) to exactly one. The motivation for this was the resulting simplicity, because in the general case, the value of K could be infinite. In this chapter, we studied the effect of varying K on algorithm R_K and concluded that under most values of parameters, $K = 1$ is a very good approximation for the general case.

- In comparison studies between R_1 and R_{1_T}, it was seen that when the transfer delays were small, R_1 outperformed R_{1_T} quite convincingly, over most of the range of loads tested. This effect is more prominent at high loads and low to moderate delays because more active load sharing can be performed.

- In order to simplify the analysis for the R_K and R_{K_T} algorithms, we had assumed that probes took zero time, while jobs were subject to transfer delays. Subsequently in this study, we relaxed this assumption and determined the effect of probing delays. In the context of reverse probing schemes, this meant that negative replies to probes took non-zero times. In all the studies that we conducted for this aspect of the problem, it was seen that as long as probing time was less than $1/10th$ of the job transfer times, the system essentially behaved like one with zero probing times. We postulate that that probing will take a small fraction of the job transfer time, because of the possible relative sizes of these two entities. Thus, it seems reasonable to assume that probes take zero time.

- A representative study of algorithm R_1 over a large range of loads and delays was performed. It was seen that the most significant gain in performance over NLB was seen at high loads (≥ 0.8). At low to moderate delays, load sharing was viable even for low loads. At very high loads, $\rho \geq 0.9$ there appeared to be a substantial benefit from load sharing, even when delays were very high, as much as $10S$.

- We studied the effects of dual thresholds on an algorithm called R_{T^2} and noticed that very little or no benefits accrued from this change.

- We studied the effects of delays on network traffic densities. It was seen that as delays increase, the optimal performance generates fewer and fewer job transfers, with complete cessation of load sharing when delays become very high. However, nodes continue to generate probes at a fairly high rate (which stabilizes after a specific value of delay which is dependent upon load). This would appear to be a shortcoming of Receiver-Initiated load sharing algorithms.

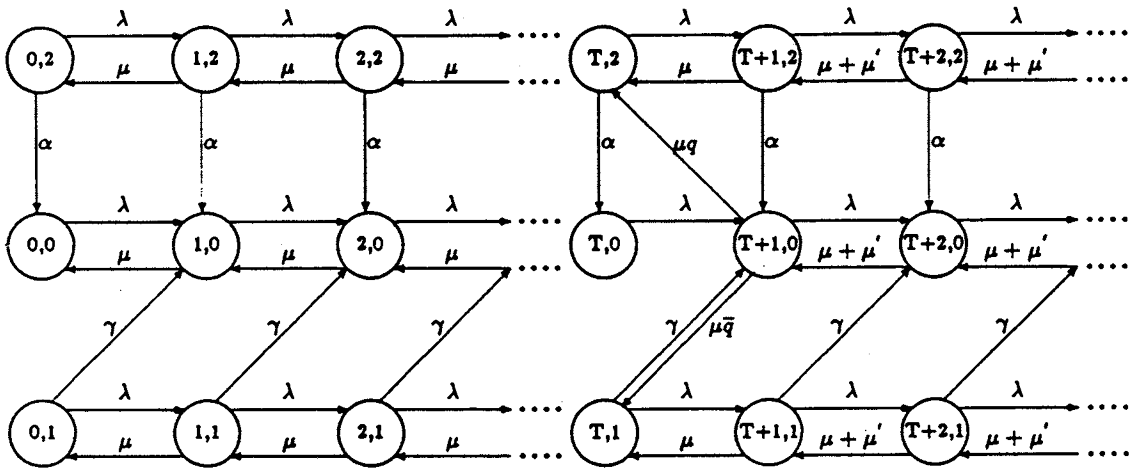

Figure 4.5: State Diagram for Algorithm R_{D_T}

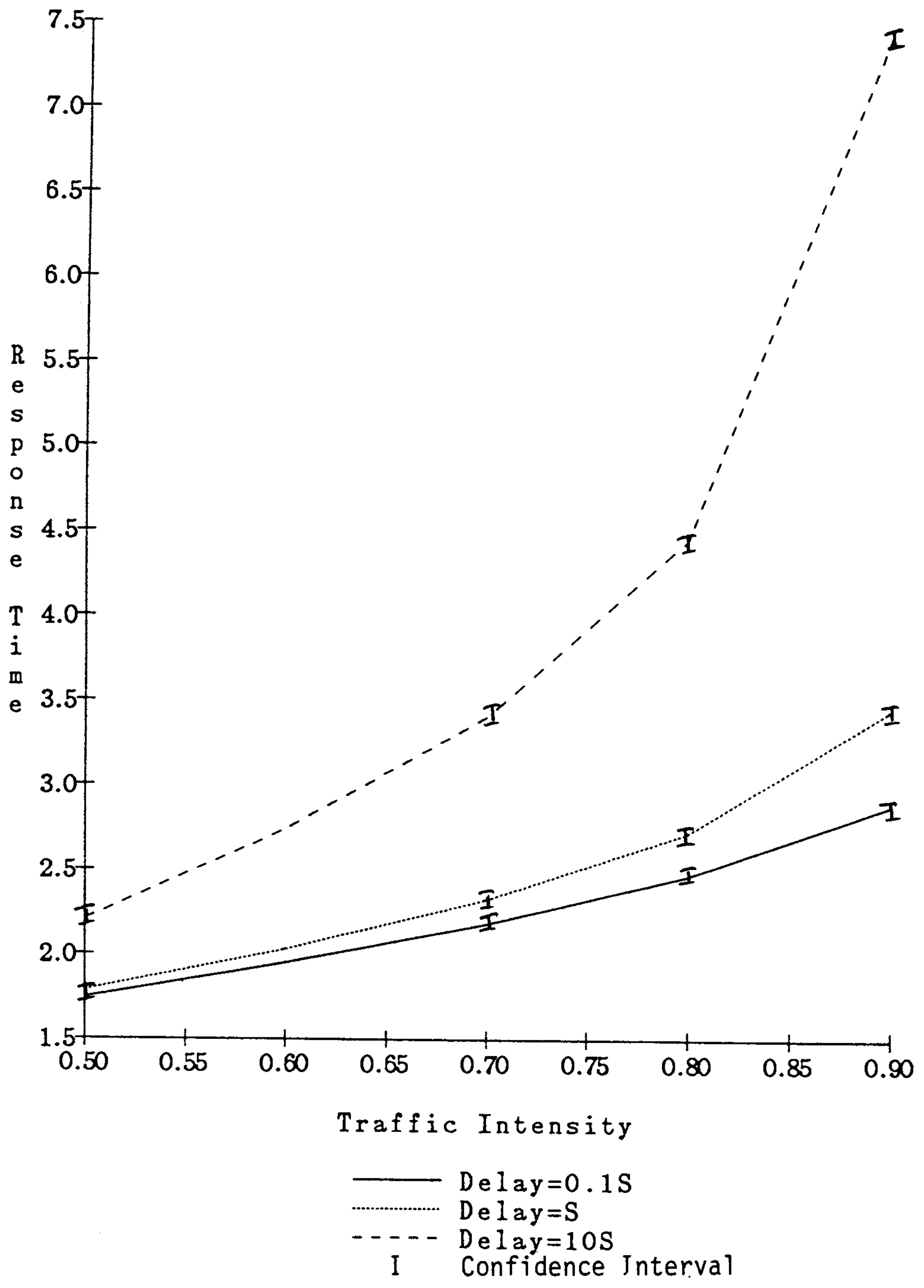

Figure 4.6: Comparison with Simulations

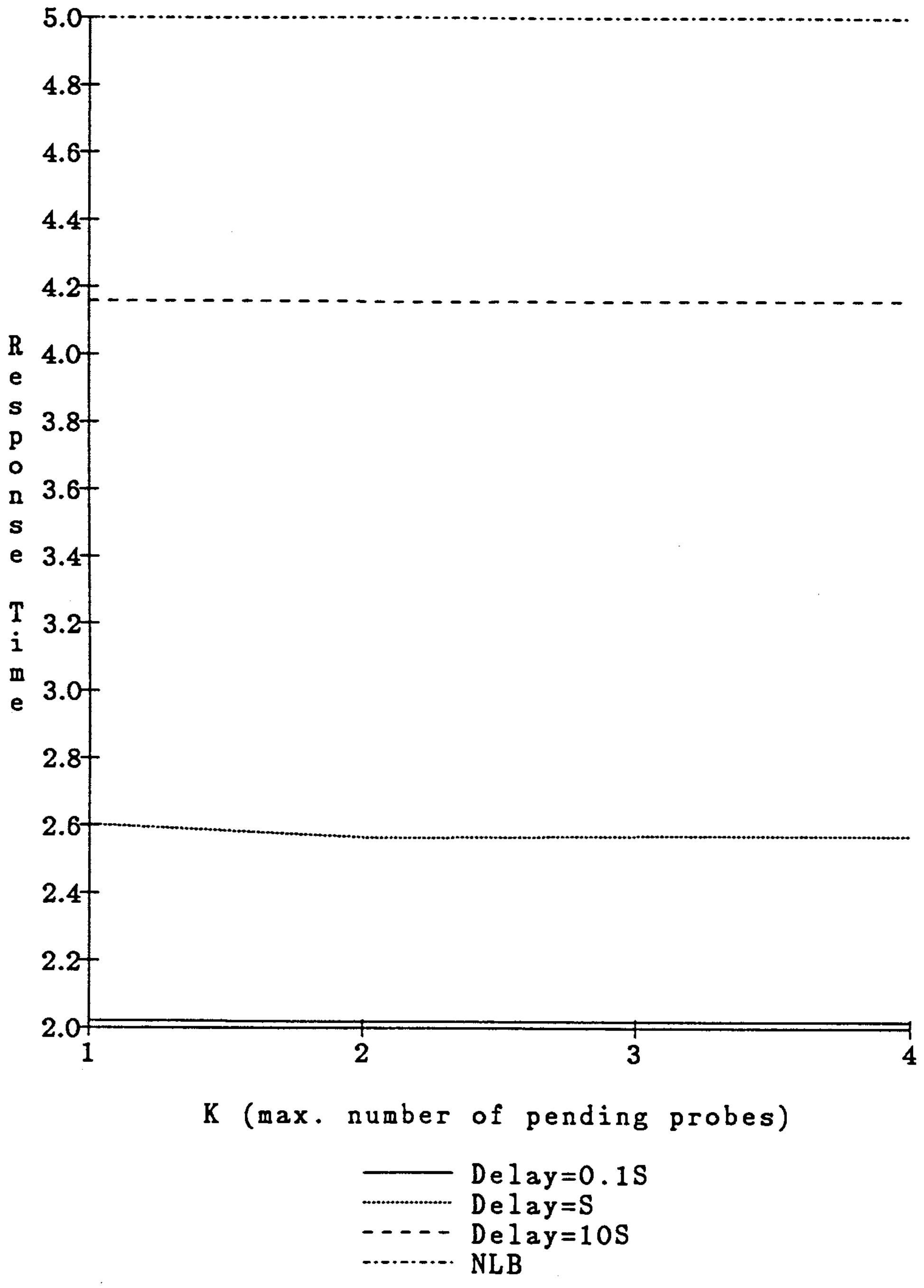

Figure 4.7: Optimal R.T. vs. K (ρ=0.8)

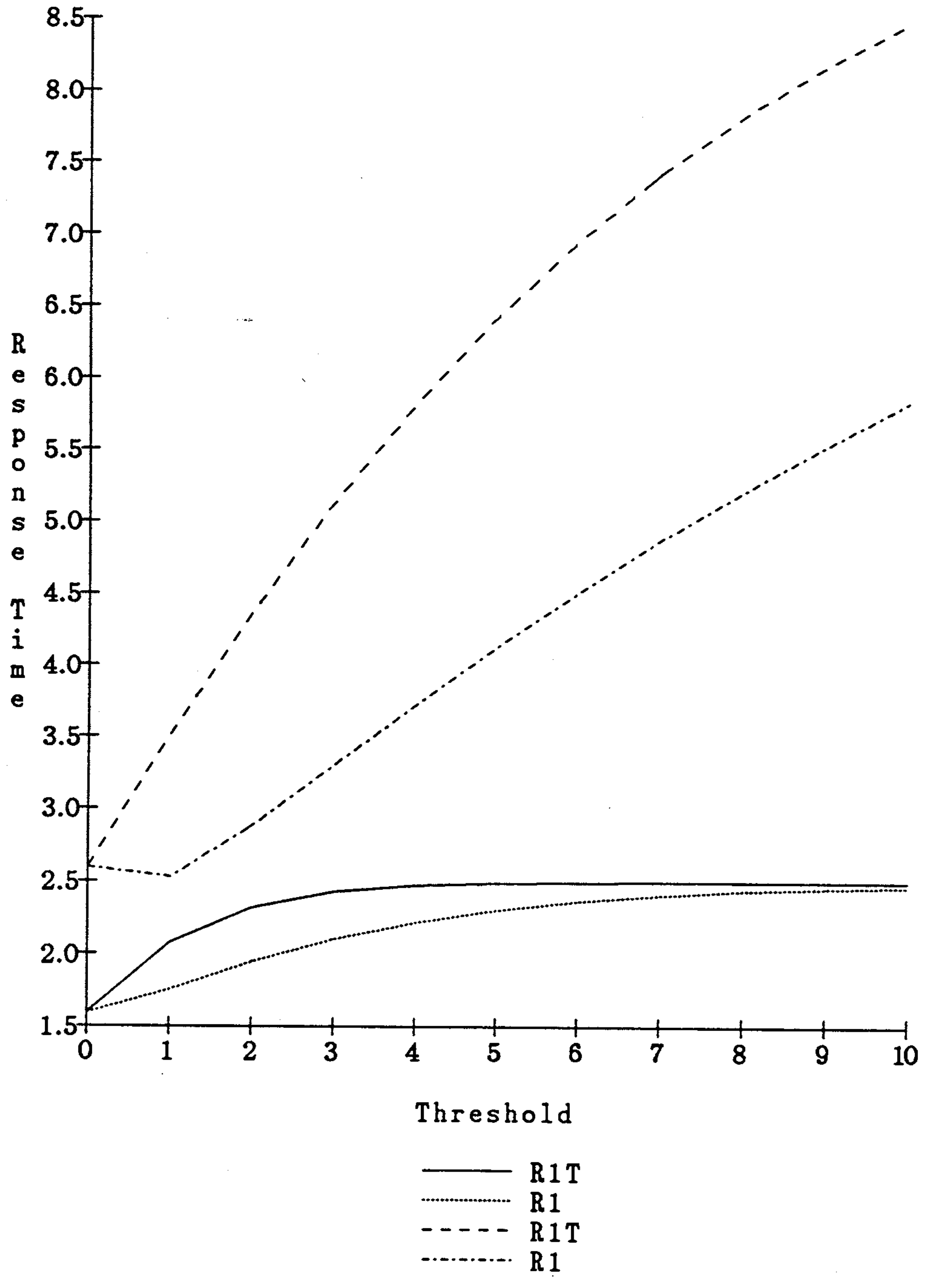

Figure 4.8: R_1 vs. R_{1_T} (Delay=0.1S)

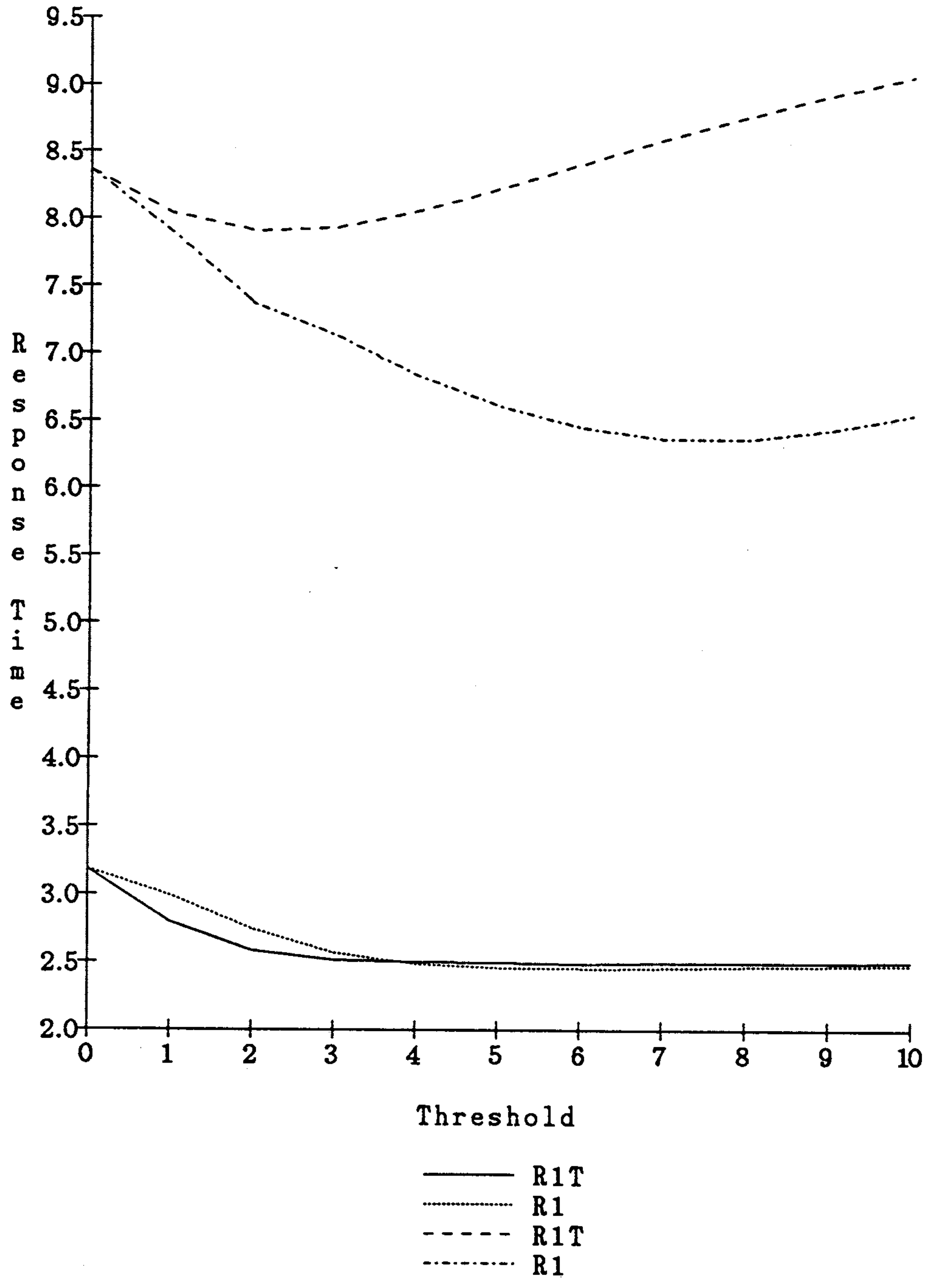

Figure 4.9: R_1 vs. R_{1_T} (Delay=$10S$)

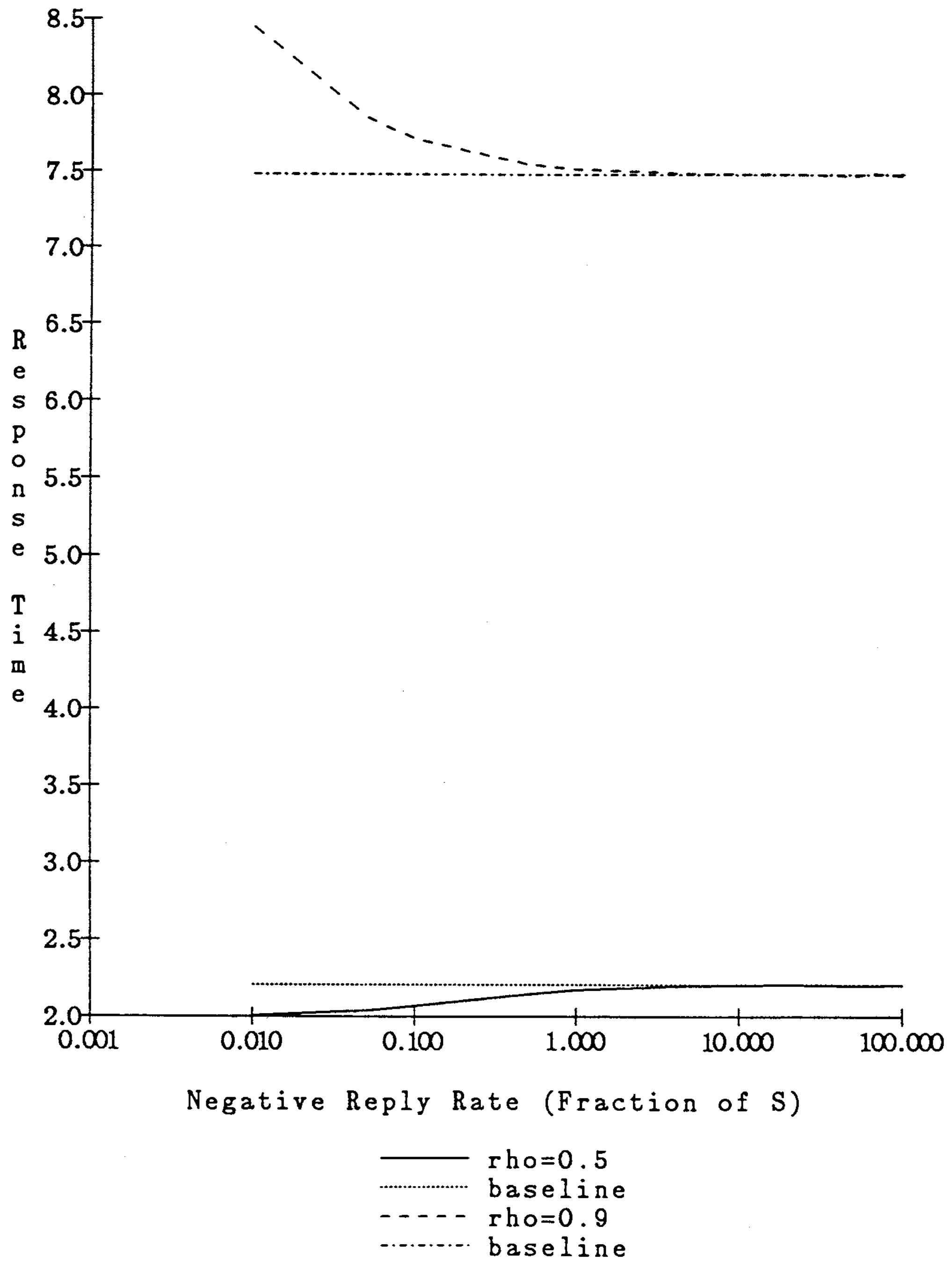

Figure 4.10: Effect of Non-Zero Probe Times (Delay=10S)

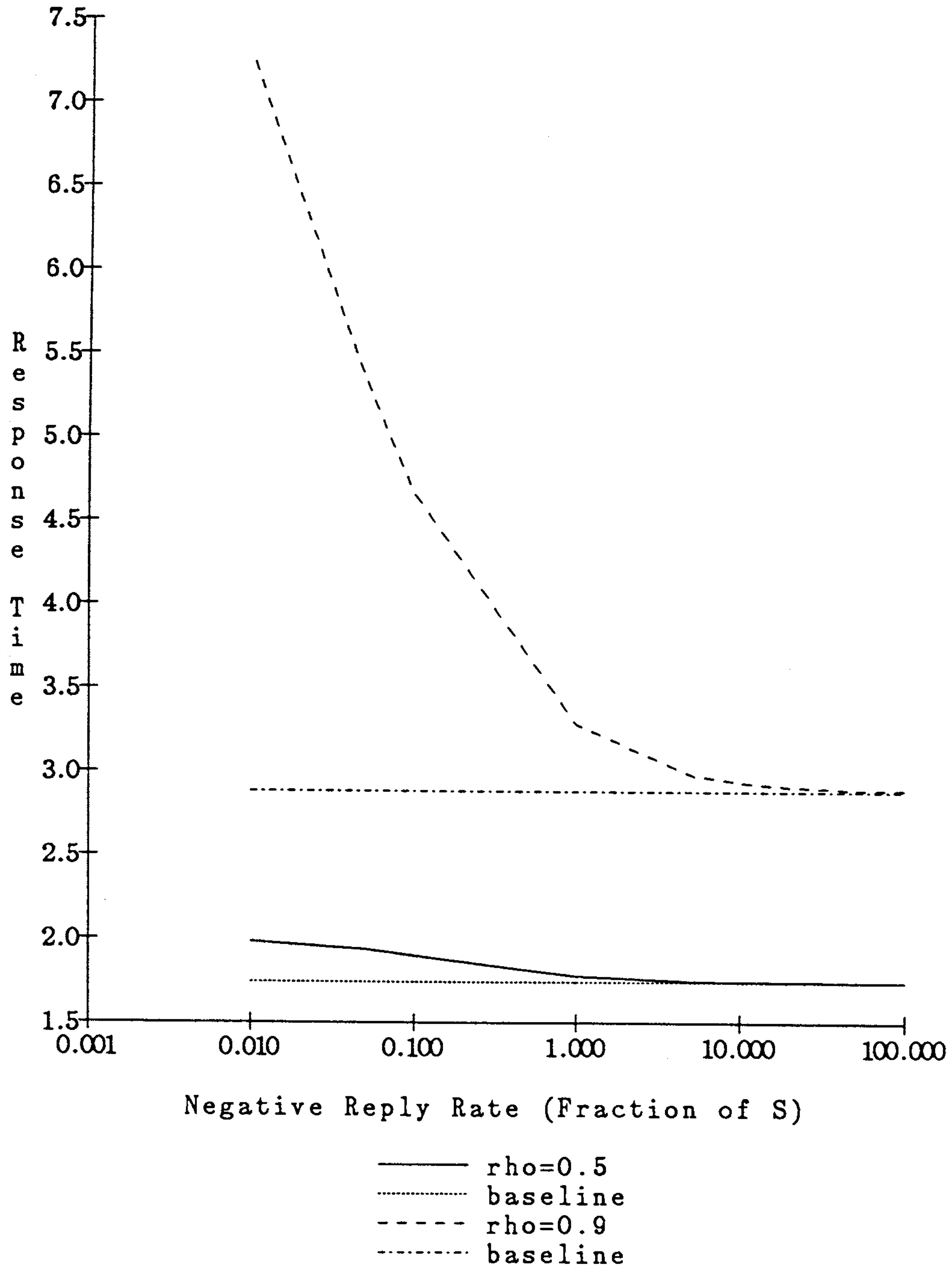

Figure 4.11: Effect of Non-Zero Probe Times (Delay=0.1S)

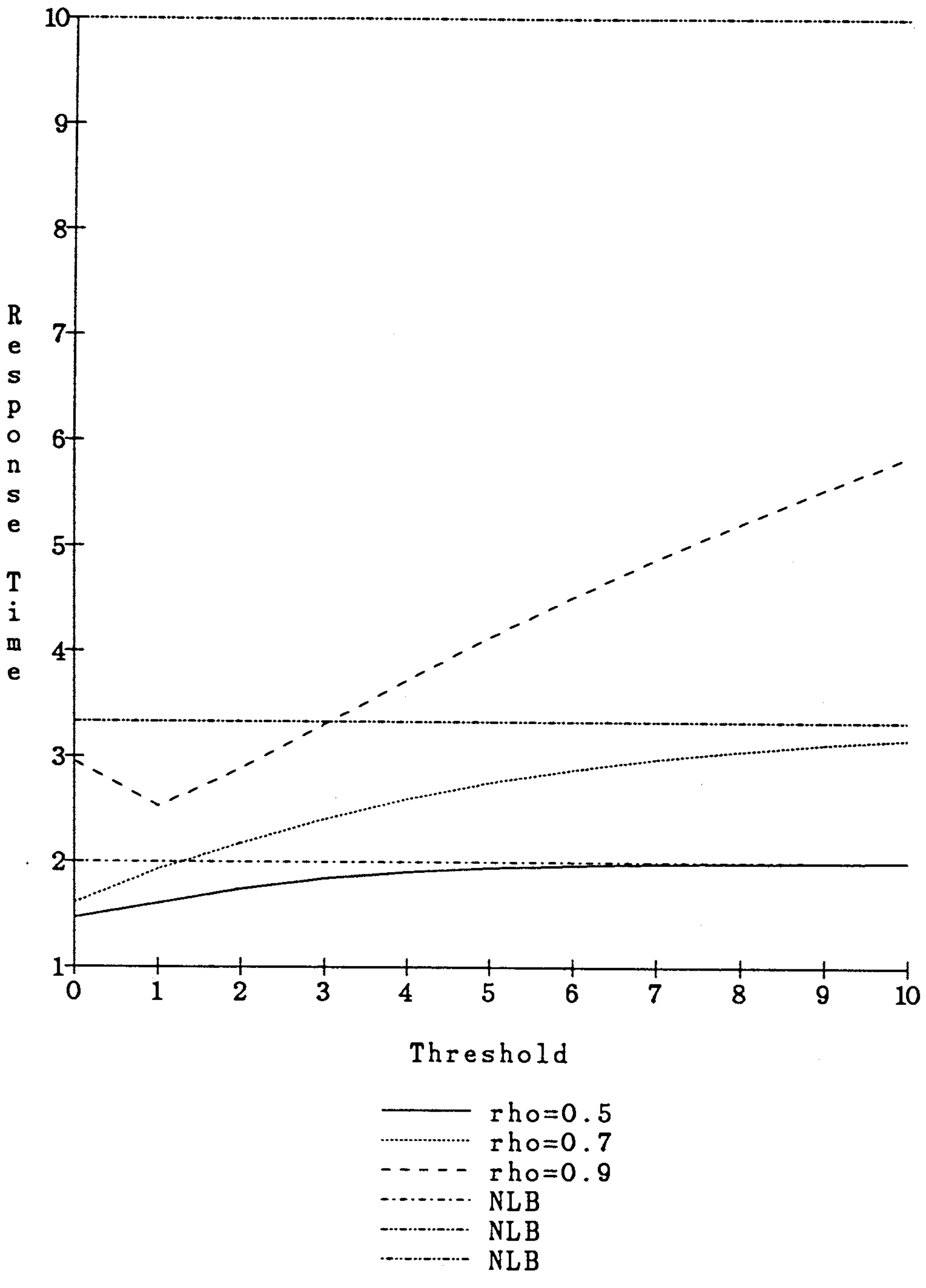

Figure 4.12: R.T. vs. Thresholds (Delay=0.1S)

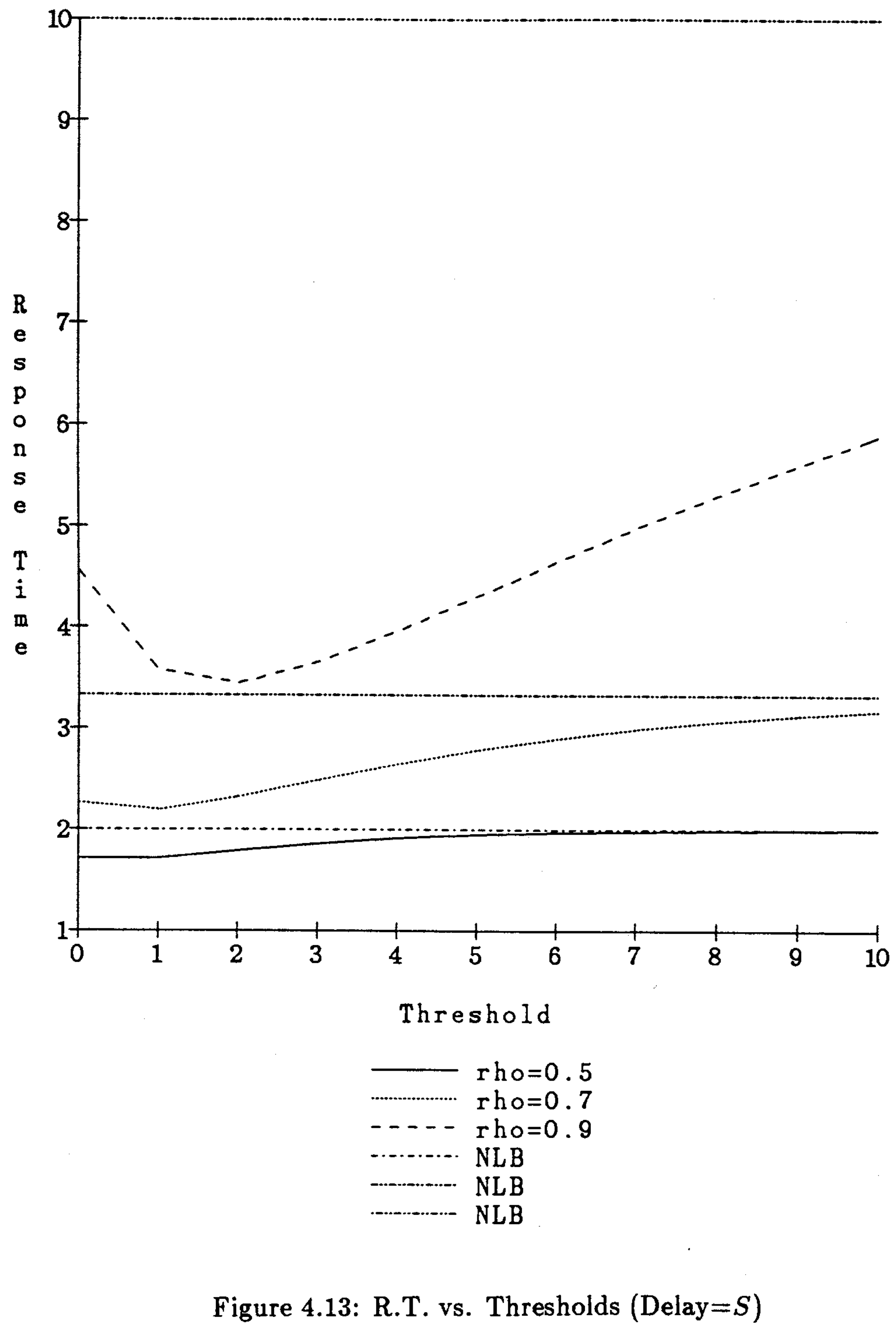

Figure 4.13: R.T. vs. Thresholds (Delay=S)

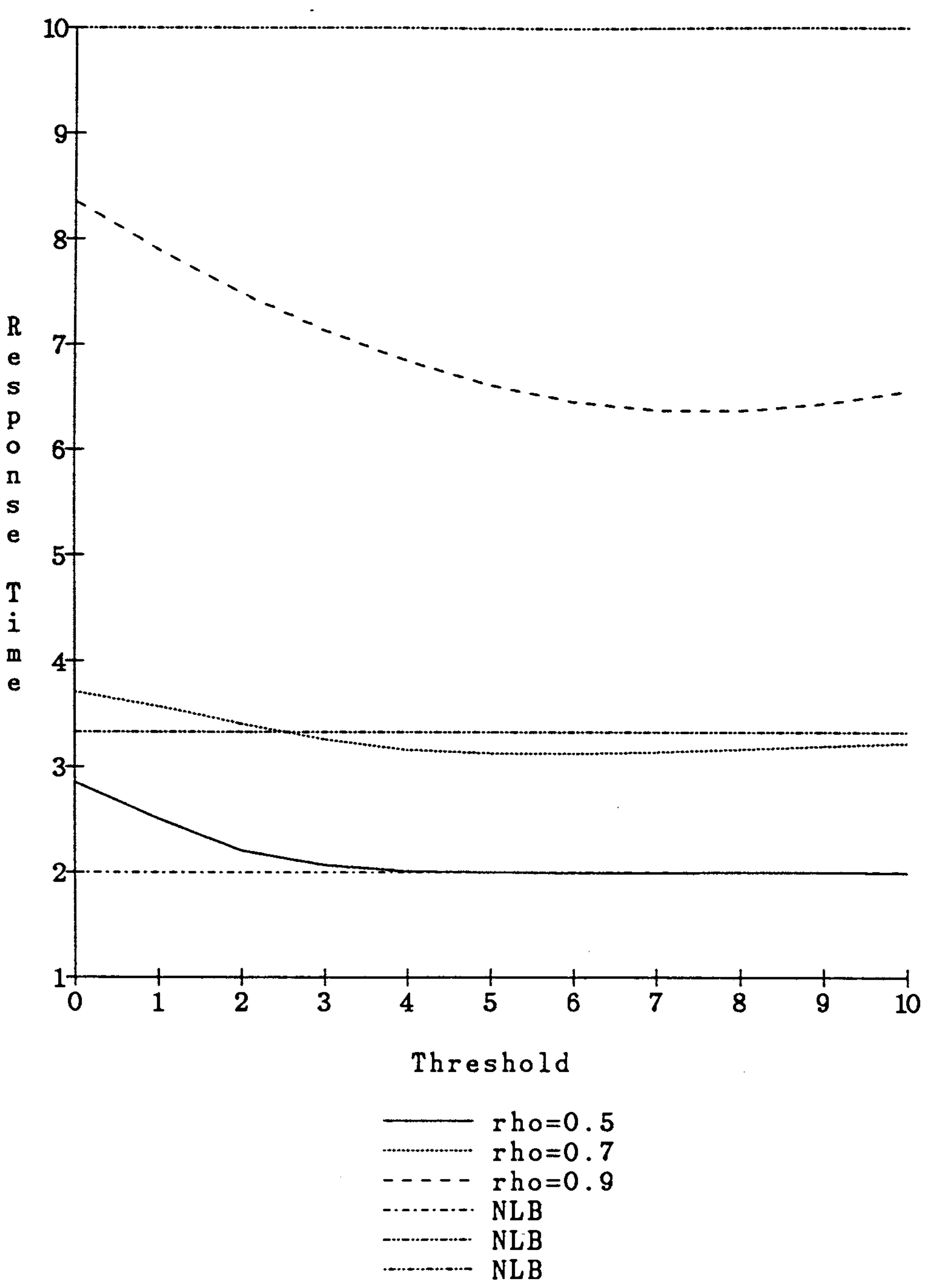

Figure 4.14: R.T. vs. Thresholds (Delay=10S)

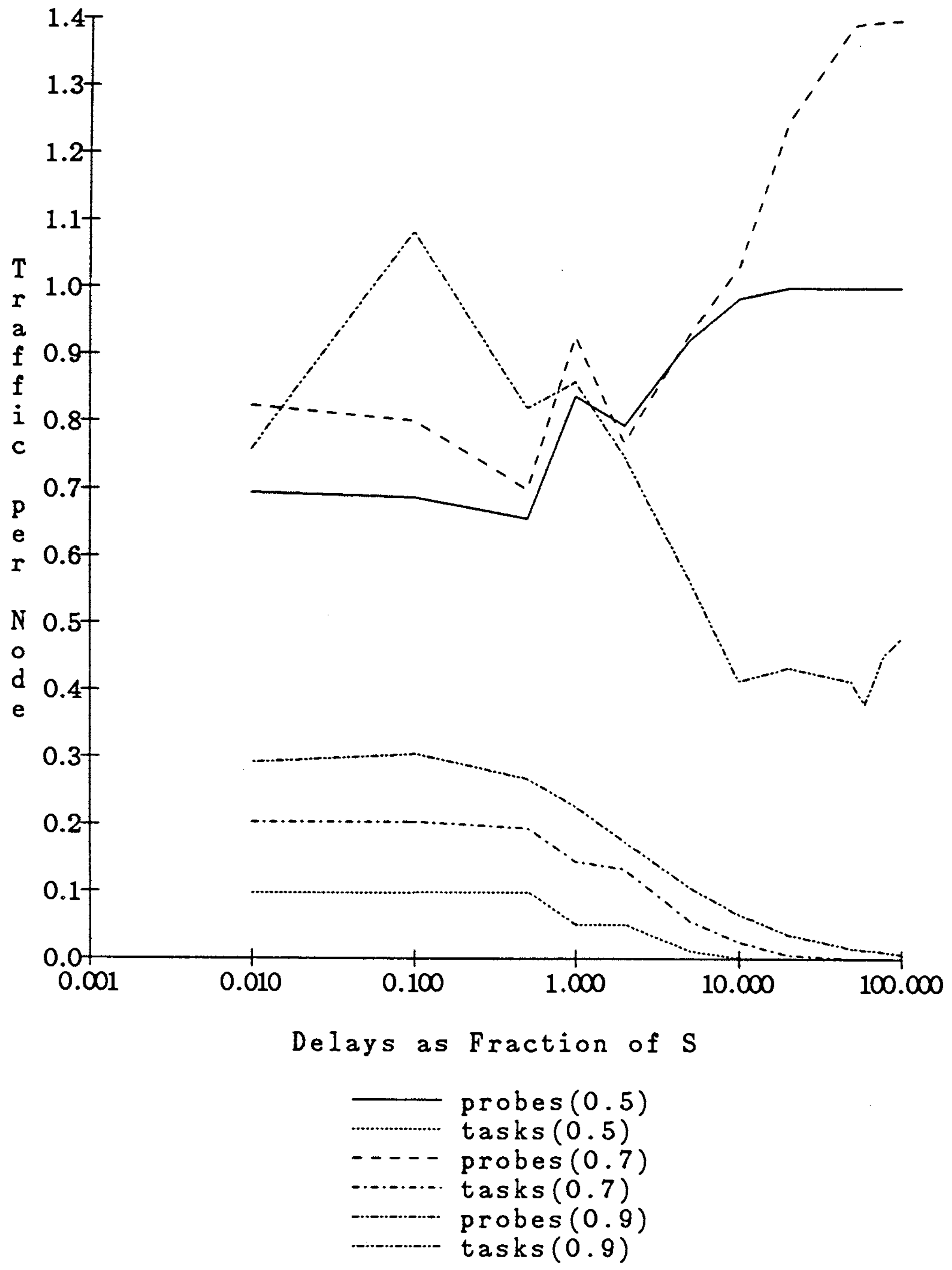

Figure 4.15: Network Traffic Rates

Chapter 5

LOAD SHARING IN HETEROGENEOUS SYSTEMS

5.1 Introduction

In the previous chapters, we studied load sharing in homogeneous systems where the arrival rates and processing speeds of jobs were identical at each node. In this chapter, we study the performance characteristics of simple load sharing algorithms for heterogeneous distributed systems. Heterogeneity in distributed systems can arise primarily in the following two ways: In the first instance, all the nodes in the system may be identical with regard to their processing capabilities and speeds. However, the rate at which external jobs arrive at one or more nodes may differ from that at other nodes in the system. In the second instance, nodes although functionally identical, may process jobs with different speeds (in such systems, the arrival rates of jobs at various nodes may also differ. We refer to these kinds of heterogeneous systems as **type-1** and **type-2** systems, respectively. Thus, **type-2** systems may indeed be a generalization of **type-1** systems). The study of such systems is interesting because many real systems tend to satisfy either one or both of the above conditions. In this chapter too, we assume that non-negligible delays are encountered in transferring jobs from one node to another and in gathering remote state information. This chapter analyzes the effects of these delays on the performance of two algorithms that we call Forward and Reverse. We formulate queueing theoretic models for each of the algorithms operating in heterogeneous systems under the assumption that the job arrival process at each node is Poisson and the service times and job transfer times are exponentially distributed. Each of the models is solved using the Matrix-Geometric solution technique and the important performance metrics are derived and studied. Wherever relevant, we compare the results of these algorithms against a Random job assignment algorithm as well as the no-load-balancing $M/M/1$ and perfect load balancing at zero cost $M/M/K$ models.

