Analysis of the Effects of Delays on Load Sharing

Ravi Mirchandaney
Electrical and Computer Engineering Department
University of Massachusetts

Don Towsley and John A. Stankovic
Department of Computer and Information Science
University of Massachusetts

COINS Technical Report 87-100
February, 1987

Analysis of the Effects of Delays on Load Sharing *

Ravi Mirchandaney
ECE Dept., U. Massachusetts
Amherst MA. 01003

Don Towsley
COINS Dept., U. Massachusetts
Amherst MA. 01003

John A. Stankovic
COINS Dept., U. Massachusetts
Amherst MA. 01003

February 1987

Abstract

In this paper, 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 tasks 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 Because of these delays, the state information gathered by the load sharing algorithms is out of date by the time the load sharing decisions are taken.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 paper 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 task arrival process at each node is Poisson and the service times and task 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.

*This work was supported, in part, by the National Science Foundation under grant ECS-8406402 and by RADC under contract RI-44896X

1 . Introduction

Distributed computer systems possess many potentially attractive features. Some of these are the capability to share processing of tasks in the event of overloads and failures, reliability through replication, and modularity. This study focuses on the issue of sharing computation power between nodes of a distributed system in response to imbalances in loads.

It will be assumed that tasks arrive at the nodes in a random fashion. Thus, situations can develop whereby some of the nodes are excessively busy while others are idle at the same time[LIVN82]. This kind of situation is detrimental to performance because tasks at the busy nodes experience very high waiting times, while the less busy nodes have idle cyles at the same time. The function of load-sharing is to minimize the occurrence of such scenarios by moving tasks from overloaded nodes to less busy ones.

Distributed load balancing has been an active area of research for some time. For example, Stone[STON78b,STON78a], Bokhari[BOKH79] and Towsley[TOWS86] examined static algorithms that utilized information only about the average behavior of tasks in deciding their assignments. Tantawi and Towsley[TANT85] investigated an optimal probabilistic assignment scheme. Silva and Gerla[dSeS84] determine an optimal load sharing strategy under the assumption that the nodes and the communication network can be modelled as product form queueing networks. Recently, Lee[LEE87] studied the effects of task transfer delays on simple algorithms that do not utilize any remote state information.

Eager et.al.[EAGE86] evaluated three simple load sharing schemes. They assume that the entire overhead due to load sharing is transferred onto the CPU and is modelled as an increased load on the same. Further, the nodes are assumed to be part of a local area network connected by a high bandwidth medium. Thus, there are no delays in transferring tasks and remote state information is always perfectly accurate.

While these works provided significant insight into various aspects of load sharing, the problem of communication delays and out of date state information and its impact on load sharing has not been investigated in any great detail. In this paper, we focus on the effect of communication delays upon the performance of simple load sharing algorithms. We feel that this problem is interesting in that there exist a sufficient number of system architectures that will generate significant delays in

task transfers. For instance, Theimer et.al.[THEI85], report their concerns with task transfer delays. Furthermore, they have acknowledged that if the files used by a task were to be transferred (as they might have to be if the nodes were disk-based), the effect of delays would become even more prominent (the V-System is currently comprised of diskless workstations). Also, the question of how to deal with out of date state information has been one of the many interesting developments in designing algorithms for distributed systems as investigated in Stankovic[STAN85].

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 traffic intensities over which state information is worth gathering and useful load sharing can be performed.

The remainder of this paper is organized as follows: In Section 2, we provide a brief description of the system architecture and the load sharing algorithms. Section 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 4, we describe the important results of this research and we summarize our work in Section 5. Finally, Appendices A and B describe the internals of the matrices involved with the solution of the Markov processes.

2 System Architecture and Load Sharing Algorithms

2.1 System Architecture and Motivation

Processing and transmission of communication messages for state updates (probes) and for tasks 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 tasks 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 task 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 task 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 tasks. 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 tasks and probes will be quite different. The physical transfer of a task 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 tasks vs. probes. Consequently, it appears that the delays incurred by tasks 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 task transfers.

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 task arrives at the node. If the number of tasks at this node (including the task currently being executed) is greater than $T + 1$, an attempt is made to transfer the newly arrived task 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 task. A probed node responds positively if the number of tasks it possesses is less than $T + 1$ and it is not already waiting for some other remote task. If more than one node responds positively, the sender node transfers the task 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 task completes at a node and the total number of tasks at the node is less than $T + 1$ and the node is not already waiting for a remote task to arrive. If so, the node probes a subset of size L, remote nodes at random to try and acquire a remote task. Only nodes that possess more than $T + 1$ tasks, (including the currently executing one) can respond positively. If more than one node can transfer a task, the probing node chooses one of these at random from which it requests a task.

- **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 task, it attempts to transfer a task and if it drops below $T + 1$ upon a task completion, it attempts to acquire a remote task.

In all the algorithms described above, it is assumed that probing takes zero time. This is based on the initial assumption that probes are much smaller entities than are tasks. Thus, the overhead for processing a probe at the BIU is much smaller than for tasks. Further, probes occupy much less of the communication bandwidth than tasks. Thus, the entire delay is assumed to occur during actual task transfer. Furthermore, we have seen in separate studies (not described in this paper) that as long as the ratio of task transfer times to probe transfer times is sufficiently large (≥ 50), the system essentially behaves as if the probes actually take zero time. We are currently investigating this phenomenon in greater detail.

3 Mathematical Analysis

It is assumed that the task arrival process at each node is Poisson, with parameter λ. Also, the service times and task transfer times are assumed to be exponentially distributed, with means $1/\mu$ and $1/\gamma$, respectively. The task transfer time includes the time between the initiation of a transfer from a node and the successful reception of the task 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. Tasks are assumed to be executed on a First-Come-First-Served (FCFS) basis at each node.

Let $N_t^{(i)}$ be the number of tasks 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 task 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 [EAGE86]. The interactions between the nodes which result in task 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 later 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.

