The Study of Load Balancing Algorithms for Decentralized Distributed Processing Systems

Thesis for the Degree of

Doctor of Philosophy

by

Miron Livny

Submitted to the

Sicientific Council of the

Weizmann Institute of Science

Rehovot, Israel

August, 1983

ACKNOWLEDGEMENT

I would like to express my warm gratitude to Mike Melman for the time and effort he has devoted to me and to my work.

I also thank the system operators for their dedicated assistance, and all the students who were willing to be the guinea pigs in the use of the DISS language. Their relentless aid in debugging it and their constructive comments were invaluable to the process of its formulation.

Finally, I am grateful to my wife, Efrat, who has enabled me to make time and space for this research and has been my guide in the maze of the English language.

Contents

Plates

Abstract

The multiplicity and autonomy of resources in fully distributed processing systems make task migration an attractive method for enhancing the response time of these systems, However, the communication delays and processing overheads associated with the migration of a task raises doubts as to the capability of load balancing methods to improve the performance of distributed systems.

This thesis investigates the problem of load balancing in distributed systems. A comparative performance study of several load balancing algorithms is presented. The methodology used in this study is based on the idea that the load balancing problem, like many other problems related to distributed systems, is a two dimensional one. The first dimension captures parameters that define the algorithm itself, while the second represents the characteristics of the distributed system. An understanding of the interdependence between the algorithm performance and the system's attributes is essential for acquiring an insight into the load balancing process. A number of new load balancing algorithms for both broadcast and point-to-point systems are presented and analyzed. Performance models of the various algorithms and systems are defined and solved. It is demonstrated that even when the communication delays and processing overheads are non-trivial, load balancing can significantly improve the response time of the system.

Different approaches are used for defining and solving the performance models of the algorithms. Depending on the complexity of the model and the level of detail required, analysis or discrete and continuous simulation are used. A method for modeling and simulating distributed systems is presented and used for deriving performance measures. The models and simulation programs built according to this methodology reflect the loose coupling and autonomy of the elements of the system. Consequently the models are endowed with the modularity of the distributed system

Chapter 1

Introduction

§1.1 Motivation

What would have been your reaction if while joining the end of a seemingly endless line of customers at the banks soothsayer would have wispered in your ear: *"Be aware of the idle tellers at the brunch on the next block !"*? Would yon have decided to run across the block hoping that you are the only one that heard the wisper, or would you have used some kind of reasoning to justify a decision to stay. Whatever your decision would have been it might have saved you time. By becoming aware of the state of the other branch you were faced with the problem of load balancing in a *multi-resource* system. As a user of such a system you have realized that while you were waiting for a resource at one location a resource which belongs to the same system but is located at a different place was available. By selecting the 'right' resource to wait for (you can later change your mind) the amount of time that you would have to wait could have been reduced considerably. However, due to the stochastic properties of most users and systems it seems that one has to be a soothsayer in order to know who is the 'right' server.

Various goals may motivate the construction of multi-resource systems. One of the main motivations for such systems is the need for *resource sharing.* This need has always existed as far as processing systems were concerned. The great progress in the field of computer networks in the last decade made multi-resource processing a reality. The primary goal of the projects in which the first computer networks were designed was to developed means by which a large and widely spread community can share hardware and

software resources [Lawr70]. The computer to computer interconnections and communication protocols that were developed gave the user the ability to access resources that they could not use before since these resources were not part of the users' local environment. By doing so these networks gave an answer to the ***permanent resource availability*** problem. However users of processing systems face an ***instantaneous resource availability*** problem whenever a local resource is not accessible at a given instance due to resource contention. In such a case the user may be willing to use any non-local resource although a resource with similar or even superior properties is part of his local system. The need for resource sharing under such circumstances is motivated by the desire to obtain a better response time.

When a given resource is permanently **not** available at the local system the selection of the non-local resource to be used can be carried out by the user. But in the case where resources are selected according to their instantaneous availability the assignment of resources has to be executed by the system. Due to the frequent changes in.the state of the resources and the system load distribution, the binding between users and resources has to be a dynamic process. By migrating tasks from one location to the other according to the instantaneous system load the assignment algorithm may reduce the response time of the multi-resource system. The realization of the potentiality of the task migration process to enhance the performance of such systems motivated this study of load balancing algorithms for Distributed Processing Systems (DPSs). This study attempts to answer the questions of ***how , under what conditions*** and ***to what eztent*** the expected queueing time of a user of such a system can be reduced by means of load balancing.

§1.2 Distributed Systems

The extensive experience that has been accumulated in the operation, maintenance and upgrading of centralized computer systems revealed the disadvantage of this type of computer organizations. An analysis of these drawbacks led to the development of the ideas that by distributing the resources and control of a processing system some of them may be eliminated. A great many advantages are claimed for distributed systems [Ensl78], some of which are listed in Table **1.1.** The attractivity of distributed systems brought many scientists and vendors to add the title "***distributed***" to any system with more than one processor. As a result of this the term "***Distributed processing***" was left devoid of any substantive meaning.

High Availability and Reliability
High System Performance
Ease of Modular and Incremental Growth
Automatic Load and Resource Sharing
Good Response to Temporary Overloads
Easy Expansion in Capacity and/or Function

Table 1.1.
Claimes for "benefits" provided by Distributed Processing systems

Only few attempts have been made to establish a set of definitional criteria for a Distributed Procesing System [Ensl78],[Jens78] and [Ensl81]. The DPSs that were analyzed in this study posessed the five criteria of a Fully Distributed Processing System as defined by Enslow in [Ensl81]. According to this definition a processing system has to meet the following criteria in order to be considered as fully distributed:

i. **Multiplicity of resources:** The system should provide a number of assignable resources for any type of service demand. The greater the degree of replication of resources, the better the ability of the system to maintain high reliability and performance.

ii. **Component interconnection:** A Distributed System should include a communication subnet which interconnects the elements of the system. The transfer of information via the subnet should be controlled by a two-party, cooperative protocol (loose coupling).

iii. **Unity of control:** All the components of the system should be unified in their desire to achieve a common goal. This goal will determine the rules according to which each of these elements will be controlled.

iv. **System transparency:** From the users point of view the set of resources that constitutes the DPS acts like a 'single virtual system'. When requesting a service the user should not be required to be aware of the physical location or the instantaneous load of the various resources.

v. **Component autonomy:** The components of the system, both the logical and physical, should be *autonomous* and are thus afforded the ability to refuse a request for service made by another element. However in order to achieve the system's goals they have to interact in a *cooperative* manner and thus adhere to a common set of policies. These policies should be carried out by the control schemes of each element.