The remainder of this chapter is organized as follows: In Section 5.2, we provide a brief description of the system architecture and the load sharing algorithms and describe the differences in the assumptions made in this chapter as compared to the previous chapters. Section 5.3 contains the description of the Markov process corresponding to the algorithms and their Matrix Geometric solution. In Section 5.4, we describe the important results of this research and we summarize our work in Section 5.5.

5.2 System Architecture and Load Sharing Algorithms

We assume that the system under consideration is comprised of C distinct classes of nodes. This classification is performed on the basis of external arrival rates of jobs and/or the processing speeds of nodes. All the nodes in a particular class are assumed to be identical in every way.

5.2.1 Load Sharing Algorithms

The two algorithms that we have studied in the context of this research are called Forward and Reverse. Each algorithm is provided with thresholds T_c, $c = 1, \ldots, C$, one for each class of nodes. The algorithms are described in the following paragraphs.

- **Forward:** The algorithm is activated each time a local job arrives at a node. If the number of jobs at this node (including the job currently being executed) is greater than $T_c + 1$ (the node in question belongs to class-c), an attempt is made to transfer the newly arrived job to another node. A finite number, L_p, of nodes (usually $L_p = 2$ or 3 is adequate) is probed at random to determine a placement for the job. A probed node responds positively if the number of jobs it possesses is less than $T_d + 1$ (the probed node belongs to class-d) and it is not already waiting for some other remote job. If more than one node responds positively, the sender node transfers the job to one of these respondents, chosen at random. If none of the probed nodes responds positively, i.e., this probe was unsuccessful, the node waits for another local arrival before it can probe again.

- **Reverse:** This algorithm is activated every time a job completes at a node and the total number of jobs at the node is less than $T_c + 1$ and the node is not already waiting for a remote job to arrive. If so, the node probes a subset of size L_p remote nodes at random to try and acquire a remote job. Only nodes that possess more than $T_d + 1$ jobs, (including the one currently executing) are allowed to respond positively. If more than one node can transfer a job, the probing node chooses one of these nodes at random from which it requests a job.

Implicit in the above algorithm descriptions is the assumption that a node randomly probes other nodes, i.e., a class-c node is as likely to be probed as a class-d node. In general, this assumption may be relaxed for potential performance benefits. For instance, if nodes possess information about the membership and load of each class, it might be possible to bias probing towards one or more classes. Thus, at each probe, class-c nodes may be selected with probability f_c (with nodes in a class being selected in an equiprobable manner). In the general case, $f_{c,d}$ is the probability that a probing node of class c will choose class d nodes, $\sum_{d=1}^{C} f_{c,d} = 1, \forall c$. We will study the effects of varying these class selection probabilities in this chapter.

In the algorithms described above, it is assumed that probing takes zero time. This is based upon the assumption that probes are much smaller entities than are jobs. Thus, the overhead for processing a probe is much smaller than for jobs. Further, probes occupy much less of the communication bandwidth than jobs. Thus, the entire delay is assumed to occur during actual job transfer. We have seen in

Chapter 4 that as long as the ratio of job transfer times to probe transfer times is sufficiently large (≥ 20), the system essentially behaves as if the probes actually take zero time.

5.3 Mathematical Analysis

It is assumed that the job arrival process at each class-c node is Poisson, with parameter λ_c. Also, the service times and job transfer times for class-c nodes are assumed to be exponentially distributed, with means $1/\mu_c$ and $1/\gamma_c$, respectively. In our study, we have assumed that γ_c is independent of the class of nodes under consideration. Thus, for the remainder of this chapter, $\gamma_c = \gamma$, $\forall c$. The job transfer time includes the time between the initiation of a transfer from a node and the successful reception of the job at the destination node. The nodes in each class are assumed to be identical, i.e., the nodes have identical processing power and the arrival process at each node is the same. Jobs are assumed to be executed on a First-Come-First-Served (FCFS) basis at each node.

Let $N_t^{(c,i)}$ be the number of jobs at node i in class-c at time t and $J_t^{(c,i)}$ be the probe state of the ith class-c node, at time t. The probe state indicates whether the node is probing or being probed, etc. For example, in a system which has M_c nodes in class-c, the instantaneous state of class-c nodes can be represented by the $2M_c$-tuple

$$(N^{(c)}, J^{(c)}) \overset{\text{def}}{=} (N_t^{(c,1)}, N_t^{(c,2)}, ..., N_t^{(c,M)}; J_t^{(c,1)}, J_t^{(c,2)}, ...J_t^{(c,M_c)})$$

The instantaneous state of the entire network (over all the node classes) may be represented by the following:

$$(N, J) \overset{\text{def}}{=} ((N^{(1)}, J^{(1)}), (N^{(2)}, J^{(2)}), ...(N^{(C)}, J^{(C)}))$$

If the probe state $J_t^{(c,i)}$ is defined appropriately then, due to the Poisson arrival assumption and the exponential service and job transfer times, the process corresponding to the above state description is Markovian. It is clear that the Markov process representing multiple classes has a very large state space and appears extremely difficult to solve, even for moderately sized systems. Consequently, we decompose the model in the following way: Each node is modelled independently of the others. Because there are C different classes, this results in C distinct single node queueing models. Further, we assume that the behavior of each node in the system is Markovian. The interactions between the nodes which result in job transfers for the purpose of load sharing in the distributed system, are modelled by means of modifications to the arrival and/or departure process at each node. These interactions will be described in detail further in this subsection.

We conjecture that the method of decomposition is asymptotically exact as the number of nodes in each class tends to infinity. Actual simulation results indicate that there exists very good agreement between the model and simulations even when the systems are of relatively small size (10-15 nodes in each class). We expect the approximation to be even better for larger systems.

The analysis of the algorithms is performed using the Matrix-geometric solution technique [NEUT81], which yields an exact solution of the model for each node. The analysis of the Forward and Reverse algorithms will be described in detail along with a presentation of the main performance metrics.

The material in this chapter involves several Jacobi matrices, whose detailed definitions will be provided as in Latouche [LATO81]. Again, a matrix such as

$$\begin{bmatrix} b_0 & c_0 & 0 & 0 & & & \cdots & \\ a_1 & b_1 & c_1 & 0 & & & \cdots & \\ 0 & a_2 & b_2 & c_2 & & & & \\ & & & & & & & \\ \cdots & \cdots & & & a_{m-2} & b_{m-2} & c_{m-2} & 0 \\ \cdots & \cdots & & & 0 & a_{m-1} & b_{m-1} & c_{m-1} \\ \cdots & \cdots & & & 0 & 0 & a_m & b_m \end{bmatrix}$$

will be displayed as

$$\left\| \begin{array}{cccccc} & c_0 & c_1 & \dots & c_{m-3} & c_{m-2} & c_{m-1} \\ b_0 & b_1 & b_2 & \dots & b_{m-2} & b_{m-1} & b_m \\ a_1 & a_2 & a_3 & \dots & a_{m-1} & a_m & \end{array} \right\|$$

5.3.1 Forward Probing

Figure 5.1 represents the state diagram for the Forward probing algorithm operating at a single node of class-c using an arbitrary threshold T_c. The state of the node is represented by a tuple $(N_t^{(c)}, J_t^{(c)})$, where $N_t^{(c)}$ is the number of jobs at a node and $J_t^{(c)}$ is the probe state that indicates if the node is being probed or not. The probe states have the following codes:

- 0 : if not being probed,
- 1 : if being forward probed,

We define

$$y(n,j) = \lim_{t\to\infty} P(N_t^{(c)} = n,\ J_t^{(c)} = j),\ 0 \le n,\ 0 \le j \le 1,$$

$$\mathbf{p}_n^{(c)} = (y(n,0), y(n,1)),\ 0 \le n,$$

$$\vec{\mathbf{p}}^{(\mathbf{c})} = (\mathbf{p}_0^{(c)}, \mathbf{p}_1^{(c)}, \mathbf{p}_2^{(c)}, \dots\dots \mathbf{p}_i^{(c)}, \dots\dots).$$

If the Markov process $(N^{(c)}, J^{(c)})$ is ergodic then $\vec{\mathbf{p}}^{(\mathbf{c})}$ is its steady state probability vector satisfying $\vec{\mathbf{p}} Q_F^{(c)} = 0$, where $Q_F^{(c)}$ is the infinitesimal generator this Markov process. $Q_F^{(c)}$, the infinitesimal generator for the Forward probing algorithm for class-c nodes, has the structure of a block-tridiagonal matrix of the form

$$Q_F^{(c)} = \left\| \begin{array}{ccccccccc} & B_{01}^{(c)} & \dots & B_{01}^{(c)} & B_{01}^{(c)} & A_0^{(c)} & A_0^{(c)} & \dots \\ B_{00}^{(c)} & B_{11}^{(c)} & \dots & B_{11}^{(c)} & A_1^{(c)} & A_1^{(c)} & A_1^{(c)} & \dots \\ A_2^{(c)} & A_2^{(c)} & \dots & A_2^{(c)} & A_2^{(c)} & A_2^{(c)} & A_2^{(c)} & \dots \end{array} \right\|$$

with exactly $T_c - 1$ columns of $(B_{01}^{(c)}, B_{11}^{(c)}, A_2^{(c)})$. Each class of nodes in the system will be associated with a unique generator.

In the subsequent discussion, h_c is the probability that a class-c node is unsuccessful in finding an assignment for a spare job in response to a set of forward probes. Thus, $\overline{h_c} = 1 - h_c$, is the probability that at least one of the probed nodes will accept a remote job. The effect of a node sending a set of unsuccessful forward probes when it surpasses $T_c + 1$ is represented by the transition $\lambda_c h_c$. When a remote node sends a forward probe into this node, it makes the transition from $(n, 0)$ to $(n, 1)$, where $0 \leq n \leq T_c$. This means that the remote node will transfer a job to this node, on the basis of a successful probe. The rate at which a class-c node receives forward probes is denoted by α_c.

5.3.2 Reverse Probing

Figure 5.2 represents the state diagram for the Reverse probing algorithm operating at a single class-c node using an arbitrary threshold T_c. The state of the node is represented by a tuple $(N_t^{(c)}, J_t^{(c)})$, where $N_t^{(c)}$ is the number of jobs at a node and $J_t^{(c)}$ is the probe state that indicates whether the node is probing or not. The probe states have the following codes:

- 0 : if not probing,
- 1 : if reverse probing,

The infinitesimal generator matrix for this process (for class-c nodes) is:

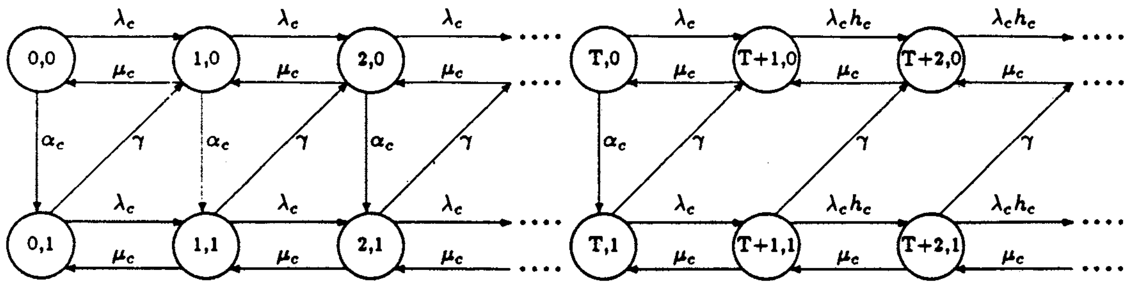

Figure 5.1: State Diagram of Forward Probing Algorithm

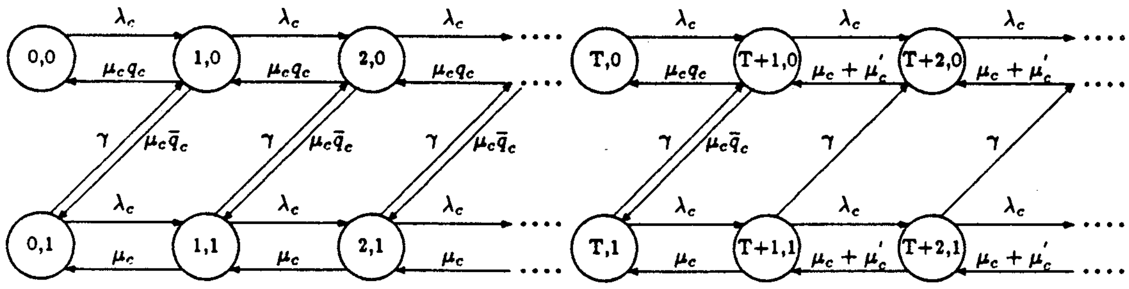

Figure 5.2: State Diagram of Reverse Probing Algorithm

$$Q_R^{(c)} = \left\| \begin{array}{cccccccc} & A_0^{(c)} & \ldots & A_0^{(c)} & A_0^{(c)} & A_0^{(c)} & A_0^{(c)} & \ldots \\ B_{00}^{(c)} & B_{11}^{(c)} & \ldots & B_{11}^{(c)} & B_{11}^{(c)} & A_1^{(c)} & A_1^{(c)} & \ldots \\ B_{10}^{(c)} & B_{10}^{(c)} & \ldots & B_{10}^{(c)} & A_2^{(c)} & A_2^{(c)} & A_2^{(c)} & \ldots \end{array} \right\|$$

with exactly $T_c - 1$ columns of $(A_0^{(c)}, B_{11}^{(c)}, B_{10}^{(c)})$. Each class of nodes will be associated with a unique generator.

We define q_c to be the probability that a class-c node is unsuccessful in finding a remote job from a set of reverse probes, and $\overline{q_c} = 1 - q_c$. When the node makes a transition on the completion of a job and the remaining number of jobs is less than $T_c + 1$, it sends out reverse probes if it is not already waiting for a remote job to arrive in response to an earlier reverse probe. Thus, at any time a node may wait for at most one remote job to arrive. In Chapter 4 we specifically addressed this assumption for Receiver-Initiated algorithms and concluded that there was almost no difference in performance for these algorithms if multiple outstanding remote jobs were allowed. A successful transition is represented by $\mu_c \overline{q_c}$ and an unsuccessful set of reverse probes is represented by the transition $\mu_c q_c$. A transition of this type is represented from $(n, 0)$ to $(n-1, 1)$, where $0 < n \leq T_c + 1$. The rate at which this node sends out jobs in response to nodes that asked it for jobs is μ_c'. Thus, the rate at which a node makes the transition (n, j) to $(n-1, j)$, for $n \geq T_c + 2$ equals $\mu_c + \mu_c'$.

As can be seen from the generators Q_F^c and Q_R^c, the Markov processes possess a regular structure comprised of the $A_0^{(c)}, A_1^{(c)}$ and $A_2^{(c)}$ matrices, preceded however by the irregular boundary conditions. The size of the irregular portion of the matrix depends upon the threshold at which the process is operating. There are exactly $T_c - 1$ columns of the matrices $(B_{01}^{(c)}, B_{11}^{(c)}, A_2^{(c)})$ in the case of Forward probing and exactly $T_c - 1$ columns of $(A_0^{(c)}, B_{11}^{(c)}, B_{10}^{(c)})$ in the case of Reverse probing.

5.3.3 Matrix-Geometric Solution

Neuts [NEUT81] examined Markov processes with such generators and determined the conditions for positive recurrence when the infinitesimal generator $A^{(c)} = A_0^{(c)} + A_1^{(c)} + A_2^{(c)}$, corresponding to the geometric part of the Markov Process, is irreducible. However, for our problem, A is lower triangular and reducible. In such cases, the stability criterion has to be determined explicitly.

Consider the non-linear matrix equation

$$A_0^{(c)} + R^{(c)}A_1^{(c)} + [R^{(c)}]^2 A_2^{(c)} = 0$$

such that $R^{(c)}$ is its minimal non-negative solution. It can be shown that $R^{(c)}$ is lower triangular, given the structure of $A_0^{(c)}, A_1^{(c)}$ and $A_2^{(c)}$ [NEUT81]. Also, $R^{(c)} = [r_{i,j}^{(c)}]$ can be written as follows:

$$\begin{aligned}
r_{1,1}^{(c)} &= \lambda_c h_c/\mu_c \\
r_{2,2}^{(c)} &= \frac{\theta_c + \gamma - ((\theta_c + \gamma)^2 - 4\mu_c\lambda_c h_c)^{1/2}}{2\mu_c} \\
r_{1,2}^{(c)} &= 0 \\
r_{2,1}^{(c)} &= \frac{\gamma}{\theta_c - (r_{1,1}^{(c)} + r_{2,2}^{(c)})\mu_c}
\end{aligned}$$

where $\theta_c = \lambda_c h_c + \mu_c$. It can be shown that the stability criterion for the Forward probing algorithm is

$$\lambda_c h_c < \mu_c.$$

Thus, the diagonal elements of R can be written explicitly in terms of the parameters of the Markov process. Once the diagonal elements are determined, the elements below the diagonal are computed recursively from the solution of the diagonal elements.

By adapting Theorem 1.5.1 from Neuts [NEUT81], the Markov process Q_F is positive recurrent if and only if $sp(R_c) < 1$ and the matrix M_c (defined below) possesses a positive left invariant probability vector. Because R_c is lower triangular, its eigenvalues are its diagonal elements.

The matrix M_c, given by

$$\left\| \begin{array}{ccccc} & B_{01}^{(c)} & \dots & B_{01}^{(c)} & B_{01}^{(c)} \\ B_{00}^{(c)} & B_{11}^{(c)} & \dots & B_{11}^{(c)} & A_1^{(c)} + R^{(c)} A_2^{(c)} \\ A_2^{(c)} & A_2^{(c)} & \dots & A_2^{(c)} & \end{array} \right\|$$

is an irreducible, aperiodic matrix. The second condition holds because of the irreducibility of M_c. The vector $(\mathbf{p}_0^{(c)}, \mathbf{p}_1^{(c)}, ..., \mathbf{p}_{T+1}^{(c)})$ is the left eigenvector of M_c.

Intuitively, the stability condition means that the rate of processing jobs (including the ones that are sent out of this node) is greater than the total arrival rate of jobs into this node. Thus, on the average, whenever there are more than $T + 1$ jobs at a node, the process drifts towards the boundary specified by the threshold T. Similar analysis may be carried out for the Reverse probing algorithm, with the appropriate substitution of parameters.

For the Reverse probing algorithm, $R_c = [r_{i,j}^{(c)}]$ can be written as follows:

$$\begin{aligned} r_{1,1}^{(c)} &= \lambda_c/(\mu_c + \mu_c^{'}) \\ r_{2,2}^{(c)} &= \frac{\phi_c + \gamma - ((\phi_c + \gamma)^2 - 4(\mu_c + \mu_c^{'})\lambda_c)^{1/2}}{2(\mu_c + \mu_c^{'})} \\ r_{1,2}^{(c)} &= 0 \\ r_{2,1}^{(c)} &= \frac{\gamma}{\phi_c - (r_{1,1}^{(c)} + r_{2,2}^{(c)})(\mu_c + \mu_c^{'})} \end{aligned}$$

where $\phi_c = \lambda_c + \mu_c + \mu_c^{'}$. It can be shown that the stability criterion for the Reverse probing algorithm is

$$\lambda_c < \mu_c + \mu_c^{'}.$$

Intuitively, this means that the rate of processing jobs (including the ones that are sent out of this node) is greater than the total arrival rate of jobs into this node. Thus, on the average, whenever there are more than $T_c + 1$ jobs at a node, the process drifts towards the boundary specified by the threshold T_c.

We now assume that all the values of all the parameters are known. First, the boundary conditions are determined, by solving a system of linear equations. Thus, for an arbitrary threshold T_c, we have, for the Forward probing algorithm

$$(\mathbf{p}_0^{(c)}, \mathbf{p}_1^{(c)}, \ldots, \mathbf{p}_{T_c+1}^{(c)}) \left\| \begin{array}{ccccc} & B_{01}^{(c)} & \ldots & B_{01}^{(c)} & B_{01}^{(c)} \\ B_{00}^{(c)} & B_{11}^{(c)} & \ldots & B_{11}^{(c)} & A_1^{(c)} + R^{(c)} A_2^{(c)} \\ A_2^{(c)} & A_2^{(c)} & \ldots & A_2^{(c)} & \end{array} \right\| = 0$$

where the number of columns in the matrix is exactly $T_c + 1$. The equation corresponding to the Reverse probing algorithm is described as follows:

$$(\mathbf{p}_0^{(c)}, \mathbf{p}_1^{(c)}, \ldots, \mathbf{p}_{T_c+1}^{(c)}) \left\| \begin{array}{ccccc} & A_0^{(c)} & \ldots & A_0^{(c)} & A_0^{(c)} \\ B_{00}^{(c)} & B_{11}^{(c)} & \ldots & B_{11}^{(c)} & B_{11}^{(c)} + R^{(c)} A_2^{(c)} \\ B_{10}^{(c)} & B_{10}^{(c)} & \ldots & B_{10}^{(c)} & \end{array} \right\| = 0$$

We know from Neuts [NEUT81] that

$$\mathbf{p}_i^{(c)} = \mathbf{p}_{T+1}^{(c)} R_{T_c+1-i}^{(c)}, \forall i \geq T_c + 1$$

Thus,

$$\sum_{i \geq T_c+1} \mathbf{p}_i^{(c)} = \mathbf{p}_{T_c+1}^{(c)} (I - R^{(c)})^{-1}$$

Also,

$$[\sum_{i=0}^{T_c} \mathbf{p}_i^{(c)} + \mathbf{p}_{T_c+1}^{(c)} (I - R^{(c)})^{-1}] e = 1$$

where $R^{(c)}$ is the minimal solution of

$$A_0^{(c)} + R^{(c)} A_1^{(c)} + (R^{(c)})^2 A_2^{(c)} = 0$$

with $R^{(c)} \geq 0$ and the spectral radius of $R^{(c)}, sp(R^{(c)}) < 1$, and I is the identity matrix.

$E[N_c]$, the expected number of jobs at a node of class-c, and $E[D_c]$, the expected response time of a job of the same class, are given by the following expressions:

$$\begin{aligned}
E[N_c] &= \sum_{i \geq 1} i\,\mathbf{p}_i^{(c)} e \\
&= \mathbf{p}_{T_c+1}^{(c)}(I - R^{(c)})^{-2} e + T_c * [\mathbf{p}_{T_c+1}^{(c)}(I - R^{(c)})^{-1} e] + \sum_{i=1}^{T} i\mathbf{p}_i^{(c)} e \\
E[D_c] &= \frac{(E[N_c] + \frac{(Flow-In(c))}{\gamma})}{\phi_c}
\end{aligned}$$

where ϕ_c is the throughput of class-c nodes and $Flow - In(c)$ is the flow into a node of remote jobs due to forward or reverse probes, depending upon the algorithm being used.

Further, the average response time of jobs over all classes in the system is given by:

$$E[D] = \frac{1}{\Phi} \sum_{c=1}^{C} \phi_c E[D_c]$$

where

$$\Phi = \sum_{c=1}^{C} \phi_c$$

is the throughput of the system over all classes.

In the next subsection, we derive the equations required to determine the values of the unknowns h_c and α_c for Forward and q_c and μ_c' for Reverse and describe the iterative algorithms used to solve the resulting models for the two algorithms.

5.3.4 Computational Procedure

We now describe the computational procedure used to determine the unknowns in the case of Forward probing. A similar procedure is used for Reverse probing, with the proper substitution of parameters. Initially, it is assumed that the values for h_c and α_c are known for all classes and the model is solved using these values. In a typical step, a model solution is used to derive new values for h_c and α_c, for all classes, and a new solution is computed. The iteration procedure that we use is described in a step-wise form, after the following definitions.

- $FFRO_c$: Flow rate out of jobs from a class-c node, as a result of forward probes made by it.
- $FFRI_c$: Flow rate of jobs into a class-c node, as a result of forward probes made to it by other nodes in the network.
- $RFRO_c$: Flow rate out of jobs from a class-c node, as a result of reverse probes made to it by other nodes in the network.
- $RFRI_c$: Flow rate of jobs into a class-c node, as a result of reverse probes made by it.

Let i denote the iteration count. Thus, $h_{(c,i)}, \alpha_{(c,i)}, FFRO_c^{(i)}$ denote the value of the class-c variables after the i-th iteration.

Iteration Procedure

1. Let $i = 0$; $c = 1$
2. Choose values for $h_{(c,0)}, \alpha_{(c,0)}, FFRO_c^{(0)}$
3. Determine $Q_F^{(c,i)}$ from $h_{(c,i)}, \alpha_{(c,i)}$
4. Determine $R_F^{(c,i)}$

5. Solve the linear system corresponding to the boundary conditions

6. Determine $FFRO_c^{(i+1)}$ from the model solution

7. Repeat steps 2 through 6 for classes 2 through C

8. If $ABS(FFRO_c^{(i+1)} - FFRO_c^{(i)}) \leq \epsilon$ for all classes, where ϵ is an arbitrary small number, stop, else

9. Let $i = i + 1$; $c = 1$ Go to 3

A similar procedure is used for Reverse probing, with the unknowns being q_c and μ_c' and the stopping criterion being based upon $RFRO_c$, similar to that in the case of Forward probing. We have observed from experiments that the solution was insensitive to the initial values chosen for the unknown quantities. Consequently, we conjecture that there exists a unique solution to the models. Further, the number of iterations required for convergence was usually small, ranging between 20 and 40.

We define the following quantities:

- $Probe - Out_c$: Rate of successful forward probes made by a class-c node.
- $Probe - In_c$: Rate of successful reverse probes made to a node of class-c.

To determine the values of the unknowns h_c and α_c for the Forward probing algorithm, we use the following equations:

$$\begin{aligned} FFRI_c &= \overline{x_c} \sum_{d=1}^{C} FFRO_d \, f_{d,c} \, , \forall c, \\ FFRO_d &= Probe - Out_d / \overline{h_d}, \\ Probe - Out_d &= \lambda_d \overline{h_d} \sum_{i > T_d + 1} \mathbf{p}_i^{(d)} \, e, \end{aligned}$$

Further,

$$x_c = \sum_{i \le T_c} \mathbf{p}_i^{(c)} [01]^T + \sum_{i > T_c} \mathbf{p}_i^{(c)} e.$$

is the probability that a class-c node is unable to accept a remote job. Also, $\overline{x_c} = 1 - x_c$.

where $r_d = \sum_{c=1}^{C} f_{d,c} x_c$, is the probability that a forward probe made by a class-d node is unsuccessful and $\overline{r_d} = 1 - r_d$ is the probability of a successful probe. And $h_d = (r_d)^{L_p}$, is the probability that L_p forward probes made by a class-d node are unsuccessful. $\overline{h_d} = 1 - h_d$ is the probability that at least one of the probes is successful.

$FFRI_c$ can also be expressed as follows:

$$FFRI_c = \alpha_c \overline{x_c}$$

In order to determine α_c, the the rate at which a class-c node receives forward probes, we can rewrite the above equation as:

$$\alpha_c = FFRI_c / \overline{x_c}$$

To determine the unknowns μ_c', and q_c for the Reverse probing algorithm, we use the following equations:

$$\begin{aligned} RFRO_c &= Probe - In_c \, \overline{x_c}, \\ Probe - In_c &= \sum_{d=1}^{C} RFRI_d \, f_{d,c} / \overline{y_c}, \\ RFRI_d &= \mu_d \overline{q_d} \sum_{i=1}^{T_d+1} \mathbf{p}_i^{(d)} [10]^T \end{aligned}$$

where

$$x_c = \sum_{i \le T_c + 1} \mathbf{p}_i^{(c)} e$$

is the probability that a class-c node is unable to provide a spare job and $\overline{x_c} = 1 - x_c$. Further,

$$y_c = \sum_{d=1}^{C} f_{c,d} x_d$$

is the probability that a reverse probe made by a class-c node is unsuccessful and $\overline{y_c} = 1 - y_c$. Also, $q_c = (y_c)^{L_p}$ is the probability that L_p reverse probes made by a class-c node are unsuccessful. $\overline{q_c} = 1 - q_c$ is the probability that at least one of the probes is successful. The rate at which a class-c node sends out jobs in response to reverse probes made to it is

$$\mu_c' = \frac{RFRI_c}{\sum_{i > T_c + 1} \mathbf{p}^{(c)} e}$$

In summary, we have now developed the procedure and determined the equations necessary to solve for the unknown parameters of the two Markov processes. Next, we study the performance of the algorithms under a large range of system parameters.

5.4 Performance Comparisons

In this section, we study the performance characteristics of the algorithms in the two main classes of heterogeneous systems, i.e., where the external job arrival rates at nodes may differ, and second, where the nodes themselves may have different processing speeds. Even though the analysis presented in the previous section is valid for an arbitrary number of node classes, we restrict our study to systems with two distinct classes. The reasons for this restriction are the following: Firstly, the computational effort required to study heterogeneous systems grows at a higher than linear, and may at times, grow exponentially. For example, a brute force search of the optimal thresholds for a set of parameters is easily seen to grow asymptotically as $O(T^C)$, where $0 \ldots T$ is the range of thresholds and C is the number of classes under consideration. Secondly, even from the results of systems with only two classes, we believe that it is feasible to predict many of the interesting characteristics of more complex systems. For the subsequent discussion, when we refer to the quantities in both classes, we use the following system: For example, (T_1, T_2) refer to the operating thresholds at nodes of the two classes. All other parameters are represented in a similar fashion.

Wherever relevant, we compare the algorithm's performance to that of the Random assignment, the no-load-balancing or $M/M/1$ results, and the $M/M/K$ system or an appropriate lower bound for heterogeneous systems of **type-2** (called LB_2, which we describe in Appendix C at the end of the dissertation). The Random assignment algorithm is the same as that described in Chapter 3. Briefly, a class-c node that has a local arrival and its total queue length is $\geq T_c + 2$, transfers the new arrival to another node, chosen at random. In the simplest case, the nodes do not possess information regarding classes. However, in this algorithm too, as in Forward and Reverse, it is possible to bias the selection of classes, with nodes in a class being picked randomly. In all the subsequent studies in this chapter we will assume that, the probe limit, $L_p = 2$, unless mentioned otherwise. Further, $S_c = 1/\mu_c$ is the mean service time for jobs executing at class-c nodes. For **type-1** systems, $S_1 = S_2 = S = 1$ unit. Also, ρ_1 and ρ_2 are the loads for nodes of class-1 and class-2, respectively.

5.4.1 Type-1 Heterogeneous systems

In this study, $\rho_1 = 0.9$, is kept fixed for all the tests conducted. However, ρ_2 is varied between 0.1 and 0.7, in steps of 0.2. Thus, we examine systems in which the degree of heterogeneity (in this case the ratio of loads of the two classes) is varied over a large range. Although we have conducted tests for all four values of ρ_2, we will present only those results which are most interesting. In general, we always present detailed discussion of the results when $\rho_2 = 0.1$ and 0.7, representing very high and very low degrees of heterogeneity, respectively. Wherever relevant, we also comment on the results of the other two loads of 0.3 and 0.5.

As might be imagined, the number of parameters that it is possible to vary in the study of these algorithms is very high. We will try to limit this study to what we believe are its most interesting aspects. These items are presented in the following list:

- Validation of analytical models with simulations
- Effects of varying the probe limit L_p
- Effects of varying (T_1, T_2)
- Effects of biasing the probing probabilities, $(f_{c,d}\,,\, 1 \leq c, d \leq 2)$
- Performance of Biased Random
- Optimal response times under a large range of job transfer delays

Validation of models with simulations

As mentioned in Section 5.2, the solution technique we have used is only an approximation for finite sized multiple class systems, which is conjectured to be exact for infinitely large systems. Consequently, we have conducted simulation studies of systems executing the load sharing algorithms and compared these with the results of the analytical models. These comparisons help us determine the minimum sized systems for which our analysis is valid. We present the results for the Forward probing algorithm. The results of Reverse probing are very similar and consequently we will not describe these.

Figure 5.3 depicts the results of the analytical model for (ρ_1, ρ_2) values of (0.9,0.1) to (0.9,0.7), with ρ_2 increasing in steps of 0.2. The three curves are for three values of delays tested, i.e., $0.1S, S$ and $10S$. These curves represent the average total system response times of the analytical results. Also shown in the curves are the 95% confidence intervals, as computed by the Student-t tests, with simulations performed using the method of independent replications. We have chosen not to specifically show the results of the simulation experiments because they

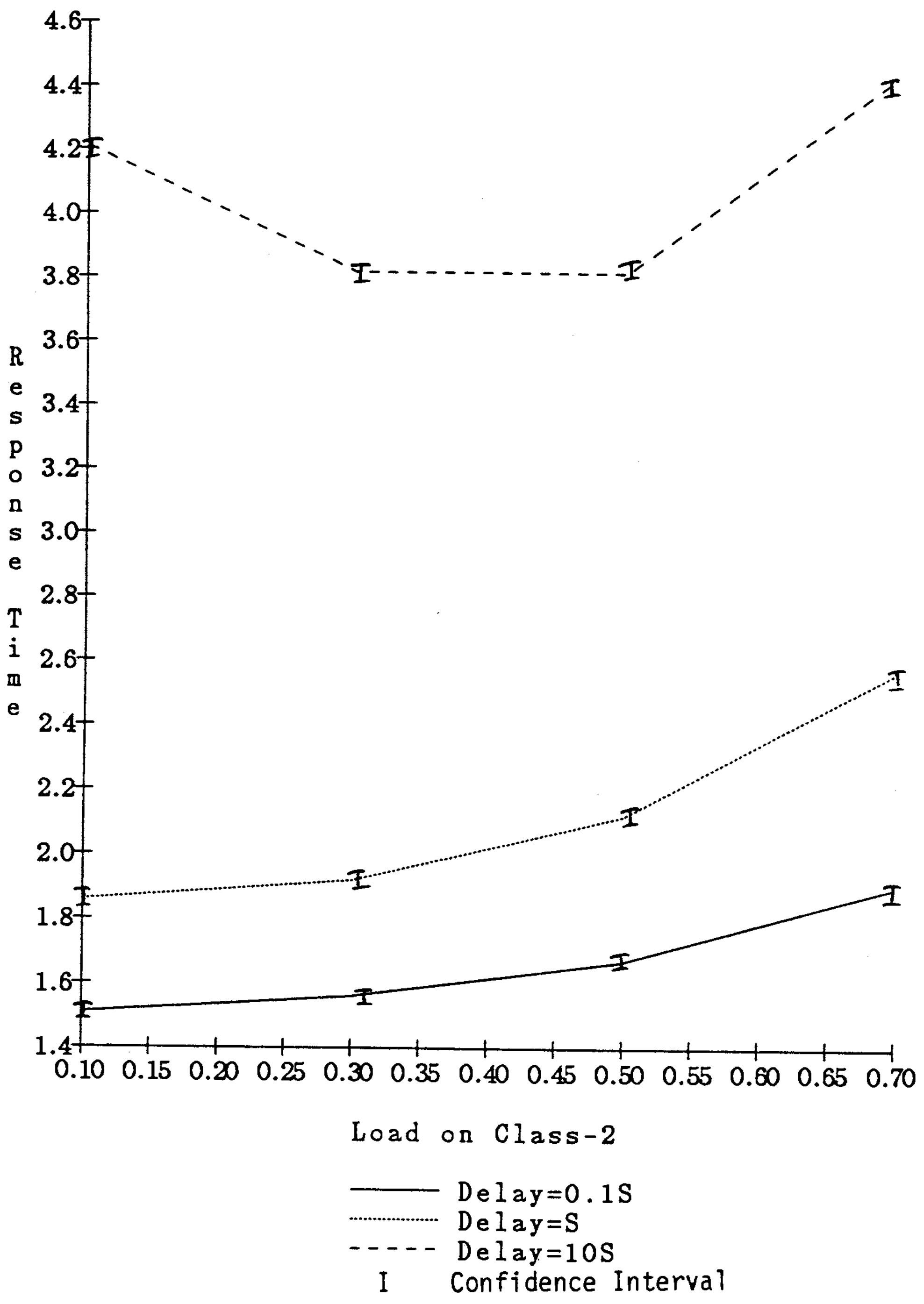

Figure 5.3: Validation with Simulations

were very close to the analytical results, with the variation being less than $2 - 3\%$. The simulation experiments were conducted on systems of increasing size, until the results showed good agreement with those of the analytical model. The curves in Figure 5.3 represent results of simulations where the classes comprised of 15 nodes each.

From the simulation experiments, we have seen that with about 15 nodes or more in each of the two classes, the results appear to be very good. Smaller systems tend to generate larger errors, as might be expected. For example, with classes of 10 nodes, the errors were as large as about 5-7% (these results are not depicted in Figure 5.3). In Chapter 3, we had seen that for homogeneous systems, systems of at least 15 nodes provided very good agreement with the analytical results. It might thus appear that for heterogeneous systems, at least 15 nodes per class might be the requirement, in order to have very high agreement between the analytical and simulation results.

Effect of varying L_p, the probe limit

In this study, we consider the effect of varying the probe limit on the performance of the two algorithms. In general, probes can generate processing overheads which may degrade system performance and in real systems, this overhead would need to be traded off against the response time improvements that increased probing may offer. Figures 5.4 and 5.5 depict the results of varying L_p between 1 and 10 for two different sets of loads for the Forward probing algorithm. In Figure 5.4, (ρ_1, ρ_2) =(0.9,0.1) and in Figure 5.5, (0.9,0.7). The three curves in each graph correspond to different values of delays, i.e., $0.1S, S$ and $10S$. These three values of delays serve as examples of small, moderate and high delays respectively. From Figures 5.4 and 5.5 we can make the following observations:

- Increasing the probe limit beyond 3 or 4 appears to provide little or no improvement (3-4%) in response times. However, increasing L_p may increase the overhead on the processors and network.

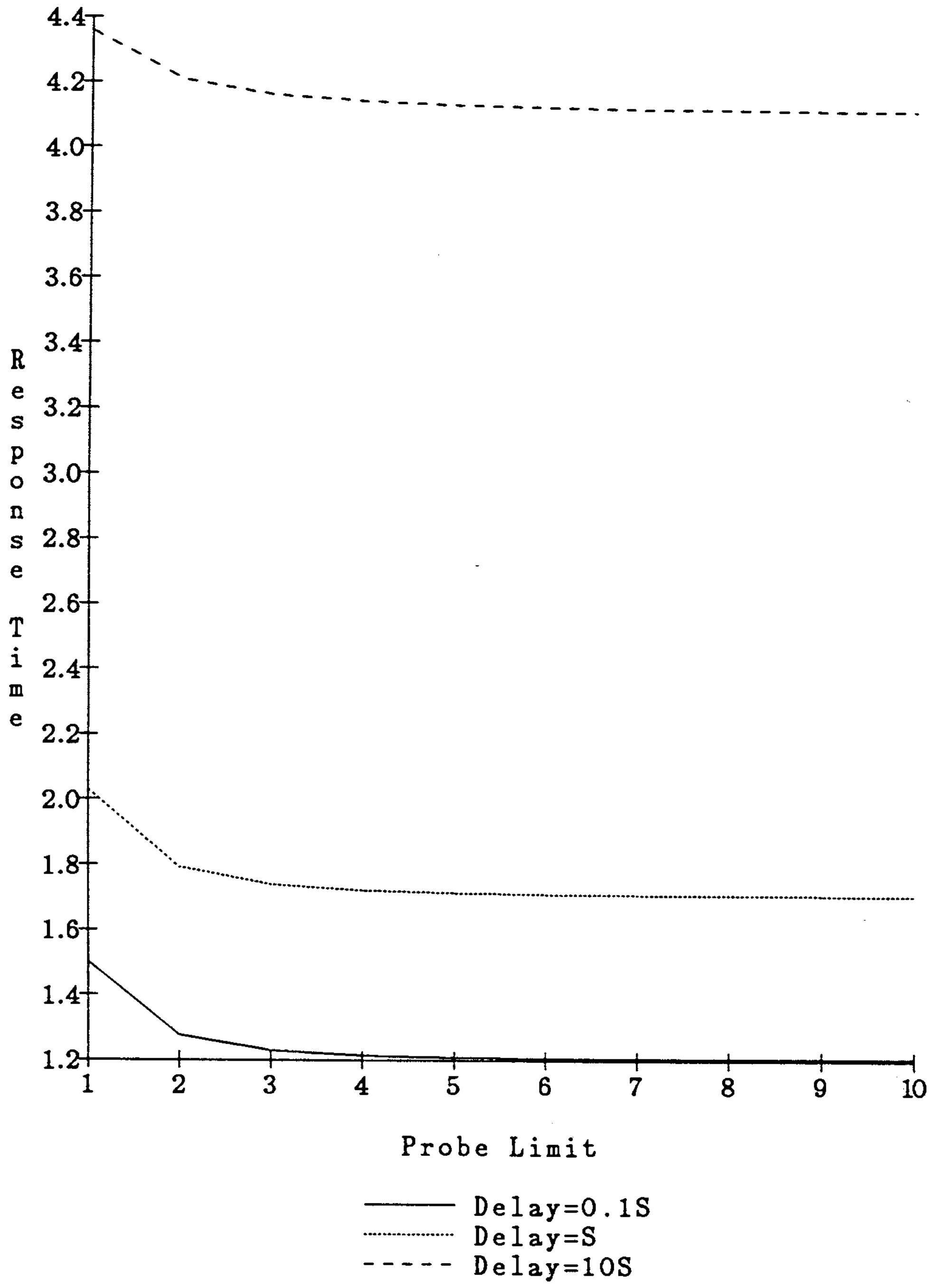

Figure 5.4: Effect of Probe Limit (FP, $\rho_2 = 0.1$)

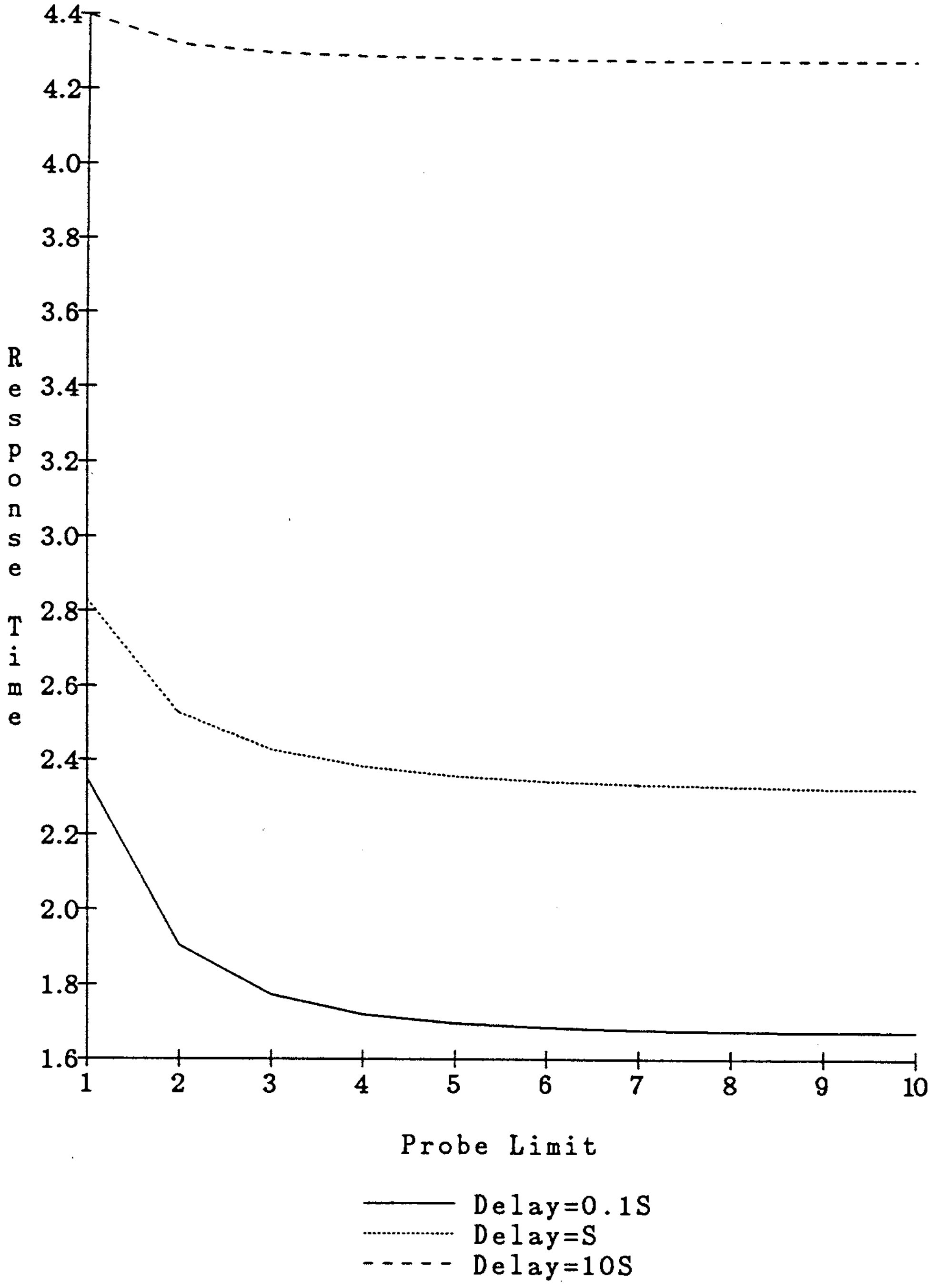

Figure 5.5: Effect of Probe Limit (FP,$\rho_2 = 0.7$)

- When the degree of heterogeneity is large, as in the case of Figure 5.4 the system response time is less sensitive to L_p than in a more balanced system. From Figure 5.4, which represents a a highly heterogeneous system, it is easily seen that going beyond $L_p = 2$ does not provide significant performance benefit (at most 3-5%). On the other hand, slightly different behavior is depicted in Figure 5.5, where at delays of $0.1S$ and perhaps even at S, $L_p = 3$ seems the proper limit (with about 10% gain over $L_p = 2$). The main reason for this behavior is the following: When systems are very highly unbalanced, a forward probe made by a node is very likely to find a recipient (even when the class selection is equiprobable). However, as the degree of heterogeneity decreases, a probing node needs to work harder to find a placement for a spare job.