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

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

will be displayed as

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

Figure 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 tasks 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 1. In 3-Space, the top and the bottom rows would be merged together.

We define

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

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

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

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

$$Q_S = \begin{Vmatrix} & B_{01} & \ldots & B_{01} & B_{01} & A_0 & A_0 & \ldots \\ B_{00} & B_{11} & \ldots & B_{11} & B_{21} & A_1 & A_1 & \ldots \\ B_{10} & B_{10} & \ldots & B_{10} & A_2 & A_2 & A_2 & \ldots \end{Vmatrix}$$

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.

In the subsequent discusssion, h is the probability of failure in finding an assignment for a spare task 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 task. Also, q is the probability of failure in finding a remote task 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 task, it sends out reverse probes in order to get a remote task. A successful transition is represented by μq and an unsuccessful set of reverse probes is represented by the transition μq.

Thus, on the completion of a task when the node goes below $T+1$, it sends out reverse probes, if it is not already waiting for a remote task 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 task 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 tasks in response

to nodes that asked it for tasks 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

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

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 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

$$\begin{Vmatrix} & B_{01} & \dots & B_{01} & B_{01} \\ B_{00} & B_{11} & \dots & B_{11} & B_{21} + RA_2 \\ B_{10} & B_{10} & \dots & B_{10} & \end{Vmatrix}$$

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

Intuitively, the stability condition means that the rate of processing tasks (including the ones that are sent out of this node) is greater than the total arrival rate of tasks into this node. Thus, on the average, whenever there are more than $T+1$ tasks 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

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

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

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

Thus,

$$\sum_{i=T+1} p_i = p_{T+1}(I - R)^{-1}$$

Also,

$$\sum_{i=0}^{T} p_i + p_{T+1}(I - R)^{-1}]e = 1$$

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

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

where Total-Flow-In is the flow into a node of remote tasks 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.

3.1 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 tasks, as a result of forward probes made by this node.
- $FFRI$: Flow rate in of tasks, as a result of forward probes made to this node by other nodes.

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

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

Iteration Procedure

1. Let $\iota = 0$; choose values for $h^{(0)}, q^{(0)}, \mu'^{(0)}, \alpha^{(0)}, FFRO^{(0)}, RFRO^{(0)}$
2. Determine $Q^{(\iota)}$ from $h^{(\iota)}, q^{(\iota)}, \mu'^{(\iota)}, \alpha^{(\iota)}$
3. Determine $R^{(\iota)}$
4. Solve the linear system corresponding to the boundary conditions
5. Determine $FFRO^{(\iota+1)}$ and $RFRO^{(\iota+1)}$ from the model solution
6. If $ABS(FFRO^{(\iota+1)} - FFRO^{(\iota)}) \leq \epsilon$ and $ABS(RFRO^{(\iota+1)} - RFRO^{(\iota)}) \leq \epsilon$, where ϵ is an arbitrary small number, stop, else
7. Let $\iota = \iota - 1$. 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 symmetric nature of the algorithm

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

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

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

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, $x = 1 - x$ and $h = 1 - h$.

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 task. 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 task, 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.

3.2 Forward and Reverse

As mentioned in section 1, 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 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 tasks 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:

$$Q_F = \begin{Vmatrix} & B_{01} & \dots & B_{01} & B_{01} & A_0 & A_0 & \dots \\ B_{00} & B_{11} & \dots & B_{11} & A_1 & A_1 & A_1 & \dots \\ A_2 & A_2 & \dots & A_2 & A_2 & A_2 & A_2 & \dots \end{Vmatrix}$$

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

Figure 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 tasks 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:

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

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

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 tasks 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. The model is solved iteratively in both 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 B.

In both of 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.

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 tasks 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 task. However, the node does not send any probes. Instead, it picks a destination node at random and transfers a task to this node. The key performance metric for comparison is the mean response time of tasks.

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 task 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 2 that the decomposition used in this paper 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 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 ± 0.015 units about the sample mean. The only exception to this is at $\rho = 0.9$, when the confidence interval is about ± 0.165 units 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%, decreases 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 traffic intensities, Forward probing is likely to perform best, while at high traffic intensities, 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 [MIRC87] 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 5 shows the performance curves of the algorithms for $C = 0.1S$ and $T = 0$. 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 task. 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 6 shows the performance curves of the algorithms for $C = 2S$ and $(T = 2)$. From Figure 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.
- 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 if 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 7, 8 and 9 show the response times for the Symmetric algorithm tested over a wide range of communication delays and thresholds, for the traffic intensities of 0.5, 0.7 and 0.9. It can be seen from Figure 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 $M/M/K$ value is approximately 1.04 units. Further, the NLB resonse 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 task 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 task to it. Thus, in some sense, $T = 0$ ensures that all task transfers are useful in that a remote task arrives at the node soon after it becomes idle.

For moderate delays ($C = S$, Figure 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 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 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 impovement 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 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 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 traffic intensity, 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 traffic intensities 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 traffic intensities (≤ 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 traffic intensity 0.9. We postulate that at higher loads, this effect will be even more prominent.

Optimal thresholds

Figure 11 indicates the variation of the *optimal* thresholds corresponding to the *optimal* response times indicated in Figure 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 traffic intensities. 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.

5 Summary and Conclusions

This study was concerned with the performance analysis of simple load sharing algorithms in the presence of significant task 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 ($\rho \geq 0.9$), susbtantial 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 over-

heads. 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 task 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 $= 100S$.

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.

Appendix B

In this appendix, we provide closed form representations for the matrices in the case of the Forward and Reverse probing algorithms.

Forward

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

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

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

$$x = \sum_{i \leq T} \mathbf{p}_i [01]^T + \sum_{i > T} \mathbf{p}_i e$$

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

$$h = x^{L_p}$$

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

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

$$A_2 = \mu I_2$$

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 task, 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

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

Acknowledgements The authors would like to thank the referees for their valuable comments. In particular, one referee pointed out the flaw in our initial analysis pertaining to the reducibility of A and suggested the correction that appears in this paper.