The salient characteristics of DPSs is the multiplicity and autonomy of its resources. Most

of the advantages which DPSs provide depend on these two properties. However due to multiplicity and autonomy of its resources a DPS may be in a *Wait while Idle* (WI) state which is a state in which a task is waiting for service while a resource that is capable of serving it is idleing. Any system which aims at achieving minimal response time will consider a WI state as undesired and thus attempt to minimize its duration. The WI state is a fundamental phenomenon associated with distributed systems and may occur even when a number of tasks are waiting for a single resource.[1] In such a case the distribution of the access control scheme is the cause for the WI state.

§1.3 Load Balancing Algorithms

A Load Balancing (LB) algorithm for a DPS is a distributed decision process that controls the assignment of the system resources. The algorithm is motivated by the desire to achieve better overall performance relative to some selected metric. The algorithm utilizes a *task migration* mechanism in order to place the tasks at the 'right' resources. This study focuses on LB algorithms whose goal is to minimize the expected turnaround time of a task.

The nature of a DPS adds another dimension of complexity to the development of decision processes. Because of the existence of more than one decision maker (controller) in the system and the absence of information on the current system state at the paint: the process takes place, the control problem of such systems is nonclassical [Scho78]. In such control problems the selection and collection of information for the decision process - the *information* rule - is almost as important as the decision rule - the *control law.*

The control law of a distributed LB algorithm determines when, from where and to whom to transfer a waiting task. The decision is made according to the current available information on the system's load. It is the function of the information policy to collect the data concerning the instantaneous load of the various resources. Each of these two element has to reside at every resource and the communication system is used by both of them in order to carry out their functions. The control element sends *data* messages that describe tasks and the information element sends *status* messages that contain data concerning the resource load.

Since the operation of the algorithm relies on an efficient exchange of information, the balancing process faces a *transmission dilemma* because of the two opposing effects the

[1] In an ETHERNET network a number of stations may be in a 'backoff' state while the channel is free

transmission of a message may have. On the one hand, the transmission of a message improves the ability of the algorithm to level out the instantaneous system load and to maintain an updated picture of the system load at different locations. On the other hand, it raises the expected queueing time of a message due to the increase in channel utilization. Long transmission delays lower the ability of the LB algorithm to achieve its goal.

§1.4 Previous Work

The problem of resource allocation in an environment of cooperating autonomous resources and its relationship to system performance is a major issue associated with the design of distributed systems [Echh78]. A number of studies of this issue have been reported. However, most of these studies deal with processing systems that utilize central elements, such as a job dispatcher, a shared memory or a main processor. In all these studies processing overhead due to the balancing process are not included in the performance models.

Stone in [Ston77] presents a centralized resource allocation algorithm for multiprocessor systems. The algorithm assumes that the cost (including communication costs) of each assigment is given. Under this assumption the optimal assignment problem is transferred to the problem of finding the *minimum cutset* of a graph.

A homogeneous two-server system with a central job dispatcher has been studied by Chow and Kohler [Chow77]. A load balancing algorithm that aspires to minimize the difference between the queue length of the two servers has been presented. The system has been modeled as a two-dimensional Markov process and has been solved by means of a recursive method.

The complexity of the load leveling problem has been analyzed by Kratzer and Hammerstrom [Krat80]. In their study they have shown that the CPU load leveling problem is NP complete. A stable decentralized algorithm has been defined for a uniformally structured network.

Bryant and Finkel [Brya81] have presented a preemptive stable load balancing algorithm for homogeneous distributed systems. The service discipline of the processors, is assumed to be **processor *sharing*** and they are interconnected in a point-to-point fashion. The preformance of the algorithm for a given topology and different operating conditions has been investigated through simulation.

A number of load balancing strategies for a class of local area networks have been defined by Ni and Abani in [Ni 81]. In their performance models they have assumed that

the communication delays due to job routing are very small compared with the job execution time. Both analysis and discrete event simulation have been employed for obtaining the desired performance measures.

§1.5 Organization of Dissertation

This thesis takes a tree structure approach in describing the study of load balancing algorithms for DPSs. Chapter 2 is the root of the structure and chapters 3 to 5 are the leaves. In each of the later chapters interaction between the load balancing process and a different type of distributed system is analized. These three chapters describe three parallel investigations of different aspects of the task migration phenomenon. The factors that unify these investigations are discussed in the 'root' chapter.

In chapter 2 the m*(M/M/1) family of distributed system models is defined and a taxonomy of load balancing is proposed. Some of the properties of multi-resource systems and the balancing process are demonstrated by means of simple analytical models.

Load balancing in a two-server distributed system is the subject of the third chapter. A LB algorithm with a parameterized and state dependent threshold is presented. Both the steady-state and the transient behaviour of the algorithm are analyzed and guidelines for the design of LB algorithms are concluded.

Chapter 4 focuses on broadcast distributed systems. Three load balancing algorithms which utilize the advantages of a broadcast communication media are presented and their performance analyzed. The performance models of the broadcast systems include a detailed description of the ETHERNET communication protocol.

In Chapter 5 a LB algorithm for store-and-forward distributed systems is defined. The chapter focuses on the interdependency between the topology of the system and the load balancing phenomenon. Performance models of DPSs with various topologies are simulated and their performance analyzed.

Performance predication is a cohesive element of an investigation of the load balancing phenomenon. Due to the characteristics of distributed processing systems performance prediction of such a system almost always entails a simulation study. A novel approach for modeling and simulating distributed processing systems is presented and discussed in Chapter 6. An example of a model and a simulator which were developed according to this methodology are included in Appendix C.