- At very high delays $\geq 10S$, the performance is less sensitive to L_p than at lower delays. This is because of reduced load sharing at high delays, caused by the high operating thresholds.

From our tests on systems with $(\rho_1, \rho_2) = (0.9,0.3)$ and $(0.9,0.5)$ in connection with L_p, (results not presented here), we have seen that a gradual change occurs in the behavior. Thus, (0.9,0.3) behaves more like (0.9,0.1) than (0.9,0.7) while the behavior of (0.9,0.5) approaches (0.9,0.7), as might be expected. Further, we have seen that when ρ_1 is smaller, for example, 0.8 and less, there appears to be less sensitivity to probe limit. Again, this is because a probing node is more likely to find a recipient for a spare job, because the net load on the entire system is lower.

Figures 5.6 and 5.7 depict the results of identical tests performed on the Reverse probing algorithm. The loads indicated in Figures 5.6 and 5.7 are (0.9,0.1) and (0.9,0.7), respectively. From Figures 5.6 and 5.7, we can make the following observations:

- Reverse probing appears more sensitive to probe limit, particularly when the degree of heterogeneity is large, as in the tests corresponding to Figure 5.6.

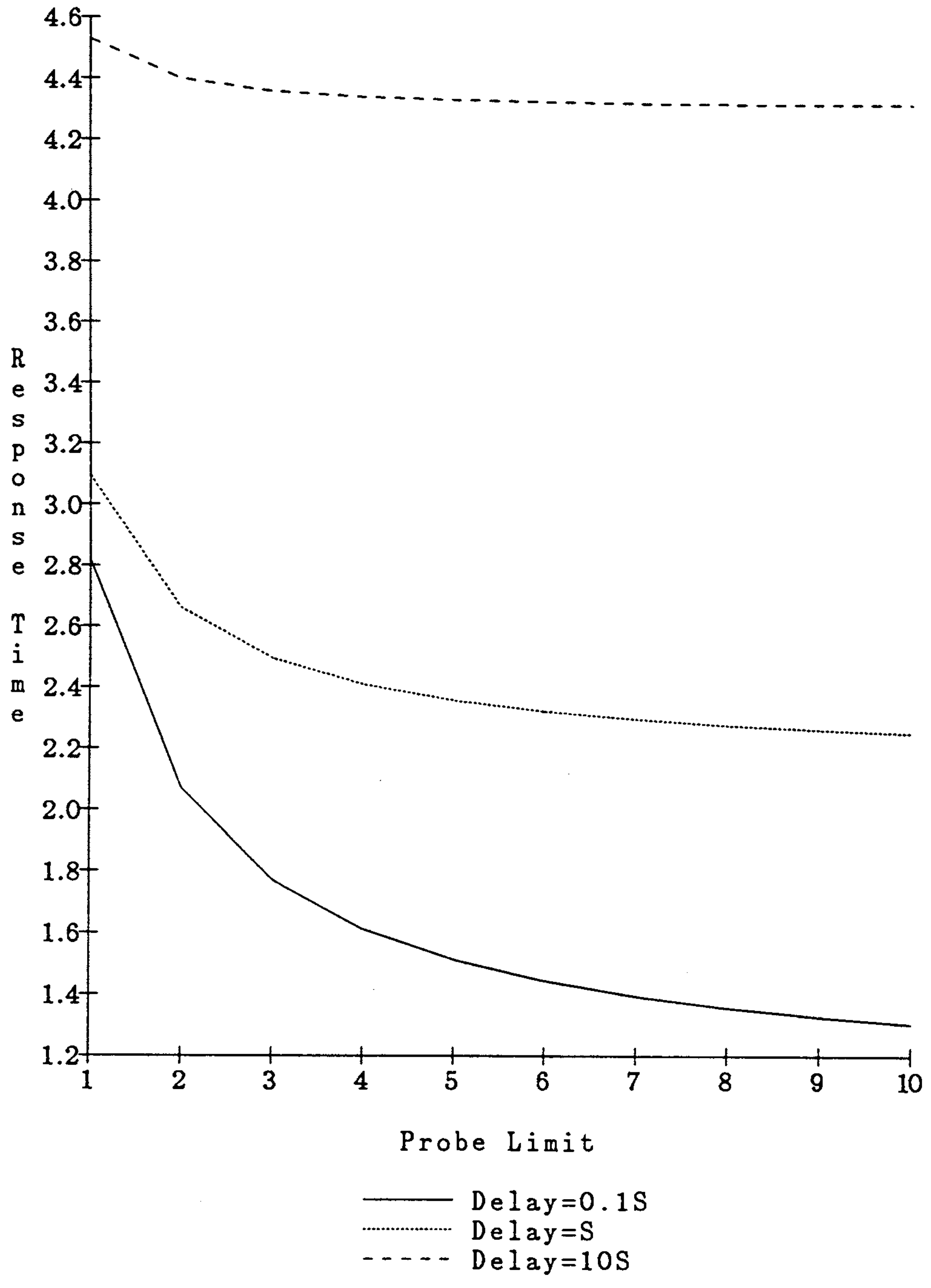

Figure 5.6: Effect of Probe Limit (RP, $\rho_2 = 0.1$)

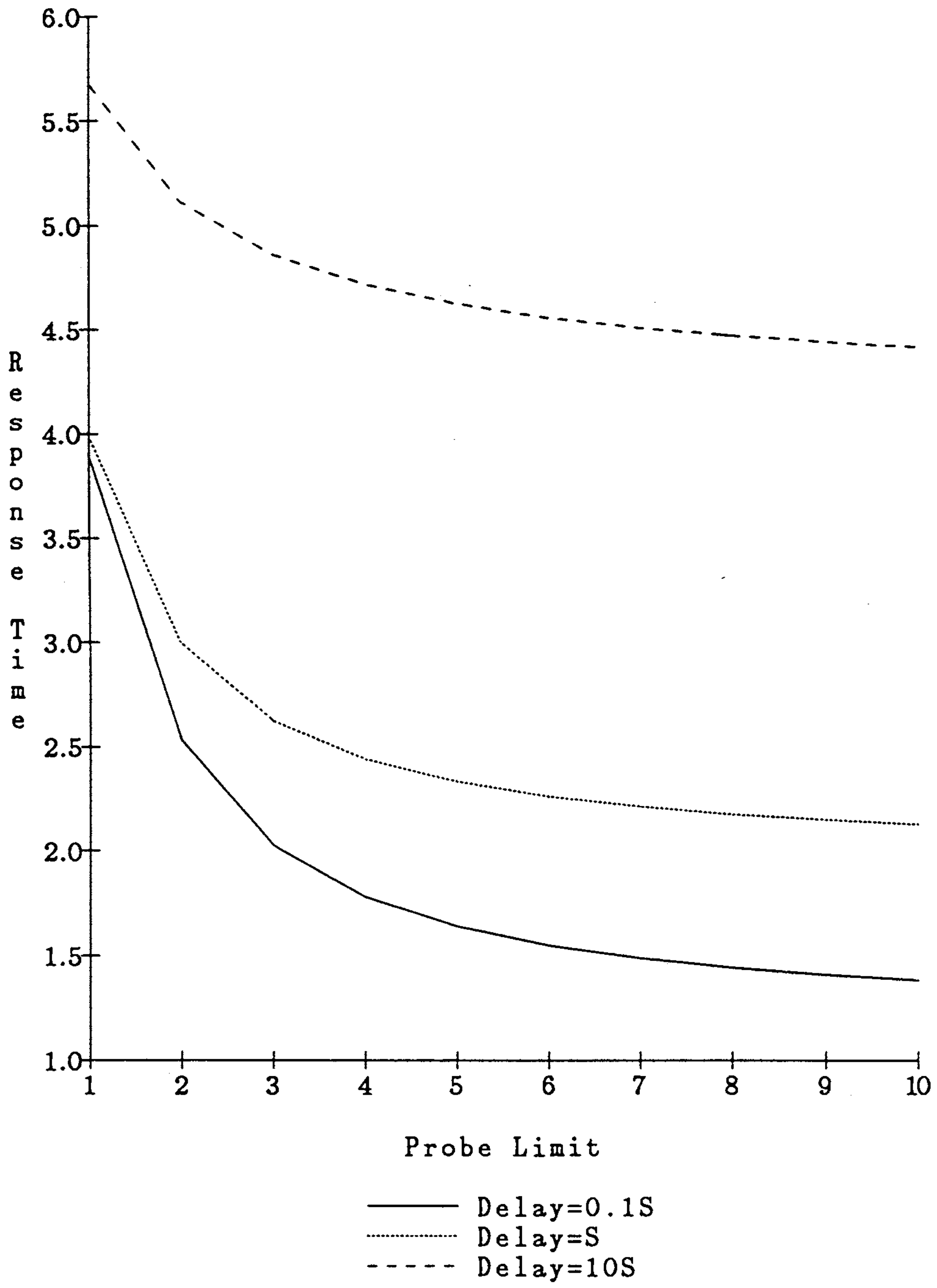

Figure 5.7: Effect of Probe Limit (RP,$\rho_2 = 0.7$)

This is because a probing node is less likely to find a spare job since class-2 nodes are very lightly loaded. Thus, a probing node needs to work harder to find a spare job. In general, it would appear that three or even four probes offer significant performance benefits.

- When delays are larger, the effect of increased probe limit is not as pronounced, except when ρ_2=0.1. This effect is due to the high thresholds which reduce load sharing in the first instance.

- Another possible explanation for the higher sensitivity to L_p when the degree of heterogeneity is large is as follows: Very few jobs execute at class-2 nodes. Thus, after an unsuccessful set of reverse probes, the node waits for a long time before another probe can be made. Thus, each time probes are made, it becomes much more critical that a spare job be found.

Effect of varying thresholds (T_1, T_2)

The motivation behind these tests is to determine the sensitivity of thresholds in either class. This may provide some clues about how critical it is to choose the proper thresholds. Initially, the optimal threshold pair, (T_1, T_2) is determined for the loads and delays of interest. Then, while keeping T_2 fixed at its optimal value, we vary T_1 over a large range. Next, we keep T_1 fixed at its optimal value and vary T_2. Figures 5.8 and 5.9 show the effects of varying T_1 and T_2, respectively, for load (0.9,0.1), and delays $0.1S, S$, and $10S$. Also shown in the graphs are the results from the $M/M/1$ and the perfect load balancing at zero cost $M/M/20$ systems. Figures 5.10 and 5.11 depict the results of identical tests conducted when the loads were (0.9,0.7), respectively, for the two classes.

From Figures 5.8 and 5.9, we can see that the performance of the algorithm is very sensitive to T_1, particularly when the delays are $\leq S$. In these cases, $T_1 = 0$, for optimal performance. For a delay of $10S$, the optimal value for T_1 is 4. In comparison, the performance is insensitive to changes in T_2, showing an almost flat behavior in Figure 5.9. This is because the nodes of class-2 are so lightly loaded that

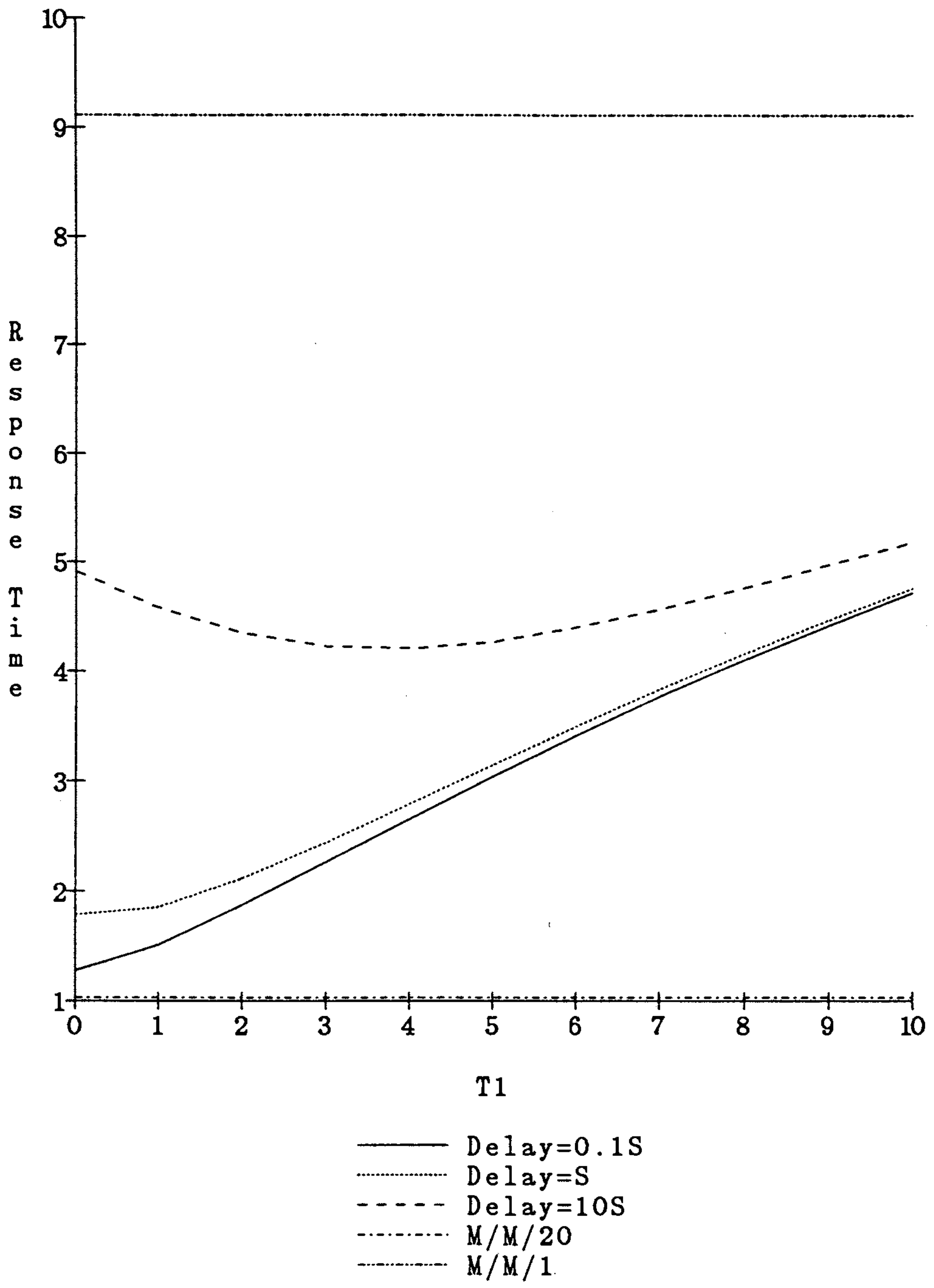

Figure 5.8: Variation of T_1 (FP,$\rho_2 = 0.1$)

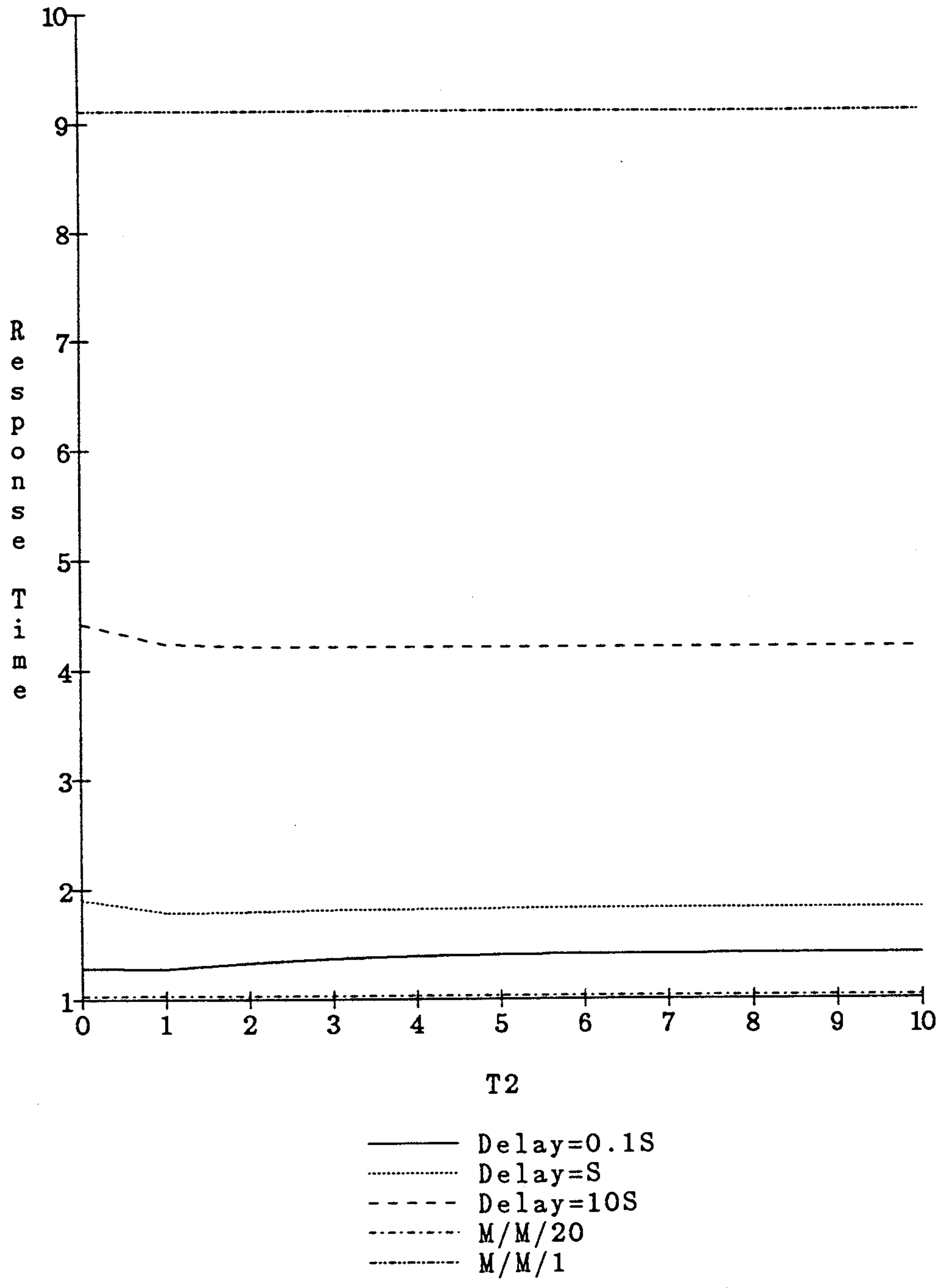

Figure 5.9: Variation of T_2 (FP,$\rho_2 = 0.1$)

increasing the threshold does not substantially increase load sharing. Even when T_2 is low (=1), there is high probability that a class-2 node is at or below its threshold (because $\rho_2 = 0.1$). However, from Figures 5.10 and 5.11, it is clear that when the degree of heterogeneity is small $(\rho_1, \rho_2) = (0.9, 0.7)$, the effects of varying T_1 and T_2 are almost identical, although the performance appears marginally more sensitive to T_1. Furthermore, we have seen that this behavioral change occurs gradually, over the range of heterogeneity we have tested the algorithm (results for $\rho_2 = 0.3$ and 0.5 are not presented here for brevity).

We have conducted similar tests for the Reverse probing algorithm and the results of these tests have shown us behavior similar to that of the Forward probing algorithm. Here too, the performance of the system is much more sensitive to changes in T_1 than it is to T_2, especially in systems with greater load imbalance. The reasons are as follows: When T_2 is increased above a certain amount (1-2), there is no increase in load sharing as the class-2 nodes are so lightly loaded that they rarely go above 1 or 2. However, increasing T_1 above a certain amount drastically reduces the transfer of jobs to class-2 nodes, thus resulting in poor performance. As might be expected, when the systems become more alike, the effects of varying the two thresholds become very similar. Consequently, we will not present the results of these tests conducted on the Reverse probing algorithm.

Effect of biased probing

In all the studies conducted thus far in this research, a probing node was equally likely to choose a class-1 or a class-2 node. Using our earlier notation, $f_{c,d} = 0.5, \forall c, d$ (this is asymptotically exact as the number of nodes in each class grows equally large). Intuitively, one is inclined to believe that some type of focussed probing could be useful in heterogeneous systems. For example, when $(\rho_1, \rho_2) = (0.9, 0.1)$, a node trying to make a forward probe is more likely to find a placement at a class-2 node, while a node making a reverse probe is more likely to find a spare job with a class-1 node. The question then is to determine if and when biased probing can make a substantial difference in performance and reduce the probe limit

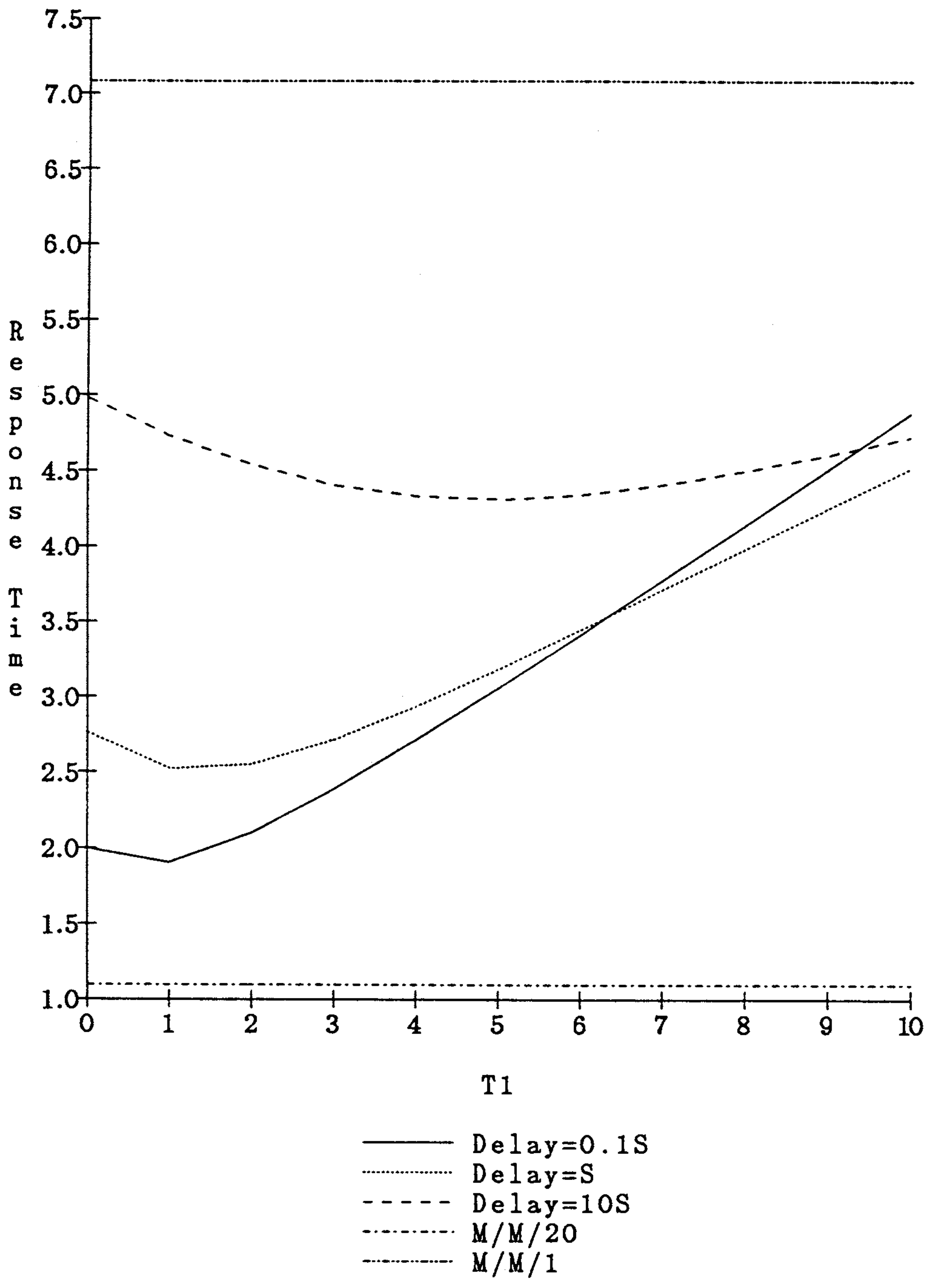

Figure 5.10: Variation of T_1 (FP,$\rho_2 = 0.7$)

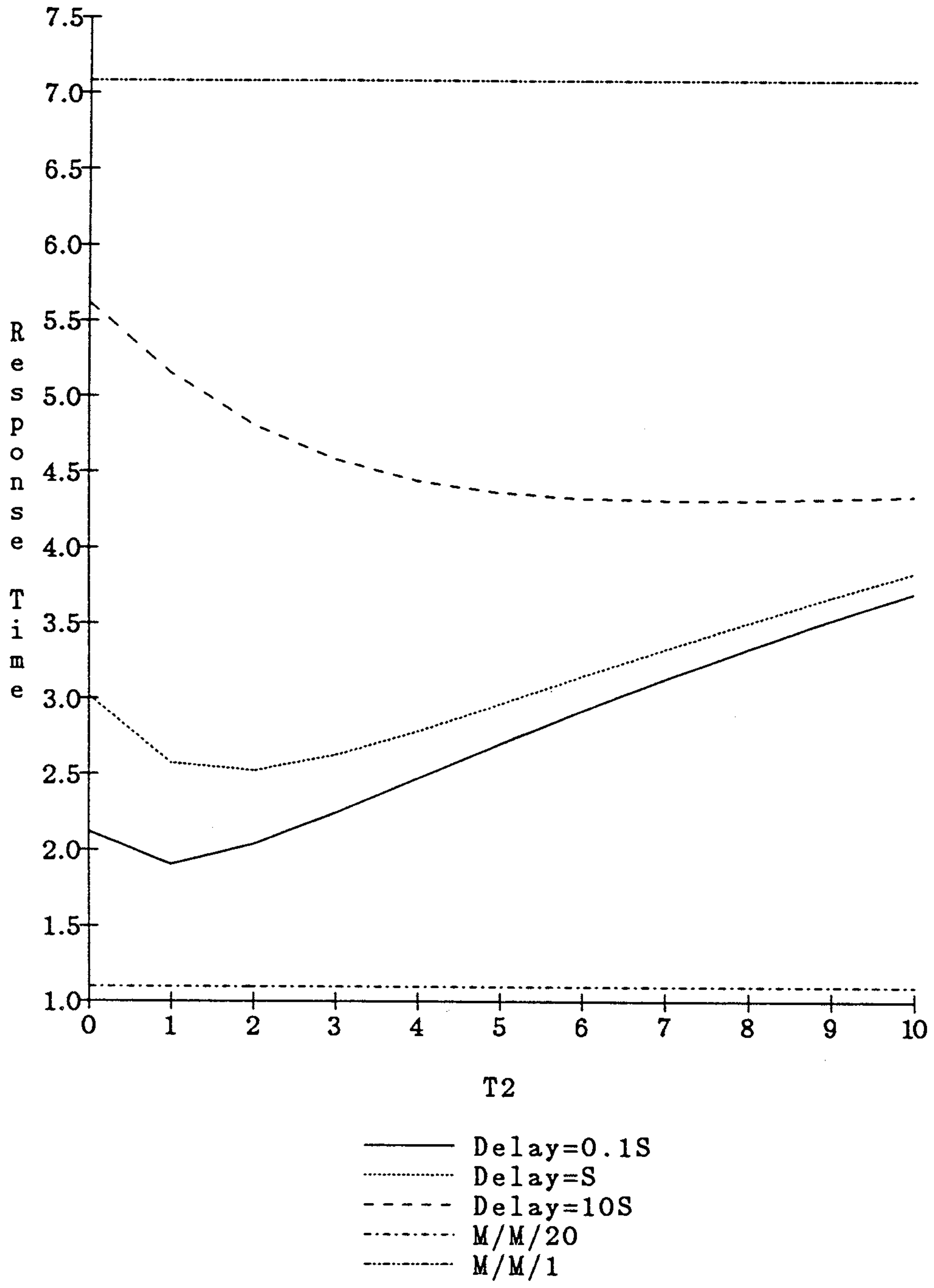

Figure 5.11: Variation of T_2 (FP,$\rho_2 = 0.7$)

in the process. To conduct this part of our study, we have adopted the following procedure. The optimal and next-to-optimal threshold pair (T_1, T_2) were chosen from the earlier experiments on the variation of thresholds. Recall that L_p was 2 in those tests and that probing was unbiased. These two threshold pairs were taken and the performance under biased probabilities was plotted. Here too, we have conducted more tests than we depict.

Initially, $f_{1,1} = f_{2,1} = f_1$, and $f_{1,2} = f_{2,2} = f_2$. Also, $f_1 + f_2 = 1$, at all times. Recall that f_c is the probability that a class-c node will be selected by a probing node of any class. Figures 5.12 and 5.13 depict the results of the experiments conducted on the effects of biasing the probes in the study of Forward probing. Each graph shows 6 curves, two each for three values of delays, $0.1S, S$ and $10S$. The two curves for each delay correspond to the two threshold pairs that we mentioned in the previous paragraph. From Figure 5.12 (load (0.9,0.1)), we see that the equiprobable selection of classes is only marginally worse (at most 5%) than the optimal fractions. However, from Figure 5.12, it is clear that when the classes are more or less balanced, (0.9,0.7), it is best to have unbiased (or close to it) selection of classes for Forward probing in heterogeneous systems.

The reader might wonder as to why we have chosen to depict results from the next-to-optimal threshold pair along with the results of the optimal pair. The decision to select the next-to-optimal was arbitrary. However, we wished to illustrate that there are a large number of parameters that can affect the "optimal" performance. From both Figures 5.12 and 5.13, we have seen that the optimal threshold pair under the equiprobable assumption may not always be "optimal" under the biased selection strategy.

Finally, we study the effect of biased probing upon the Reverse probing algorithm. Here, we will only depict one set of curves, for load (0.9,0.1), because this is quite different from its Forward probing counterpart. The same values of delays are tested as well as again the optimal and next-to-optimal thresholds under the equiprobable assumption are selected. Figure 5.14 shows the results from these

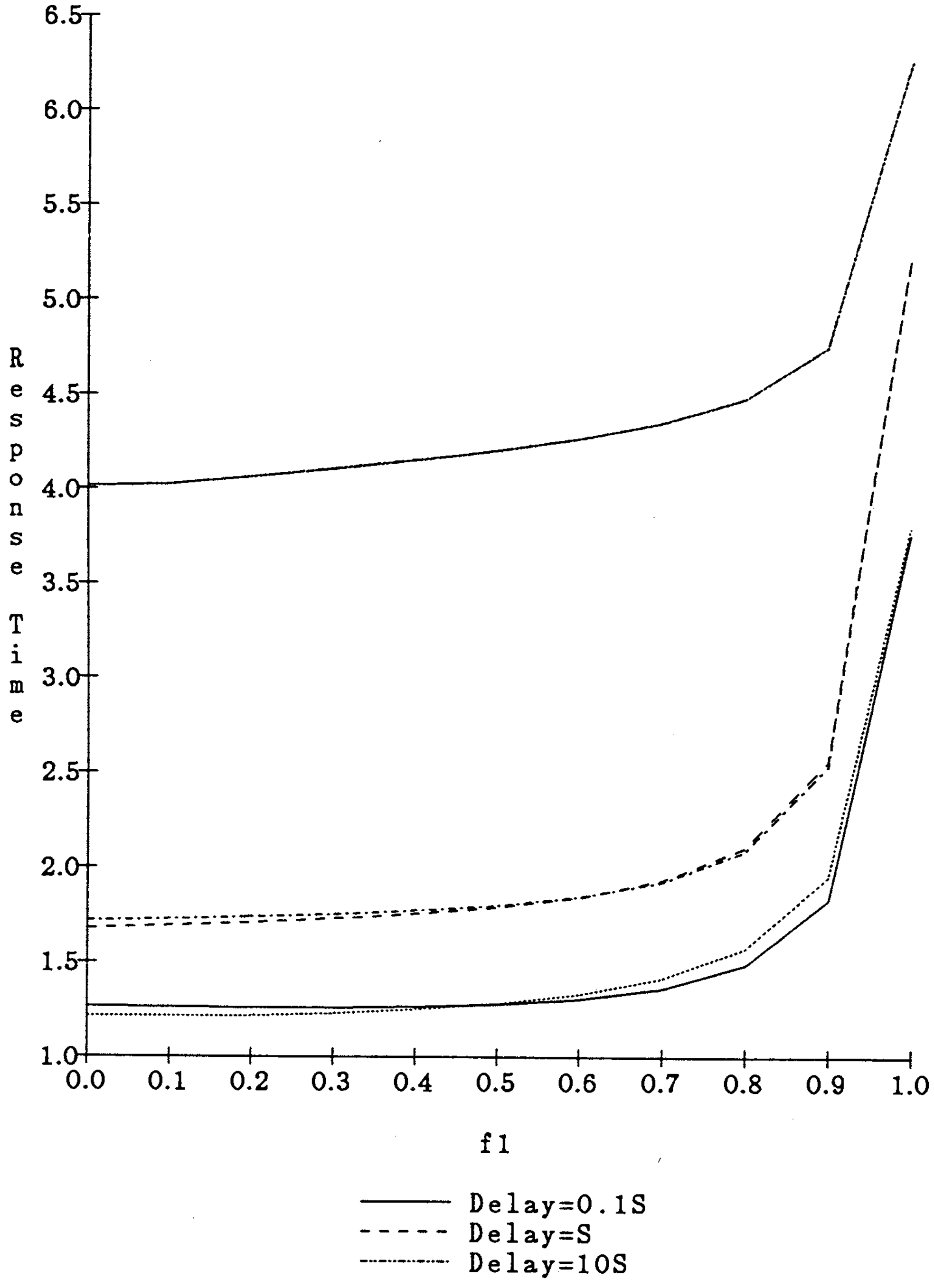

Figure 5.12: Effect of Biased Probing (FP,$\rho_2 = 0.1$)

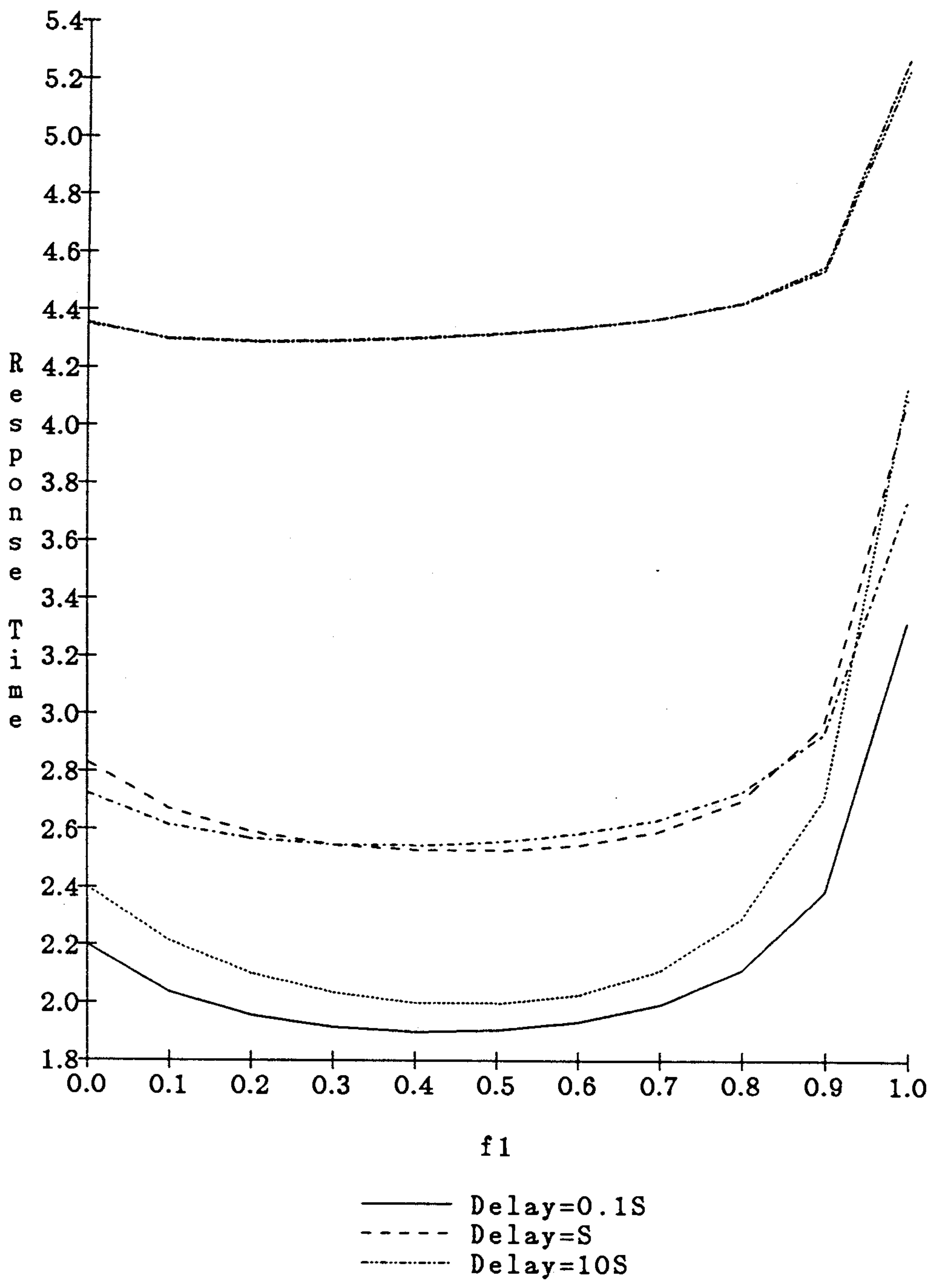

Figure 5.13: Effect of Biased Probing (FP,$\rho_2 = 0.7$)

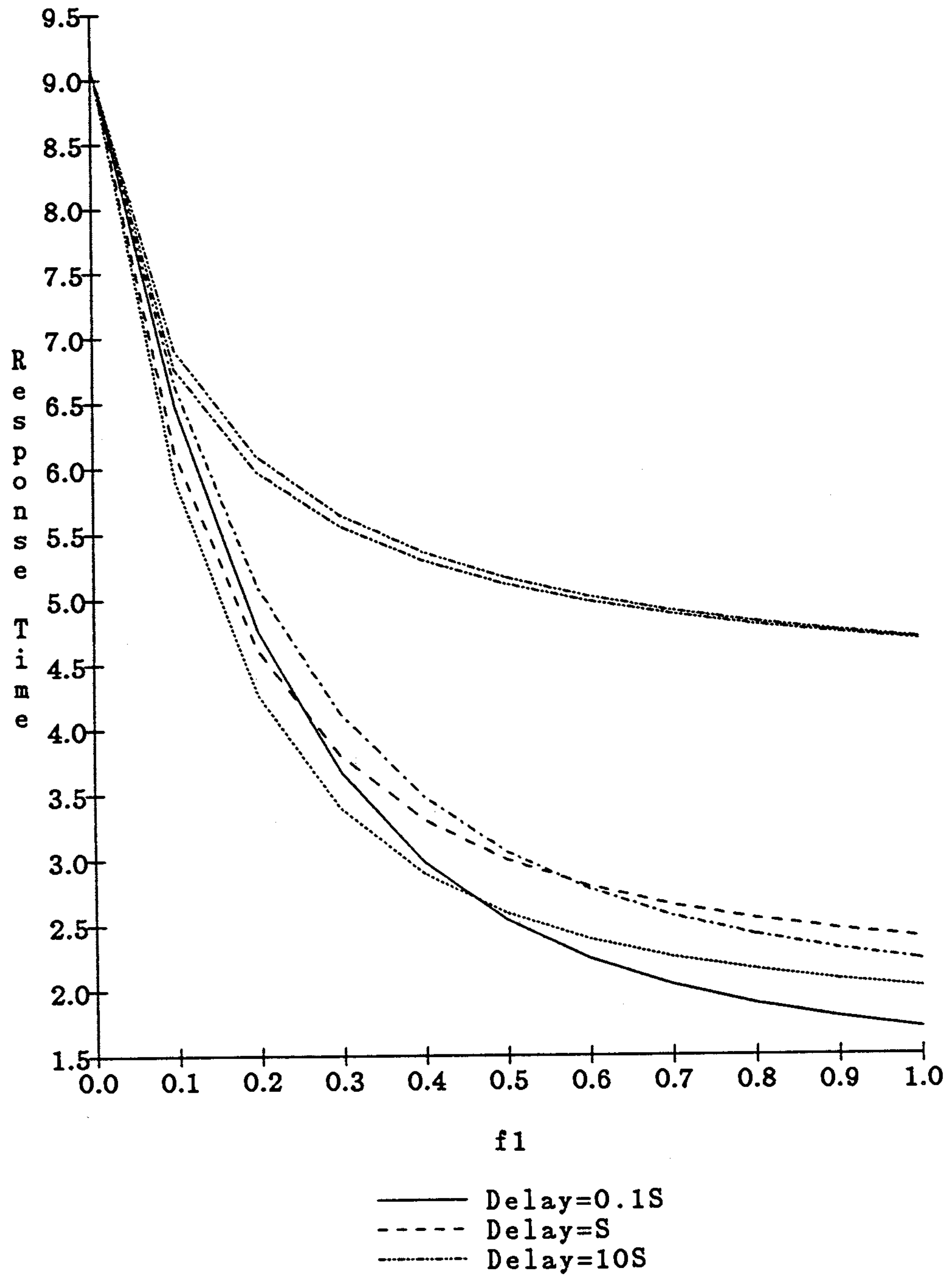

Figure 5.14: Effect of Biased Probing (RP,$\rho_2 = 0.1$)

tests. From the curves shown in the figure, we can see that there is between 30-40% improvement by correctly biasing the probabilities, when the delays are $\leq S$. The effect is less pronounced for higher delays. This result is explained by the fact that Reverse probing is much more sensitive to probe limit when the imbalance in systems is large. Thus, each probe is very important and when $L_p = 2$, biasing appears to provide substantial benefits. In any event, as the degree of heterogeneity is decreased, the effect of biased probing becomes less prominent, as was seen in the case of Forward probing.