References

[BOKH79] Bokhari, S., "Dual Processor Scheduling With Dynamic Reassignment," *IEEE Trans. Soft. Engg.*, Vol. SE-5, No. 4, July 1979.

[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.

[EAGE86] Eager, D., E. 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.

[LATO81] Latouche, G., "Algorithmic Analysis of a Multiprogramming-Multiprocessor Computer System," *J. ACM*, Vol. 28, October 1981.

[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.

[MIRC87] Mirchandaney, R., L. Sha, and J. A. Stankovic, "Load Sharing in the Presence of Non-Negligible Delays," 1987. in preparation.

[NEUT81] Neuts, M. F., *Matrix-Geometric solutions in Stochastic Models: An Algorithmic Approach, Mathematical Sciences*, Johns Hopkins University Press, 1981.

[STAN85] Stankovic, J., "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.

[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.

[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.

[WANG85] Wang, Y. and R. Morris, "Load Sharing in Distributed Systems," *IEEE Trans. Computers*, Vol. C-34, March 1985.

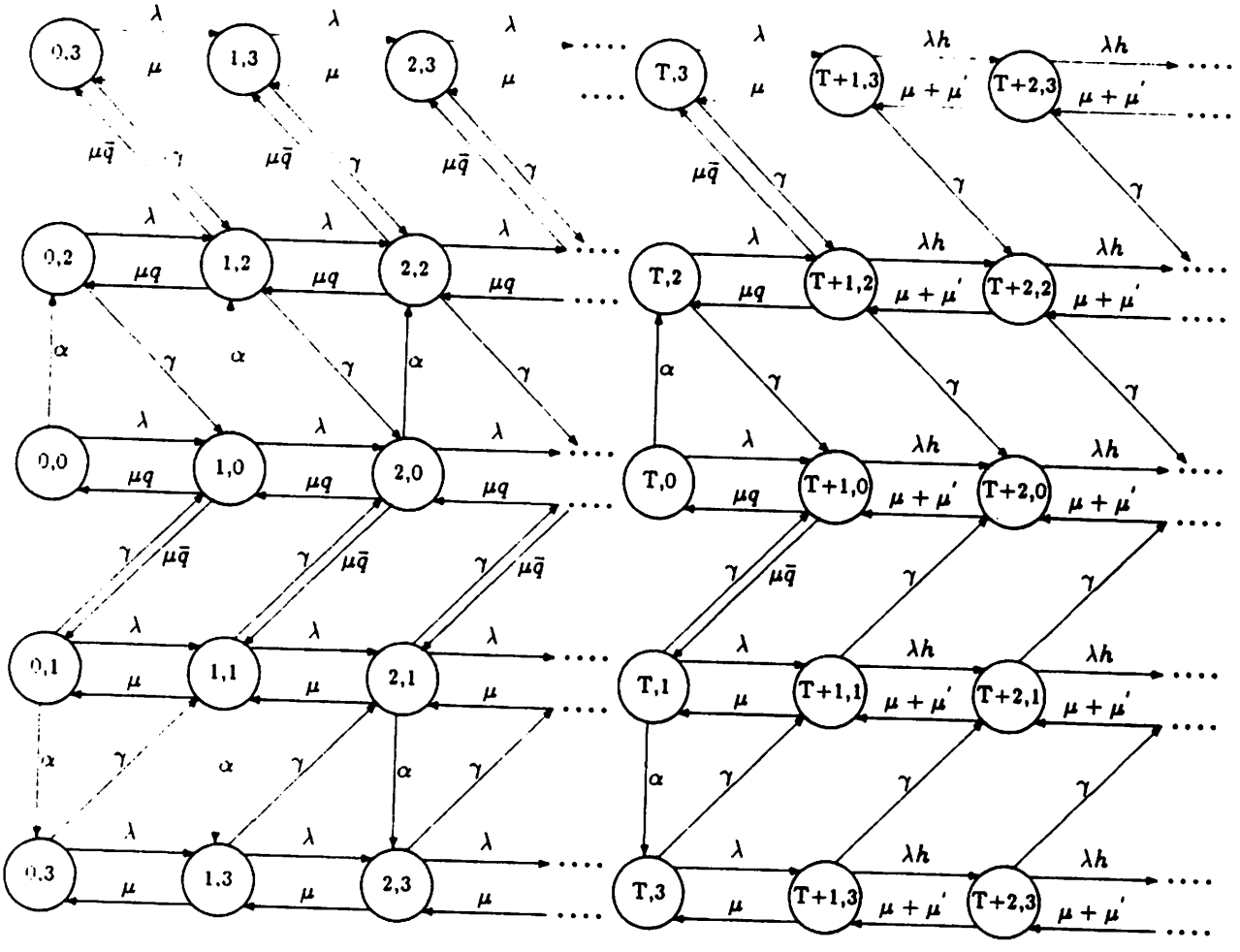

Figure 1. Symmetric Probing

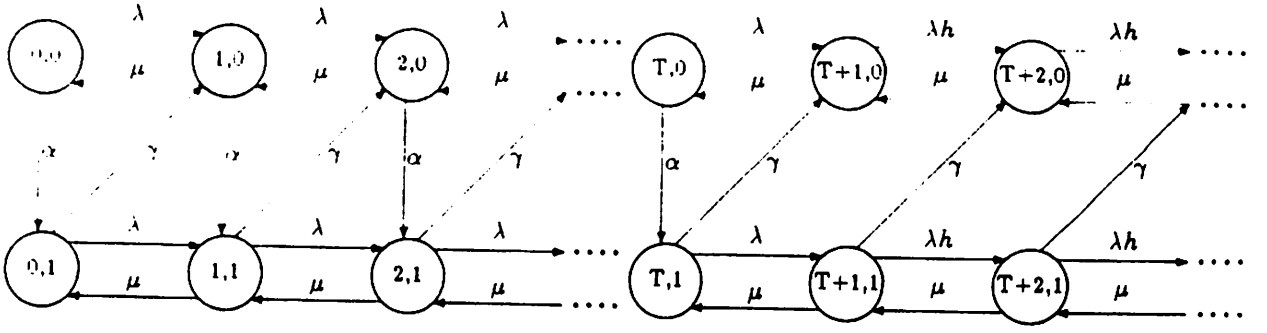

Figure 2. Forward Probing

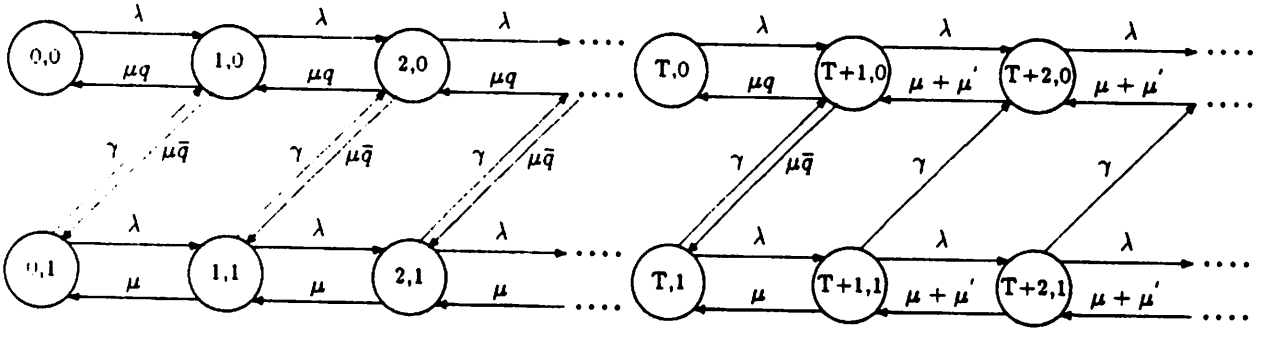