Chapter 2

The m*(M/M/1) Distributed System

The problem of load balancing in DPSs, like other problems related to the performance of this type of systems, is a two dimensional one. The first dimension represents the characteristics of the distributed system, while the second captures parameters that define the LB algorithm itself. There is a great variety in both the structure and the intended usage of DPSs. Various communication systems as well as processing elements are used for building DPSs and different approaches are employed when designing them to meet the needs of various users. Because of this variety a quantitative analysis of the interdependence between the performance of LB algorithms and the characteristics of the DPS is essential for acquiring an insight of the task migration phenomenon. A sensitivity analysis performed along the first dimension, the system axis, will provide means for determining which of the system's attributes are *detrimental* and which are *advantageous* to the load balancing process. A similar analysis along the second dimension, the algorithm axis, will point at ways in which the task migration process can take advantage of certain properties of the system, and how obstacles caused by other attributes can be overcome.

In order to carry out the above analysis, various performance models of DPSs which are controlled by different LB algorithms must be defined and solved. The performance measures obtained from the solution will give a quantitative description of the relation between the algorithms and the systems. The family of DPSs models selected for this analysis has a major impact on the scope and nature of the study of the LB problem. On the one hand, the conclusions drawn from a study based on complex and detailed models will be applicable to a particular implementation but, in most cases, will have a limited significance as far as the basic characteristics of the phenomenon are concerned. On the other hand, results obtained

from solving simple and abstract models reflect the basic characteristics of the problem but in the model itself some important characteristics of DPSs may be overlooked. The stochastic properties and complexity of the models determine **which** methods can be employed for solving them and thus they define the nature of the study. When the model meets the assumptions of the numerical methods which have been developed for solving performance models the study will be 'analytical'. However, the investigation will turn into a 'simulation study' when these methods fail to solve the model.

§2.1 Definition of an m*(M/M/1) System

The family of distributed system models selected for this study are the **m*(M/M/1) systems.** An m*(M/M/1) system consists of m processing elements that are interconnected by a communication subnet (Fig **2.1).** The family is characterized by the structure of its processors and the profile of the workload. The specification of the family does not impose any restriction upon the structure or the protocol of the communication subnet. Every node of an m*(M/M/1) system can serve it's own users **autonomously** and therefore the operation of the system does not rely on communication. The processing elements were integrated into one system in order to provide their users with a better response time. The system is controlled by a load balancing algorithm which tends to reduce the expected queueing **time** of a task by means of task migration. **This algorithm** is the sole user of the communication system.

The specification of the m*(M/M/1) is the following:

1. **Processors-** Each of the m processing elements consists of *a* processor, P_i , and an infinite queue. The queueing discipline is First Come First Serve (FCFS) and all the processors provide the same functional capabilities.
2. **Work-Load Profile.** Tasks arrive independently at each node and join the queue. The inter-arrival time has a negative exponential distribution and thus the task arrival process of the entire system consists of m independent *Poisson* processes. The service demand of the tasks is exponentially distributed and the structure of the nodes and tasks is such that every processor can serve any task. When a task has not been served by the node at which it had arrived, that is its *entrance site,* the results of its execution have to be transferred from the node at which it has been executed, back to the entrance node. The node which has served the task is called the *execution site* of the task.