Thus, from our studies concerning biased probing, we are able to reach the following conclusions: Biasing appears to have a second or third order effect on response times, except in the case of Reverse probing in highly heterogeneous systems. However, even with the inclusion of proper biasing, Reverse is unable to outperform unbiased Forward for such systems. We have seen from the graphs on biasing that choosing incorrectly biased probabilities has a much more detrimental effect on performance than equiprobable selection. If the biasing probabilities were optimized for a certain load pattern and the system load changed over time (for example, a load of (0.9,0.1) changed to (0.1,0.9) for the two classes) the old values of the bias could result in highly sub-optimal performance.

Performance of Biased Random

In the description provided for the Random assignment algorithm, it was stated that any node in the system was equally likely to be a recipient for a transferred job (Unbiased Random). Thus, in a system with two classes, a node executing the "Unbiased" Random algorithm will transfer approximately half its spare jobs to either class of nodes (in the limiting case as the number of nodes in each class become equally large). We now present results of Random assignment when the class selection for transferring a job is biased towards one class or another (with nodes within a class being selected in an equiprobable manner, similar to the probing case).

Figures 5.15 and 5.16 depict the results of the tests conducted on the Random

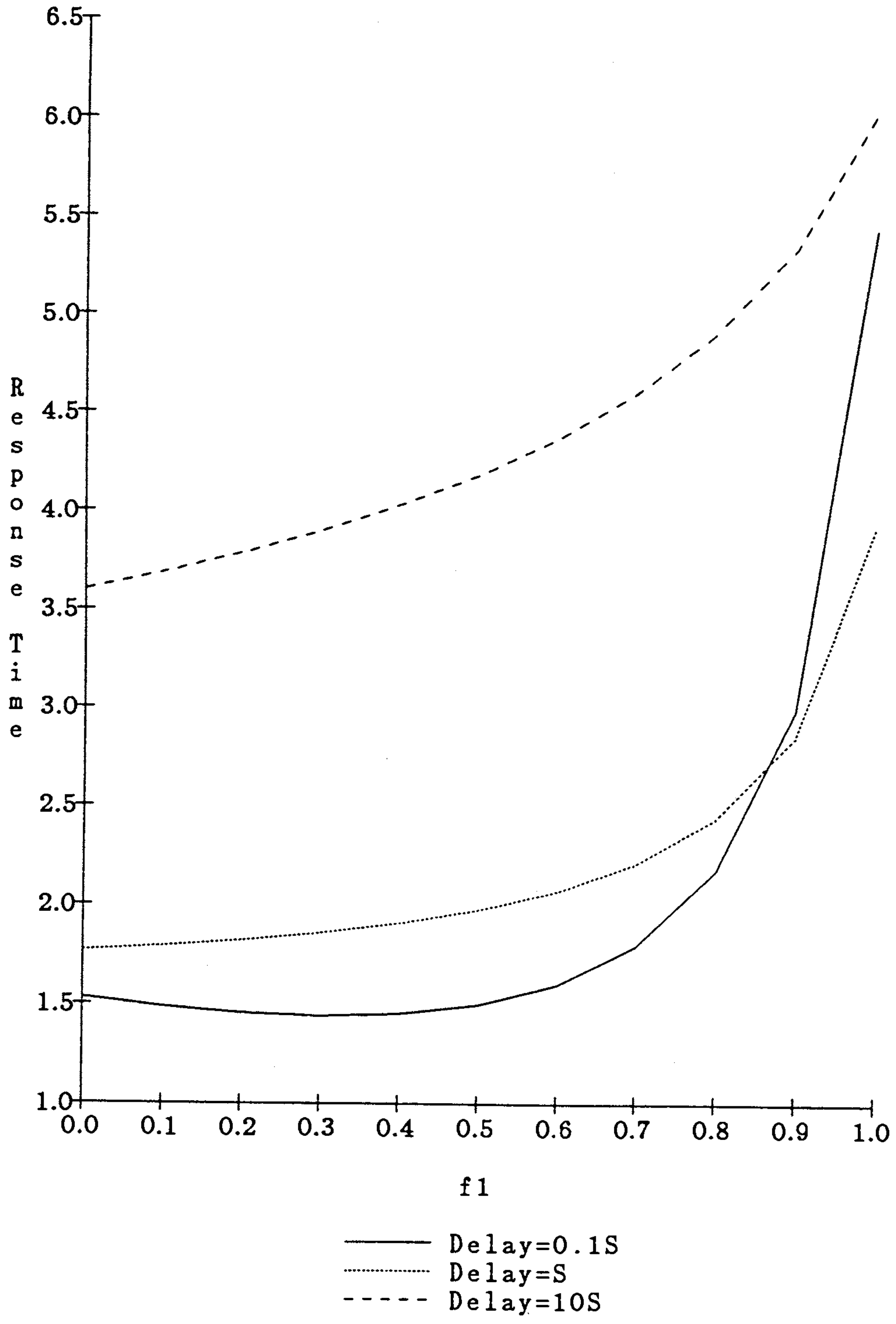

Figure 5.15: Effect of Biased Random ($\rho_2 = 0.1$)

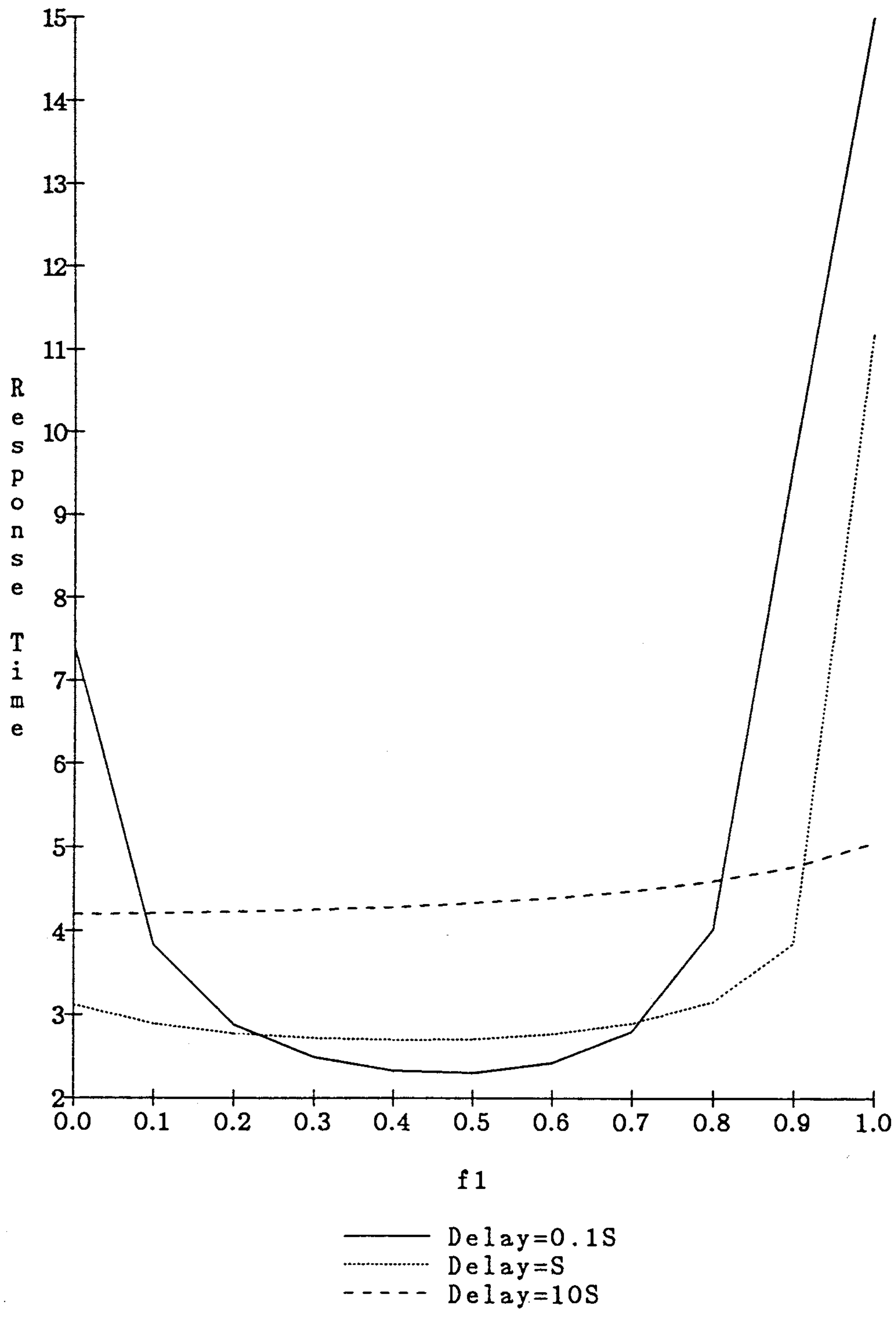

Figure 5.16: Effect of Biased Random ($\rho_2 = 0.7$)

algorithm. (ρ_1, ρ_2) are (0.9,0.1) and (0.9,0.7) in Figures 5.15 and 5.16, respectively. The X-axis depicts the probability that a class-1 node is selected to be a recipient of a transferred job. From the two figures, we can make the following deductions:

- When the degree of heterogeneity is high, as in Figure 5.15, there is an impact of biasing the probabilities when the job transfer delays are high ($10S$). The improvement in that case over the unbiased Random is about 20%. However, for moderate delays, the difference is almost nil (delay=S) and the unbiased version actually performs better by about 7% at delay = $0.1S$.
- When the systems become more balanced, as in Figure 5.16, unbiased Random is clearly superior to any biased Random (except at delay = $10S$, where it is worse by about 4%).

Because the gain due to biasing does not appear very significant, all further references to Random will imply an "Unbiased" version of that algorithm.

Optimal Normalized Response Times

In this study, we vary the job transfer delay from $0.01S$ to $100S$ and determine the optimal performance achieved by the two probing algorithms as well as the Random assignment algorithm. The results of these experiments are shown in Figures 5.17 and 5.18. The Y-coordinate depicts the optimal response time normalized by the corresponding $M/M/1$ response time. Thus, the smaller the value on a curve, the greater is the gain over the corresponding $M/M/1$ value. The results for each of the four loads are depicted in each curve, along with the corresponding results from the Random assignment. The reason we are interested in comparing the probing algorithms against the Random algorithm is because it will provide an idea of the range of parameters over which probing is useful.

Figure 5.17 depicts the results for the Forward probing and Random assignment algorithms. From the figure, we are able to infer the following facts:

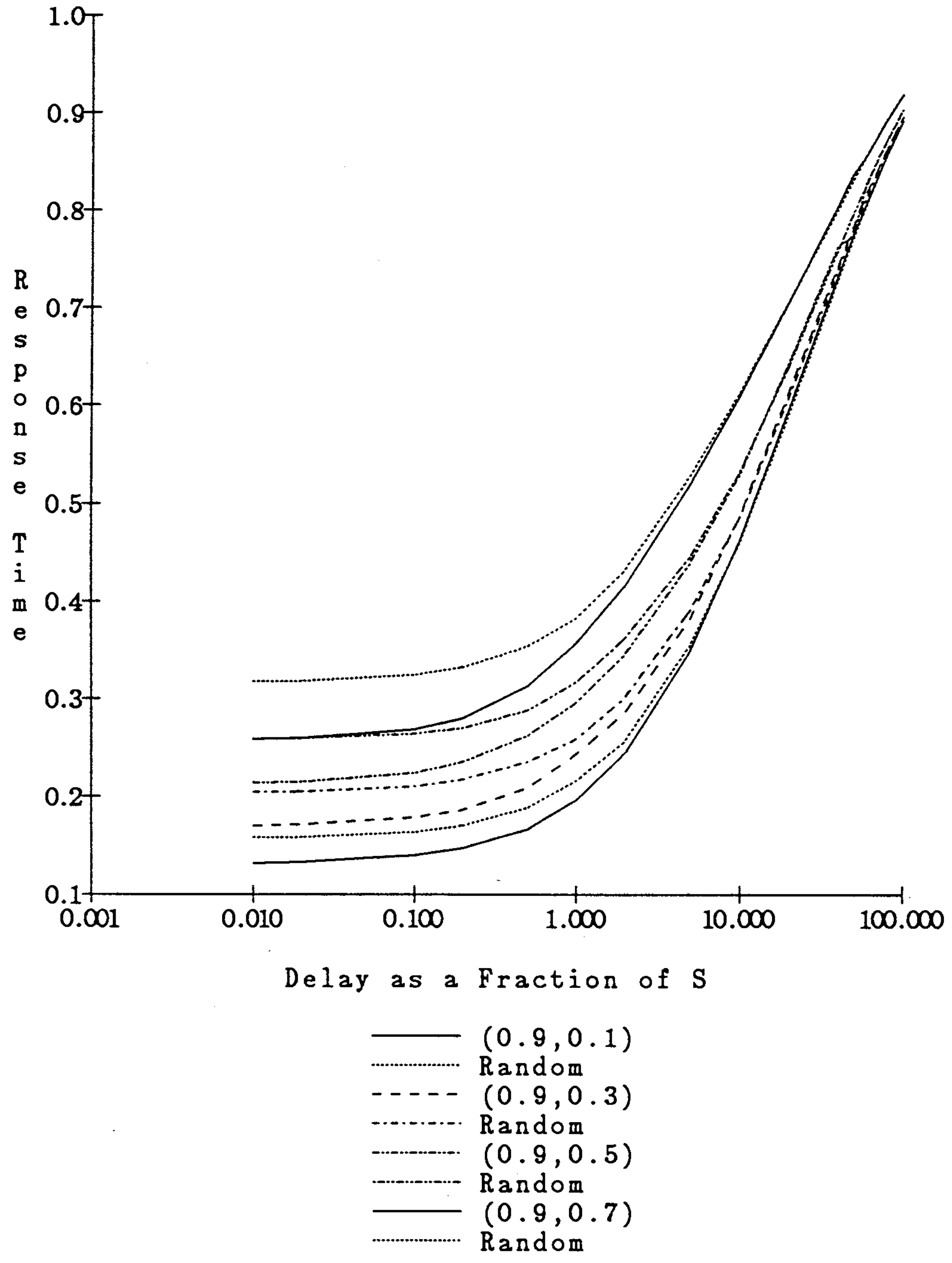

Figure 5.17: Optimal Normalized R.T. (FP)

- There is performance benefit from probing when delays are $\leq 2 - 3$ times the average service time, as can be seen from the curves. For higher values of delays, the curves for the probing and random algorithms become almost indistinguishable, from the standpoint of response time. This probably means that the delays are so large that there is almost no correlation between the state of a remote node when a probe is made and the state of that node after a job is transferred to it.

- While probing may cease to be useful at high delays, load sharing provides considerable benefits over the $M/M/1$ performance. This is very easily seen from the curves, which show that even at delays up to $100S$, there is about 7-10% performance gain. We had seen this phenomenon even for homogeneous systems. However, there is one major difference: For example, when $(\rho_1, \rho_2) =$ (0.9,0.1), the net load over all the nodes in the system is 0.5. For homogeneous systems with load of 0.5, the maximum delay at which load sharing is useful is about 3S. Thus, the imbalance in the loads greatly increases the range of delays for which load sharing is applicable. Further, the greater the imbalance in the systems, the larger is the gain over the corresponding $M/M/1$ value. Thus, the curves for (0.9,0.1) are situated lower than those for (0.9,0.3) and so on.

- The greater the imbalance in the systems, the smaller is the difference between the random and probing algorithms. This effect is more pronounced for lower delays as can be seen from Figure 5.17, where the pairs of curves appear to be more separated at higher values of ρ_2. This may be explained as follows: When both classes of nodes are busy, there is a greater likelihood that a random assignment will result in a transfer to an already busy node.

Figure 5.18 depicts the results of similar tests conducted on the Reverse probing algorithm. From Figure 5.18, we are able to see that the behavior for Reverse probing is quite different from that of Forward probing (Figure 5.17) The main

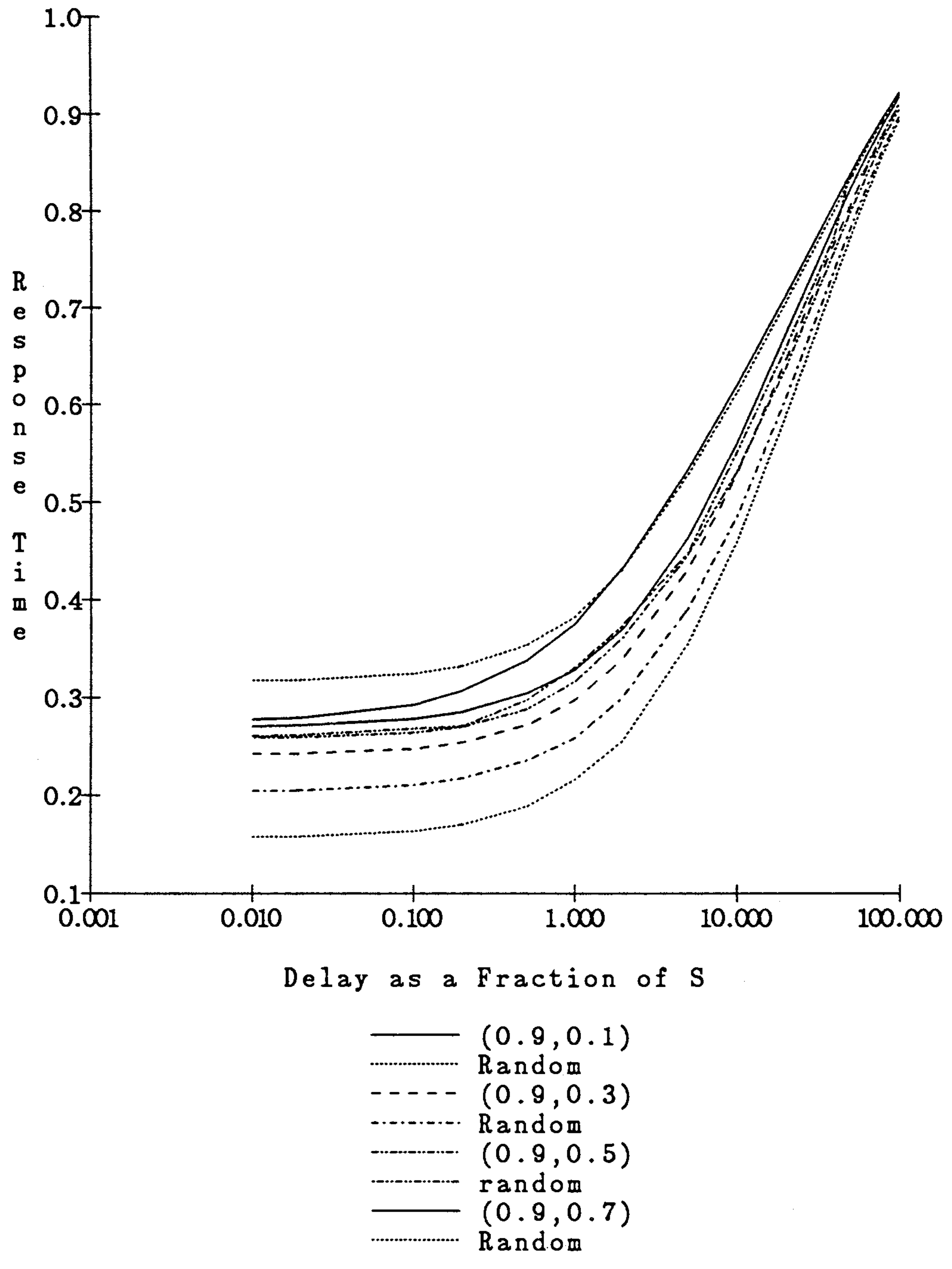

Figure 5.18: Optimal Normalized R.T. (RP)

points worth noting are as follows:

- When the imbalance between the classes is high, Random assignment is significantly better than Reverse probing. For example, the curves corresponding to loads (0.9,0.1) show that the Reverse curve approaches the Random curve only asymptotically. This is easily explained as follows: Recall that Reverse probes are sent by a node only when it is below the threshold when a job completes. Now, in very lightly loaded nodes, very few jobs actually execute. Thus, there are few opportunities to balance the load. On the other hand, the Random assignment strategy is quite effective in highly unbalanced systems, with a high probability of transferring jobs to idle nodes in class-2.

- As the degree of heterogeneity decreases (i.e., the systems become more alike), the gap between Reverse and Random decreases. From the curves, it is seen that Reverse actually performs better than Random when the loads are (0.9,0.7). At this point, class-2 nodes are able to perform reasonably effective load balancing and the higher loads make the Random assignment less suitable.

In the previous two graphs, we have determined the optimal response times of the algorithms under different values of total system load. We kept ρ_1 fixed at 0.9 and varied ρ_2 between 0.1 and 0.7, in steps of 0.2. In Figure 5.19, we depict results of tests where the total system load is kept fixed at 0.5, while changing the distribution of the load between the classes. Thus, the three sets of curves depict results for (ρ_1, ρ_2) = (0.5,0.5), (0.7,0.3) and (0.9,0.1), i.e., from a homogeneous system to a very heterogeneous one.

From Figure 5.19, we are able to make the following observations:

- As might be expected, the benefits of load sharing over the M/M/1 response time is greater as the degree of heterogeneity is increased. Thus, the curves for (0.5,0.5) lie at the top with (0.7,0.3) next and the curves for (0.9,0.1) are the

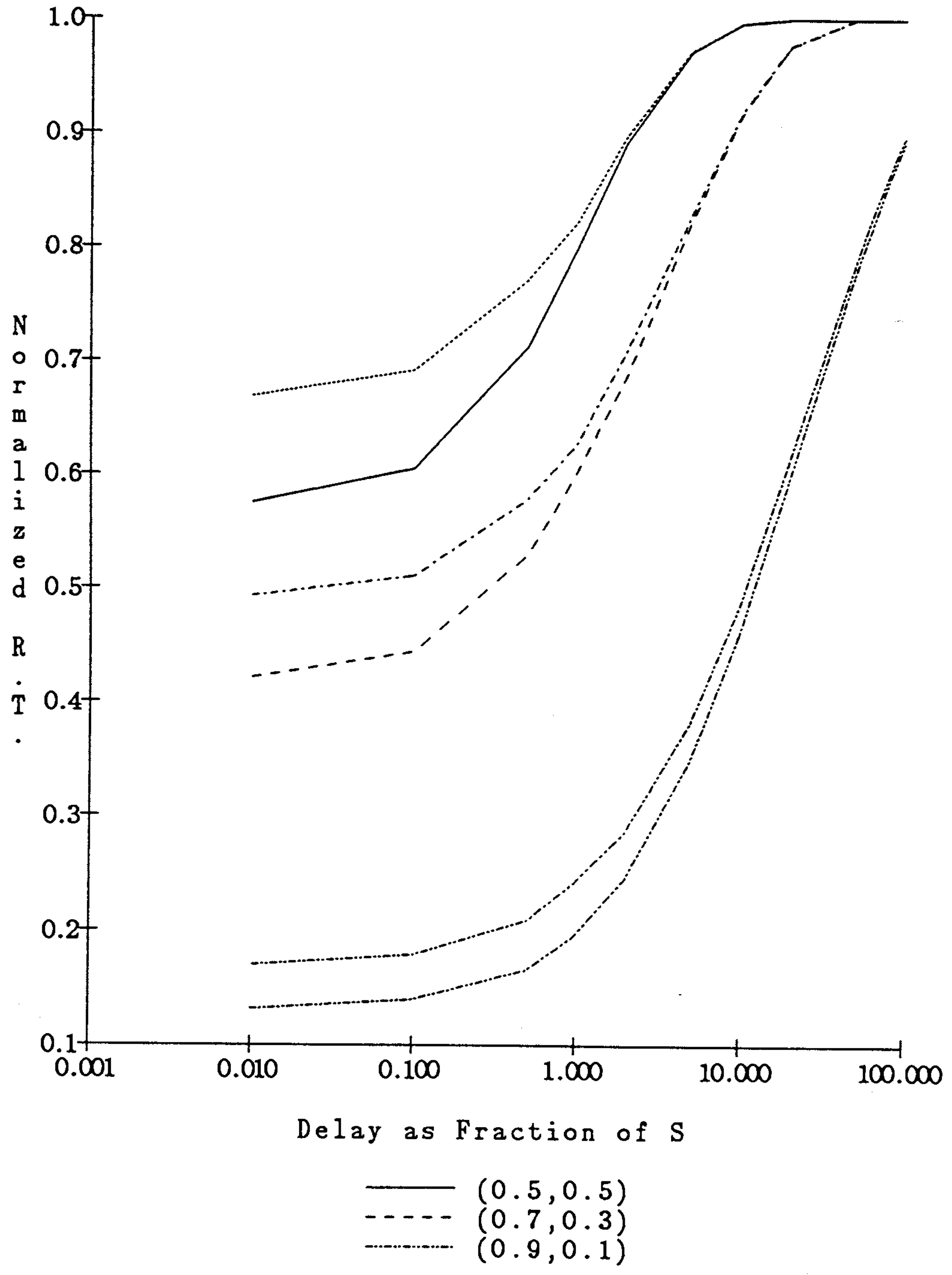

Figure 5.19: Optimal Normalized R.T (Fixed System Load, FP)

lowest in the graph. In systems with greater heterogeneity, the performance of Random is less distant from the probing curves. This is because when the classes are highly unbalanced, a Random job transfer is likely to reach an idle node, particularly when the average system load is low. The separation between the Random and probing curves is most pronounced when (ρ_1, ρ_2) is (0.5,0.5).

- At delay = $100S$, (0.9,0.1) generates about 10% improvement over the corresponding $M/M/1$ response time. However, for (0.5,0.5) and (0.7,0.3), delays of about $5S$ and $20S$ produce the $M/M/1$ response time, respectively.

5.4.2 Type-2 Heterogeneous systems

In this set of experiments, we study systems in which the nodes in the two classes process jobs at different speeds. Thus, transferring a job from a longer queue to one that is shorter may not necessarily result in lower response times (even after accounting for delays). Nodes of class-1 have the same service rate as in **type-1** systems, i.e., $\mu_1 = 1$. The service rate for nodes of class-2, on the other hand, is varied from 0.2 to 0.8. Thus, we will study systems where the nodes possess different degrees of heterogeneity, as in **type-1** systems. The loads at nodes of either class is kept fixed at 0.8. This is for ease of comparison and we believe that in no way does this limit the validity of our experiments. It is relatively easy to conjecture the behavior of systems in which the above restriction may not apply.

Because the results of **type-2** systems are at times similar to those of **type-1**, we will only emphasize those aspects of **type-2** systems that are startlingly different from **type-1** systems. The remaining results will be described in brief. Before we conducted any substantial tests for **type-2** systems, we compared the results of the analytical models with simulations. This is for the same reason as in **type-1** systems: The solution technique we have adopted is only an approximation and we need to determine how close the model is to practical, finite sized systems. Exactly

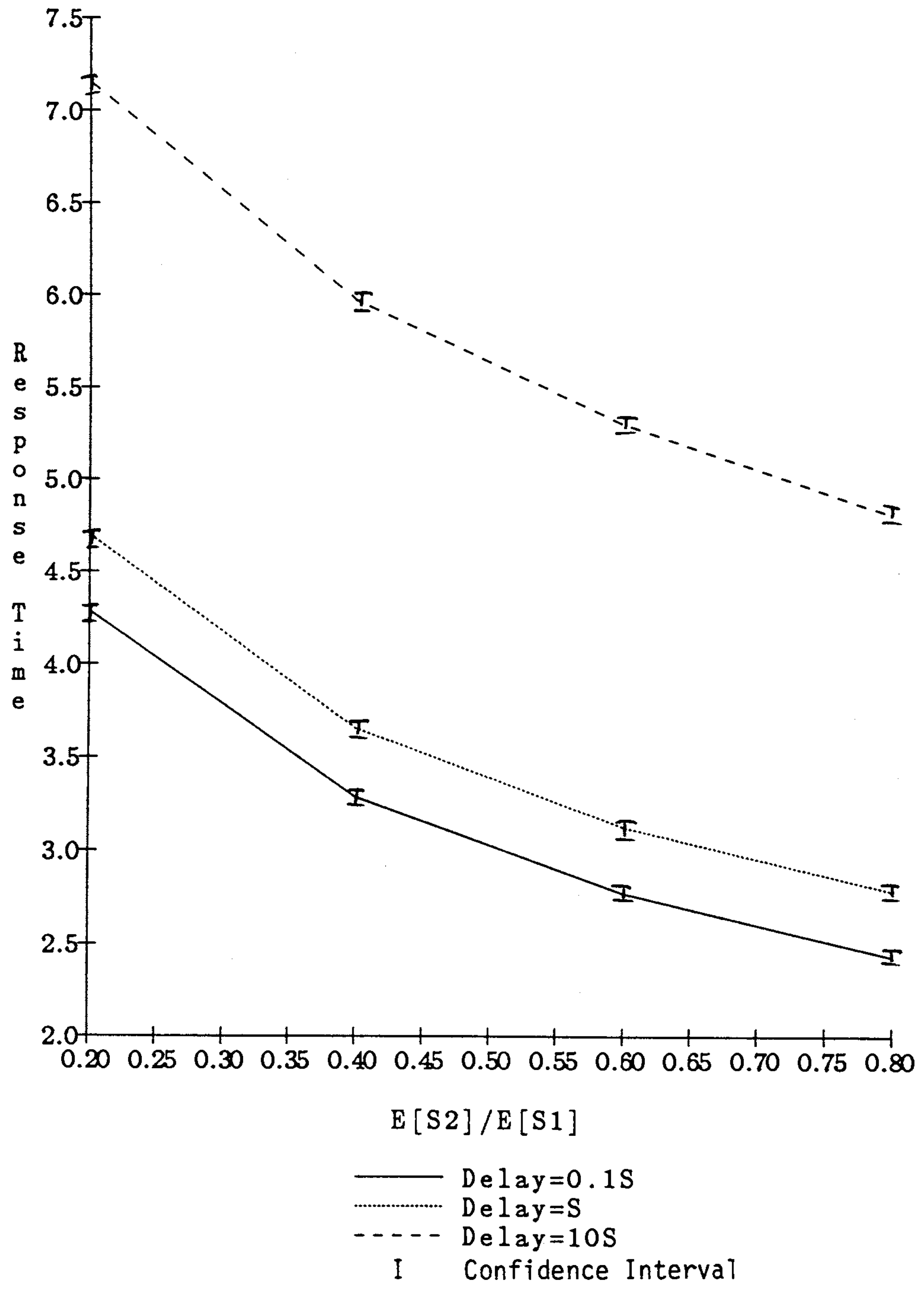

Figure 5.20: Validation with Simulations (FP, Type-2)

the same kind of tests were conducted as for **type-1** systems and the results were very similar. The average response times generated by the analytical model for μ_2 = 0.2, 0.4, 0.6 and 0.8 are plotted for three values of delays, $0.1S, S$ and $10S$ in Figure 5.20. It was seen that between 10-15 nodes were needed in each class for the simulation results to be very close $2 - 5\%$ to the analytical models. The 95% confidence intervals for the simulation results were computed using the Student-t tests and were seen to be within $\pm 5\%$ of the sample mean. These confidence intervals are also depicted in the curves of Figure 5.20.

While we have conducted all the subsequent tests on all four values of μ_2, we will only present curves for $\mu_2 = 0.2$ and 0.8. The results corresponding to the other two values of μ_2 will only be described in brief unless they happen to be counter-intuitive. The reasons for adopting this approach are identical to those given in the case of **type-1** systems. Furthermore, we assume that $S_1 = S$.

Effect of probe limit L_p

These tests were conducted to determine a good choice for the probe limit L_p. Figure 5.21 depicts the results of such tests for the Forward probing algorithm in a system in which $\mu_2 = 0.2$ (i.e., class-2 nodes process jobs at 1/5th the speed of class-1 nodes). It is seen from this figure that there appears to be some advantage if $L_p = 3$ when the delays are $\leq 0.1S$. For higher delays, however, a limit of 2 seems adequate. Low delays encourage active load sharing and this in turn makes the performance more sensitive to probe limit, as opposed to very high delays like $10S$, where the load sharing performed is minimal. Similar behavior was observed for values of $\mu_2 = 0.4$, 0.6 and 0.8. Thus, although $L_p = 3$ might be more appropriate for low delays, we will maintain $L_p = 2$, unless specifically mentioned otherwise.

Effect of varying (T_1, T_2)

The effect of thresholds on performance was studied for **type-1** systems. It was seen that incorrect values of operating thresholds could lead to poor performance. Thus, we are interested in determining the relationship between thresholds

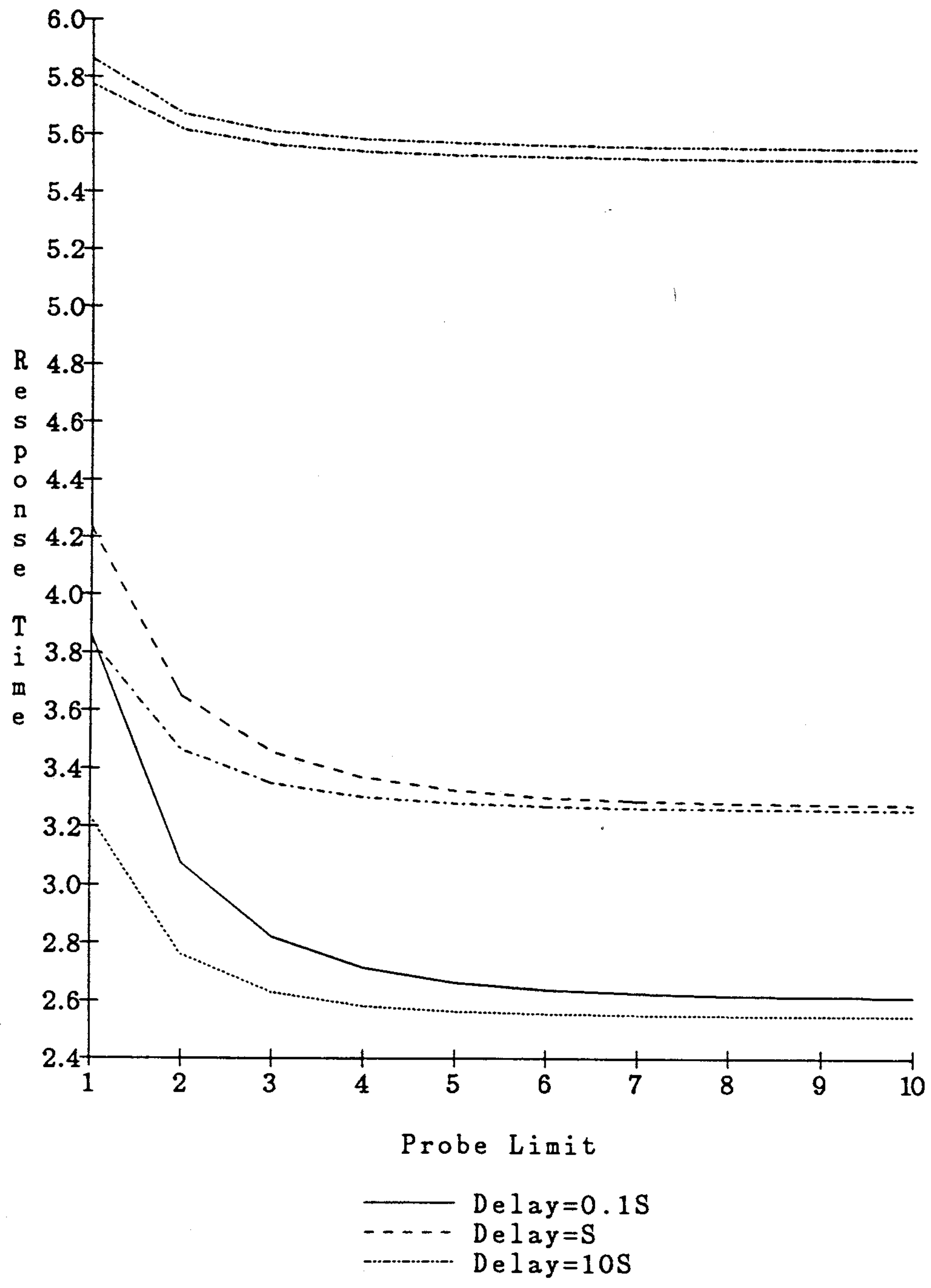

Figure 5.21: Effect of Probe Limit (FP,$\mu_2 = 0.2$)

and performance in the case of **type-2** systems as well. The strategy adopted here is the same as in **type-1** systems. For the loads and delays tested, (T_1, T_2) corresponding to the optimal response time is determined. Then, keeping T_2 fixed at its optimal value, we vary T_1 over a large range. Next, T_2 is varied, keeping T_1 fixed at its optimal value. In each of the figures, we will present the results of the $M/M/1$ and LB_2 response times corresponding to the parameters in question. Recall that LB_2 is the lower bound for **type-2** systems that we had mentioned earlier in this section. Details of this algorithm and its analysis are presented in Appendix C.

The results of these tests for $\mu_2 = 0.2$ are depicted in Figures 5.22 and 5.23. From Figure 5.20, the behavior of the system in response to varying T_1 can be explained by the following arguments: For the most part, the optimal value of T_2 is 0. This implies that very few jobs are transferred to class-2 nodes although they are more likely to transfer jobs out. This is facilitated by the fact that the optimal value of T_1 is 2 or higher, depending upon the delay. If T_1 were lower, say 0 or 1, fewer jobs could be transferred between class-1 nodes, resulting in poor performance, as may be seen from the curves in Figure 5.22 (class-1 nodes are likely to be higher than T_1 for low values of T_1). However, if T_1 was set higher than a certain value (2 or 3 in the case of delays $\leq S$), this would mean that virtually no jobs may be transferred out of class-1 nodes to balance the instantaneous loads among class-1 nodes.