Figure 3. Reverse Probing

Figure 4. Validation with Simulations

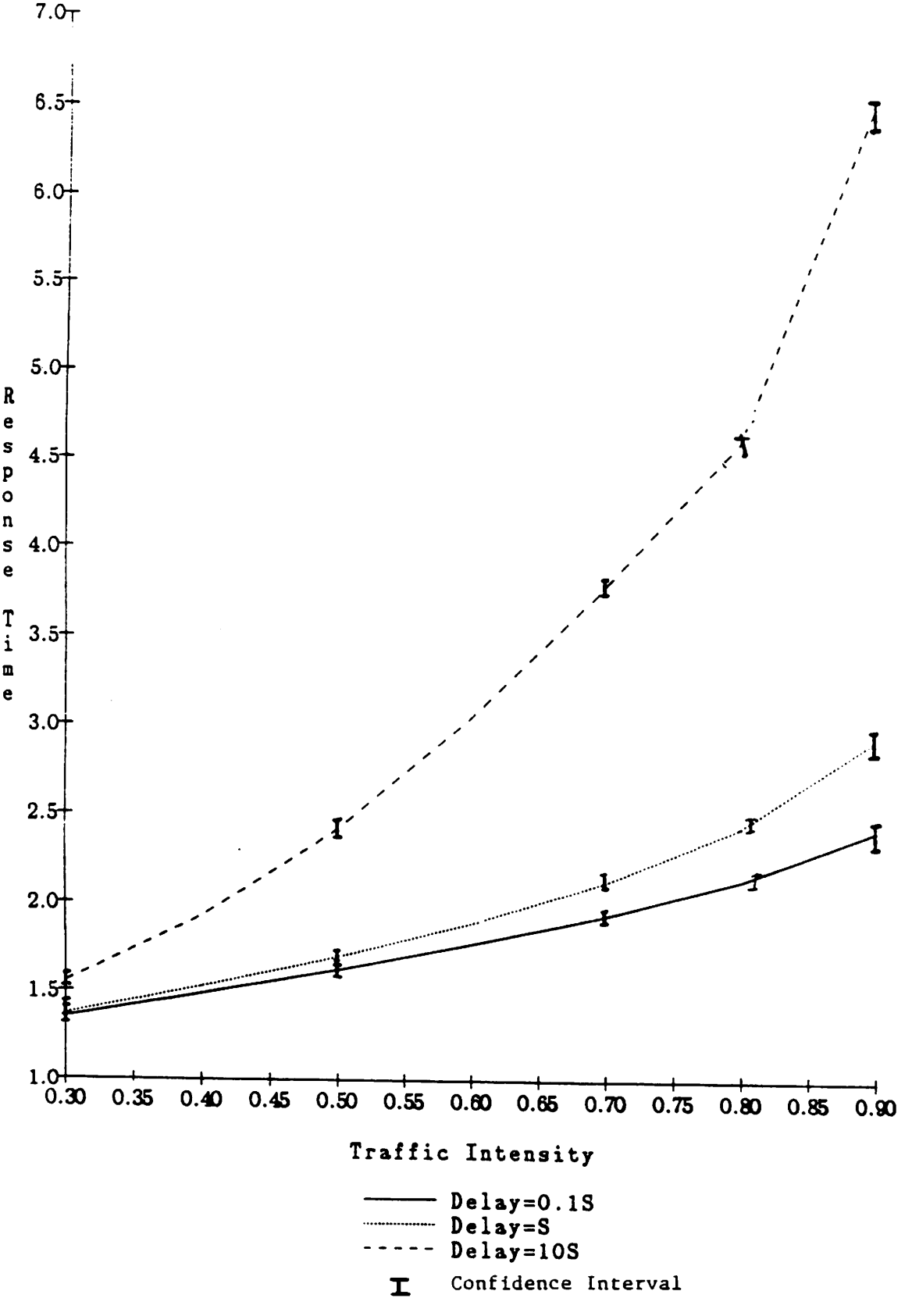

Figure 5. Comparison of Algorithms

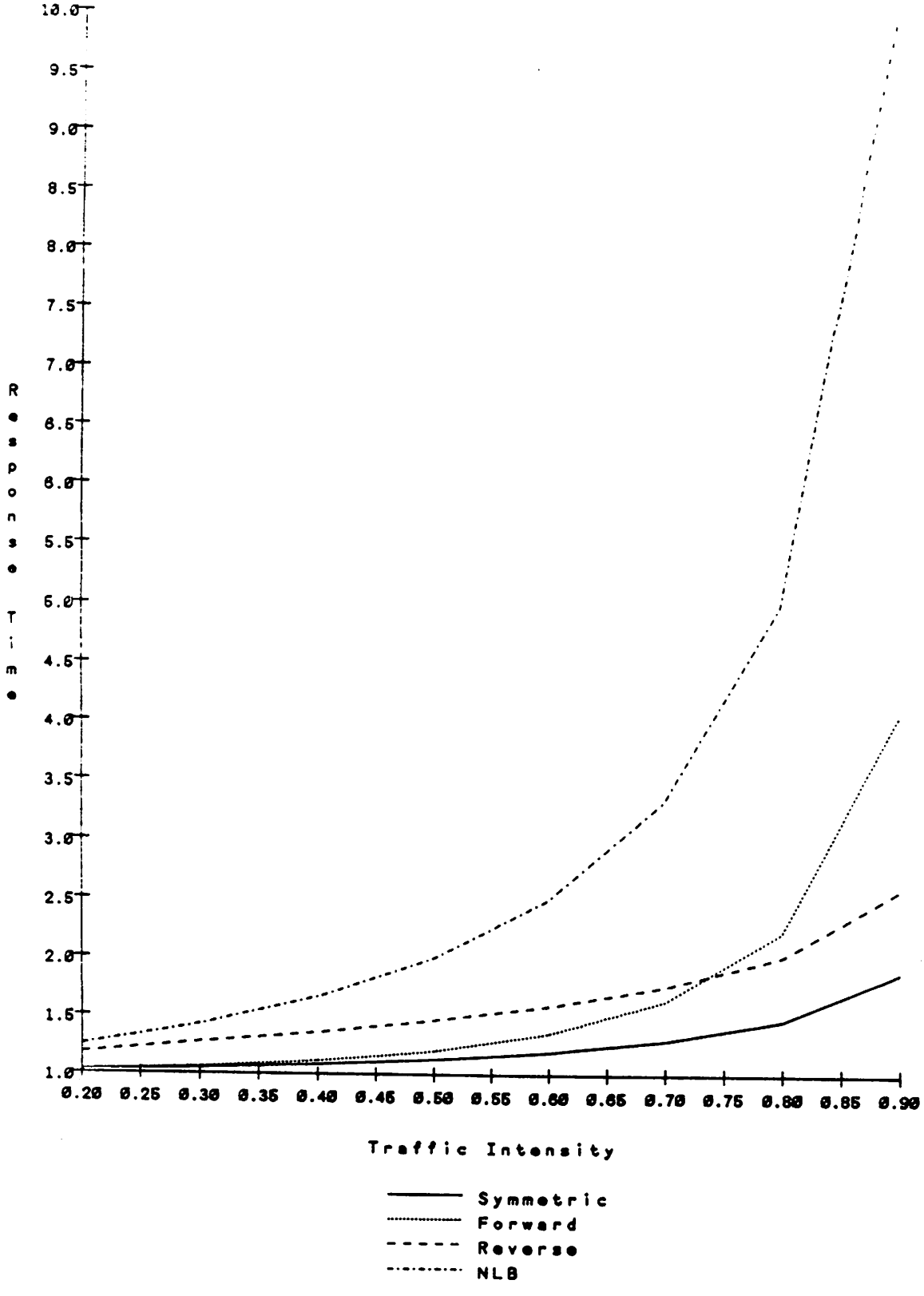

Figure 8. Comparison of Algorithms

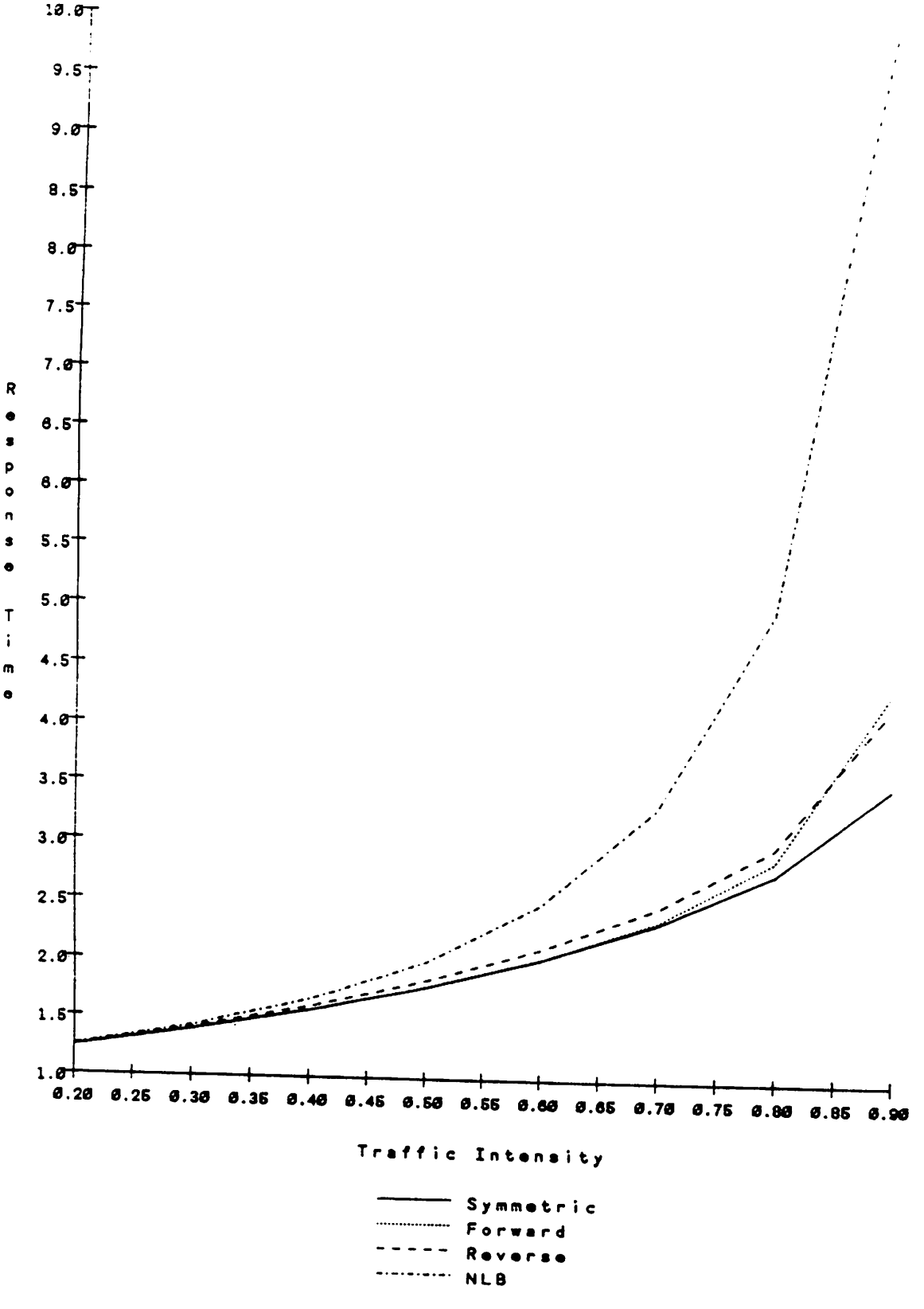

Figure 7. Variation of Threshold (Delay=0.1S)

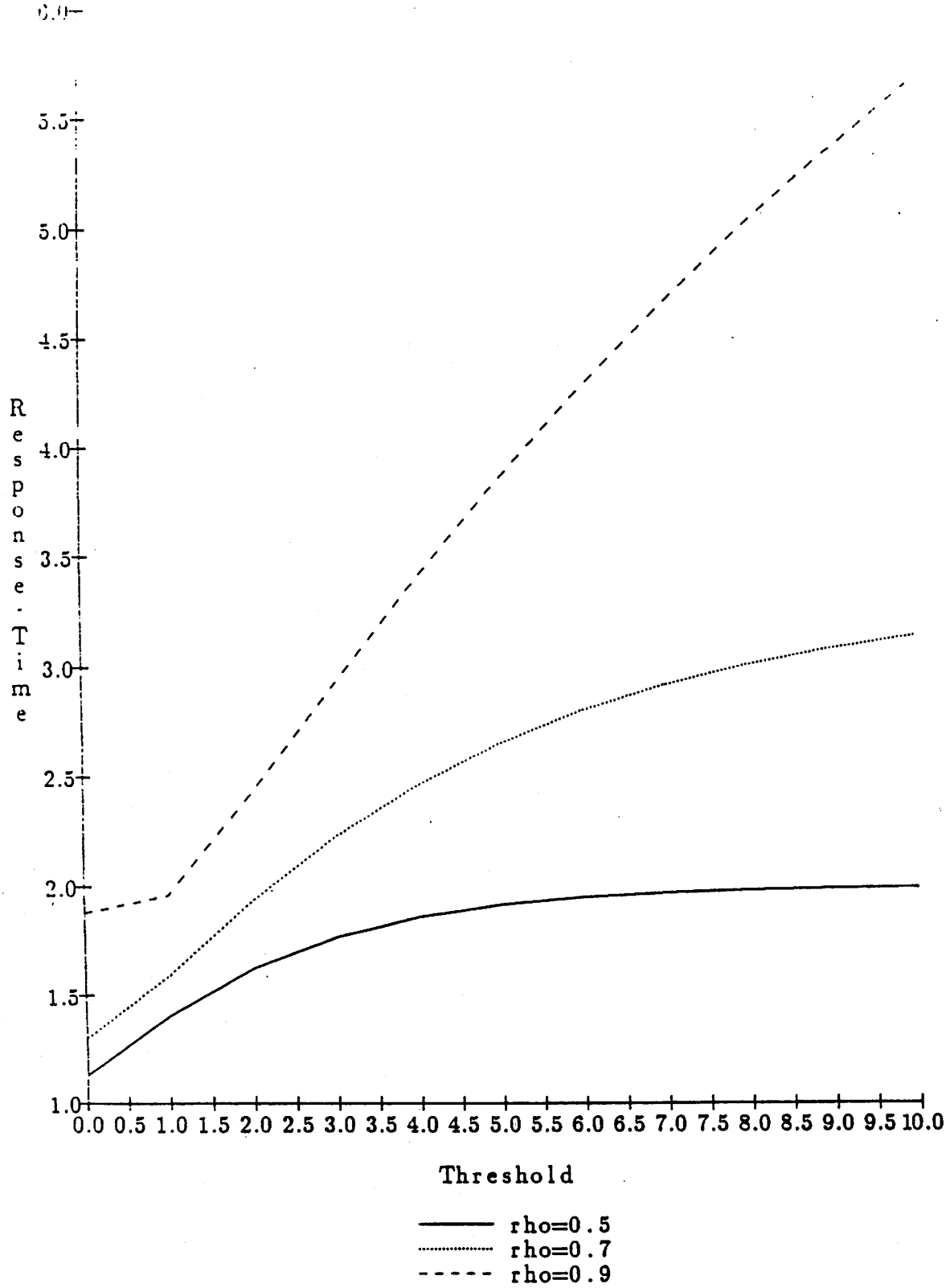

Figure-8.-Variation-of-Threshold-(Delay=S)