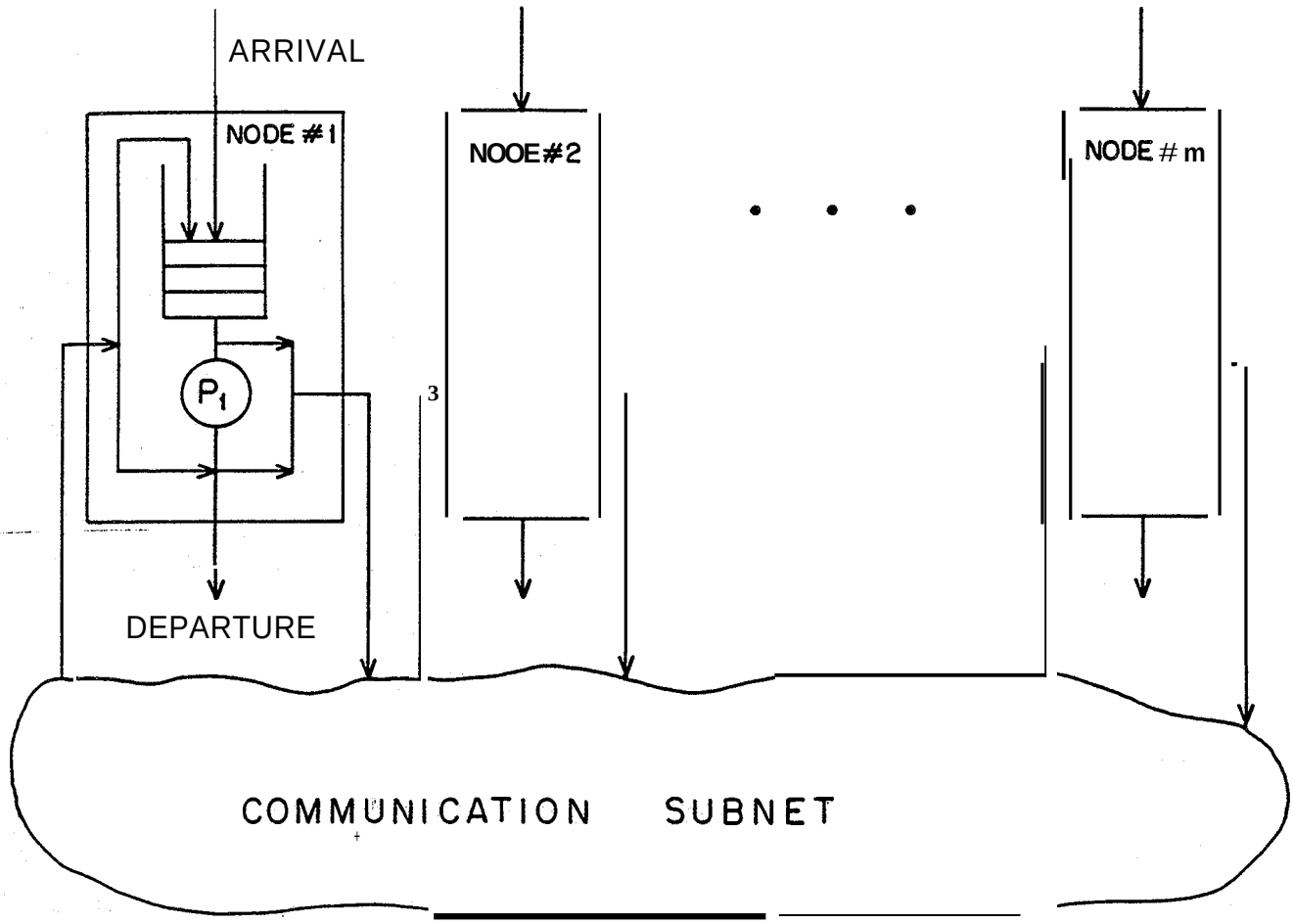

Figure 2.1. An m*(M/M/1) System

The ***operating conditions*** of the processing elements and the workload of an m*(M/M/1) system are defined by the structure

$$O = < (P_1, \ldots, \mu_m), (\lambda_1, \ldots, \lambda_m), (\alpha_1, \ldots, \alpha_m), , (\gamma_1^I, \ldots, \gamma_m^I), (\gamma_1^O, \ldots, \gamma_m^O) >$$

where

- μ_i – is the service rate of processor i given in ***Execution Units (eu)*** per ***Time Unit*** (*tu*).
- λ_i – is the rate at which tasks arrive at processor i.
- α_i – is the expected execution demand, in *eus*, of the tasks that arrive at node i.
- γ_i^I – is the expected number of *Data Units* (*du*) a processor needs in order to identify and serve a task that has arrived at node i.
- γ_i^O – is the expected number of *dus* required to describe the results of a task that has arrived at node i.

As indicated by the above specifications the m*(M/M/1) systems, are in a way, the simplest models of a distributed systems. They have simple processing elements, their tasks have nice - *memoryless* - stochastic behaviour and they do not perform any distributed processing except for the LB process. Nevertheless, despite their simplicity, these models capture the main properties of a distributed system. They have resource multiplicity, their resources are loosely coupled and autonomous, and all the processing elements cooperate in the achievement of a common goal.

In this study only models with *homogeneous* processors and tasks will be considered. The processing elements of an m*(M/M/1) system will be defined as homogeneous processors if $\mu_i = \mu$ for every $0 < i \leq m$. The tasks will be defined as homogeneous if $\alpha_i = \alpha, \gamma_i^I = \gamma^I$ and $\gamma_i^O = \gamma^O$ for $0 < i \leq m$. The *effective service rate* of a system with homogenous processors and users will be defined by the ratio $\tilde{\mu} \triangleq \frac{\mu}{\alpha}$. The system will be *homogeneously* loaded if both the processors and tasks are homogeneous and $\lambda_i = \lambda$ for $0 < i \leq m$.

2.1.1 The Probability of a WI State

The load balancing process aspires to improve the response time of the distributed system by minimizing the probability that the system will be in a WI state. The value of that probability for a system in which no task migration takes place, will point at the potential capability of the load balancing process to enhance the performance of the system. The extent to which this probability can be reduced by means of task migration and the net impact of the balancing process on the expected response time of the system, depend on the characteristics of both the system and the algorithm.

Assume a homogeneously loaded m*(M/M/1) system in which no task migration takes place and let $P_{wi}(n)$ be defined as[1]

$$P_{wi}(n) = P[\textit{at least } n \textit{ tasks are waiting and at least } n \textit{ processors idle}] \qquad (1)$$

then from the uniformity and the independency of the nodes it follows that

$$P_{wi}(n) = \sum_{i=n}^{m-1} \binom{m}{i} I_i W_{m-i}(n) \qquad (2)$$

[1] The notation $P[E]$ denotes the 'probability of event E'.

where

$\binom{m}{i}$ is the number of different ways a set of i processors can be selected from the m processors of the system.

$I_i = (1-\rho)^i$ is the probability that a given set of i processors will be idle.

$W_j(n)$ is the probability that a given set of j processors will be busy and at least n tasks will be waiting for service in their queues.

$\rho = \frac{\lambda}{\mu}$ is the utilization of the processors.

Because each of the nodes is an independent M/M/1 queueing system it follows that

$$P[k \text{ tasks in a given distribution wait for service in a given set of } j \text{ busy processors}] = P_0^j \rho^{j+k} \tag{3}$$

where $P_0 = (1-\rho)$ is the probability that an M/M/1 system is empty. From (3) it follows that

$$W_j(n) = \rho^j - \sum_{k=0}^{n-1} \binom{j-1+k}{k} P_0^j \rho^{j+k} \tag{4}$$

where

ρ^j is the probability that j processors are busy.

$\binom{j-1+k}{k}$ is the number of ways in which k tasks can be distributed among j queues.

From (2), (3) and (4) it follows

$$P_{wi}(n) = \sum_{i=n}^{m-1} \binom{m}{i} P_0^i [\rho^{m-i} - (P_0\rho)^{m-i} \sum_{k=0}^{n-1} \binom{m-i-1+k}{k} \rho^k] \tag{5}$$

When the system is in a WI state, at least one task is waiting and at least one processor is idle. Therefore the probability that a homogeneously loaded m*(M/M/1) will be observed in a WI state if no task migration takes place, P_{wi}, is given by $P_{wi}(1)$. Thus from (5) it follows that

$$\begin{aligned} P_{wi} \triangleq P_{wi}(1) &= \sum_{i=1}^{m-1} \binom{m}{i} P_0^i (\rho^{m-i} - (P_0\rho)^{m-i}) \\ &= 1 - (1-P_0)^m (1 - P_0^m) - P_0^m (2-P_0)^m \end{aligned} \tag{6}$$

Fig. 2.2 presents P_{wi} for different values of m with processor utilizations, ρ, as a parameter. The curves of the figure indicate that for practical values of ρ, the probability of a WI state is remarkably high and that in systems with more than ten processors there is almost always a task waiting for service while a processor is idling. P_{wi} reaches its maximum value when