The results obtained by varying T_2 are presented in Figure 5.23. From this figure, it is observed that the performance is much more sensitive to changes in T_2 and actually gets much worse than the $M/M/1$ value for delays $\leq S$. The reasons for this behavior are as follows: As mentioned earlier, the optimal value for T_2 is 0 for delays $\leq S$ and 1 for delay $= 10S$. If T_2 is increased beyond the optimal value, there is greater likelihood that jobs will be transferred to class-2 nodes. Because the service rate of class-2 nodes is low, long queues build up and the response times get worse. In fact, there is the possibility that the class-2 queues can approach saturation, particularly in the case of delays $= 0.1S$ and S. This behavior appears

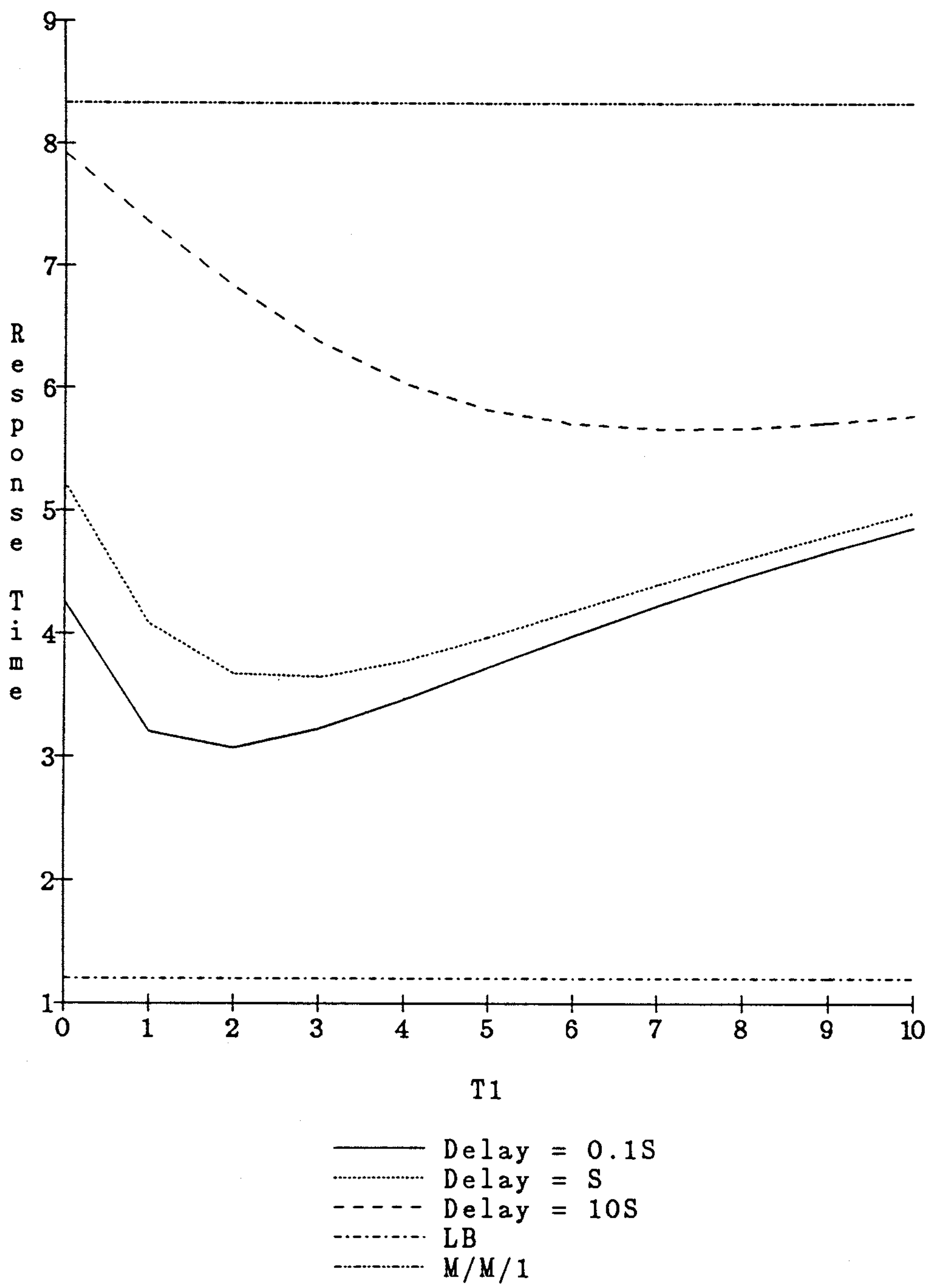

Figure 5.22: Variation of T_1 (FP,$\mu_2 = 0.2$)

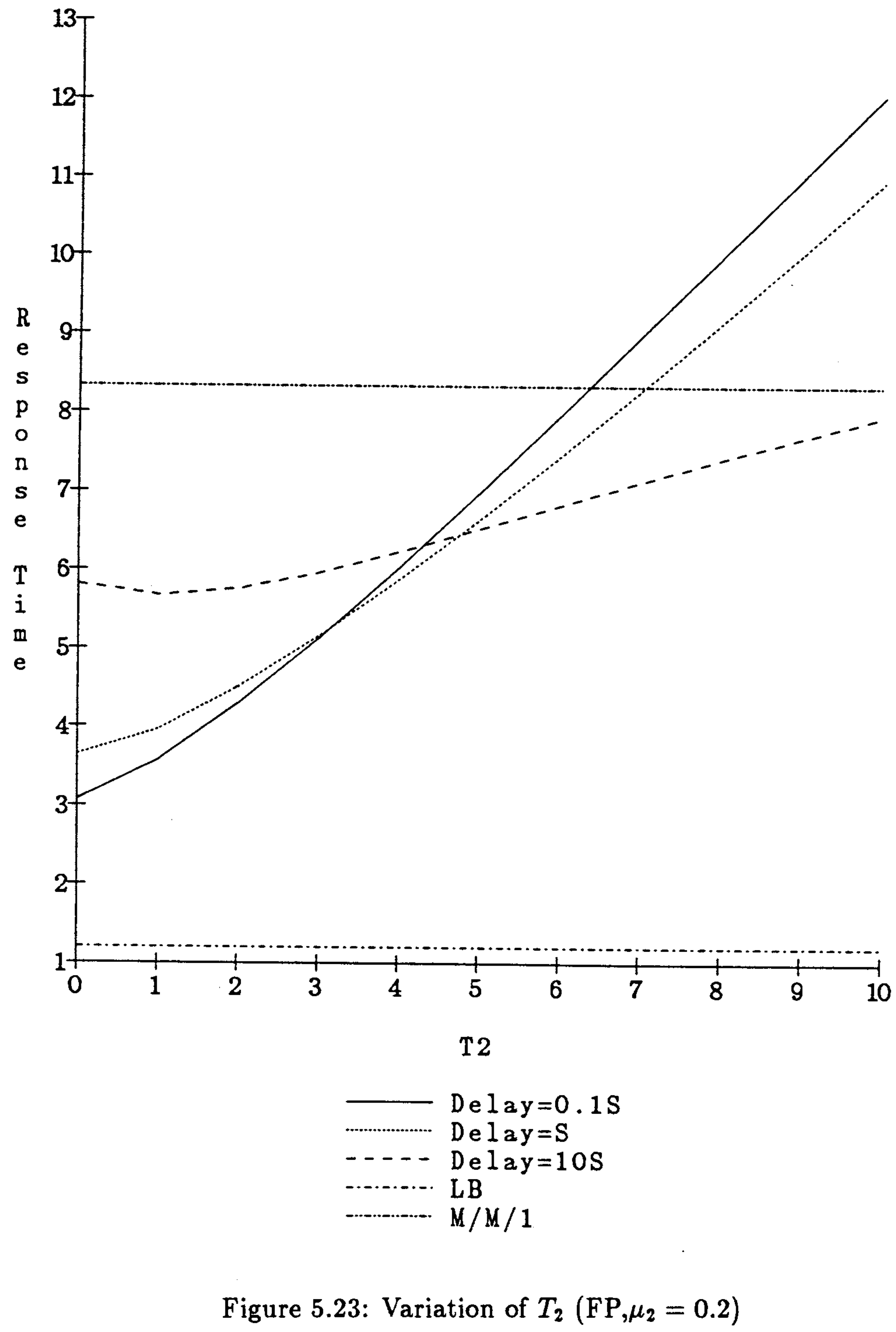

Figure 5.23: Variation of T_2 (FP,$\mu_2 = 0.2$)

to occur when T_2 is around 3 or 4 and the rate of increase of response time appears to be very high.

Figures 5.24 and 5.25 depict similar tests conducted when μ_2 was set to 0.8. Thus, jobs execute 25% longer on class-2 nodes as compared to class-1 nodes and the degree of heterogeneity between the classes is not very high. From these figures, we can the effect of varying the thresholds is very similar to the earlier set of tests when μ_2 was equal to 0.2. The only effect is that the curves are shifted along the X-axis. The optimal value of T_1 is lower than in the previous case. For delay $=$ $0.1S$, $T_1 = 1$ is optimal, while for delay $= S$, it is 2. This implies that class-1 nodes are likely to be more active in load balancing. Because class-2 nodes are not much slower than class-1 nodes, they are able to accept more jobs than in the case of the earlier set of tests. The phenomenon of saturation in class-2 nodes is also observed in Figure 5.25. The explanation is the same as given for $\mu_2 = 0.2$. High thresholds in class-2 nodes result in many job transfers to these nodes from class-1 nodes. Because μ_2 is only 25% less than μ_1, many more jobs need to be transferred to class-2 nodes before this unstable behavior occurs. Thus, the thresholds are higher here than the case when μ_2 was 0.2.

Similar tests concerning the effects of varying T_1 and T_2 were conducted for the Reverse probing algorithm. It was seen that when T_2 is increased (particularly when the imbalance is high), the performance gets worse. This is because high values of T_2 result in more jobs transferred to class-2 nodes as well as fewer jobs transferred out of class-2 nodes to class-1 nodes (which should be the ideal strategy). Thus, class-2 nodes have a tendency become saturated if T_2 is higher than its optimal value. This effect, although less dramatic, is also seen when μ_2 is 0.8, i.e., the imbalance is not very high.

The effect of varying T_1 is seen to be much less pronounced than T_2, especially when the degree of heterogeneity is high. For delays of $\leq S$, the optimal value of T_1 is 1. Increasing T_1 beyond that results in worse performance because higher thresholds result in much less active balancing among class-1 nodes, even though

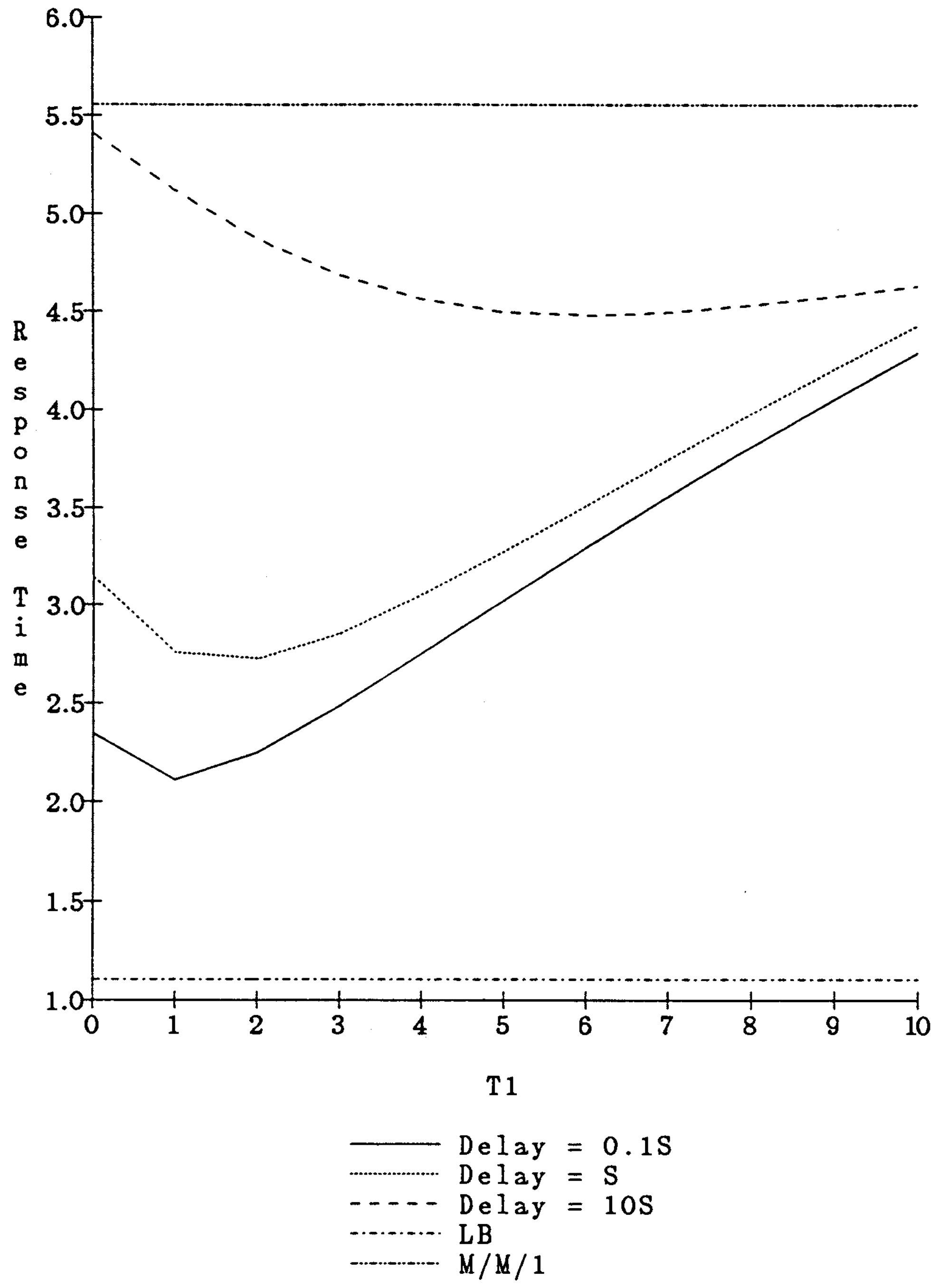

Figure 5.24: Variation of T_1 (FP,$\mu_2 = 0.8$)

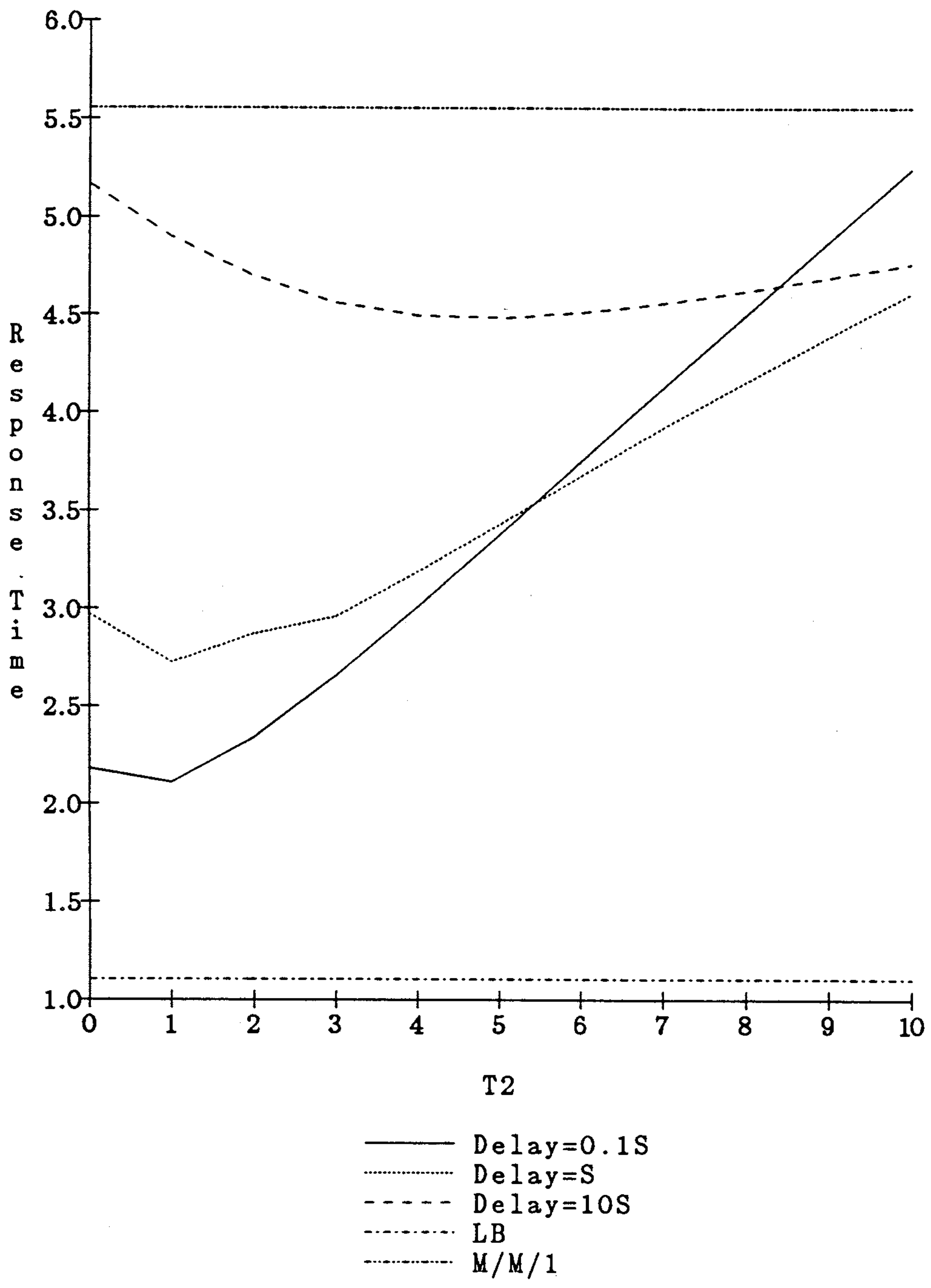

Figure 5.25: Variation of T_2 (FP,$\mu_2 = 0.8$)

they further restrict jobs being transferred to class-2 nodes (particularly since the optimal value of T_2 is 0 or 1, for the range of delays tested).

Optimal Normalized Response Times

Figure 5.26 depicts the results of subjecting the Forward probing algorithm to a large range of delays. The optimal response times are plotted for each value of delay along with the optimal response time generated by the Random assignment algorithm. The tests are conducted for all four values of μ_2, as indicated earlier. From Figure 5.26, we are able to infer the following facts:

- The benefits from load sharing are more pronounced for systems in which the degree of heterogeneity is large. From example, the curve for $\mu_2 = 0.2$ is the lowest of all curves, for most of the range of delays tested (except for very low delays around $0.05S$). The fact that the curve is the lowest signifies that the response time improvement over the corresponding $M/M/1$ value is the greatest.

- The curves for the Random assignment algorithm are seen to be located much higher than their respective probing curves. This means that Random performs much worse than Forward probing for certain values of parameters (at delay $= 0.1S$ and $\mu_2 = 0.2$, Random is about 55% worse). Further, the greater the degree of heterogeneity, the more the displacement. This is quite easily understood by the following argument: Because class-2 nodes are slow, the job transfers that occur to these nodes by random assignment slow down the system by a large amount.

- For large degrees of heterogeneity, the curves corresponding to Random approach their respective probing curves after fairly high values for delays, as compared with **type-1** heterogeneous systems. For example, when $\mu_2 = 0.2$, there appears to be about $5-6\%$ benefit by probing, even with delays as high as $5S$. Thus, the value of remote state information is more pronounced and

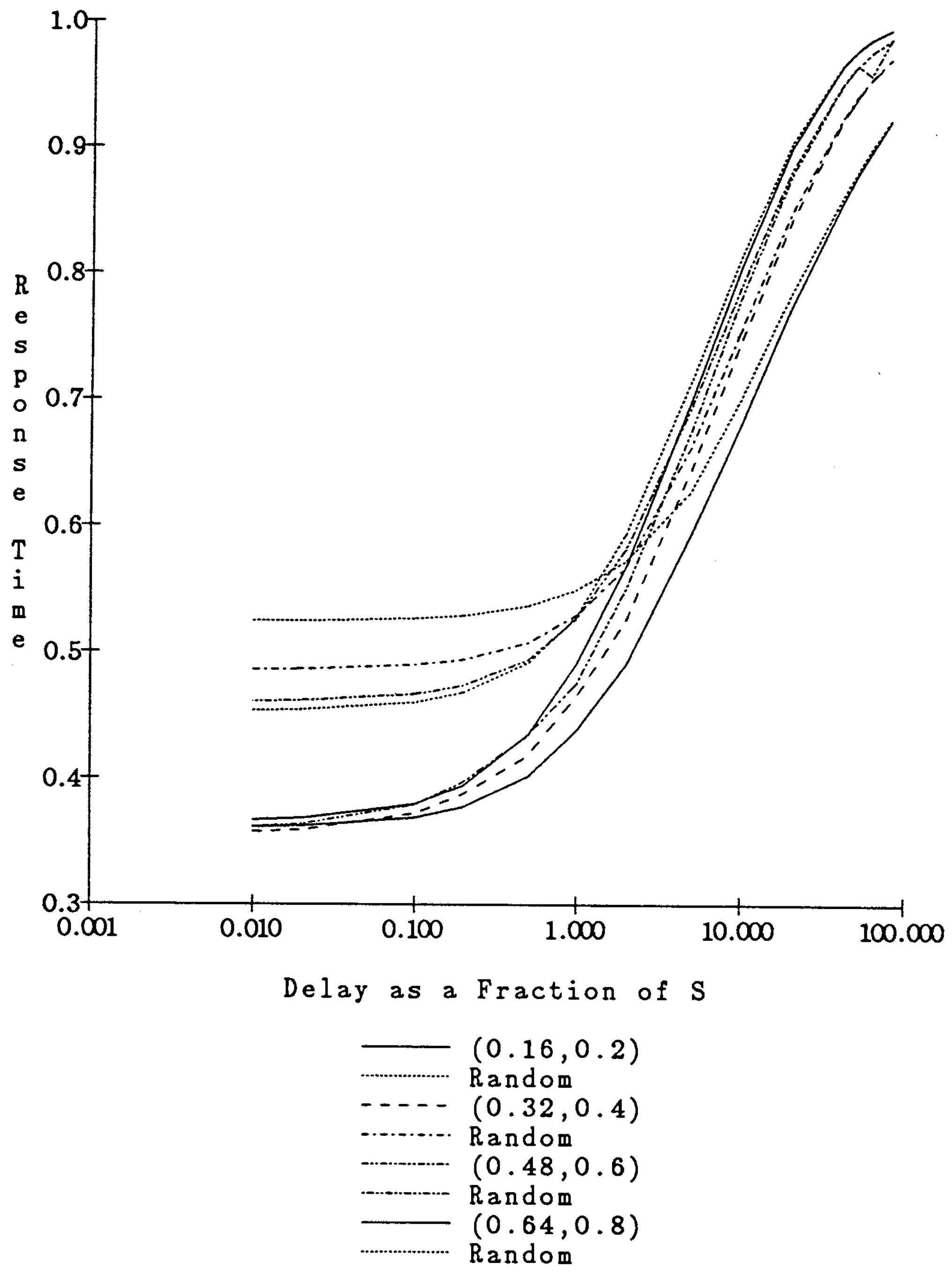

Figure 5.26: Optimal Normalized R.T. (FP)

its effect is significant for higher delays in highly unbalanced **type-2** systems as compared with **type-1** systems.

Figure 5.27 depicts the results of similar tests conducted on the Reverse probing algorithm. The results in this case are very similar to that of the Forward probing example. In general, Random assignment performs quite poorly in comparison to probing, be it forward or reverse. Also, the greater the degree of heterogeneity, the more advantageous it is to probe, as opposed to Random assignment. Here too, as in Forward probing, we see a more significant effect of probing at higher delays than at **type-1** systems.

5.5 Conclusions

In this chapter, we have studied the load sharing problem in the presence of delays when the underlying system is heterogeneous. We have studied two main types of heterogeneous systems, i.e., **type-1** systems where all the nodes have the same processing speeds and capabilities but the arrival rate of local jobs at nodes may not all be the same, and **type-2** systems, where different nodes may process jobs at different speeds (however, the nodes are assumed to be functionally identical). The two algorithms which were studied in this context were called Forward and Reverse. The resulting Markov processes from these algorithms were solved using the Matrix Geometric solution technique, as in Chapters 3 and 4.

The analytical models were then subjected to many interesting tests and the results of these tests and their implications were described in the previous section. In this section, we summarize the most interesting results obtained. Although the tests were conducted for systems with 2 classes, we will attempt to generalize many of the results for an arbitrary number of classes.

- As in the case of homogeneous systems, the solution technique of studying the nodes independently is an approximation of the Markov process for the system. In this chapter too, we compared the analytical results with simulations

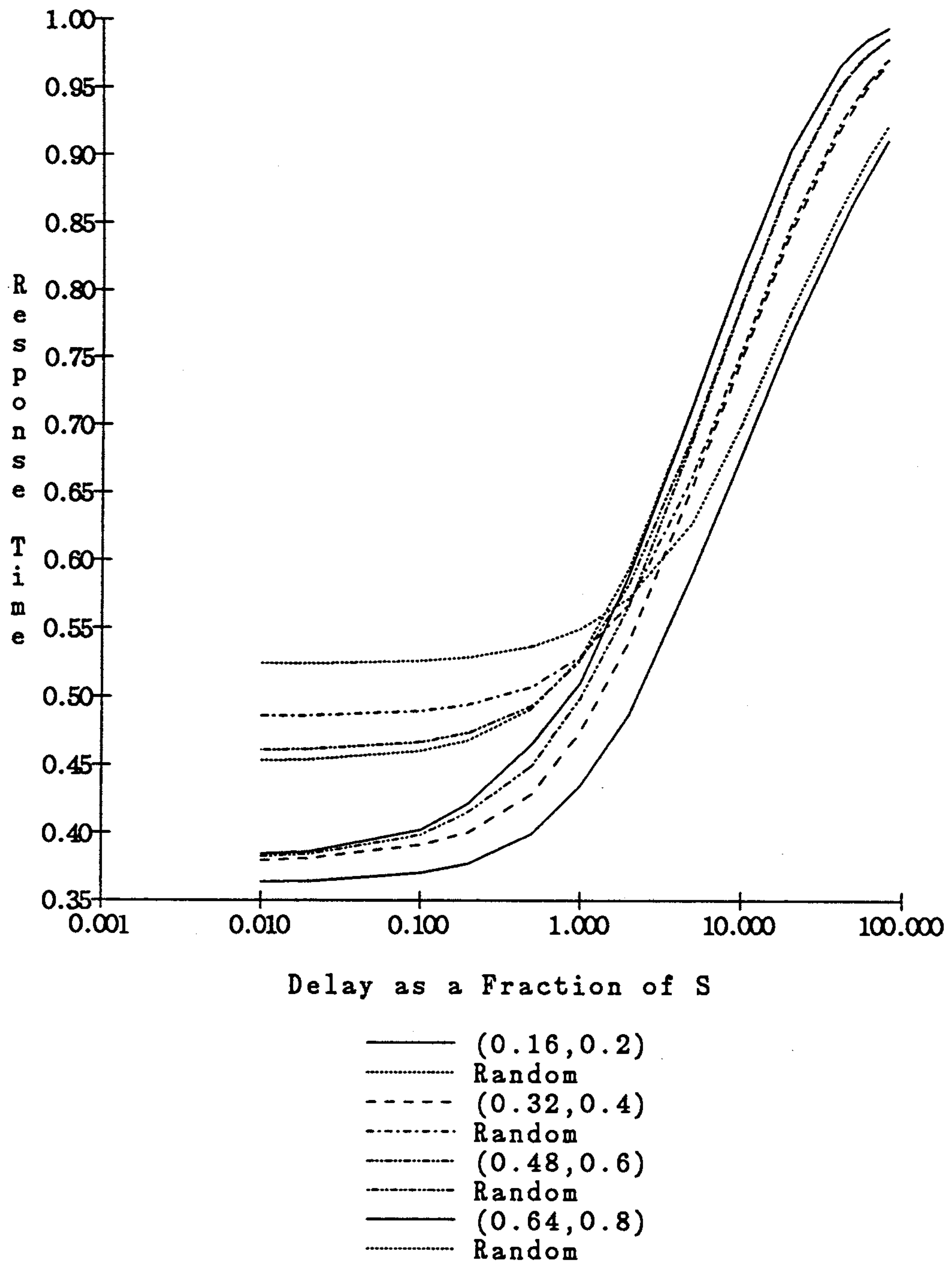

Figure 5.27: Optimal Normalized R.T. (RP)

of finite sized systems and determined that the results were very accurate, with the variation between the analytical and simulation results being about 2-4%. This was true for **type-1** as well as **type-2** systems, with the simulations consisting of 15 nodes per class.

- For **type-1** systems, the sensitivity to probe limit was observed to depend upon the degree of heterogeneity. In the case of Forward probing, the sensitivity was smaller with greater imbalance and the opposite was observed to be true for Reverse probing. If the number of distinct classes is increased beyond two, the effect on probe limit will in all likelihood depend upon whether the new system has become less or more heterogeneous. For instance, if (ρ_1, ρ_2) = (0.9,0.1) and $\rho_3 = 0.5$, one can say that the system with the addition of class-3 is more balanced than it was with only two classes of nodes. Then, one may safely conjecture that under equiprobable selection of classes, Forward probing will become more sensitive to probe limit in this case (recall that when the system was more balanced, it became harder to find a placement for a spare job). On the other hand, under similar conditions of loads, Reverse probing will show decreased sensitivity to probe limit. Of course, in order to determine the exact response times, it will be necessary for the system designer to run the analytical models with the new parameters.

- In **type-1** systems, the effect of varying (T_1, T_2) showed that when the imbalance in systems was large, the performance was relatively insensitive to changes in T_2. However, choosing the appropriate value for T_1 was quite critical. The above observations were also hold for Forward as well as Reverse probing. Furthermore, as the imbalance between the classes decreased, the effect of changes in T_1 and T_2 became more alike.

- We determined the optimal performance of the two algorithms operating in **type-1** systems and compared these results against the optimal performance of the Random assignment, for a large range of delays. It was seen that when the systems were highly heterogenous, Random was much better than

Reverse. However, Forward always performed better than Random, although the advantage of probing was more pronounced when the systems were more balanced. Further, the advantages of probing seemed to disappear when the delays were $\geq 2S - 3S$. If the system was enhanced to include class-3 nodes as well, one might be able to postulate certain behavioral trends, depending upon the values of the loads at the three classes. For instance, if $(\rho_1, \rho_2) =$ (0.9,0.1) and $\rho_3 = 0.5$, quite clearly the system will become less heterogeneous with the inclusion of class-3 nodes and the behavior of the new system is likely to be closer to a two class system with $(\rho_1, \rho_2) =$ (0.9,0.3) than to (0.9,0.1).

- In the case of **type-2** systems it was seen that when T_2 is increased beyond a certain small number and the nodes are executing Forward probing, there is the potential for class-2 nodes to get saturated. The effect of varying T_1 is much less dramatic, however. The above effects are much more pronounced when class-2 nodes are much slower than class-1 nodes. Thus, selection of appropriate thresholds is much more critical for **type-2** heterogeneous systems than it is for homogeneous as well as **type-1** heterogeneous systems.

- In the case of **type-2** systems, we determined the optimal performance for the two algorithms over a large range of delays and compared these results against the Random assignment algorithm. It was seen from these tests that probing does significantly better than Random for **type-2** systems where very few jobs should be executed on class-2 nodes (depending upon how slow they are in comparison to class-1 nodes and some other factors). In **type-2** systems, the advantages of probing are seen to exist for greater values of delays ($\leq 4S - 5S$) than in **type-1** systems. Again, as in previous hypotheses regarding extensions to systems with more than two classes, it will be relevant to ask if the extended system becomes more or less heterogeneous as compared to the one with two classes. Of course, exact values of response times can only be determined by running the models.

- In the case of **type-2** systems, the job transfer delays are compared to the processing time of the fastest class. This decision was purely arbitrary. The comparison could very easily have been made to the processing times of the slowest class or even an average of the two classes (which, in hindsight may be more reasonable than any other alternative). However, it should be quite clear that this will not make any qualitative difference to the results we have presented. Of course, the exact response times generated will be different.

Chapter 6

ENTROPY MINIMAX ESTIMATION FOR LOAD SHARING

6.1 Introduction

In the previous chapters, we have been concerned with analytical studies of the Markov models resulting from simple load sharing algorithms. These models provide significant insight into the relation between the performance metrics and the various important parameters of load sharing algorithms. One of these parameters, i.e., the threshold, may in general, need to be determined on line, particularly in cases where the load at a node is not explicitly made available to the job scheduling algorithm. In this chapter, we study a method to estimate the threshold as a function of average job transfer delay (known explicitly beforehand) and load (which is unknown). The main contributions of this chapter include the application of Entropy Minimax, an information theoretic estimation technique that reduces the uncertainty of state information arising due to the delays. This is accomplished by classifying the states into High or Low, based upon the load and delays (a node that is classified as High at any instant may be a potential source of jobs or be neutral, while a node that is classified as Low is a potential sink for remote jobs).

The remainder of this chapter is organized as follows: In Section 6.2, we discuss the motivation for this research. The relation between delays and state classification is also addressed in Section 6.2. In Section 6.3, we address the question of load sharing in systems where the algorithms utilize thresholds that are generated by Entropy Minimax. It is observed that these thresholds are able to provide optimal response times for most of the cases tested. In Section 6.3, we study load sharing under exponential as well as non-exponential service time distributions. We also determine the effect of probe and job transfer overhead costs on the performance of the load sharing algorithms. Finally, in Section 6.4, we summarize the main contributions of this chapter.

6.2 Delays and Load Sharing

6.2.1 Motivation for Entropy Minimax

In the model that we consider, the minimization of the number of empty (or zero) states while some other nodes have more than one job, is the main objective of load sharing algorithms. This objective follows from the discussion by Livny and Melman [LIVN82] where it was shown that significant performance degradation occurs when idle nodes coexist with nodes which possess waiting jobs.

The optimal strategy under the negligible delay assumption is to exactly emulate the M/M/n behavior. However, this may not be the right strategy to utilize in the face of delays. For example, suppose a node becomes idle. It tries to acquire a remote job by probing the other nodes in the system. Because of the job transfer delays, the node remains idle until the remote job arrives (unless it receives a local job). Obviously, performance can suffer for this period. If the delays are non-trivial (for example, on the order of one service time or more), the degradation in performance can be severe.

It may then become necessary to try and acquire a remote job even before a node becomes idle. The question then is: When should the process of searching for a remote job begin? Is it possible to determine the correct subset of states which corresponds to the idle state in the negligible delay assumption? An estimation technique that appears to possess the suitable characteristics in this regard is called Entropy Minimax [CHRI85], which has its roots in Shannon's theory of communication.

One of the main problems associated with distributed algorithms which actively acquire remote state information is related to the uncertainty in this remote information. Entropy Minimax has the intuitive appeal of extracting *all but no more* than the truly available information from the remote and delayed observations [CHRI85][1]. Furthermore, this method has the advantage of being non-parametric. This is particularly relevant in our context, because the very act of load sharing transforms an initial known arrival distribution into some unknown distribution, thus limiting the applicability of parametric estimation methods.

Broadly speaking, Entropy Minimax utilizes two meta-level states, as follows: One is called Low, which indicates that a remote job is needed in order to prevent the node from becoming idle, and the other is called High, which indicates that a job is not needed. The intuition behind using this method is to find a threshold which partitions the state description (in this case the queue length) at each node into these two meta-level states, High and Low, as a function of delay and load. This partition has the property that when a job is transferred from a High node to a Low node, the probability that it arrives when the destination node just becomes idle, is maximized.

6.2.2 Entropy Minimax

Conceptually, Information

$$I(x) = \log \frac{1}{p(x)}$$

is a measure of the uncertainty associated with the outcome of a random variable $X = x$. If the outcome is certain i.e., $p(x) = 1$, then we have no information gain from observing the outcome and $I(x) = 0$. On the other hand, if the outcome is very uncertain, i.e., $p(x) << 1$, then the information gain from observing the outcome is large. Entropy $H(X)$ is the average information associated with a random variable X, i.e.,

[1]For a detailed comparison between this and other well known estimation techniques like maximum likelihood estimation, least squares etc, see [CHRI85]

$$H(X) = \sum_{i=1}^{n} x_i \log \frac{1}{p(x_i)}$$

The concepts of conditional information, $I(X|Y)$ and conditional entropy $H(X|Y)$ follow in a similar manner. If there are no errors associated with the observation of the events [GALL68], the outcome of the random variable Y perfectly determines the outcome of the random variable X, then $P(X = x|Y = y) = 1$ and we have $H(X|Y) = 0$. Since $H(X|Y) = 0$ is in general unattainable, Entropy Minimax attempts to minimize its value.

Let us now try to see how this is related to the problem of estimation in the context of load sharing. If τ is the delay in acquiring a job, can we determine a threshold T such that the following criteria hold?

If $N_i(t) \leq T$, then $N_i(t + \tau) = 0$, where $N_i(t)$ is the number of jobs at node i at time t, and,

If $N_i(t) > T$, then $N_i(t + \tau) > 0$.

The above requirements are for an idealized threshold such that the node is able to perfectly predict its future state, in case no action is taken by it to try to try and acquire a remote job. The threshold serves the purpose of classifying the states into two meta-level states, one corresponding to Low (which includes all states to the left of T as well as T, i.e., $0 \ldots T$) and the other corresponding to High (which includes all states to the right of T). While it would be ideal to possess such a threshold that acts as a perfect predictor for future states, it may not be achievable in practice, because most systems are stochastic in nature. Thus, the question is whether this idealized threshold can be approximated to any degree such that in a majority of the cases, the prediction is correct. In other words, on the average, a requested remote job arrives when the requesting node becomes idle.

We now describe the Entropy Minimax algorithm that is used to compute the threshold. Let the nodes have local arrivals which are serviced in a FCFS manner.

There is no load-sharing being performed at this stage. Every τ units of time (average job transfer delay), the number of jobs in the node is recorded. Associated with each state l (queue length), are two counters, a_l and b_l. If the node is idle at time t, the b_l counter corresponding to the number of jobs l in the node at time $t-\tau$ is incremented by one. In any case, the a_l counter corresponding to the number in the node at time $t-\tau$ is incremented by one. Basically, what this process does is provide a set of conditional probabilities as follows. We define

$$p_i = P(N_{t+\tau} = 0 | N_t = i),\ 0 \leq i \leq T_m$$

as the probability that the node becomes idle at time $t+\tau$, given that its queue length at time t is equal to i and T_m is the the maximum number of states needed. The conditional probabilities are computed as follows:

$$p_i = b_i / a_i,\ 0 \leq i \leq T_m$$

where i is a particular value of the queue length. These conditional probabilities are collected for the states (queue lengths) that a node enters. Let the instantaneous threshold (during the Entropy Minimax computation stage) be $l = [0, T_m]$. In practice, it is observed that no more than 8-10 states are really needed, for the loads and delays that we have studied. Thus, only $T_m + 1$ partitions of the queue lengths are examined by the Entropy Minimax algorithm [CHRI85].

For a threshold l, the following sums are defined:

$$B_l(L) = \sum_{i=0}^{l} b_i$$

$$A_l(L) = \sum_{i=0}^{l} a_i$$

$$B_l(R) = \sum_{i=l+1}^{T_m} b_i$$

$$A_l(R) = \sum_{i=l+1}^{T_m} a_i$$

where $0 \leq l \leq T_m$ and L and R denote the left and right partitions respectively, generated by the threshold l. The conditional probabilities associated with the partitions are:

$$P_l(L) = B_l(L)/A_l(L)$$

and

$$P_l(R) = B_l(R)/A_l(R).$$

The logarithm of each of these conditional probabilities is the conditional information received from the observation. The expected value of the information is the conditional Entropy of observation, $H(X|Y = l)$, when the instantaneous threshold $= l$.

$$\begin{aligned} S_l(L) &= -P_l(L)\ln P_l(L) - (1 - P_l(L))\ln(1 - P_l(L)) \\ S_l(R) &= -P_l(R)\ln P_l(R) - (1 - P_l(R))\ln(1 - P_l(R)) \\ H(X|Y = l) &= P(L)S_l(L) + (1 - P(L))S_l(R) \end{aligned}$$

where $P(L) = \frac{A(L)}{A(L)+A(R)}$. The threshold is computed using the following optimization procedure:

$$\text{minimize} \quad H(X|Y = l) \tag{1}$$

subject to simple probabilistic constraints as described in [CHRI85].

The entropy minimax algorithm steps through the queue lengths, starting at zero. The conditional entropy is computed at each step and the queue length corresponding to the minimum entropy is selected as the threshold. What this threshold implies is that if a node reaches this threshold, it is very likely to become idle at time $t + \tau$ unless a remote job is made available to it. Further, if the queue length is greater than the threshold, the likelihood is that the node will *not* become idle in the given interval of time. In order to determine the relation between delays, load and state classification, we have conducted several experiments. The results and their implications are described next.

6.2.3 Results from Threshold Experiments

In this section, we present the results from simulation studies which specifically address the issue of the threshold and the two factors that affect it greatly, namely delays and the load. The algorithm that computes the threshold samples the queue length of the node at intervals of time $= \tau$ (the average job transfer delay). The conditional probabilities are computed from the above observations, as described in the previous subsection. The threshold is computed after a large enough number of samples have been gathered. This can be verified by noting if the threshold has stabilized. It was seen that if S was the expected service time for jobs, then a window of about $400S$ to compute the threshold appeared to be adequate. Table 6.1 depicts the variation of the threshold over a range of delays and loads. Three different service time distributions (with identical means) were studied. They were: exponential (Exp), 2 stage Erlang(Erl) and hyperexponential(Hyp) with $C_v^2 = 2.0$. The utilizations tested were 0.4, 0.6 and 0.8 and the average job transfer delays were $0.5S$, S, $1.5S$ and $2S$, for each of the loads tested.

From Table 6.1, it can be seen that:

- The threshold is a non-decreasing function of delays. This is because greater delays increase the uncertainty in the information, leading to a greater lumping of low states, represented by the higher thresholds.
- As the delays tend to zero, the threshold approaches zero for all loads. This is intuitively satisfying because at insignificant delays, it is appropriate to

Table 6.1: Results from Threshold Experiments

Load	0.0	0.5	1.0	1.5	2.0	
	0	0	0	1	1	Exp
0.4	0	0	1	1	1	Erl
	0	0	0	1	1	Hyp
	0	0	1	1	1	Exp
0.6	0	0	1	1	1	Erl
	0	0	1	1	2	Hyp
	0	1	1	2	2	Exp
0.8	0	1	1	1	2	Erl
	0	1	1	2	2	Hyp

emulate the optimal M/M/n strategy. A threshold of zero makes this possible.

- For a given delay, higher loads tend to increase the threshold, because the holding time of a state decreases with increase in load, resulting in a greater lumping of states.

- In some cases, the thresholds appear to be dependent upon the service time distributions, as may be seen in Table 6.1 (e.g., $\rho = 0.6$, delay $= 2S$). However, the exact relation between distributions and thresholds is not altogether clear at this point.

In this chapter, we restrict our study to the situation under which delays are significant but remote state information is still useful in the sense that a stable Entropy Minimax threshold can be found. In extremely high delays, we have seen that the threshold is undefined[2]. Furthermore, at such high delays, it may be the case that load sharing will actually make the performance worse because the transferred jobs will be in neither the sender's queue nor the receiver's queue for the duration of the delay time.

[2]In these cases, the conditional entropy associated with different threshold values are approximately equal to each other.

6.2.4 Description of Algorithms

The algorithms studied in this chapter are the same as those in Chapter 3 but their description is repeated here for convenience. Each node is provided with a threshold T and probe limit L_p.

Symmetric: As soon as a node's queue length goes below $T + 1$ on the completion of a job and the node is not already waiting for a remote job, it probes L_p nodes in the system, until a node can provide it with a job or all the nodes have been exhausted. If more than one node can transfer a job, one of these nodes is selected at random. A remote node will only transfer a job if it possesses at least $T + 2$ jobs. Also, as soon as a local arrival occurs at a node and it has at least $T + 2$ jobs (including the new arrival), it probes L_p nodes in the system, until it finds a node which has $\leq T$ jobs and is not already waiting for another remote job. If all the probed nodes have at least T jobs, no transfer will take place. If more than one node can accept a spare job, only one of these will be selected at random for transfer.

Forward: If a local arrival occurs and the node has at least $T + 2$ jobs (including the newly arrived one) it probes L_p nodes to determine if any one is $\leq T$ and is not already waiting for a remote job. If so, it transfers this job there, else, it keeps this job. If more than one node is able to accept a spare job, one of these nodes is selected at random for transfer.

Reverse: As soon as a node goes below $T + 1$ on the completion of a job and it is not already waiting for a remote job to arrive, it probes L_p nodes to determine if any node has a spare job (at least $T + 1$), the remote node transfers a job to this node. If more than one node responds positively, one of these nodes is selected at random.

Random: In this algorithm, the nodes do not perform any probing. As soon as a node receives a local job, it checks if it has at least $T + 1$ other jobs. If so, it transfers this new job to one of the other nodes, selected at random. There is no state update overhead generated by this algorithm and it serves to provide a reasonable bound for comparison against probing algorithms.

6.2.5 Description of the Experiments

The simulation system consisted of 10 identical nodes, connected in a network. The inter-arrival and job transfer times were exponentially distributed. However, the service times were selected from three different distributions, depending upon the test being conducted. These were, exponential, 2-stage Erlangian and hyperexponential distributions. Further, the arrival and service rates were identical at the nodes. The entropy minimax thresholds for the various loads and delays were computed off-line for convenience, but they could as well have been computed on-line. The delays were varied from 0.5*S to 2*S, where S the expected service time of jobs, was 1.0 units. The load was varied from 0.4 to 0.8, which encompassed a large range for load sharing.

Every time a job was transferred, it immediately disappeared from the sender's queue and appeared at the receiver's queue after the mean time equal to the transfer delay. Thus, a transferred job was not available for execution at either node during this interval. While jobs encountered delays, probes were assumed to take zero time. Although it was possible to exactly determine the state of remote node at the time of decisions, the delay in actual job transfer caused the uncertainty in the state information, because the load sharing decisions were based upon states that may have changed by the time a transferred job arrived at the remote node.

The maximum number of probes that a node was allowed to make was tuned initially as a parameter. It was seen that 2 was a good number in most instances, in a 10 node system, particularly if the probe overhead would be accounted for in some way. The incremental gain in performance by allowing $L_p = 3$ over 2 was marginal in all cases tested. Experiments conducted with complete probing ($L_p = 9$) showed very little improvement over $L_p = 2$ or 3. Thus, the simulation runs in the following section were made with 2 probes. Also, the nodes to be probed were selected at random.