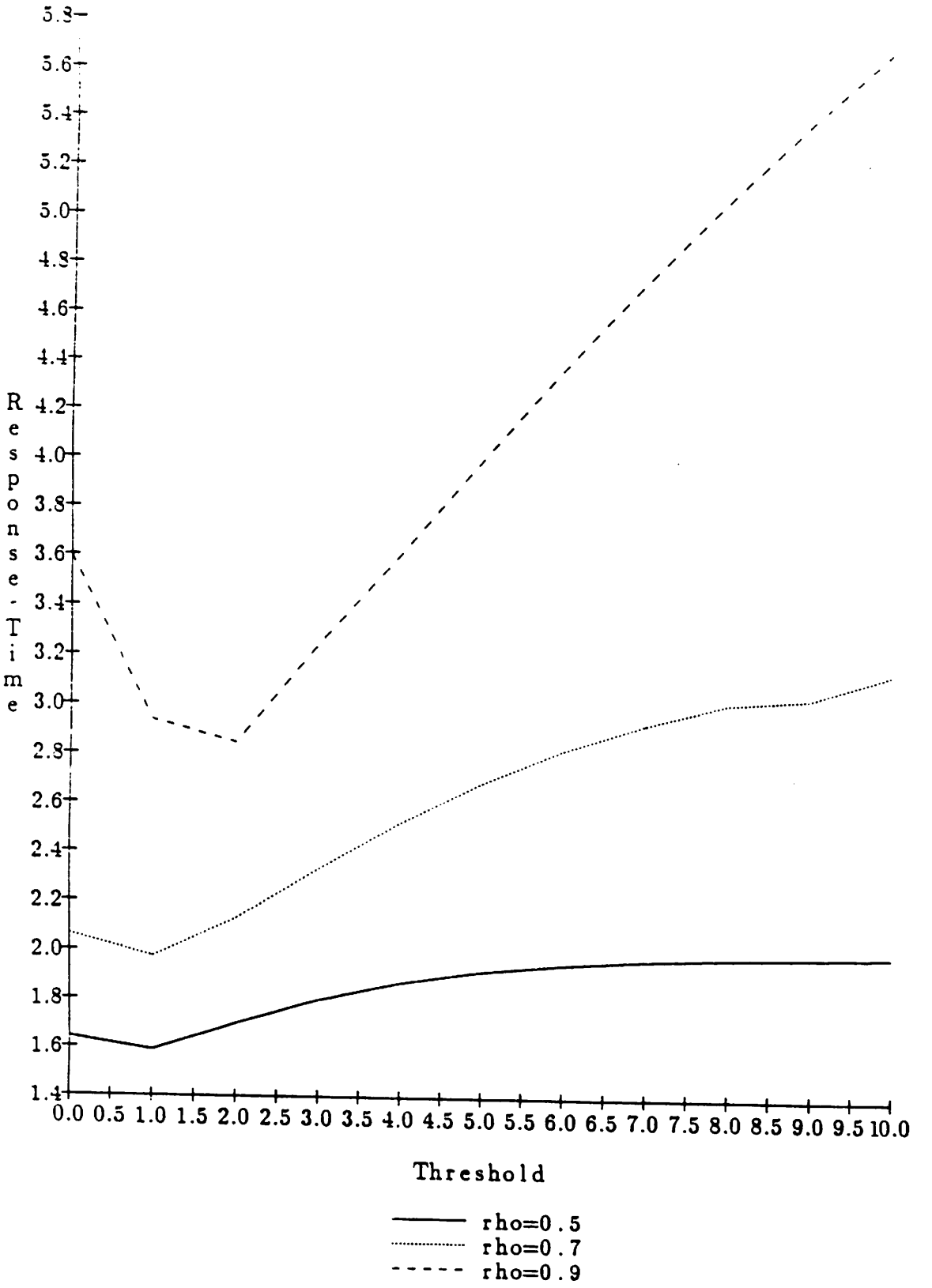

Figure-9.-Variation-of-Threshold-(Delay=10S)