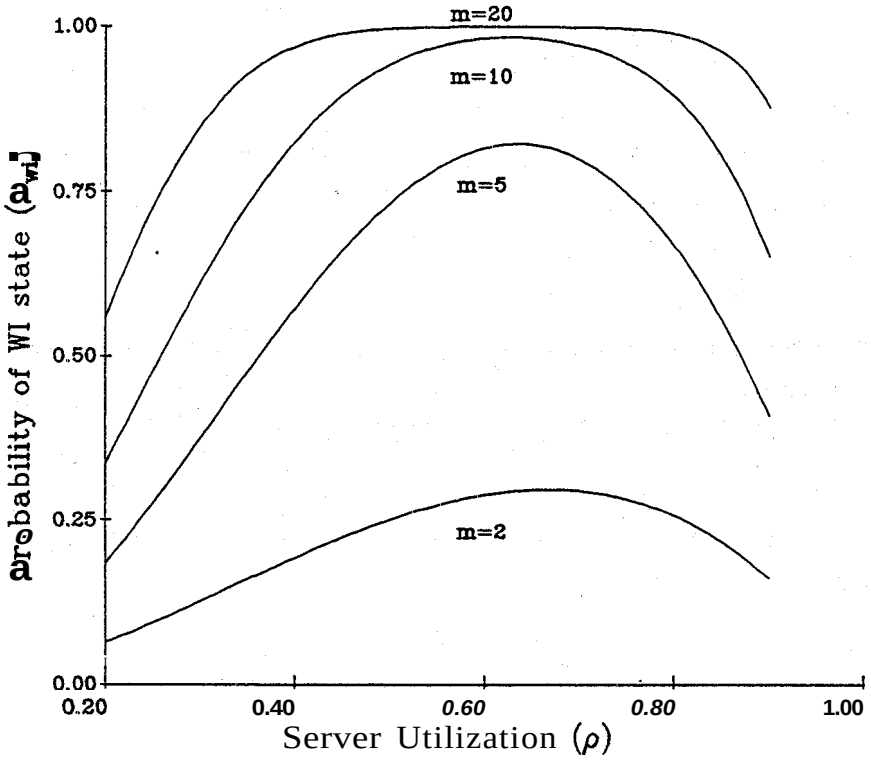

Figure 2.2. P_{wi} **vs.** ρ for an m*(M/M/1) system with no task migration

the processors are utilized during 65%, of the time. As the utilization of the processors increases past the level of 65% P_{wi} decreases. This property of P_{wi} indicates that a *'good'* load balancing algorithm should perform less work when the system is heavily utilized. It is clear that the same holds true for systems that are idle most of the time.

2.1.2 Balancing Distance

In various areas the distance between two objects is measured in *time units*. Years, hours and minutes are used for describing the distance between stars, towns and houses. Time is also used as a measure for the distance between elements of a DPS but in a different way. From the point of view of a scheduling or resource allocation algorithm for such a system, the distance between resource A and B is the time required for moving a given amount of data from A to B, whereas in the other examples the distance is the time it takes to go from A to B at a given velocity. Unlike the other cases the measure used for a DPS has no relation to the physical distance between the resources. It reflects the capacity of the communication link through which the resources are connected and the processing overheads associated with

the communication process. Thus two processors which are located at different continents may be 'closer' than two computers situated in the same room.

The degree to which task migration can reduce the probability of a WI state in a multi-resource system depends mainly on the ratio of the transfer time to execution time[2] of a task. Therefore from the point of view of the LB process the distance between two resources is determined by this ratio. Let $TJ_{i,j}(x)$ be the transmission time of a task of length x dus from resource i to j than the *balancing distance* between i and j is defined

$$BD_{i,j} \triangleq \frac{TJ_{i,j}(x)\,\alpha_i}{\mu_i} \tag{7}$$

The resources will be considered as being *close* to one another when the expected transmission time is negligible relative to the mean execution time, i.e. a small BD, and as *distant* when the time required for transferring a task is much longer than the time required for executing it, i.e. a large BD. The ***balancing diameter*** of an m*(M/M/1) system will be defined as the largest balancing distance between two of its processors.

213 Processing Overheads

In addition to communication resources, the LB process requires processing resources. The execution of both the control element of the algorithm and the various functions of the communication protocol require processing capacity [Tane81]. The processing capacity utilized by the LB process is the *overhead* which the distributed system has to pay in order to achieve a reduction in P_{wi}. As a result of this overhead the LB process reduces the amount of processing capacity available to the users. The effect of this reduction on the expected response time of the system is the opposite of the effect of the reduction in P_{wi}.

The protocols of communication systems that meet the requirements of a *'reliable network'* are complex and require a considerable amount of bookkeeping whereas the control laws of LB algorithms are relativly simple. Therefore, only processing overheads due to activities of the different layers of the communication protocol will be considered in this study. The manner in which these overheads will be introduced into the m*(M/M/1) model will depend on the characteristics of the communication subnet.

P_{wi} can be reduced to zero only if the balancing diameter and overhead of the system are zero. In such a case the system can be viewed as a single queue multiple processor

[2] the execution and transmission times are the actual service times and do not include queueing delays.

distribution		$\rho = .3$	$\rho = .5$	$\rho = .6$	$\rho = .7$	$\rho = .8$	$\rho = .9$
Determinestic	$(C_b = 0)$	.214	,500	.750	1.166	2.000	4.500
Exponential	$(C_b = 1)$	.428	1.000	1.500	2.333	4.000	9.000
Gamma(k=2)	$(C_b = 2)$	1.071	2.500	3.750	5.883	10.00	22.500
Hyperexponential	$(C_b = 3)$	2.142	5.000	7.500	11.667	20.00	45.000

Table 2.1. Norm. Expected Waiting Time $\hat{W}_q$ for Different Distributions

queueing system where the queue consists of m cells. Each cell has an independent stream of tasks and a processor that serves tasks that were allocated to this cell according to an **FCFS** discipline. **Tasks** are moved instantaneously from one cell to the other according to the migration criterion of the LB algorithm. An m*(M/M/1) system with no balancing overhead and where $BD_{i,j} = 0$ for all $0 < i, j, \leq m$ will be called an M/M/m-like system.