6.2.6 Experimental Results

In this section, we present the results that we have obtained from simulations conducted on a network of 10 nodes. The main metric of interest is the average response time generated by the algorithms. As mentioned in Chapter 3, the overhead generated by load sharing will be assumed to be entirely transferred to the BIU's, except when we specifically address the issue of overhead costs. The results include the effects of thresholds on performance, the performance of the various algorithms under different loads and delays, the effects of various overhead costs for probes and job transfers and the study of load sharing with non-exponential service times.

All the results indicated are averages of at least three independent simulation runs with different random number seeds (i.e., the method of independent replications). It was seen that in typical cases, the sample standard deviation was less than 0.2%, and the 95% confidence interval lay between ±0.3% of the sample mean. These confidence intervals were computed using the Student-t distribution. Unless otherwise stated, the service times are assumed to be exponentially distributed.

Choosing an Algorithm

Figures 6.1, 6.2, and 6.3 depict the performance of the four algorithms under delays for the loads of 0.4, 0.6 and 0.8 respectively. In general, one can see that Forward with 2 probes performs well over the range of parameters tested. For the case of low loads, Random is very effective and generates only slightly worse response times than Forward. However, there is point of discontinuity at $\rho = 0.8$ and delay $= 0.0$. It is seen that the performance of Random improves when the delay in increased to 0.5*S. This counter-intuitive behavior has to do with the fact that in general, Random does not fulfill the requirement imposed by the Entropy Minimax state

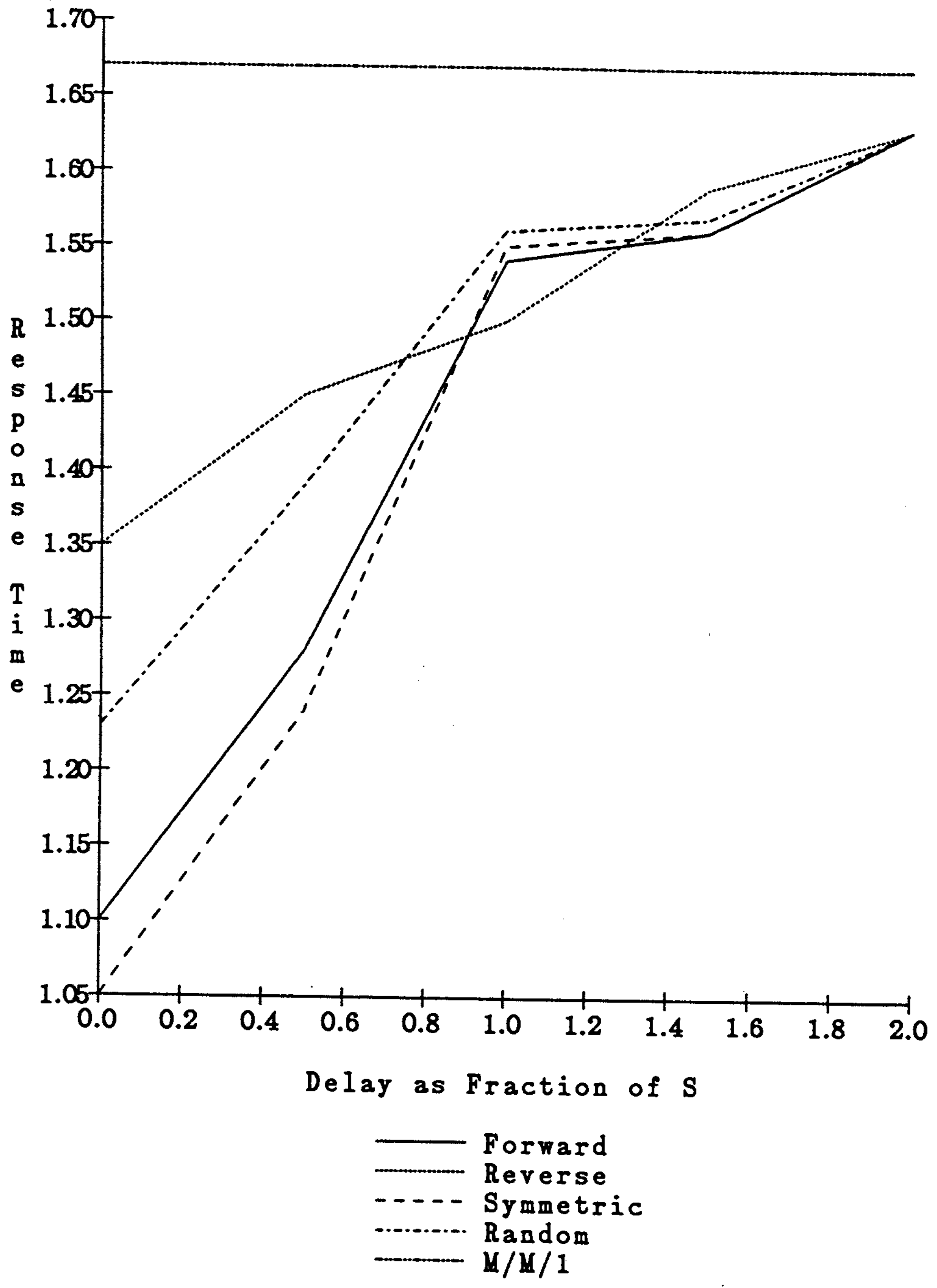

Figure 6.1: Effects of Delays ($\rho = 0.4$)

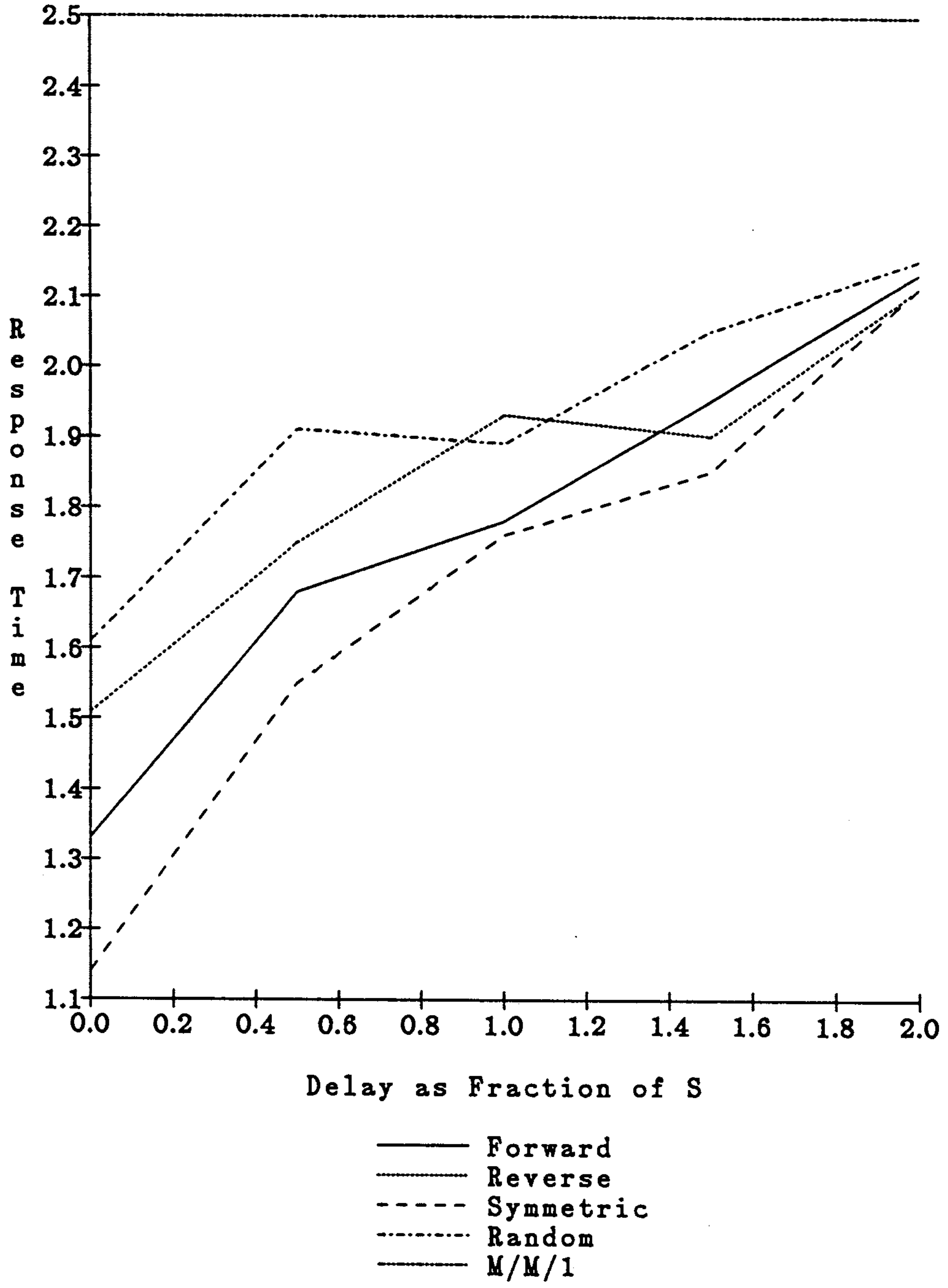

Figure 6.2: Effects of Delays ($\rho = 0.6$)

representation that a transfer should take place from a High node to a Low node. This is particularly true at $\rho = 0.8$ and zero delay where a transferred job is likely to arrive at a High node quite often. However, at higher delays, the threshold increases and so does the probability that a transferred job will arrive at a Low node. In other words, the threshold that might be optimal for probing algorithms may not necessarily be optimal in the case of Random assignment, particularly at low to moderate delays.

Reverse does not perform as well as Forward, particularly when the load is $\leq$ 0.6. This is because more nodes are likely to be in the Low state, making it harder for Reverse to find a spare job. In zero delay experiments with extreme ($\rho \geq 0.9$) loads it performed better than Forward (observed in Chapter 3).

It can be seen from Figures 6.1, 6.2 and 6.3 that as the job transfer delays increase, the following observations may be made. Firstly, the benefits of load sharing become less significant and the performance under load sharing approaches that of the $M/M/1$ system. In fact, the response time may actually get worse than $M/M/1$ at even higher transfer delays and we have seen this occur in other experiments (results not presented here). This is particularly evident at $\rho = 0.4$ (Figure 6.1) where at delay $= 2S$, the performance under load sharing is only about 5% better than the $M/M/1$ response time at that load. On the other hand, at $\rho = 0.8$, the benefits of load sharing are more substantial even at the maximum delay tested (Figure 6.3). It is seen that for delay $= 2S$, the performance under load sharing is about 45% better than the corresponding $M/M/1$. Thus, it is possible to hypothesize that at even higher loads greater delays may be tolerated. At low loads, it might make sense to turn off load sharing when the job transfer delays increase beyond $2S$. Furthermore, the performance of the probing algorithms become almost identical and at low loads, Random performs as well as any probing algorithm tested (at $\rho = 0.8$, there is about 5% improvement on account of probing).

Effect of Thresholds on Three Service Distributions

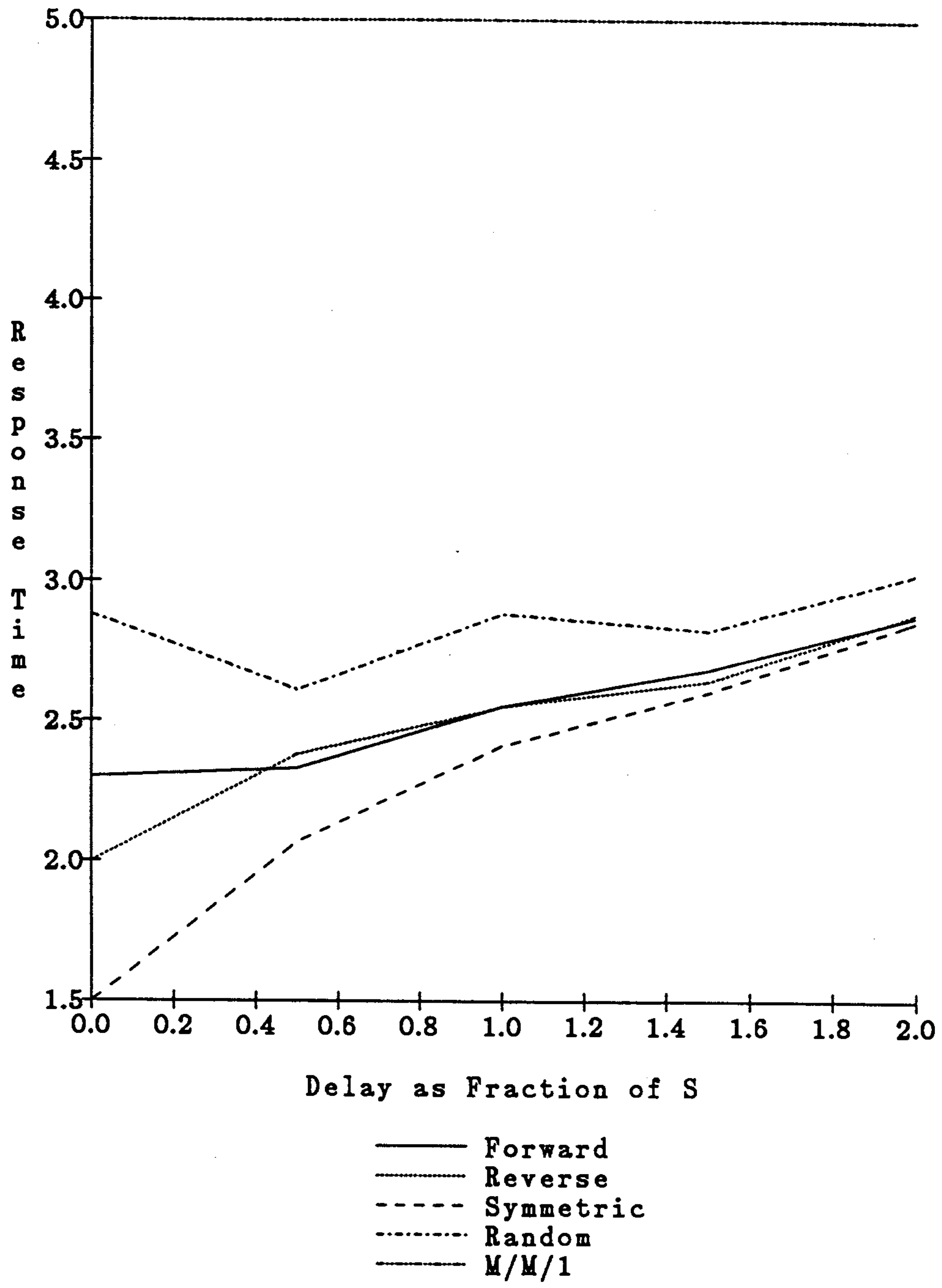

Figure 6.3: Effects of Delays ($\rho = 0.8$)

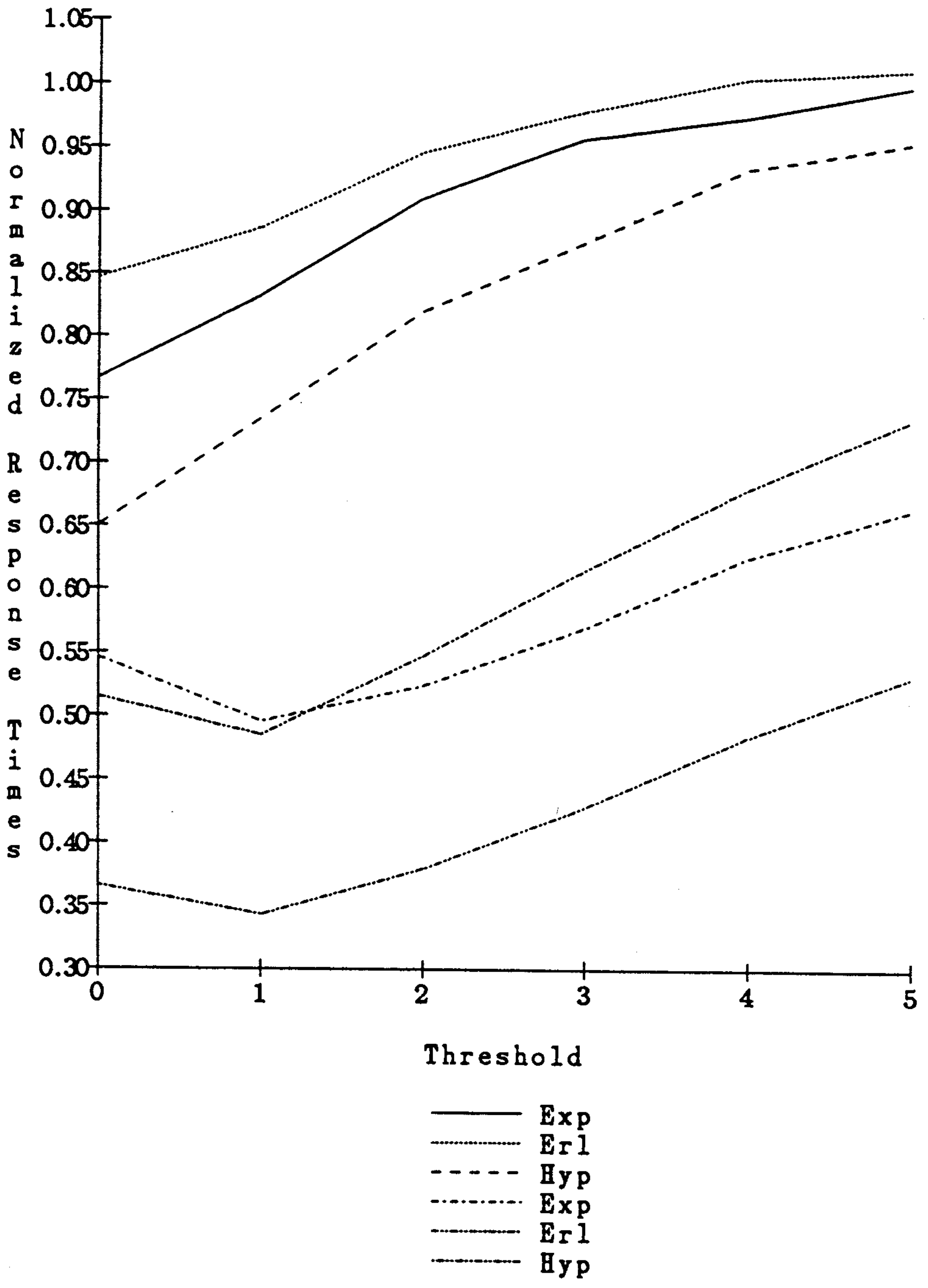

Figure 6.4: Effects of Thresholds (Delay= 0.5S)

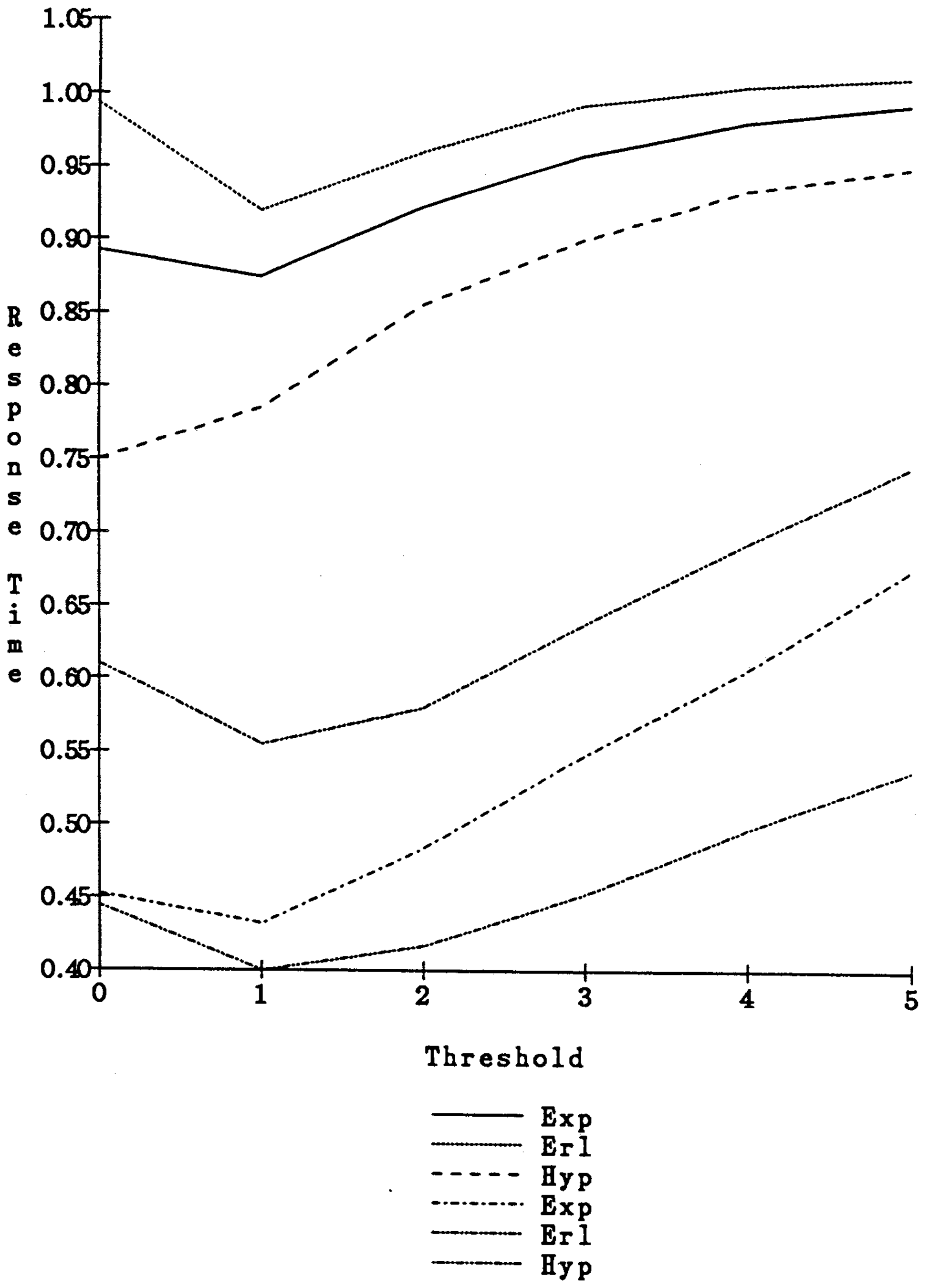

Figure 6.5: Effects of Thresholds (Delay= S)

Until this point in this chapter and dissertation, we have concentrated on the performance of the load sharing algorithms when service times are exponentially distributed (except for the case of obtaining thresholds for Erlangian and hyperexponential servers as depicted in Table 6.1). In general, the exponential assumption may be considered restrictive and we would like to see how appropriate are the Entropy Minimax thresholds when the service times are not exponentially distributed. In addition, we are interested in comparing the performance of load sharing for the three service distributions, i.e., exponential, 2 stage Erlangian and hyperexponential with $C_v^2 = 2.0$, all having the same mean, 1.0 units.

Figures 6.4, 6.5, 6.6 and 6.7 depict the behavior of the Symmetric probing algorithm for the three service time distributions discussed above. The arrival rates were 0.4 and 0.8 jobs/unit and the job transfer delays were $0.5S$, S, $1.5S$ and $2.0S$ in Figures 6.4, 6.5, 6.6 and 6.7, respectively. The curves in the graphs represent not the actual response times obtained but the response times normalized by the corresponding no load balancing response times generated by the $M/M/1$, $M/H_2/1$ and $M/E_2/1$ values for the appropriate service time distributions. Thus, the results of load sharing with exponential service time are normalized by the $M/M/1$ values, the Erlangian by $M/E_2/1$ and hyperexponential by $M/H_2/1$. Further, the thresholds were varied between 0 and 5, and the response times under these conditions were recorded.

From the above set of figures, we are able to make the following interesting observations:

- The improvement by load sharing over the corresponding no load sharing is greater as the load increases. The curves for $\rho = 0.4$ are located higher than those for $\rho = 0.8$ ($\rho = 0.6$ curves are located in between, although we have not depicted these in the graphs). For example, from Figure 6.5, delay $= S$, the best improvement for Hyp is 60% over no load balancing when $\rho = 0.8$ but only 25% when $\rho = 0.4$. Similar numbers may be determined from the other three figures in this set. This occurs because of the higher inherent waiting

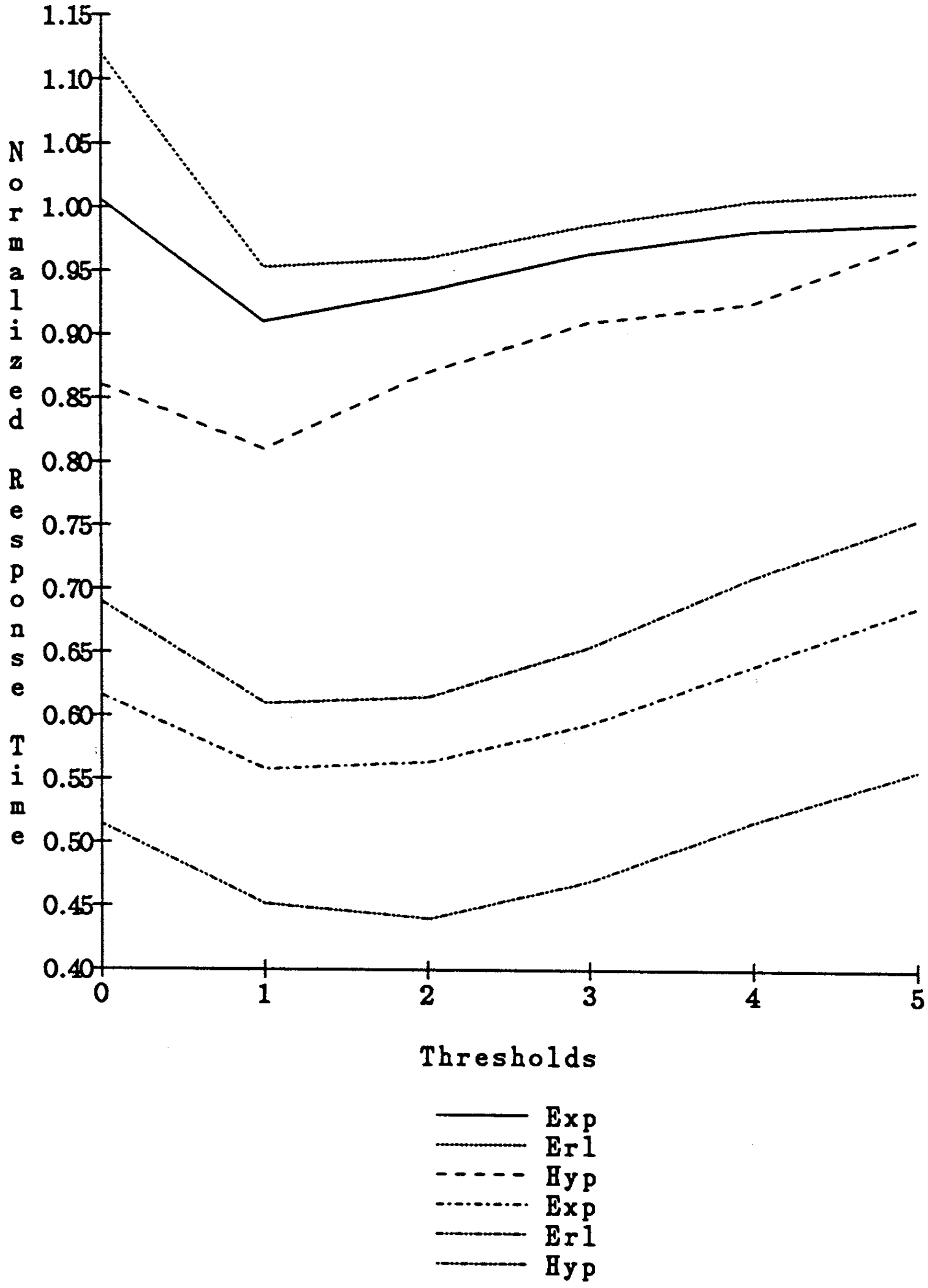

Figure 6.6: Effects of Thresholds (Delay=1.5S)

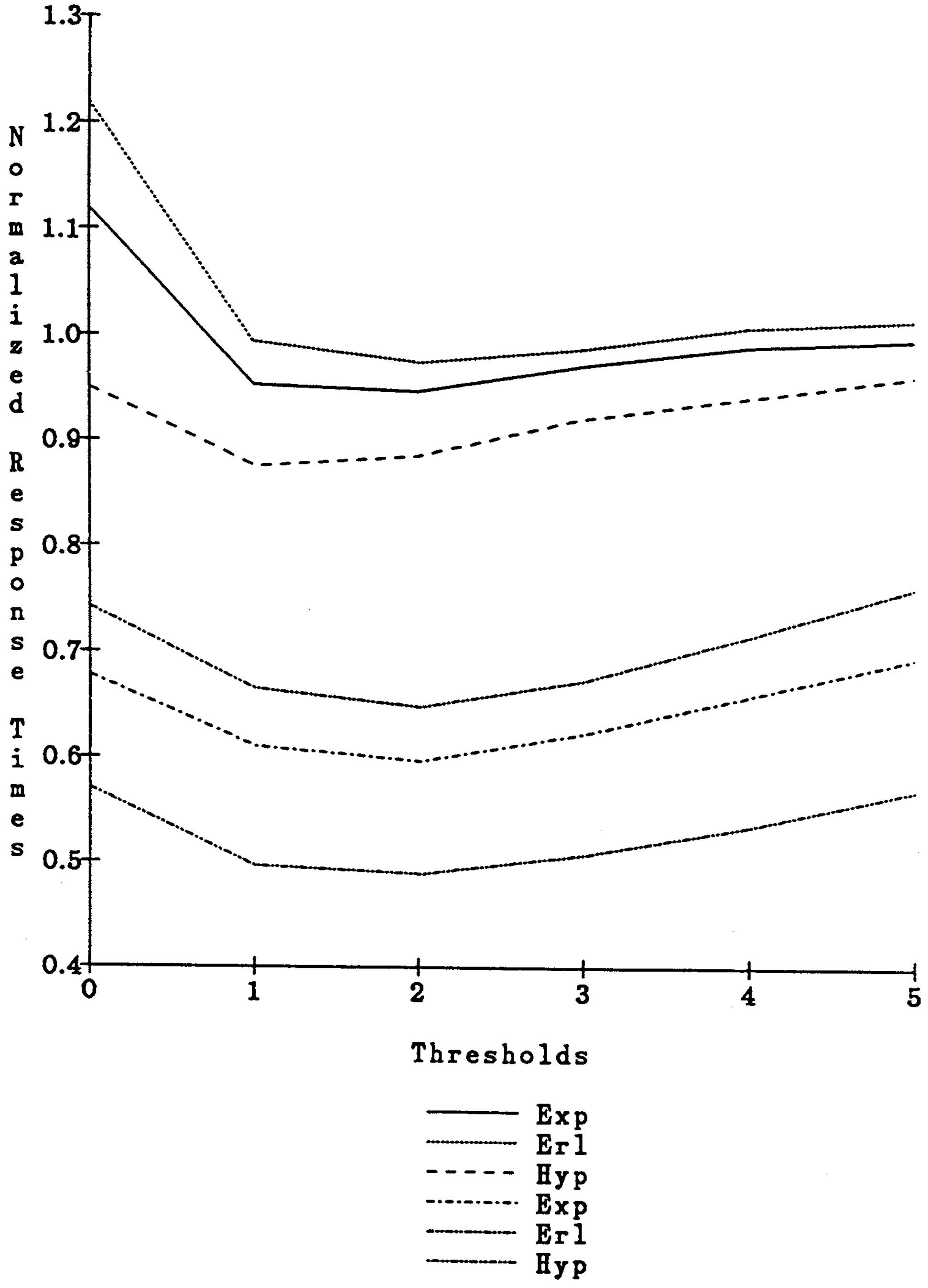

Figure 6.7: Effects of Thresholds (Delay=$2S$)

times at high loads and the load sharing is consequently most advantageous.

- For most of the range of thresholds and delays tested, it seems that the improvements are greater when the variability in service rates is higher. Thus, in general, the curves for hyperexponential are located lowest, Erlangian are highest with exponential in the middle of these two (an exception is seen in Figure 6.4, $\rho = 0.8$ when the Exp curve crosses the Erl). For instance, in Figure 6.4, $\rho = 0.8$, the gain by Hyp is greater than 65% whereas Erl is better by about 50% over its no load balancing value. Again, this is because higher variability in service times implies longer waiting times. We postulate that similar behavior may be expected when different arrival distributions are examined.

- Varying the threshold over a large range of values helps us determine whether the thresholds predicted by Entropy Minimax are correct (i.e., they provide the optimal response times). In this connection, we refer back to Table 6.1 which encapsulates the Entropy Minimax thresholds for the loads 0.4, 0.6 and 0.8 for the 4 values of job transfer delays. From Table 6.1 and the graphs under consideration, it is seen that for most of the parameters tested, the thresholds generated by Entropy Minimax are optimal. In fact, out of the 36 different threshold tests conducted (results of which are depicted in Table 6.1), the optimal threshold as obtained by simulations of the Symmetric algorithm was different in only 3 instances. For example, for Erlang-2 service times with $\rho = 0.4$ and delay $= 2S$, the optimal threshold was 2 but Entropy Minimax predicted 1. However, the differential in response times using one or the other threshold is less than 2% (see Figure 6.7, topmost curve).

Effects of Probe and Job Transfer Overheads

Thus far in this chapter, the response times generated by the algorithms did not include the effect of probe and job overhead. The reasons for doing this were as follows: Firstly, we had assumed that the overhead of processing jobs and probes for load sharing is completely transferred to the network controllers (BIU). In a perfect world, the CPU at a node would not be slowed down owing to interference from the network controller. Secondly, it was our conviction that it is very hard to estimate reasonable costs for potential interference since these costs are highly dependent upon the underlying system architecture and protocols and at this stage we have made only very general assumptions regarding the node architecture.

The response to the above arguments may be the following: In reality, network controllers will tend slow down the CPU to some extent because of the common resources they may need to access, e.g., shared memory, system bus and so on. System designers will consequently be interested in determining the effects that the interference may have on the system performance as a whole. In order to provide a feel for how a range of overhead costs might affect the net response time of jobs, we have conducted several experiments where the effect of probe and job transfer overhead is modelled as an interference to the jobs executing on the CPU. We have chosen to use a simple interference model because it is our belief that these simplifications will affect all the algorithms equally and that the relative comparisons will consequently be unaffected.

Probe Overheads

To study the effect of probe cost, each node in the simulation system accumulates the number of probes it sends out and receives. As soon as the number increases above a prescribed value, the next job to be executed at this node executes at a slower rate. The normal mean service time of jobs was 1.0 units, as in all the earlier experiments. However, the interference is assumed to increase the mean service time of the slower job to 1.25 units, an increase of 25%. The amount of increase is arbitrary and is used solely for the purpose of illustration. The counters which hold the probe counts are then reset and the monitoring process continues. When the effect of job transfer overhead is to be studied, the nodes accumulate the number of jobs transferred. When this count increases above a prescribed value, the next job executes at the slower rate and the job transferred count is reset to zero and the monitoring process continues.

Figure 6.8 depicts the results for the case of probe overheads (job transfers do not cause interference in these experiments). The horizontal axis represents the probe count at which the job execution time is increased. For instance, a count of 10 means that as soon as a node processes 10 probes, it schedules the next job to execute at the slower processing rate. This count is varied between 10 and 100, with 10 generating the highest overheads. Sy, Re, Fo represent the results for Symmetric, Reverse and Forward probing respectively.

From Figure 6.8, we can make the following observations: At $\rho = 0.4$, the effect of probe overhead is most felt by Reverse probing. Most of the reverse probes do not result in job transfers and only contribute to overhead. As the probe costs decline, Forward and Symmetric show identical behavior. However, when probes costs are high, Symmetric performs slightly worse than Forward. Similar behavior is seen at $\rho = 0.6$ where Forward is better than Symmetric at high probe costs, because Symmetric generates a large number of wasted reverse probes. However, at low loads, Reverse is inherently worse than Forward or Symmetric in the first place. At $\rho = 0.8$, Reverse does better than Forward and Symmetric at high probe costs, by about 20% when the probe costs are the highest. This is because nodes are more likely to make forward probes at this load and high costs of probes will adversely affect Forward and Symmetric probing. When probe costs decrease, Symmetric is clearly superior to either Forward or Reverse and even Forward ends up being slightly better (about 2%) than Reverse at the lowest probe costs tested.

Job Overheads

In the next set of tests, each node monitors the number of jobs it sends out and receives. As soon as the number increases above a prescribed value, the next job to be executed at this node executes with the slower rate. The counters which hold the job counts are then reset and the monitoring process continues. In Figure 6.9, we present the results of the experiments wherein probes generated no overhead.

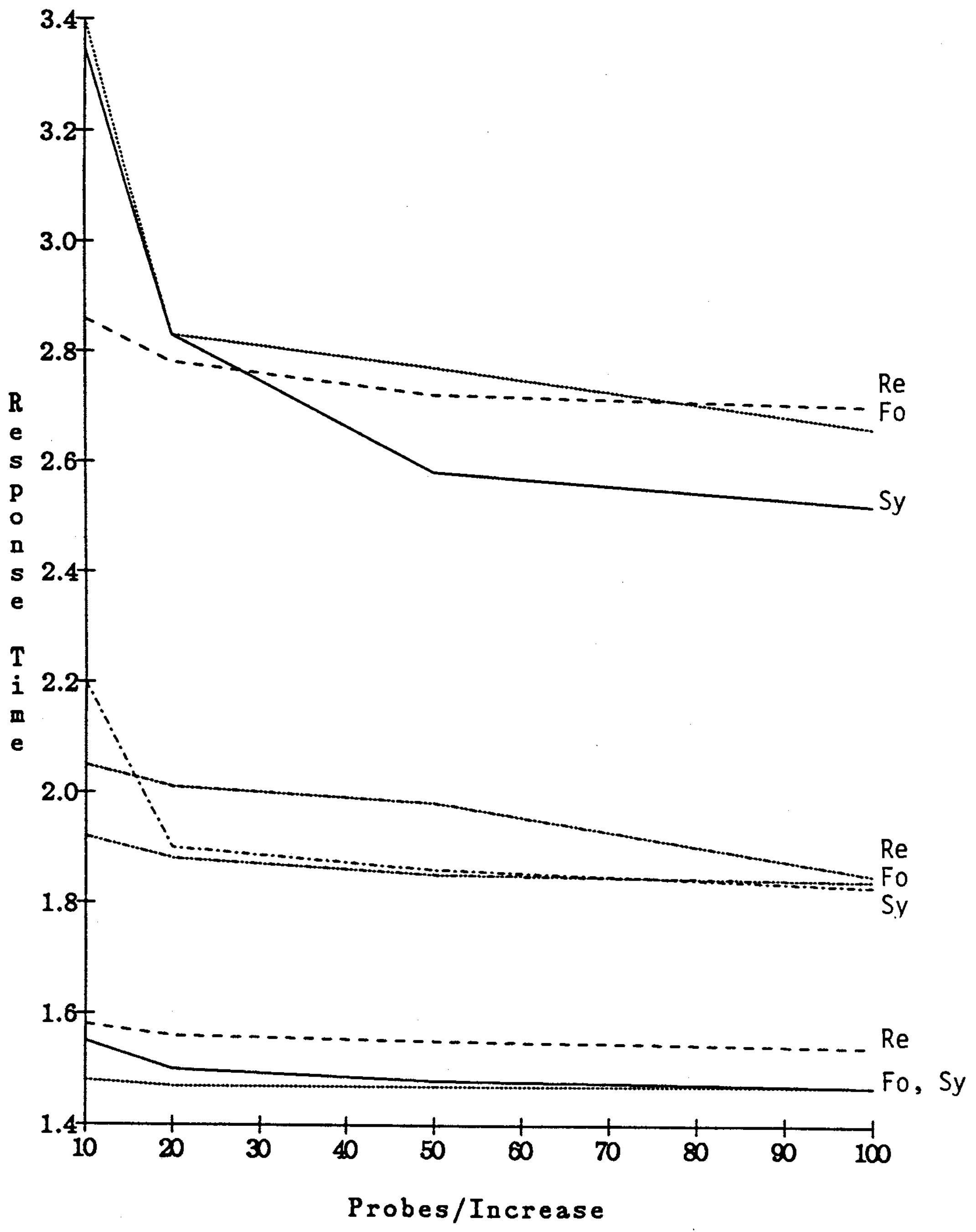

Figure 6.8: Effects of Probe Overhead

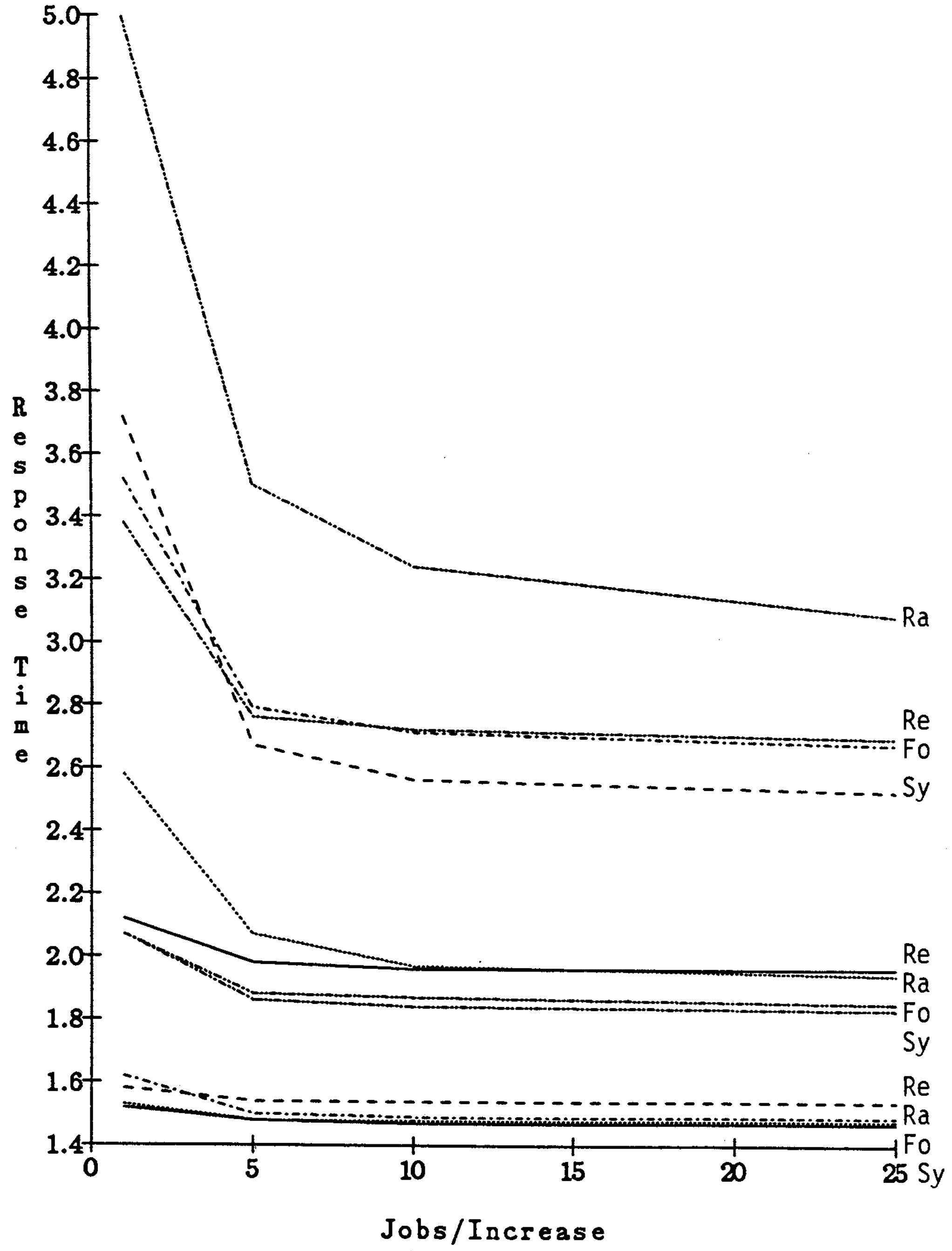

Figure 6.9: Effects of Job Transfer Overhead

Instead, job transfers causes interference to the currently executing CPU job and increases the average service time of the delayed job by 25%, as in the case of the previous experiments on probe overheads. The horizontal axis in Figure 6.9 represents the job count at which the execution time is increased. This count is varied between 1 and 25, with 1 generating the highest overheads. The significance of the job count is similar to the probe count in Figure 6.8. As soon as the job transferred count is reached, the next CPU job is executed at the slower speed. For comparison purposes, we have included the curves for the Random assignment algorithm in the presence of job transfer overheads. These curves are represented by the code *Ra*. The other codes are the same as those for Figure 6.8.