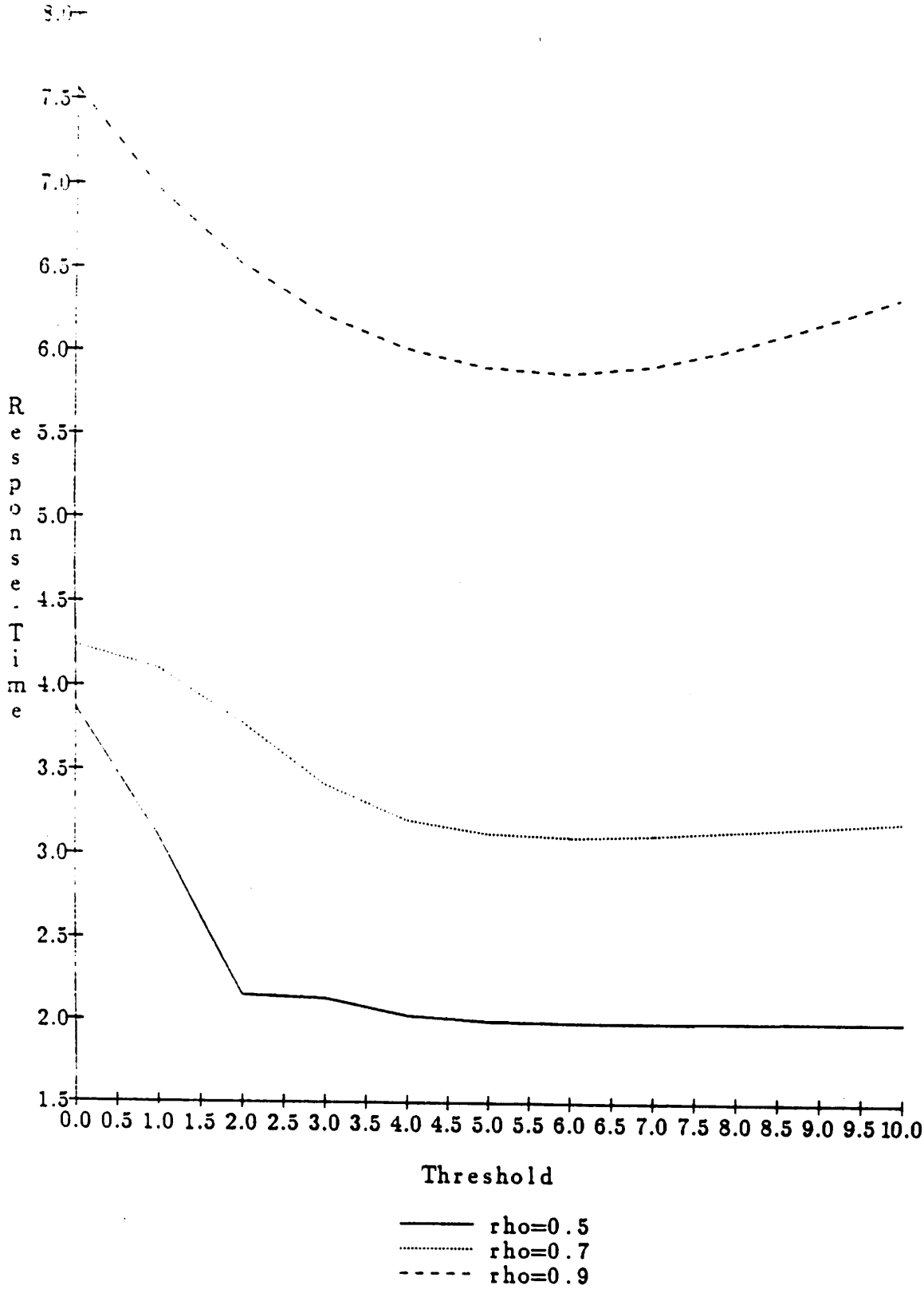

Figure 10. Optimal Normalized Response Times

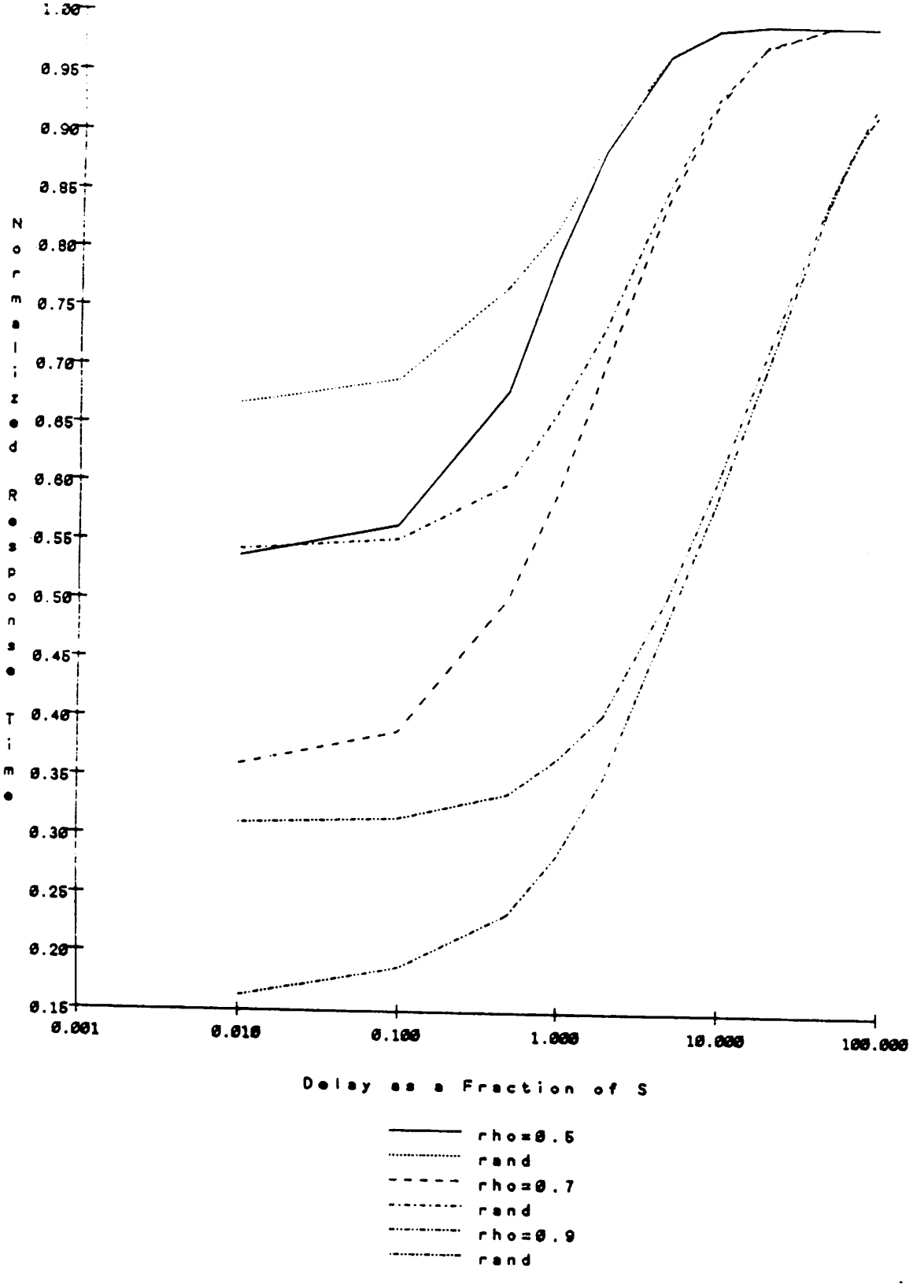

Figure 11. Variation of Thresholds