§2.2 M/G/1 and M/M/m Queueing Systems

Processing systems are **usually** shared by several users. The stochastic behaviour of the **tasks** submitted by these users - their arrival times and service demands - cause *resource contention* that leads to the establishment of queues **and** consequently the tasks have to ***waste*** time while waiting for service. The factor by which the system ***inconveniences*** the users due to the fact that they are sharing the same resources is represented by the ***mean normalized queueing time*** of the system ,$\hat{W}_q$, which is the ratio of average time a task spends in a queue to average service time required by the task.

The $\hat{W}_q$ of a single processor system that serves a Poisson stream of customers according to an **FCFS** discipline, an M/G/1 queueing system, is given by

$$\hat{W}_q = \rho \frac{(1 + C_b^2)}{2(1 - \rho)} \tag{8}$$

which is the *Pollaczek-Khinchin (P-K) mean value formula* and where C_b is the *coefficient of variation* for service time demand distribution. It follows from (8) that $\hat{W}_q$ is an ***unbounded monotonic increasing*** function ***of*** the variance of the service time demand distribution. Therefore a system with a moderate utilization might have a large $\hat{W}_q$ when the standard deviation of the service time is large. Table **2.1.** presents some numerical values of $\hat{W}_q$ for systems with different ρ and C_b. Note that the coefficient of variation for CPU service time distributions is assumed to be greater than one [Coff73]. Samples of these times form, in most cases, a *hyperexponential* distribution.

The service time demand distribution in an m*(M/M/1) system is exponential and therefore the mean normalized queueing time of such a system, in which the balancing distance between any two resources is infinite, without load balancing, is given by

$$\hat{W}_q = \frac{\rho}{1-\rho} \qquad \text{(M/M/1)} \tag{9}$$

(9) is the upper bound for the $\hat{W}_q$ of an m*(M/M/1) system and serves as a means for evaluating the improvement in performance due to the LB process. The best mean normalized queueing time that m Poisson streams of customers[3] can obtain from m exponential processors is given by

$$\hat{W}_q = \frac{\left(\frac{(m\rho)^m}{m!}\right)\left(\frac{1}{m(1-\rho)^2}\right)}{\left(\sum_{k=0}^{m-1}\frac{(m\rho)^k}{k!} + \left(\frac{(m\rho)^m}{m!}\right)\left(\frac{1}{1-\rho}\right)\right)} \qquad \text{(M/M/m)} \tag{10}$$

which is the mean normalized queueing time of an M/M/m system [Klei75]. Eq. (10) can serve as a lower bound on the $\hat{W}_q$ of an m*(M/M/1). It follows from (10) that the lower bound on $\hat{W}_q$ is a *monotonic decreasing* function of the number of processors.
The greater the number of M/M/1 systems which are integrated into one M/M/m-like system, the smaller the expected queueing time of a task is. Fig 2.3. shows the value of $\hat{W}_q$ for an M/M/m system as a function of m for different values of p.

2.2.1 Rate of Transfers in an M/M/m-Like System

The rate at which tasks must be transferred from one queue to the other in order to minimize the probability of a WI state in an m*(M/M/1) system, is a major argument in justifying a study of task migration criteria for such systems. Only when a significant percentage of tasks are transferred, a change in the control law of the LB algorithm will affect the utilization of the communication system and thus change the performance of the system. In this section a lower bound on the transfer rate for a M/M/m-like system is derived and used for evaluating the amount of transfers required for minimizing the mean queueing time of a task in an m*(M/M/1) system.

[3] m Poisson streams can be considered as one Poisson stream with a rate equal to the sum of the rates of the individual streams.

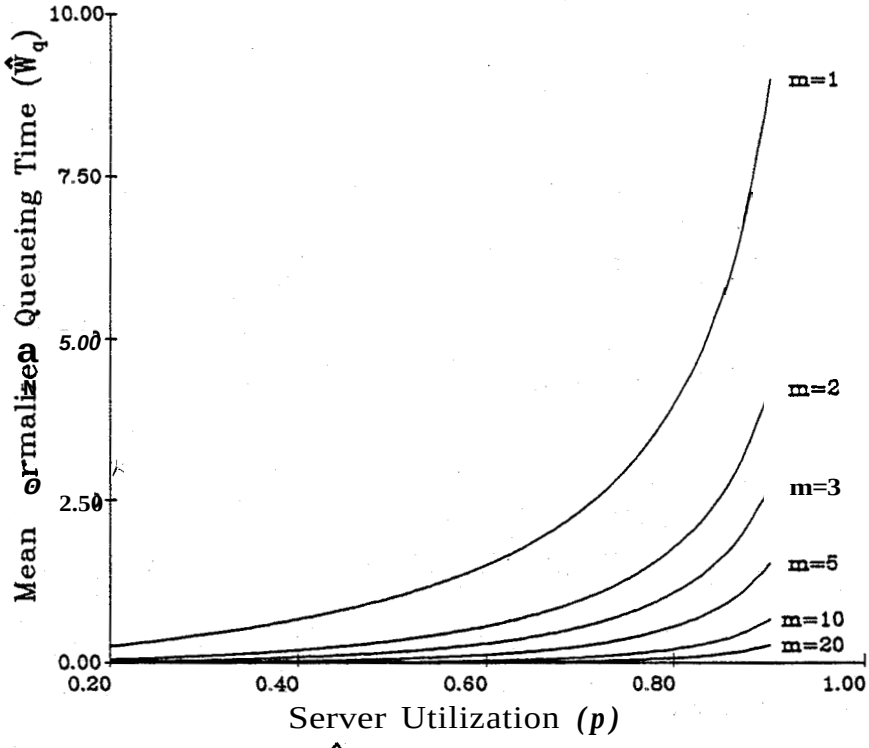

Figure 2.3. $\hat{W}_q$ **vs.** ρ for an M/M/m system

Assume a homogeneously loaded M/M/m-like system and let TR be the rate in **which** task transfers are initiated by the LB algorithm. Since the probability of a WI state in such a system is zero, a task must be transferred whenever one[4] of the following events occures:

E1 **A** task arrives at a busy processor while at least one of the processors is idling.

E2 **A** processor completes the service of a **task,** no other tasks are waiting in its queue and there is at least one task waiting in another queue.