From Figure 6.9, we can make the following observations: The effects of job overheads is most felt by Symmetric and Random. This is because they tend to transfer the most number of jobs, especially as the loads increase. This is shown by the fact that extremely high (with the potential of becoming unstable) response times are generated by Random at $\rho = 0.8$ and high job overheads. While Symmetric is stable, it does perform worse than the other two probing algorithms at $\rho = 0.8$, when the overheads are relatively high (about 5-10% worse). As the overhead costs decrease, the behavior becomes more predictable with Symmetric clearly performing better than all the other algorithms.

To reiterate our earlier reservations about assigning overhead costs, we believe that these costs are very tightly linked to the underlying system architecture. Designers of such systems have to estimate the potential interference that might be caused by probes and job transfers, given the underlying node architecture. If estimates of these quantities are available, then Figures 6.8 and 6.9 can provide some help in selecting the algorithm that might be most appropriate. For instance, if probe costs are very high and the load is ≥ 0.8, it might be appropriate to select Reverse probing. At low loads (≤ 0.7) and high probe costs, Forward appears to be the best bet. If however, the probe costs are low, Symmetric will easily outperform either of the other two probing strategies. For high job costs and high loads (≥ 0.8), Symmetric is definitely worse than either of the probing algorithms. However, when job costs are not very high, it is probably best to go with Symmetric. This fact was also noted in Chapter 3, where Symmetric outperformed Forward and Reverse when overhead costs were zero.

6.3 Summary and Conclusions

This study was primarily concerned with studying the effects of delays in load sharing. We presented an application of Entropy Minimax, an information theoretic estimation technique, to reduce the uncertainty in the delayed state information. It was seen that the performance of the algorithms using the state classification provided by Entropy Minimax was for the most part optimal. In the few instances that the thresholds were not optimal, the response times were off the optimal results by at most a few percent. This fact was verified for a large range of parameter values as well as three different service distributions: exponential, 2 stage Erlang and hyperexponential. However, it was seen that the performance of the algorithms is less sensitive to appropriate selection of operating thresholds than we had originally imagined. If the chosen threshold was off by one from the optimal threshold, the performance could be worse by 5-10%.

As regards the state update protocol and algorithm design, it was seen that Symmetric probing with probe limit of 2 probes performed uniformly well over the range of parameters tested with Forward being next best. For higher values of load (≥ 0.9), Reverse is likely to outperform Forward, as we have seen in the results of Chapter 3. We studied the effects of probe and job transfer overheads on the CPU service rates and concluded that for high overhead costs, Symmetric performs poorly because it has the tendency to generate large numbers of probes and job transfers. On the other hand, with low to moderate overhead costs, none of the other algorithms could match its performance over the range of parameters tested.

Chapter 7

ADAPTIVE LOAD SHARING IN DISTRIBUTED SYSTEMS

7.1 Introduction

Computer systems are normally subject to various changes during their operation. For instance, the system load may change continuously or periodically, nodes may fail and then recover, the processing power at nodes may change (e.g. addition of vector processing unit, fast I/O processors etc.) and so on. It appears that some or all of these changes might warrant a dynamic modification of the control strategy utilized by the scheduling algorithms utilized by these systems.

In this chapter, we study the problem of dynamically adapting the the load sharing strategies and parameters when the system load may change over time. We separate the adaptation problem into two logical parts: The first part is concerned with the estimation of the relevant parameters (e.g., utilization at a node, the arrival rate of jobs, etc.) as they change and the second relates to the adaptation of the control strategy in response to these changes. In Section 7.2, we present the system model and describe the load sharing algorithms under consideration. The estimation problem is examined by three simple methods, as described in Sections 7.3 and 7.4. The efficacy of these methods is compared using the mean percentage error of estimation as the main metric. Also used as means of comparison is the convergence speed of the various techniques in obtaining a good estimate of the load. From the experiments conducted, it is seen that simple techniques are able to estimate the loads fairly quickly. The performance of a simple adaptive control strategy is compared against the optimal control strategy (one that always utilizes the optimal parameters for load sharing) and these results are presented in Section 7.5. From these experiments, it is seen that any one of the three simple estimation techniques used in conjunction with a simple adaptive load sharing algorithm provide very good performance in comparison with the optimal strategy. Finally, we summarize the results of this chapter in Section 7.6.

7.2 System Model and Load Sharing Algorithms

The system model used in this chapter is similar to that assumed in the earlier chapters. However, in brief, the distributed system under consideration consists of N identical nodes, connected in a network. Each node executes a load-sharing algorithm. Further, node-i is provided with a threshold T_i. The algorithm that we use in this study is called Symmetric, described in the following paragraph.

Symmetric: As soon as node i's queue length goes below $T_i + 1$ on the completion of a job and the node is not already waiting for a remote job, it probes L_p nodes in the system, until a node can provide it with a job or all the nodes have been exhausted. If more than one node can transfer a job, one of these nodes is selected at random. A remote node j will only transfer a job if it possesses at least $T_j + 2$ jobs. Also, as soon as a local arrival occurs at node i and it has at least $T_i + 2$ jobs (including the new arrival), it probes L_p nodes in the system, until it finds a node j which has $\leq T_j$ jobs and is not already waiting for another remote job. If all the probed nodes have at least as many jobs as their threshold, no transfer will take place. If more than one node can accept a spare job, only one of these will be selected at random for transfer.

From our results in Chapter 6 of this dissertation, we noticed the need to consider two levels of changes in such systems, relating to load sharing. We call Level-1 changes those that need a radical modification in the control strategy. For instance, Forward probing versus Reverse (algorithms described and analyzed in earlier chapters) if the changes in the load are drastic. Level-2 changes are less dramatic and involve adaptation of the internal parameters of the currently executing control algorithm, as for example, the thresholds. In this dissertation, we only focus on Level-2 changes. This is because in our initial model (described in Chapter 3), all the processing overhead for job transfers and probes is transferred to the DMA processor at the nodes and without the inclusion of these overheads, Symmetric is clearly superior to either of the other two algorithms (results depicted in Chapters 3 and 6). Thus, our only concern will be to adapt the internal parameters of this algorithm. We now describe the general approach that is adopted in the following sections to study the problem of adaptive load sharing.

- Initially, we assume that the underlying system is homogeneous and that this assumption is satisfied even when changes occur in the loads. Under these assumptions, we study two estimation techniques and determine their performance in relation to tracking changes in the arrival rates.
- Next, we relax the homogeneity assumption. In this case, some of the nodes are subject to changes in loads that are quite different from the others. We commence these tests with a homogeneous system where a designated node is subjected to surges of jobs over and above its regular arrivals and determine how quickly and accurately these surges can be monitored. Studies of large heterogeneous systems will not be conducted in this research.

At an abstract level, the adaptation mechanism at a node is represented by the interaction of two main modules. The Est-State module continuously monitors the relevant local parameters (e.g. arrival rate of jobs, utilization etc., depending upon the estimation strategy being used). At periodic intervals, the information regarding the estimated parameter is passed onto the Controller module which may make one of the following decisions based upon the intensity of the changes:

1. Use a different load sharing algorithm at this node (not examined in this dissertation).
2. Modify the parameters of the current load sharing algorithm.

3. Do not make any changes.

We now study the estimation problem as it relates to homogeneous systems and present two methods to estimate the changing loads.

7.3 Homogeneous Systems

In the case of homogeneous systems, a simple approximation that we have studied is the following: Let us assume that all the changes occur at each node at the same time. This implies that even though the nodes are subject to changing loads, the homogeneity assumption is never violated. This may be a reasonable coarse-grained assumption for some types of systems. For example, computing facilities at universities and services used by research labs where systems are quite evenly loaded at any time. From our earlier studies [MIRC87b], we know the answers to the following questions:

1. What is the best algorithm for a given set of system parameters?
2. What should be the values of the internal parameters of the above algorithm?

In the following subsection, we describe two simple procedures that we have utilized in order to estimate the changing loads on the system. Because of the homogeneity assumption and the fact that all the nodes use identical control strategies, the effective load at a node is unaffected by the act of load sharing. This leads us to investigate very simple methods to adaptively determine the load at the nodes. Further, in the simulation model, only Node-1 monitors its load and broadcasts its value at prespecified periods to the rest of the system.

7.3.1 Estimation Procedures

The simulation model for the study of the estimation techniques consisted of a network of 10 nodes executing the Symmetric algorithm. Each node possessed the same operating threshold. The arrival rates at the nodes were changed periodically, always maintaining the homogeneity assumption. The inter-arrival, service and job transfer times were all exponentially distributed random variables. The expected service time of a job at any node was assumed to be $S = 1$ unit.

Method-1

The first method assumes that there exists some mechanism built into the node that is able to keep track of idle CPU cycles during a prescribed window of observation. For instance, the VAX architecture provides hardware support for this purpose [Vax85]. In case there does not exist special purpose hardware, this function can be programmed into the job scheduler, which will need to maintain the data regarding the idle CPU cycles.

Let p_0 be the steady state probability that a node is idle. Hence, the processor utilization is $\rho = 1 - p_0$. The designated node continuously monitors its idle time. During a given window of observed time t_m, the idle time at the node is denoted by t_i, where $t_i \leq t_m$. Thus, $\rho_m = 1.0 - t_i/t_m$ is the fraction of time the node is busy during the time interval t_m. In the limit, as m tends to infinity, ρ_m will approach ρ, the exact load at the node (assuming that the load has not changed over this interval).

A relevant aspect of this study is to determine the minimum t_m needed to generate a reasonably close approximation of the load. The reason we do not insist upon exact values of the load is that although the parameters are affected by the load, small ($\leq 5 - 10\%$, depending upon system load) variations in estimating the load are seen not to affect the performance of the system to any appreciable degree. This is because the optimal thresholds change quite slowly as a function of load. For more details regarding this assertion, refer to Chapter 3 of this dissertation.

The internals of the tables in this chapter have the following significance: The column denoted by t_m represents the window of estimation. The other three columns represent the estimates provided by the technique for particular values of t_m. In each box which denotes the estimates, there are three elements: The topmost

Table 7.1: Performance of Method-1, Load = (0.8,0.6,0.4)

t_m	0.8	0.6	0.4
	0.81	0.61	0.4
7500	0.81	0.61	0.4
	1.0	1.0	0.0
	0.82	0.62	0.41
2500	0.86	0.65	0.36
	4.0	3.0	2.5
	0.81	0.62	0.41
1250	0.71	0.69	0.45
	5.0	6.5	2.5
	0.83	0.64	0.43
750	0.70	0.69	0.31
	6.0	6.5	7.0
	0.76	0.55	0.47
250	0.68	0.44	0.59
	10.0	14.0	17.0

element is the best estimate for a given t_m and load, the second element is the worst estimate obtained and the bottom element denotes the mean % error in the estimates. The sign of the error in the estimate is disregarded in the computation of the mean % error. Thus, estimates of 0.72 and 0.88 for a load of 0.8 will each generate an error of 10%. Unless specified otherwise, this structure is assumed to hold for all the tables that appear in the discussion that follows.

In the first set of tests, the loads were varied from 0.8 to 0.6 to 0.4, at intervals of 7500 units. Thus, for the first 7500 units of simulation time the average load was 0.8 and for the next 7500 units it was 0.6 and finally, 0.4. The simulations were run with different values of t_m, the time window used to estimate the load. The window was varied from the entire 7500 units to 250 units and the results of these experiments are shown in Table 7.1. Table 7.2 shows the results of similar experiments conducted with loads of 0.5, 0.7 and 0.9. t_i is reset at the end of each t_m.

The results shown are the means of the load as observed from 3 sets of ex-

Table 7.2: Performance of Method-1, Load = (0.5,0.7,0.9)

t_m	0.5	0.7	0.9
	0.5	0.7	0.91
7500	0.5	0.7	0.91
	0.0	0.0	1.0
	0.51	0.71	0.92
2500	0.54	0.67	0.86
	2.0	1.5	3.3
	0.53	0.68	0.88
1250	0.58	0.74	0.84
	6.0	2.5	4.0
	0.54	0.66	0.85
750	0.42	0.78	0.96
	7.5	5.0	5.0
	0.42	0.63	0.79
250	0.38	0.80	0.72
	20.0	13.0	15.0

periments conducted with different random number seeds. The main observations that can be made from Tables 7.1 and 7.2 are the following:

- In general, the larger the value of t_m, the more accurate is the outcome of the estimation procedure. This can be expected because the estimate of the load will converge to its exact value when t_m tends to infinity. For example, load=0.4 and t_m=7500 units (Table 7.1), the error is 0%.

- If we accept that exact values of the load are not needed to perform effective load sharing, then errors of around 5% should be tolerable. From the numbers, it is seen that this requirement is met by setting $t_m \geq 750$ units.

- For values of $t_m < 750$ units, the estimation procedure does not provide very stable results. This means that the value of the load is either over or under estimated, depending upon the actual set of events during that particular period. For example, about 15-20% errors are noticed in Table 7.1, when t_m = 250 units. For values of $t_m \geq 750$ units, the results are much more stable and the variation between the actual load and the estimate is small.

Method-2

Many computer systems do not possess the special purpose utilization monitoring hardware that was alluded to in the discussion on Method-1. In such cases, the processor utilization may need to be determined by software intervention. There are several alternative methods by which this can be achieved. We use the following simple procedure. At regular intervals of time separated by t_o units, the queue length at the designated node is determined. If there are no jobs in the queue, a *idle* counter is incremented by one. In any event, the *total* counter is incremented by one, independent of the queue length at the node.

Thus, the utilization may be approximated by ρ=1 - *idle/total*, with the conjecture that as the number of samples tends to infinity, the estimate of the utilization will tend to become exact. In this method there is likely to be some amount of context switching overhead incurred because of the periodic monitoring of the queue length, although the computational effort required by the estimation procedure to determine the load is negligible. The two parameters which are critical are the number of observations used to provide one estimate and the period at which the observations are made. Several simulations were conducted with different values of t_o, which is the time between observations, and t_m, which is the time needed for one estimate of the current load. Further, the counters are reset at the end of each t_m. Clearly, if t_o = the smallest unit of measured time in the system, Method-2 reduces to Method-1. Because the processing in case of Method-2 is by means of software, the idea is to minimize the potential overhead generated by this procedure.

To study the effectiveness of Method-2, we conducted simulations of homogeneous systems where the nodes cycle through different loads as in the case of Method-1. The two sets of loads tested were (0.9,0.7,0.5) and (0.8,0.6,0.4). A large range of values for t_m and t_o were used and the important results of our simulations of Method-2 are presented in Tables 7.3 and 7.4. From Tables 7.3 and 7.4, we are able to make the following observations:

Table 7.3: Performance of Method-2, Load = (0.9,0.7,0.5)

t_o	0.9	0.7	0.5
	0.91	0.72	0.49
1	0.94	0.65	0.55
	2.0	3.0	4.0
	0.92	0.68	0.48
2	0.86	0.74	0.58
	3.3	3.0	6.0
	0.86	0.66	0.54
5	0.82	0.62	0.43
	5.0	6.0	10.0
	0.84	0.75	0.46
10	0.78	0.60	0.38
	10.0	10.0	18.0

Table 7.4: Performance of Method-2, Load = (0.8,0.6,0.4)

t_o	0.8	0.6	0.4
	0.80	0.62	0.41
1	0.84	0.56	0.44
	2.5	3.3	2.5
	0.82	0.62	0.41
2	0.86	0.56	0.34
	4.0	3.3	5.0
	0.79	0.58	0.42
5	0.73	0.67	0.31
	5.0	5.0	12.0
	0.77	0.64	0.37
10	0.91	0.68	0.49
	8.5	10.0	15.0

- When $t_o = 1$ unit (which is the same as the mean service time of jobs), and $t_m = 500$ units, the estimates are very accurate, being about 2-4% off the exact value of the load. However, the rate at which the observations are made is fairly high and the overhead costs may become a factor.

- When $t_o = 2$ units and t_m is again 500 units, only half as many observations are made during the estimation period as compared to when $t_o = 1$. However, almost no appreciable difference is seen in the accuracy of the estimates (the error is about 4% as opposed to 2%). Thus, reducing the number of observations by half does not have a significant impact on the quality of the estimates.

- When $t_o = 5$ units and $t_m = 500$ (i.e., only 1/5th the observations are made as compared to the first case), there is greater variability between the estimates and the actual loads in some instances. However, in most of the cases, the estimates are again fairly good, being about 5-10% off the exact value. When t_o=10 units the results are not very good; the errors being as high as 18% at times (Table 7.3).

Thus, it would appear that in most instances, about 100 observations made over 500 units of time provide reasonably accurate estimates of the loads. If observations are made more frequently, the estimate does improve. However, when one considers the potential cost of this method in terms of context switches, one is inclined to favor making fewer observations.

In order to study the effect of varying t_m, we ran several simulations, the results of which are summarized in Tables 7.5 and 7.6. In these simulations, t_o is fixed at 5 units and t_m is varied over a large range (only a few of the results are actually presented here). From Tables 7.5 and 7.6, we are able to infer the following points: Clearly, when t_m is as low as 50 units and $t_o = 5$ units, the estimation quality is very poor, with errors greater than 50% at times, as for instance when. On the other hand, when $t_m = 1250$ units, the estimation is fairly accurate, with errors

Table 7.5: Performance of Method-2, Load = (0.8,0.6,0.4)

t_m	0.8	0.6	0.4
2500	0.80 0.83 1.0	0.61 0.58 1.5	0.40 0.47 5.0
1250	0.80 0.83 1.0	0.59 0.68 5.0	0.41 0.33 7.5
500	0.79 0.73 5.0	0.58 0.67 5.0	0.42 0.31 12.0
250	0.84 0.70 10.0	0.64 0.73 15.0	0.37 0.28 20.0

Table 7.6: Performance of Method-2, Load = (0.9,0.7,0.5)

t_m	0.9	0.7	0.5
2500	0.90 0.93 1.0	0.70 0.73 1.5	0.51 0.46 2.0
1250	0.91 0.86 3.3	0.71 0.74 2.5	0.52 0.45 6.0
500	0.86 0.82 5.0	0.66 0.62 6.0	0.54 0.43 10.0
250	0.86 0.78 12.0	0.69 0.62 9.0	0.48 0.41 14.0

being in the range of about 5%. On the whole, we have noticed that $t_m \geq 500$ units with $t_o = 5$ is able to provide reasonably good estimates of the changing loads. Higher values of t_m may facilitate the selection of lower t_o, but this will tend to reduce the responsiveness of the adaptation procedure. Decreasing t_m by some amount will, on the other hand necessitate smaller t_o, increasing the potential for overhead costs.

7.4 Heterogeneous Systems

Methods 1 and 2 clearly applicable in cases where the system remains homogeneous during the time of its operation. This is because both these methods directly determine the utilization and not the arrival rates at the nodes. In case of homogeneous systems where all the nodes use the exact same load sharing algorithm, this is adequate because the utilization of the nodes remains unchanged by the process of load sharing. This will not hold true in the case of heterogeneous systems. It may become necessary to directly estimate the arrival rate of local jobs. Of course, any estimation procedure designed for heterogeneous systems can be used for homogeneous systems as well. We have designed a recursive estimator based upon a simple filtering technique. A brief background of recursive filtering is presented in Appendix D.

7.4.1 Estimation Procedure

Method-3

In this method, we utilize the concepts of recursive filtering to design an estimator which is able to determine the mean inter-arrival time of local jobs at a node. The arrival of local jobs is continuously monitored. The time between the last and the new arrival is calculated each time a local arrival takes place. This value corresponds to the observation $y(k)$ referred to as the value of the kth observation, in Appendix D. Because of the exponentially distributed inter-arrival times, any given observation may be considered to be noisy. This is because from an estimation standpoint, it is not possible to determine the characteristics of the arrival process from any one observation. Thus, there is a continual refinement of the estimate as more arrivals take place. The computation involved in this technique is quite simple (a few multiplications and divisions on each arrival). Presumably, this computational overhead can be included with the normal set of system functions when a new job arrives at a node. Thus, no separate and expensive context switches may be necessary in order to compute the estimates.

We now present some of the results obtained by using Method-3 as the estimation technique. As stated earlier, we directly estimate the mean inter-arrival time of jobs. Because of this property, we are able to use this method in the study of heterogeneous systems. In Chapter 5, we presented performance results of heterogeneous systems and using these results, it is possible to determine the parameters necessary for optimal performance. Because the results in Chapter 5 are in terms of local arrival rates (and not the net arrival rates which may be different because of load sharing in heterogeneous systems), the inter-arrival time of local jobs only is determined by Method-3.

Because a new internal estimate of inter-arrival time is computed on each local arrival, high loads will generate more invocations of the recursive procedure and may produce better estimates than low loads. In any event, the old values of the estimate are discarded at the end of each period of estimation and the estimation process starts from the initial conditions, as was the case in Methods-1 and 2.

The nodes were subject to similar changing loads as in the previous two estimation methods. In one set of tests, the nodes cycle through loads of (0.9,0.7,0.5) and in the second set of runs, the loads are (0.8,0.6,0.4). Tables 7.7 and 7.8 depict the results of the estimation when Method-3 was adopted. The period of estimation, t_m, was varied between 125 units and 1000 units, where 1 unit is the mean service time of a job. For these tests, $\gamma = \sigma_v^2 = 2.0$. From Tables 7.7 and 7.8, we are able to make the following observations:

Table 7.7: Performance of Method-3, Load = (0.9,0.7,0.5)

t_m	0.9	0.7	0.5
	0.91	0.69	0.52
1000	0.91	0.69	0.52
	1.0	1.5	4.0
	0.88	0.67	0.53
500	0.84	0.76	0.42
	4.0	8.5	10.0
	0.85	0.66	0.46
250	0.82	0.78	0.58
	7.0	10.0	14.0
	0.82	0.64	0.45
125	0.76	0.80	0.27
	13.0	15.0	24.0

Table 7.8: Performance of Method-3, Load = (0.8,0.6,0.4)

t_m	0.8	0.6	0.4
	0.81	0.61	0.42
1000	0.81	0.61	0.42
	1.0	1.5	5.0
	0.82	0.63	0.38
500	0.75	0.55	0.46
	2.5	5.0	7.5
	0.83	0.56	0.44
250	0.73	0.65	0.33
	6.0	8.0	12.0
	0.73	0.54	0.47
125	0.90	0.48	0.54
	10.0	15.0	22.0

- Clearly, when t_m is small, i.e., 125 units, the estimates are not very accurate. For instance, there can be about 10-20% error in estimating the mean inter-arrival times, as seen in Tables 7.7 and 7.8.
- As we had expected, the effect of short t_m were more pronounced on low arrival rates, as may be seen by $\rho = 0.5$ (Table 7.7), where the error in estimation is as high as 24%.
- When t_m is increased beyond 500 units, the overall quality of the estimates is significantly improved, particularly for low arrival rates, because it is now possible to make many more observations than with short t_m.

7.4.2 Reaction to Surges

Until this point in this chapter, we have concentrated on estimating the varying loads when all the nodes undergo the same changes. In this subsection, we determine the effectiveness of the adaptation procedure in the case where one or more nodes in the system is subjected to a sudden surge of jobs. These types of surges occur quite often in real systems, particularly in control engineering applications where sudden changes in the external system could necessitate the execution of several new jobs. In many instances, these surges are of a short duration (especially when compared to the length of the system uptime). The main questions that are of interest here are the following:

- What is the time needed to estimate that a surge has occurred?
- Is this time short enough to actually change the system parameters and effectively react to the surge?

To study the above problems, we conducted tests on a network of 10 nodes. The surge jobs are introduced only on the designated node which also performs the estimation procedure. To detect the occurrence of surges, we use Method-1 which determines the utilization and Method-3 which is able to directly compute the mean inter-arrival time of jobs (including those that belong to the surge). Two sets of tests were conducted. In the first set, the base load was varied from 0.4 to 0.6 and 0.8 at the nodes, each for 5000 time units. At time 5000, a surge of length = 2500 units which increased the effective load from 0.6 to 0.9, was introduced at Node-1 (the designated node). The tests were conducted with two different thresholds, 0 and 10. In the second set of tests, the base load at each node was varied from 0.9 to 0.7 and to 0.5, at time units 5000, 10000 and 15000, respectively. The surge was started at the designated node at time 10000 (i.e., base load = 0.5), and increased the load to 0.8, again for a period of 2500 units. For these set of tests, t_m is 1000 units.

Surges and Method-1

The results of these tests conducted on Method-1 are shown in Figures 7.1 and 7.2. The X-axis depicts the simulation time and the Y-axis depicts the loads. From Figures 7.1 and 7.2, we notice the following facts:

- When $T = 0$, it is seen that although there is a visible effect of the surge on the estimation procedure, the results are less than satisfactory. The best estimate provided for the net load of 0.9 (including the surge) is 0.723 (Figure 7.1). For the second surge (load= 0.8), the best estimate is 0.71 (Figure 7.2). This is because the threshold of 0 precipitates very active load sharing and thus many of the jobs that arrived as a result of the surge are actually transferred to other nodes. Recall that Method-1 estimates the utilization and jobs transferred out of the designated node do not affect the utilization.

- When $T = 10$, much better results are seen from an estimation standpoint. The best estimate of load during the surge when the effective load is 0.9, is 0.884 (Figure 7.1), which is less than 2% off the actual load. In the second surge (Figure 7.2), the results are almost identical as regards their accuracy. Obviously, the high threshold means that very few jobs are transferred out

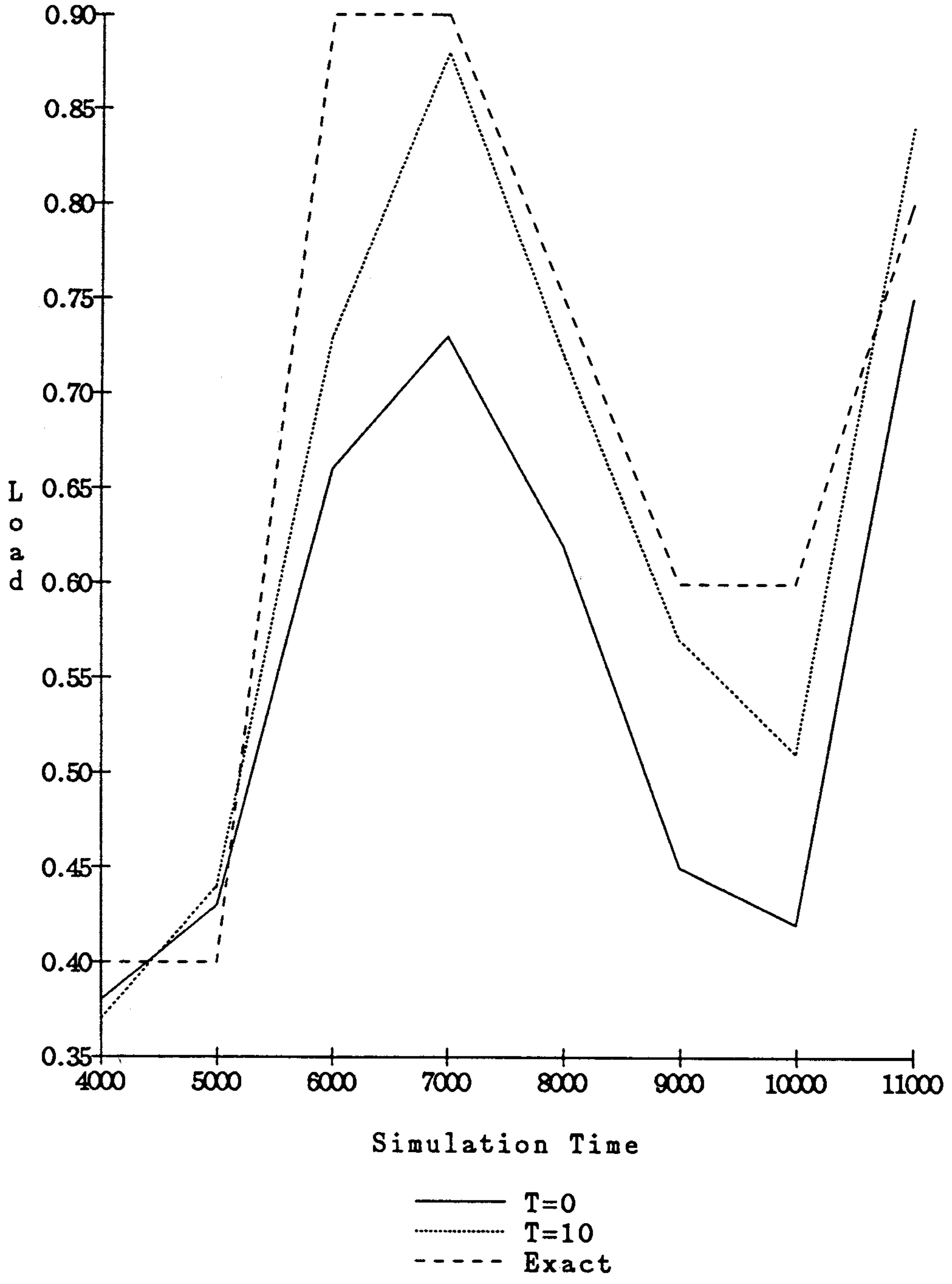

Figure 7.1: Estimation of Surge (Method-1)

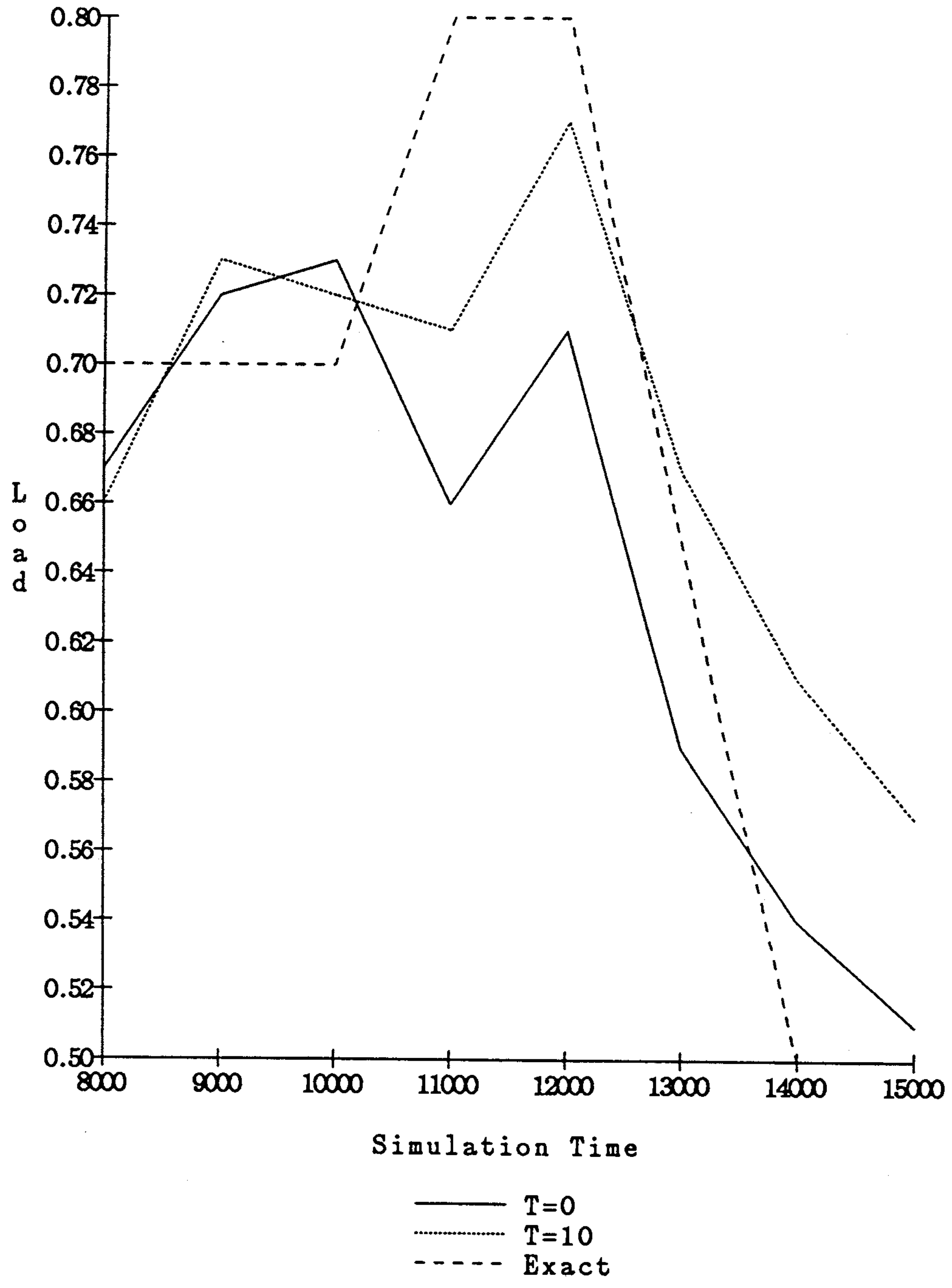

Figure 7.2: Estimation of Surge (Method-1)

of Node-1 and most of the jobs resulting from the surge actually execute at Node-1. Thus, the utilization of Node-1 reflects this fact.

- For both the curves, it is seen that the estimates have a bell shaped curve, with a sharp increase and decrease as the surge begins and ends. For $T = 10$, the estimated load approaches the exact load at the end of 7000 units.

Thus, we can conclude that reasonably good estimation of surge intensity by this technique mandates high thresholds and consequently the jobs that arrive as a result of the surge are not quickly dispersed around the rest of the network. If the thresholds were set lower (as they might be for low to moderate job transfer delays), the surges will tend to clear themselves out in a short period of time.

Surges and Method-3

We now present the results of the exact same experiments as above, but conducted on a simulation system in which the designated node (which receives the surge jobs) utilizes Method-3 to estimate the arrival rates. The results of these experiments are shown in Figures 7.3 and 7.4. Figure 7.3 corresponds to the set of loads (0.4,0.6,0.8), with the surge starting at time = 5000 and raising the effective load at the designated node to 0.9 for 2500 units of time. Figure 7.4 corresponds to the load set (0.9,0.7,0.5), with the surge beginning at time 10000 and increasing the load from 0.5 to 0.8, for a period of 2500 units. From these figures, we are able to see that:

- The results of the estimation procedure are not affected by the internal parameters of the algorithm (threshold) as they are in the case of Method-1 and the exact same estimates are provided when $T = 0$ and $T = 10$. This is because the estimation procedure directly determines the mean inter-arrival time of local jobs and is unaffected by the process of load sharing, even though jobs are transferred out of Node-1.

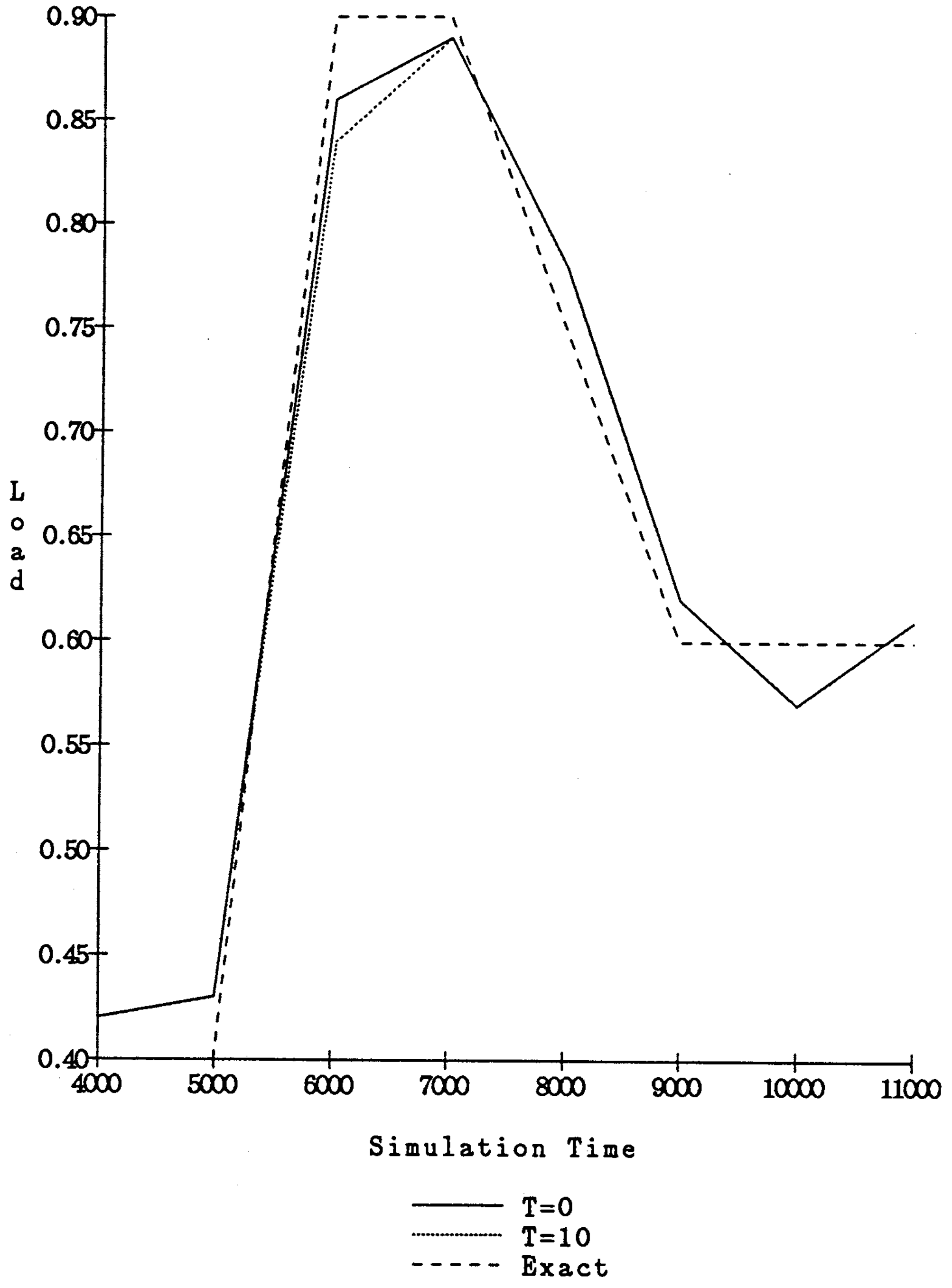

Figure 7.3: Estimation of Surge (Method-3)

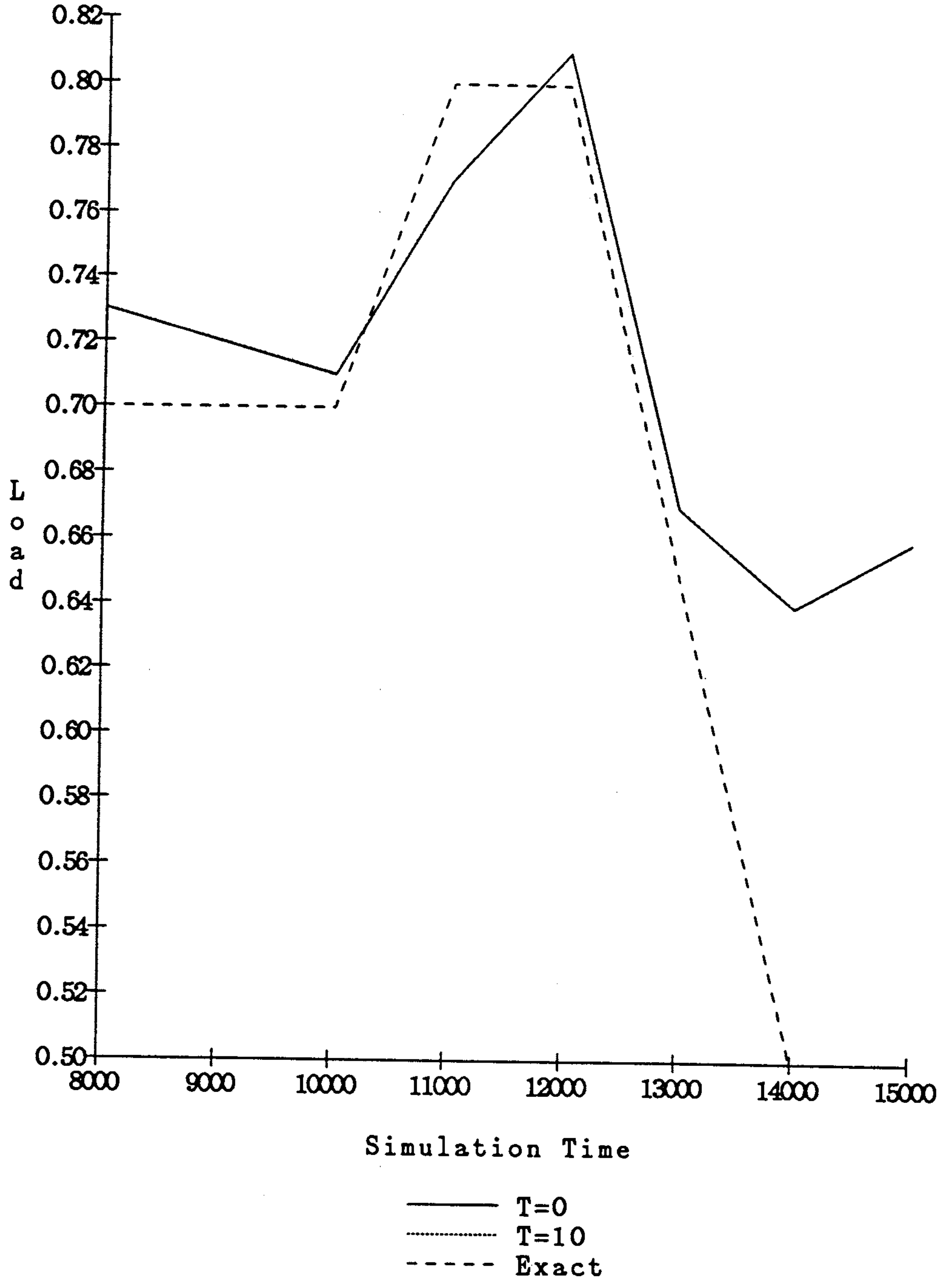

Figure 7.4: Estimation of Surge (Method-3)

- It appears that $t_m = 500$ units is appropriate and at the end of the first t_m after the start of the surge, a very good estimate of the arrival rate is available. In Method-1, on the other hand, it was noticed that the detection of the surge was much more gradual.

Thus, Method-3 appears to be more responsive in terms of detecting changes in the arrival rate, particularly in connection with surges, as compared to Method-1.

7.5 Performance Considerations

While we have focussed upon the issue of detecting the changing loads, the final objective of the adaptation procedure is to improve response time. In this section, we investigate control strategies which utilize the estimates of the changing loads to modify the internal parameters (e.g., the thresholds) of the load sharing algorithms (in this chapter, we are only concerned with the Symmetric probing algorithm). We utilize the average system response time as the main metric for comparison. At all times, the adaptive algorithm will be compared against the optimal strategy (hereafter referred to as *Opt*) used under the different loads on the system. Because the estimation procedure only approximates the actual load on the system, it is likely that the parameters chosen by the algorithm may result in performance that is less than optimal.

7.5.1 Description of Experiments

In this subsection, we describe the experiments conducted to determine the performance benefits that accrue as a result of estimation and reacting to changes in the system load. In the first set of tests, the system undergoes changes in the load, but the homogeneous nature of the nodes is not disturbed.

The designated node continuously monitors its local load, using any one of the estimation techniques described earlier in this chapter. The period of observation

t_m, is a run-time parameter of the system. At the end of a particular period of observation, the new system parameters are computed and broadcast to the rest of the network. These new parameters are utilized for the purpose of load sharing until the end of the next t_m, when the parameters are recomputed and rebroadcast. As mentioned earlier, the estimation techniques discard all statistics of the previous period of observation once the new period has started. By this we ignore the questions about historical data and how it should be utilized. More important has been the observation that t_m is not a very large number and the system is able to adapt reasonably quickly, as seen in Sections 7.3 and 7.4.

We have developed a simulation system in SIMSCRIPT II.5 that is able to interact with the estimation procedures and the control strategies. From the results in Chapter 3, we know the optimal strategy to use under different loads and job transfer delays. This knowledge is codified and made available to the simulation system at the start of the test. The relation between loads/delays and optimal thresholds as well as the load sharing algorithm is available to the program in the form of 2-dimensional look-up tables, easily accessed at run time. One of the questions here is the following: How should the range of load (0.0 to 1.0) be divided. In our tests, the load is varied in steps of 0.1, from 0.1 to 1.0. Thus, the look up tables for each value of job transfer delay have 10 entries (each of these being the optimal threshold for that combination of delay and load). Thus, while the program tracks the changing loads in the system, the estimated load and the transfer delay are used to determine the new operating threshold.

In the subsequent discussions, the tables depict the performance of the adaptation procedure from the standpoint of response time. Method-1 with $t_m = 750$ units was used for the purposes of estimation. Also, the load was changed at durations of 5000 units of simulation time. The results are presented for three different values of delay and the two numbers in each box represent the *optimal* response time (upper number) and that generated by the adaptation procedure (lower number). Algorithm *Opt* utilizes the optimal threshold at all times. The adaptive strategy

Table 7.9: Response Times, Load = (0.4,0.6,0.8)

Delay	0.4	0.6	0.8	Total
	1.08	1.20	1.47	1.29
$0.1S$	1.10	1.23	1.51	1.32
	1.45	1.76	2.27	1.91
S	1.49	1.82	2.38	1.99
	1.67	2.45	4.11	3.01
$10S$	1.82	2.53	4.23	3.15

Table 7.10: Response Times, Load = (0.5,0.7,0.9)

Delay	0.5	0.7	0.9	Total
	1.13	1.30	1.88	1.50
$0.1S$	1.14	1.35	1.97	1.56
	1.59	1.97	2.85	2.25
S	1.66	2.05	2.94	2.34
	1.99	3.10	5.88	4.03
$10S$	2.13	3.22	6.21	4.27

uses the thresholds from the look-up tables which are made available beforehand to the simulation program. There is no estimate available at the start of the run and any arbitrary threshold may be input to the simulation at that time. For the tests conducted in this chapter, T was initially set to 2. All the results are averages of at least three runs. It was observed that the standard deviations as computed by the Student-t tests, were very small for all the loads tested.

The numbers in Tables 7.9 and 7.10 correspond to two sets of loads. The first set is comprised of loads 0.4, 0.6 and 0.8, while the second is comprised of 0.5, 0.7 and 0.9. From Tables 7.9 and 7.10, we are able to make the following observations:

- From Tables 7.9 and 7.10, we see that the response time values for *Opt* and for the adaptive algorithm are almost identical at low delays. This is explained by the fact that the optimal threshold is 0 for delay = $0.1S$, for almost the entire range of load ($\rho \leq 0.9$) of interest (when $\rho > 0.9$, $T_{opt} = 1$). The slight variation between the results stems from two main reasons, i.e., the initial value of T chosen by the adaptive algorithm and the fact that the estimation technique may generate slight errors in the estimated load. Thus, for $\rho = 0.9$, the times when $T = 1$ is selected because of slight errors in estimation (i.e., the estimate of the load is higher than 0.9), the result is sub-optimal performance.

- Because the initial value of $T = 2$, very active load sharing is performed resulting in the slightly higher errors for high delays, as seen in Tables 7.9 and 7.10 (the optimal threshold for Delay $= 10S$ and $\rho = 0.4$ and 0.5 is around 10 as seen in Chapter 3). This situation is rectified as soon as the first estimate of the load is generated.

- It is seen that the performance difference between the adaptive and *Opt* strategies increases with increasing loads and delays. This is because system performance is more sensitive to thresholds as delays and/or loads increase. In any event, the performance differences between the adaptive and *Opt* strategies is seen to lie between 3-8%, in the tests conducted.

7.6 Summary and Conclusions

In this chapter, we have briefly examined the problem of how to detect the changing loads in a distributed system and adapt the load sharing strategy in response to the changed loads. We divided the problem into two parts: The first part dealt with the estimation of the changing parameters and the second involved the modification of the load sharing strategy in response to the external changes. The estimation problem was tackled by three simple methods, two of which were only applicable to homogeneous systems while the third method, based on recursive filtering, was much more general in its applicability, being effective in heterogeneous systems as well. It was seen from the tests conducted that these estimation techniques performed quite adequately. Further, the estimation time required was not very large, even when the system was subjected to surges (although it is not clear if the durations of the surges in our simulations were reasonable, from a practical standpoint).

In relation to response time improvements as a result of adapting the control strategy, it was seen that in many instances, the performance difference between the adaptive and *Opt* strategies was not very significant, particularly at low to moderate delays. At such values of delays, the optimal thresholds are not highly dependent upon the system load. From the results in Chapter 3, we know that the probing algorithms are quite robust with respect to slight errors in the operating threshold (i.e., the threshold need not always be optimal to generate very good response time, as long as it is close to the optimal value).

Chapter 8

SUMMARY AND FUTURE WORK

In this chapter, we summarize the main contributions of this dissertation and propose several interesting extensions of this research which we hope to investigate in the future.

8.1 Summary and Conclusions

In Chapter 3, we developed analytical models and solved these models using the Matrix-Geometric solution technique, for the following load sharing policies:

- A sender-initiated policy called *Forward.*
- A receiver-initiated policy called *Reverse.*
- A combination policy called *Symmetric.*

From the results in Chapter 3, we observed that the analytical solutions were valid over a very large range of system parameters. From these solutions, we were able to study the effects of various important parameters on load sharing, particularly in relation to delays. It was seen that Symmetric probing performed consistently well over the entire range of parameters tested and that neither Forward nor Reverse could outperform it, although Forward approached its performance in the low end of the traffic intensity. It was also seen that the optimal thresholds were a function of the job transfer delays and the traffic intensity, with delays being the more dominant factor. In comparison with a Random assignment algorithm, it was noticed that probing ceased to be advantageous after delays greater than $2 - 3S$, where S was the average service time of jobs. At high delays, Random performed almost as well as any probing algorithm tested. Another interesting observation we made was that even when delays were extremely large ($100S$), there was about 7% gain in load sharing over the corresponding $M/M/1$, when the traffic intensity was high (0.9).

In order to simplify the above analytical models, we had made certain assumptions which we believed were reasonable. For example, we had assumed that probes were transferred in zero time, in spite of large delays during job transfers. Further, K the maximum number of allowable pending remote jobs was one, and that the probing and probed nodes have the same operating thresholds. To address the validity of these assumptions, we developed analytical models and solved these, using the Matrix-Geometric solution technique, for the following Receiver-Initiated load sharing policies:

- Policy R_K, where K was a parameter representing the maximum number of allowable pending remote jobs.
- Policy R_{K_T}, which was a threshold probing variation of R_{K_T}.
- Policy R_{T^2}, where the probing and probed nodes may have different operating thresholds.
- Policy R_D, where probes took non-zero times.
- Policy R_{D_T}, which was a threshold probing variation of R_D.

From the results of the studies conducted on the above algorithms (Chapter 4), we concluded that our initial assumptions were quite reasonable and that our intuition was by and large correct. For instance, very little or almost no performance benefits were seen when K was increased beyond 1. In the case of probes taking non-zero times, it was noticed that as long as probes were about 10-20 times faster than jobs in reaching their destination, the system essentially behaved as if the probes took zero time. Because we believe that probes will tend to be much smaller in size than jobs (1 packet of data as opposed to hundreds or thousands for a job), the zero probing delay assumption appeared to be reasonable. Also, a dual threshold algorithm appeared to have almost no performance gains over one that employed a single threshold. Further, the threshold probing variations of the algorithms performed consistently worse than their complete probing counterparts.

In Chapters 3 and 4, we had assumed that the underlying system was homogeneous. We know that in practice, this is a restrictive assumption and that many systems are comprised of heterogeneous nodes. Consequently, we extended our analytical models and determined solutions for policies operating in such systems. These were:

- A sender-initiated policy called *Forward*
- A receiver-initiated policy called *Reverse*

These algorithms were tested for **type-1** and **type-2** systems where **type-1** systems consisted of several classes of nodes where the arrival rates of jobs for the various classes were different but the processing speeds of the nodes were identical while **type-2** systems comprised of heterogeneous nodes which had different processing speeds. The results of this study were presented in Chapter 5. From the solutions of the above models, we observed several interesting phenomena. For instance, load sharing in heterogeneous systems was seen to be effective for much higher delays than for homogeneous systems, especially when the degree of imbalance in the loads was large. Also, in **type-1** systems, it was seen that under a high degree of heterogeneity, Random performed very well in comparison with the probing algorithms (it performed better than Reverse at times and was very close to Forward). The performance of Forward probing was less sensitive to the probe limit, particularly in highly unbalanced systems. However, for the same parameters, Reverse probing was much less efficient where probes were concerned. In the case of **type-2** heterogeneous systems, it was seen that Random assignment performed quite poorly and probing appeared to provide much greater performance benefits than in the case of **type-1** systems.

In Chapter 6, we described simulation studies of the simple load sharing algorithms with a particular focus on the problem of estimating the threshold as a function of delay and traffic intensity. Some the interesting results in this part of our research were as follows: We utilized Entropy Minimax, an Information theoretic estimation technique to estimate the thresholds. This method had the advantage of being non-parametric (which was particularly relevant in the context of load sharing), was computationally inexpensive and had adaptive capability. It was seen performance of load sharing policies using the thresholds generated by Entropy Minimax, was found to either be optimal or very near optimal, in most instances. We also studied the effect of load sharing in the presence of delays when the service times of the jobs were not exponentially distributed. In some sense, this part of our study provided some understanding about how to deal with the uncertainty inherently present in distributed systems, as for example by designing state classification techniques which reduce the impact of this uncertainty.

Finally, in Chapter 7, we considered systems in which the arrival rates at the nodes were time varying. We developed several *simple* strategies to recognize these variations and adapt the parameters of the policies and in some cases, change the policy itself, as a means of providing effective control in such situations. The reason we emphasize the word simple is because we had observed that the load sharing policies were quite stable in relation to small errors in the values of the estimated parameters. We conjectured that sophisticated techniques to solve the problem will in all likelihood, provide little gain over simple techniques. From the results of this chapter, we determined that in many instances, the threshold policies that we have studied are very robust with respect to small changes in traffic intensity. Thus, if the user was willing to compromise optimal performance by about 5%, simple adaptive policies which were able to provide coarse grained estimation of loads appeared to be adequate.

8.2 Some Directions for Future Work

There are several avenues available for extending this work as part of some later study. We now discuss a few of the more interesting alternatives.

In Chapter 3, we studied the performance of three simple load sharing algorithms when job transfers experienced significant delays. Implicit in our study was the assumption that the average transfer delay was the same between all pairs of nodes. Although this is generally true for simple Ethernet and Ring type mediums, we believe that it might be interesting to study systems where this assumption may not hold, for example in a multi-level bus structure where job transfers from nodes on the same bus may take less time than between nodes on different busses. Consequently, the probing policies may also change to reflect this fact: A node may provide preferential treatment to others on its bus. Further, this type of system architecture is practical because of the known limitations of single bus systems. In our study, job transfer delays were independent of the network traffic. Clearly, for medium to heavy loading of the network, this assumption may not be valid. Thus, we are also interested in analyzing more sophisticated models of the underlying network in conjunction with our load sharing policies.

In the analytical models for non-zero probing times described in Chapter 4, we had assumed that the replies to a set of probes did not depend upon the probe limit. In reality, probing delay will be a function of the probe limit and it would be interesting to determine the effects of varying the average probing delay on system performance. Implicit in our models was the assumption that there was no contention for resources between probes and jobs. In reality, it may happen that depending upon the architecture of the network controller, a probe may not be accommodated at a node because all the buffers at its controller may be full. If such a scenario were to be considered, one could try to determine its effect on the performance of the algorithms, i.e., how often may a set of probes not result in any response and how this might impact the probe limit. Further, the work on non-zero probing times was only conducted with Reverse probing algorithms. Although we believe that the results generally apply to Forward and Symmetric probing as well, it would be worthwhile to extend the Forward probing any maybe even the Symmetric probing models to include this phenomenon.

In the case of heterogeneous systems studied in Chapter 5, there are several interesting areas of future research. In the work done in this dissertation, we have assumed that each of the classes possesses a large number of nodes. In fact, many real systems are typically heterogeneous but have specific network architectures. For instance, a star network could consist of a powerful central node and a number of less powerful satellite machines. Such architectures are being designed and implemented for such diverse applications as telephone switching and research computing where each user possesses a workstation and only occasionally needs to use the central node. In such systems, load sharing may be performed between the satellites and the central node or over the entire network. It is not altogether clear what kind of load sharing policies might be appropriate in these situations.

The work in this dissertation was analytical and simulation oriented. Consequently, we were forced to make several simplifying assumptions in our models, particularly in the case of the analytical studies. There is considerable interest in load sharing from an experimental viewpoint and with the proliferation of multiprocessor systems and computer networks, it is now be feasible to implement simple load sharing algorithms on real systems. Consequently, we feel that it will be interesting to compare the results of the tests performed on real systems against the results obtained from the more idealistic models that we have studied thus far. In particular, we will be interested in studying various alternative methods to classify system load from a standpoint of deciding when it might be worthwhile to try and move jobs. For instance, an interesting question that begs to be asked is the following: How effective will threshold policies be, in real systems (in fact, some implementations have shown that simple load sharing algorithms may provide adequate performance in certain types of systems, e.g., as in the Stanford V-System). If not, how will we adapt the load sharing policies studied in this dissertation to the more realistic circumstances.

Appendix *A*

In this appendix, we give closed form representations of the matrices A_0, A_1, A_2 and the matrices $B_{00}, B_{01}, B_{10}, B_{11}$, and B_{21}, for the Symmetric probing algorithm.

$$B_{00} = \begin{bmatrix} -(\alpha+\lambda) & 0 & \alpha & 0 \\ 0 & -(\alpha+\gamma+\lambda) & 0 & \alpha \\ 0 & 0 & -(\gamma+\lambda) & 0 \\ 0 & 0 & 0 & -(2\gamma+\lambda) \end{bmatrix}$$

$$B_{01} = \begin{bmatrix} \lambda & 0 & 0 & 0 \\ \gamma & \lambda & 0 & 0 \\ \gamma & 0 & \lambda & 0 \\ 0 & \gamma & \gamma & \lambda \end{bmatrix}$$

$$B_{10} = \begin{bmatrix} \mu q & \mu \bar{q} & 0 & 0 \\ 0 & \mu & 0 & 0 \\ 0 & 0 & \mu q & \mu \bar{q} \\ 0 & 0 & 0 & \mu \end{bmatrix}$$

$$B_{21} = \begin{bmatrix} -\sigma & 0 & 0 & 0 \\ 0 & -(\gamma+\sigma) & 0 & 0 \\ 0 & 0 & -(\gamma+\sigma) & 0 \\ 0 & 0 & 0 & -(2\gamma+\sigma) \end{bmatrix}$$

$$A_0 = \begin{bmatrix} \lambda h & 0 & 0 & 0 \\ \gamma & \lambda h & 0 & 0 \\ \gamma & 0 & \lambda h & 0 \\ 0 & \gamma & \gamma & \lambda h \end{bmatrix}$$

$$A_1 = \begin{bmatrix} -\delta & 0 & 0 & 0 \\ 0 & -(\gamma+\delta) & 0 & 0 \\ 0 & 0 & -(\gamma+\delta) & 0 \\ 0 & 0 & 0 & -(2\gamma+\delta) \end{bmatrix}$$

$$A_2 = (\mu + \mu')I_4$$

where

$$\delta = (\lambda h + \mu + \mu'),$$

$$\sigma = (\lambda + \mu),$$

and I_4 is the identity matrix of size 4.

We now provide closed form representations for the matrices in the case of the Forward and Reverse probing algorithms.

Forward

$$B_{00} = \begin{bmatrix} -(\alpha+\lambda) & \alpha \\ 0 & -(\gamma+\lambda) \end{bmatrix}$$

$$B_{01} = \begin{bmatrix} \lambda & 0 \\ \gamma & \lambda \end{bmatrix}$$

$$B_{11} = \begin{bmatrix} -(\alpha+\lambda+\mu) & \alpha \\ 0 & -(\mu+\gamma+\lambda) \end{bmatrix}$$

$$x = \sum_{i \le T} \mathbf{p_i}\,[10]^T + \sum_{i > T} \mathbf{p_i} e$$

which is the probability that a node will respond negatively to a forward probe. Thus, $\bar{x} = 1 - x$ is the probability that a node will respond positively to a forward probe.

If a node probes L_p nodes, then the probability that the set of probes results in failure is

$$h = x^{L_p}$$

$$A_0 = \begin{bmatrix} \lambda h & 0 \\ \gamma & \lambda h \end{bmatrix}$$

$$A_1 = \begin{bmatrix} -(\mu + \lambda h) & 0 \\ 0 & \mu + \lambda h \end{bmatrix}$$

$$A_2 = \mu I_2$$

where I_2 is the identity matrix of size 2.

Also, $R = [r_{i,j}]$ can be written as follows:

$$\begin{aligned} r_{1,1} &= \lambda h/\mu \\ r_{2,2} &= \frac{\theta + \gamma - ((\theta + \gamma)^2 - 4\mu\lambda h)^{1/2}}{2\mu} \\ r_{1,2} &= 0 \\ r_{2,1} &= \frac{\gamma}{\theta - (r_{1,1} + r_{2,2})\mu} \end{aligned}$$

where $\theta = \lambda h + \mu$. It can be shown that the stability criterion for the Forward probing algorithm is

$$\lambda h < \mu.$$

Reverse

To determine q, the probability of a set of reverse probes resulting in failure, we use the following procedure:

Let

$$y = \sum_{i \leq T+1} \mathbf{p}_i e$$

If the node probes L_p nodes to receive a remote job, then the probability that all of them will be unsuccessful is denoted by: $q = y^{L_p}$, and $\bar{q} = 1 - q$ is the probability that at least one of the reverse probes is successful.

$$B_{00} = \begin{bmatrix} -\lambda & 0 \\ 0 & -(\lambda + \gamma) \end{bmatrix}$$

$$B_{10} = \begin{bmatrix} \mu q & \mu \bar{q} \\ 0 & \mu \end{bmatrix}$$

$$B_{11} = \begin{bmatrix} -(\mu + \lambda) & 0 \\ 0 & -(\mu + \gamma + \lambda) \end{bmatrix}$$

$$A_0 = \begin{bmatrix} \lambda & 0 \\ \gamma & \lambda \end{bmatrix}$$

$$A_1 = \begin{bmatrix} -(\mu' + \lambda) & 0 \\ 0 & -(\mu' + \lambda + \gamma) \end{bmatrix}$$

$$A_2 = (\mu + \mu')I_2$$

Also, $R = [r_{i,j}]$ can be written as follows:

$$\begin{aligned} r_{1,1} &= \lambda/(\mu + \mu') \\ r_{2,2} &= \frac{\phi + \gamma - ((\phi + \gamma)^2 - 4(\mu + \mu')\lambda)^{1/2}}{2(\mu + \mu')} \\ r_{1,2} &= 0 \\ r_{2,1} &= \frac{\gamma}{\phi - (r_{1,1} + r_{2,2})(\mu + \mu')} \end{aligned}$$

where $\phi = \lambda + \mu + \mu'$. It can be shown that the stability criterion for the Reverse probing algorithm is

$$\lambda < \mu + \mu'.$$

Appendix B

We now give closed form representations of the matrices A_0, A_1, A_2 and the boundary matrices for the $R_K, R_{K_T}, R_{T^2}, R_D$ and R_{D_T} algorithms. For ease of representation, we have assumed $K = 1$.

To determine q, the probability of a set of reverse probes resulting in failure, we use the following procedure:

Let

$$y = \sum_{i \le T+1} p_i \, e$$

If the node probes L_p nodes to receive a remote job, the the probability that all of them will be unsuccessful is denoted by: $q = y^{L_p}$, and $\bar{q} = 1 - q$ is the probability that at least one of the reverse probes is successful.

Algorithm R_K

$$B_{00} = \begin{bmatrix} -\lambda & 0 \\ 0 & -(\lambda + \gamma) \end{bmatrix}$$

$$B_{10} = \begin{bmatrix} \mu q & \mu \bar{q} \\ 0 & \mu \end{bmatrix}$$

$$B_{11} = \begin{bmatrix} -(\mu + \lambda) & 0 \\ 0 & -(\mu + \gamma + \lambda) \end{bmatrix}$$

$$A_0 = \begin{bmatrix} \lambda & 0 \\ \gamma & \lambda \end{bmatrix}$$

$$A_1 = \begin{bmatrix} -\delta & 0 \\ 0 & -(\gamma + \delta) \end{bmatrix}$$

$$A_2 = (\mu + \mu')I_2,$$

where I_2 is the identity matrix of size 2 and $\delta = (\mu + \mu' + \lambda)$

Algorithm R_{K_T}

The internals of A_0, A_1 and A_2 for this algorithm are identical to those of the R_K algorithm. This is obvious because the two Markov processes have identical structures after $T + 1$. This means that the R matrices are also the same in both the cases. Further, the B_{00} matrices are also identical. However,

$$B_{10} = \begin{bmatrix} \mu & 0 \\ 0 & \mu \end{bmatrix}$$

$$B_{20} = \begin{bmatrix} \mu q & \mu \bar{q} \\ 0 & \mu \end{bmatrix}$$

Algorithm R_{T^2}

The internals of the matrices $A_0, A_1, A_2, B_{00}, B_{10}$ and B_{11} for this process are identical to those for algorithm R_1. Thus, the R matrix for this process is the same as that for algorithm R_1. However, there is the matrix B_{20} which does not have a counterpart in R_1.

$$B_{20} = \begin{bmatrix} \mu & 0 \\ 0 & \mu \end{bmatrix}$$

Algorithm R_D

$$B_{00} = \begin{bmatrix} -\lambda & 0 & 0 \\ 0 & -(\gamma+\lambda) & 0 \\ \alpha & 0 & -(\alpha+\lambda) \end{bmatrix}$$

$$B_{10} = \begin{bmatrix} 0 & \mu\bar{q} & \mu q \\ 0 & \mu & 0 \\ 0 & 0 & \mu \end{bmatrix}$$

$$B_{11} = \begin{bmatrix} -\sigma & 0 & 0 \\ 0 & -(\gamma+\sigma) & 0 \\ \alpha & 0 & -(\alpha+\sigma) \end{bmatrix}$$

$$A_0 = \begin{bmatrix} \lambda & 0 & 0 \\ \gamma & \lambda & 0 \\ \alpha & 0 & \lambda \end{bmatrix}$$

$$A_1 = \begin{bmatrix} -\delta & 0 & 0 \\ 0 & -(\gamma+\delta) & 0 \\ 0 & 0 & -(\alpha+\delta) \end{bmatrix}$$

$$A_2 = (\mu + \mu')I_3,$$

Also, $R = [r_{i,j}]$ can be written as follows:

$$\begin{aligned} r_{i,j} &= 0 \,\forall i < j \\ r_{1,1} &= \frac{\delta - (\delta^2 - 4\lambda(\mu+\mu'))^{1/2}}{2(\mu+\mu')} \\ r_{2,2} &= \frac{\delta + \gamma - (\delta^{2'} - 4\lambda(\mu+\mu'))^{1/2}}{2(\mu+\mu')} \\ r_{3,3} &= \frac{\delta + \alpha - (\delta^2 - 4\lambda(\mu+\mu'))^{1/2}}{2(\mu+\mu')} \end{aligned}$$

$$
\begin{aligned}
r_{2,1} &= \frac{\gamma}{\delta - (r_{1,1} + r_{2,2})(\mu + \mu')} \\
r_{3,1} &= \frac{\alpha}{\delta - (r_{1,1} + r_{3,3})(\mu + \mu')} \\
r_{3,2} &= 0
\end{aligned}
$$

where I_3 is the identity matrix of size 3 and $\sigma = (\mu + \lambda)$.

Algorithm R_{D_T}

The matrices A_0, A_1, A_2 and B_{11} for this algorithm are the same as the corresponding ones for algorithm R_D. Consequently, the R matrices for the processes are also identical. However,

$$
B_{20} = \begin{bmatrix} 0 & \mu\bar{q} & \mu q \\ 0 & \mu & 0 \\ 0 & 0 & \mu \end{bmatrix}
$$

$$
B_{10} = \begin{bmatrix} \mu & 0 & 0 \\ 0 & \mu & 0 \\ 0 & 0 & \mu \end{bmatrix}
$$

Appendix C

In this appendix, we describe a lower bound called LB_2, used as a bound in the case of **type-2** heterogeneous systems. This lower bound was described in Stankovic [STAN84]. While the analysis will be presented for a system of C classes, we will study the specific case of the 2 class systems that have been analyzed in this chapter. It is assumed that N_c is the number of processors in class-c. The arrival rate of jobs at a class-c node is Poisson with λ_c and the service is exponential, with rate μ_c. If it is assumed that all the arrivals occur at a central queue servicing all the nodes, then the net arrival process is also Poisson, with rate $\Lambda = \sum_{c=1}^{C} N_c\lambda_c$. The processors are sorted in decreasing order of service rate. The state diagram corresponding to this algorithm is shown in Figure 5.26. The basic idea in this algorithm is as follows: When a new arrival occurs, it is sent to the fastest available processor. If no processor is available, the job waits in the queue. On completion of a job, the job at the head of the queue is scheduled on the freed processor. If a job completes on a processor and no job is waiting in the queue, an executing job (if one exists) from a slower processor is moved to the faster processor. Thus, at any time, all the fastest processors are utilized first. Further, it is assumed that transfers of jobs between processors involves no cost or delays.

We define the effective processing rate at state i as

$$\Omega_i = \sum_{c=1}^{d-1} N_c\mu_c + \sum_{j=1}^{i-k_{d-1}} \mu_d, \forall i \leq N$$

where N is the total number of processors in the system, and

$$k_{d-1} = \sum_{c=1}^{d-1} N_c$$

When $i > N$,

$$\Omega_i = \sum_{c=1}^{C} N_c \mu_c$$

Further, we have the state probabilities,

$$p_i = \begin{cases} (1/i!) * (\Lambda/\mu_1)^i p_0 & \forall i \leq N_1 \\ \lambda_i / \prod_{i=1}^{N_d} (\sum_{c=1}^{d-1} N_c \mu_c + i \mu_d) p_N & N_1 < i \leq N \\ (\Lambda/\Omega_i)^{i-N} p_N & \forall i > N \end{cases}$$

with

$$p_0 = 1.0 - \sum_{i \geq 1} p_i$$

Thus, the expected number of jobs in the system,

$$E[N] = \sum_{i \geq 1} i p_i$$

and from Little's law, the expected response time of jobs is

$$E[D] = E[N]/\Lambda$$

Appendix D

Digital filters are divided into two classes: nonrecursive and recursive. A nonrecursive filter whose output is to be the signal estimate $\hat{x}$ is defined as

$$\hat{x} = \sum_{i=1}^{m} h(i)y(i)$$

where the $y(i)$ are the m data signals and the $h(i)$ are the coefficients. One may choose the $h(i)$ by several different methods. For instance, they may all have equal weights, i.e., $1/m$. However, in case the mean-square error of the estimated signal is to be minimized, there needs to be another method to determine the weights of the coefficients. One such optimal nonrecursive estimator is known as the scalar Wiener filter. There are several computational difficulties associated with the Wiener filter, as described in [BOZI79]. For example, m, the number of samples needs to be prespecified and in case more data becomes available, all the calculations need to be repeated. Also, it requires the inversion of an $(m * m)$ matrix. If m is large, this can take substantial computer time.

To allow updating of the estimate as more information becomes available and to save on processing costs, another scheme called the recursive filter has been developed in the literature. The problem is specified in a way that is similar to the nonrecursive case: Given successive samples $y(k) = x + v(k)$, provide a linear estimator

$$\hat{x} = \sum_{i=1}^{k} h(i)y(i)$$

such that the mean-square error $p_e = E(x - \hat{x})^2$ is minimized. From Section 7.1 in [BOZI79], we know the nonrecursive solution to this problem and state the results derived there. For k samples,

$$\hat{x} = \hat{x}(k) = \sum_{i=1}^{k} h(i)y(i),$$

where $h(i) = 1/(k+\gamma)$, with the corresponding mean-square error

$$p_e = p(k) = \frac{\sigma_v^2}{k+\gamma},$$

where $\gamma = \sigma_v^2/\sigma_x^2$ and σ_x^2 is the variance of the input signal and σ_v^2 is the variance of the noise samples. Notation $\hat{x}(k)$ refers to the kth estimate of the parameter x, i.e., the estimate after a batch of k samples have been processed. Similarly, one can interpret $p(k)$ as the mean-square error of the estimate after k samples have been processed.

The recursive equations to update the estimate of the unknown variable can be derived as shown in [BOZI79] and written as follows:

$$\begin{aligned} p(k+1) &= \frac{p(k)}{1+p(k)/\sigma_v^2} \\ \hat{x}(k+1) &= \frac{p(k+1)}{p(k)}\hat{x}(k) + \frac{p(k+1)}{\sigma_v^2}y(k+1) \end{aligned}$$

where given $p(k)$, one can derive $p(k+1)$ and $p(k+2)$ and so on. Thus, one can use equation to determine the current estimate of x, i.e., $\hat{x}(k+1)$, from the previously computed value of $\hat{x}(k)$ and the new data sample $y(k+1)$. This procedure continually generates the best linear mean-square estimator of x and provides the corresponding mean-square error, $p(k+1)$. It can be proven that $p(k) \to 0$ for k very large.

Bibliography

[AGRA82] Agrawala, A. K., S. K. Tripathi, and G. Ricart, "Adaptive Routing Using a Virtual Waiting Time Technique," *IEEE Trans. Soft. Engg.*, Vol. SE-8, pp. 76–81, 1982.

[AGRA85] Agrawal, R. and A. Ezzat, "Processor Sharing in NEST: A Network of Computer Workstations," *Proc. 1st Int'l Conf. on Computer Workstations*, Nov. 1985.

[ALLE78] Allen, A. O., *Probability, Statistics and Queueing Theory*, Academic Press, 1978.

[AYAC82] Ayache, J. M., J. P. Courtiat, and M. Diaz, "REBUS, a Fault-Tolerant Distributed System for Industrial Real-Time Control," *IEEE Trans. Computers*, Vol. C-31, pp. 637–647, 1982.

[BANN83] Bannister, J. A. and K. S. Trivedi, "Task Allocation in Fault-Tolerant Distributed Systems," *Acta Informatica*, Vol. 20, pp. 261–281, 1983.

[BOX76] Box, G. and G. M. Jenkins, *TIME SERIES ANALYSIS forecasting and control, Holden-Day series in Time Series Analysis and Digital Processing*, Holden-Day, 1976.

[BOZI79] Bozic, S. M., *Digital and Kalman Filtering*, Edward Arnold, 1979.

[BRYA81] Bryant, R. and R. A. Finkel, "A Stable distributed scheduling algorithm," *Proc 2nd Intl. Conf. Dist. Comp. Syst.*, April 1981.

[BUTT84] Butterfield, D. and G. Popek, "Network Tasking in the LOCUS Distributed UNIX System," *Proc. Summer USENIX Conf.*, , pp. 62–71, June 1984.

[BUZE74] Buzen, J. P. and P. S. Chen, *Information Processing 74*, chapter Optimal Load Balancing in Memory Hierarchies, pages 271–275, North-Holland, New York, 1974.

[CHOW77] Chow, Y. C. and W. H. Kohler, *Computer Performance*, chapter Dynamic Load Balancing in Homogeneous Two-Processor Systems, North-Holland, 1977.

[CHOW79] Chow, Y. C. and W. H. Kohler, "Models for Dynamic Load Balancing in a Heterogeneous Multiple Processor System," *IEEE Trans. Computers*, Vol. C-28, pp. 354–361, May 1979.

[CHRI81] Christensen, R., *General Description*, Volume I of *Entropy Minimax Sourcebook*, Entropy Limited, 1981.

[CHRI85] Christensen, R., "Entropy Minimax Multivariate Statistical Modeling-I: Theory," *Int. J. General Systems*, Vol. 11, pp. 231–277, 1985.

[dSeS84] de Souza e Silva, E. and M. Gerla, "Load Balancing in Distributed Systems With Multiple Classes and Site Constraints," *Performance '84*, , pp. 17–33, 1984.

[DUTT82] Dutta, A., G. Koehler, and A. Whinston, "On Optimal Allocation in a Distributed Processing Environment," *Management Science*, Vol. 28, pp. 839–853, 1982.

[EAGE86a] Eager, D. L., E. D. Lazowska, and J. Zahorjan, "Adaptive Load Sharing in Homogeneous Distributed Systems," *IEEE Trans. Soft. Engg.*, Vol. SE-12, No. 5, pp. 662–675, May 1986.

[EAGE86b] Eager, D. L., E. D. Lazowska, and J. Zahorjan, "A Comparison of Receiver-Initiated and Sender-Initiated Adaptive Load Sharing," *Performance Evaluation*, Vol. 6, pp. 53–68, March 1986.

[EFE82] Efe, K., "Heuristic Models of Task Assignment Scheduling in Distributed Systems," *Computer*, Vol. 15, pp. 50–56, June 1982.

[EVAN67] Evans, R., "Geometric Distribution in some Two-Dimensional Queuing Systems," *Operations Research*, Vol. 15, pp. 830–46, 1967.

[FISH78] Fishman, G., *Principles of Discrete Event Simulation, Wiley-Interscience*, John Wiley and Sons, 1978.

[GALL68] Gallager, R. G., *Information Theory and Reliable Communication*, John Wiley and Sons, Inc., 1968.

[GALL77] Gallager, R. G., "A Minimum Delay Routing Algorithm Using Distributed Computation," *IEEE Trans. Comm.*, Vol. COM-25, pp. 73–85, Jan. 1977.

[GOLD53] Goldman, S., *Information Theory*, Prentice-Hall, Inc, New York, 1953.

[HWAN84] Hwang, K. and F. Briggs, *Computer Architecture and Parallel Processing*, Mcgraw-Hill, 1984.

[INDU86] Indurkhya, B., H. Stone, and L. Xi-Cheng, "Optimal Partitioning of Randomly Generated Distributed Programs," *IEEE Trans. Soft. Engg.*, Vol. SE-12, pp. 483–495, 1986.

[KIRR84] Kirrman, H. and F. Kaufmann, "Poolpo: A Pool of Processors for Process Control Applications," *IEEE Trans. Computers*, Vol. C-33, pp. 869–878, Oct. 1984.

[KLEI76a] Kleinrock, L., *Computer Application*, Volume II of *Queueing Systems*, John Wiley and Sons, 1976.

[KLEI76b] Kleinrock, L., *Theory*, Volume I of *Queueing Systems*, John Wiley and Sons, 1976.

[KOBA78] Kobayashi, H., *Modeling and Analysis*, Addison-Wesley, 1978.

[KRAT80] Kratzer, A. and D. Hammerstrom, "A Study of Load Levelling," *IEEE COMPCON Fall 80*, , pp. 647–654, Sept. 1980.

[KURO86a] Kurose, J. F. and S. Singh, "A Distributed Algorithm for Optimal Static Load Balancing in Distributed Computer Systems," *Proc. IEEE COMPCON '86*, 1986.

[KURO86b] Kurose, J. F., S. Singh, and R. Chipalkatti, "A Study of Quasi-Dynamic Load Sharing in Soft Real-Time Distributed Computer Systems," *Proc. Real-Time Syst. Symp*, Dec. 1986.

[LANT85] Lantz, K. A., W. I. Nowicki, and M. M. Theimer, "An Empirical Study of Distributed Application Performance," *IEEE Trans. Soft. Engg.*, Vol. SE-11, October 1985.

[LATO81] Latouche, G., "Algorithmic Analysis of a Multiprogramming-Multiprocessor Computer System," *J. ACM*, Vol. 28, October 1981.

[LAVE83] Lavenberg, S., *Computer Performance Modeling Handbook*, Academic Press, 1983.

[LEE87] Lee, K. J., *Load Balancing in Distributed Computer Systems*, PhD thesis, ECE Dept., University of Massachusetts, February 1987.

[LIVN82] Livny, M. and M. Melman, "Load Balancing in Homogeneous Broadcast Distributed Systems," *Performance Evaluation Review*, Vol. 11, No. 1, pp. 47–55, 1982.

[MIRC86] Mirchandaney, R. and J. A. Stankovic, "Using Stochastic Learning Automata for Job Scheduling in Distributed Systems," *International Journal of Parallel and Distributed Computing*, Dec 1986.

[MIRC87a] Mirchandaney, R., L. Sha, and J. A. Stankovic, "Load Sharing in the Presence of Non-Negligible Delays," 1987. in preparation.

[MIRC87b] Mirchandaney, R., D. Towsley, and J. A. Stankovic, "Analysis of the Effects of Delays on Load Sharing," *IEEE Trans. Computers*, 1987. submitted for review.

[NEED82] Needham, R. and A. Herbert, *The Cambridge Distributed Computing System*, Addison-Wesley, 1982.

[NEUT81] Neuts, M. F., *Matrix-Geometric solutions in Stochastic Models: An Algorithmic Approach, Mathematical Sciences*, Johns Hopkins University Press, 1981.

[NI85a] Ni, L. M. and K. Hwang, "Optimal Load Balancing in a Multiple Processor System with Many Job Classes," *IEEE Trans. Soft. Engg.*, Vol. SE-11, pp. 491–496, 1985.

[NI85b] Ni, L. M., C. Xu, and T. Gendreau, "A Distributed Drafting Algorithm for Load Balancing," *IEEE Trans. Soft. Engg.*, Vol. SE-11, pp. 1153–1161, 1985.

[NI86] Ni, L. M. and K. Hwang, "Correction To Optimal Load Balancing in a Multiple Processor System with Many Job Classes," *IEEE Trans. Soft. Engg.*, Vol. SE-12, p. 500, 1986.

[POWE83] Powell, M. and B. Miller, "Process Migration in DEMOS/MP," *Proc. 9th Symp. on Oper. Sys. Prin., ACM*, , pp. 110–119, 1983.

[SAUE81] Sauer, C. and K. M. Chandy, *Computer Systems and Performance Modeling*, Prentice-Hall, 1981.

[STAN84] Stankovic, J. A., "Simulations of Three Adaptive Decentralized Controlled, Job Scheduling Algorithms," *Computer Networks*, Vol. 8, No. 3, pp. 199–217, June 1984.

[STAN85a] Stankovic, J. A., "Bayesian Decision Theory and Its Application to Decentralized Control of Task Scheduling," *IEEE Trans. Computers*, Vol. C-34, No. 2, pp. 117–130, February 1985.

[STAN85b] Stankovic, J. A., K. Ramamritham, and S. Cheng, "Evaluation of a Flexible Task Scheduling Algorithm for Distributed Hard Real-Time Systems," *IEEE Trans. Computers*, Vol. c-34, No. 12, Dec 1985.

[STON78a] Stone, H., "Critical Load Factors in Two Processor Distributed Systems," *IEEE Trans. Soft. Engg.*, Vol. SE-4, No. 3, May 1978.

[STON78b] Stone, H., "Multiprocessor Scheduling with the Aid of Network Flow algorithms," *IEEE Trans. Soft. Engg.*, Vol. SE-3, No. 1, May 1978.

[TANE81] Tanenbaum, A., *Computer Networks*, Prentice-Hall, 1981.

[TANT84] Tantawi, A. and D. Towsley, "A General Model for Optimal Load Balancing in Star Network Configuration," *Proc. of Performance '84*, , pp. 277–291, 1984.

[TANT85] Tantawi, A. and D. Towsley, "Optimal Static Load Balancing in Distributed Computer Systems," *J. ACM*, Vol. 32, pp. 445–465, Apr. 1985.

[THEI85] Theimer, M., K. Lantz, and D. Cheriton, "Preemptable Remote Execution Facilities for the V-System," *Proceedings of the 10th Symposium on Operating System Principles*, December 1985.

[TOWS86] Towsley, D., "The Allocation of Programs Containing Loops and Branches on a Multiple Processor System," *IEEE Trans. Soft. Engg.*, Vol. SE-12, pp. 1018–1024, October 1986.

[TOWS87] Towsley, D. and R. Mirchandaney, "The Effects of Communication Delays on the Performance of Load Balancing Policies in Distributed Systems," *Proc. 2nd. Intl. Workshop on Appl. Math. and Perf./Rel. Models in Computer/Communications*, May 1987.

[VANT84] VanTilborg, A. M. and L. Wittie, "Wave Scheduling - Decentralized Scheduling of Task Forces in Multicomputers," *IEEE Trans. Computers*, Vol. C-33, pp. 835–844, Sept. 1984.

[Vax85] *VAX Architecture Handbook*, 1985.

[WANG85] Wang, Y. and R. Morris, "Load Sharing in Distributed Systems," *IEEE Trans. Computers*, Vol. C-34, March 1985.

[ZHAO85] Zhao, W. and K. Ramamritham, "Distributed Scheduling Using Bidding and Focussed Addressing," *Proceedings of the Symposium on Real-Time Systems*, December 1985.