From the above it follows that

$$P[\textit{transfer in } (t,t+\Delta t)] \leq P[\text{E1 in } (t,t+\Delta t)] + P[\text{E2 in } (t,t+\Delta t)] \qquad (11)$$

The number of busy processors at time t is $min(n(t), m)$ where $n(t)$ is the number of tasks in the system at time t. When k processors are busy at time t, the probability that a task will arrive at a busy processor in $(t, t+\Delta t)$ is $k\lambda\Delta t$ and therefore

[4]Because the system is a birth and death system [Klei75] multiple events are prohibited.

$$P[\textit{arrival at a busy processor in } (t,t+\Delta t)] = \lambda\left(\sum_{i=1}^{m-1} i\,P_i(t)\Delta t + \sum_{i=m}^{\infty} m P_i(t)\Delta t\right) \qquad (12)$$

where $P_i(t)$ is the probability that $n(t) = i$. In order that an arrival will meet the conditions of E1, at least one of the processors has to be idle. Since the number of idling processors at time t is $m - min(n(t), m)$ it follows from (12) that

$$P[\text{E1 } in\ (t,t+\Delta t)] = \lambda \sum_{i=1}^{m-1} i\,P_i(t)\Delta t \qquad (13)$$

The number of tasks waiting for service at time t is $max(n(t) - m, 0)$ and therefore when $m < n(t) < 2m$ all processors are busy, and in at least $2m - n(t)$ nodes no task is waiting to be served. The probability that one of the busy processors which has only one task in its queue will complete the service of that task in the interval $(t, t+\Delta t)$ is $(2m - n(t))\,\tilde{\mu}\Delta t$. So it follows that

$$P[E2\ in\ (t,t+\Delta t)] \geq \tilde{\mu} \sum_{i=1}^{m-1} (m-i) P_{m+i}(t)\Delta t \qquad (14)$$

The system is assumed to be in a steady state, $\frac{\partial P_i(t)}{\partial t} = 0$. By replacing $P_i(t)$ by $P_i = \lim_{t\to\infty} P_i(t)$ and integrating over a time unit interval it follows from (13) and (14) that

$$TR \geq \sum_{i=1}^{m-1} [\lambda\, i\, P_i + \tilde{\mu}\,(m-i) P_{m+i}] \qquad (15)$$

by replacing P_i by the expression for the probability of having i tasks in an M/M/m system it can concluded from (13) that

$$BL = \frac{\sum_{i=1}^{m-1}\left(\lambda\, i \frac{(m\rho)^i}{i!} + \tilde{\mu}\,(m-i)\frac{m^m \rho^{m+i}}{m!}\right)}{\left(\sum_{k=0}^{m-1} \frac{(m\rho)^k}{k!} + \left(\frac{(m\rho)^m}{m!}\right)\left(\frac{1}{1-\rho}\right)\right)} \qquad \text{(M/M/m)} \qquad (16)$$

is a lower bound on TR.

In many cases it is natural to use the expected excution time of a task, $\tilde{\mu}^{-1}$, as a time unit when analyzing a queueing system. **Fig. 2.4** presents the lower bound on the *normalized transfer rate* per node, $\hat{BL} \triangleq \frac{BL}{m}\tilde{\mu}$, as a function of ρ for systems with different numbers of processors. Note that a considerable number of tasks have to be transferred in

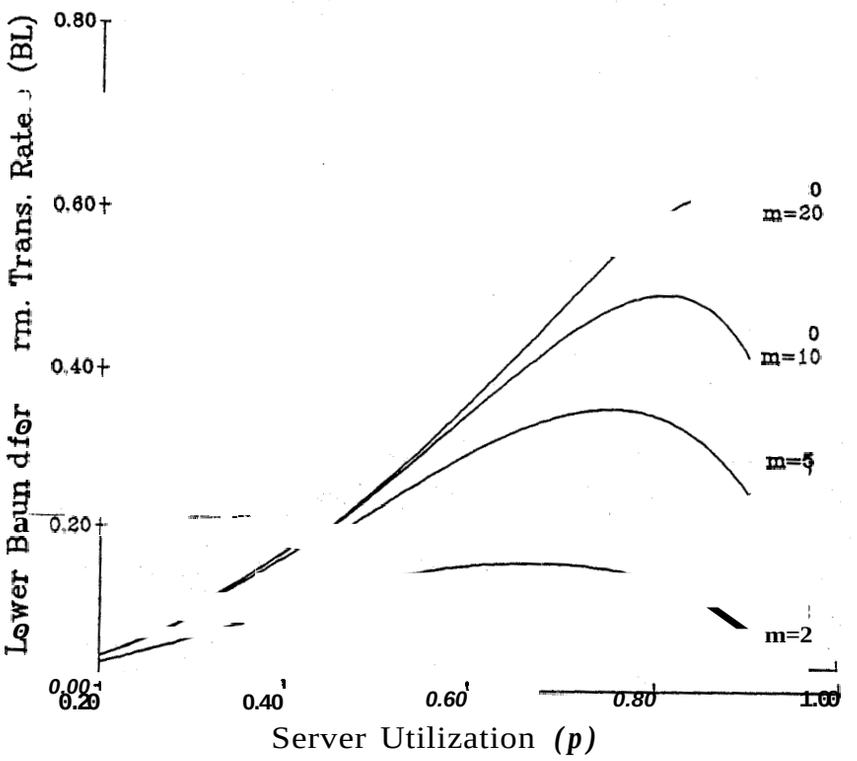

Figure 2.4. **BL** vs. p for an M/M/m-like system

order to achieve the performance of an M/M/m system. For with more than ten processors almost one out of λ^{-1} tasks are transferred.

These results indicate that in an m*(M/M/1) system where task transmission time is not negligible the load balancing process might utilize a large portion of the capacity of the communication system. The utilization of the communication system will determine the delays associated with the transmission of a task or any other message, These delays will cause an increase in P_{wi} and therefore an increase in . The amount of traffic generated by the balancing algorithm has a major effect on its ability to improve the performance of the system. Fig. **2.4** shows that in order to achieve the optimal performance, $P_{wi} = 0$, a large portion of the tasks have to be transferred.

2.2.1.1 Two policies for an M/M/2-like system

In order to demonstrate the effect of a change in the migration criterion on the expected number of task transfers initiated by an LB algorithm, assume an M/M/2-like system (zero balancing distance) and consider the following two migration policies:

1. **'Look ahead' policy-** A transfer is initiated whenever the difference between the number of tasks in the two queues is greater than one and the channel is idle.
2. **'Trouble shooting' policy-** According to this policy a **task** that **is** waiting to be served at one queue will be transferred to the other queue only when the other processor is not busy.

Under the above assumption the mean normalized queueing time of a task, in both cases, will be the same as in an M/M/2 queueing system since the probability of a WI state is zero. Let TR_1 and TR_2 be the expected rates at which transfers are initiated by the 'look ahead' and 'trouble shooting' migration policies respectivly. It is shown in Appendix **A** that TR_1 is given by

$$TR_1 = 2\tilde{\mu}P_0\frac{\rho^2}{(1-\rho)} \tag{17}$$

and that TR_2 is bounded by

$$TR_2 \geq \tilde{\mu}(1-\rho-P_0(1+\rho)(\rho\ln\left(\frac{1+\rho}{1-\rho}\right)-1) \tag{18}$$

and

$$TR_2 \leq 2\tilde{\mu}P_0\rho\,[\rho-2(\rho+\ln(1-\rho))] \tag{17}$$

where $P_0 = \left(1+\rho+\frac{1}{2}\rho^2\left(\frac{1}{1-\frac{1}{2}\rho}\right)\right)^{-1}$ is the probability that an M/M/2 system will be empty.

Fig. 2.5 presents the value of TR_1 (the dashed line) and the two bounds of TR_2 (the cross-hatched line) **as** a function of ρ for $\tilde{\mu} = 1$. The curves presented in the figure demonstrate the wide range of d u e s which the transfer rate of the balancing algorithm can receive and points at the harmful effect that a balancing algorithm with a too *'liberal'* migration criterion may have on the performance of the system.

The interdependency between the performance **of** the m*(M/M/1) system and the migration criteria of the LB algorithm will be discussed in the course of the presentation of the results obtained from the solution of the performance model of these systems. It is clear that the optimal policy depends on the balancing distance between the systems' resources and the penalties associated with the transfer of a task.

§2.3 Unbalance Factor

The LB algorithm is ***distributed*** among the processing elements of the system. Every processor has its own local LB control element which governs the migration of tasks into and out of

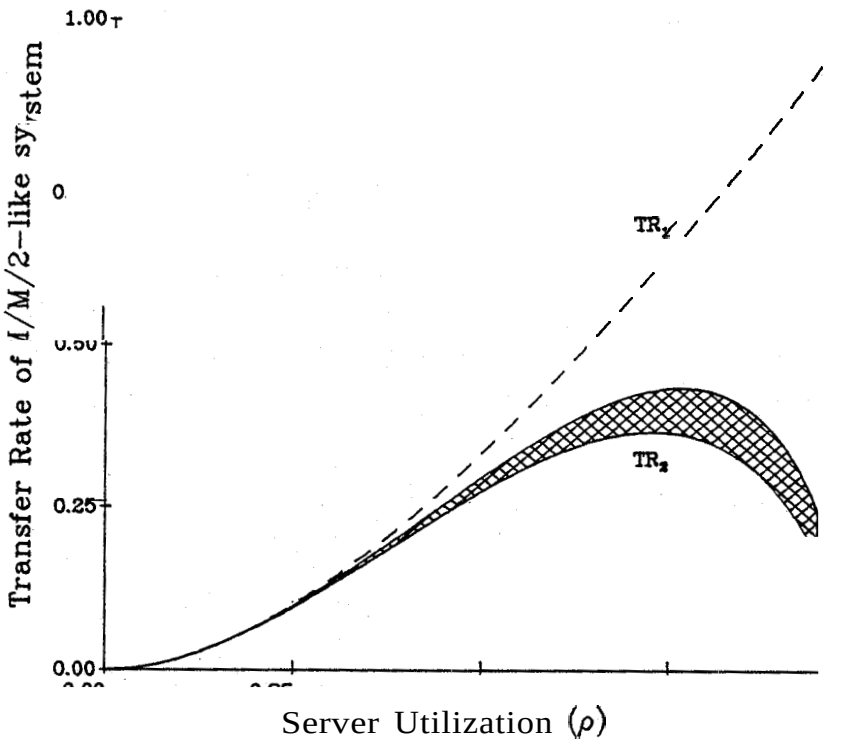

Figure 2.5. **Transfer** Rate vs. p for **M/M/2-like** system

its queue. The' decisions made by this control element.are based on the information supplied to it by the LB information elements **which** reside at other processors. The control **scheme** of the algorithm is usually based on a comparison between the load of the processor and the load of other processors which are included in a subset of processors called the ***balancing region*** of the processor. This region consists of those processors which the owner of the region considers as candidates for receiving one of its tasks. The region can be defined ***statically***, changed ***randomly*** or ***adapted dynamically*** to the instantaneous st

The migration criterion is basically a comparison between the degree to which the load distribution of the balancing region is unbalanced, and a predefined ***threshold.*** The evaluation of the load distribution is made according to the information available to the processor at that instance. Although the evaluation method may differ from one LB algorithm to the other a scheme for scaling the degree to which a load distribution is unbalanced has to be established in order to enable the characterization of task migration criteria.

In an m*(M/M/1) system the degree to which a load distribution is unbalanced should be measured according to the effect which an instantaneous task transfer has on the probability that the system will reach a WI state in the future. The likelihood that at least

one processor will become idle depends on the load level of the least loaded processor, whereas the probability that a task will be waiting is affected mainly by the number of tasks at the most loaded processor. Therefore in an m*(M/M/1) system with homogeneous tasks and processors the evaluation of a load distribution can be based on the minimal number of tasks resident at one processor and the difference between this number and the queue length of the most loaded processor.

The migration of a task is a binary operation between the source and the target nodes. Therefore, although the probability that a WI state will be reached is not determined only by the maximal and minimal queue length of the system, a scaling scheme based on the extreme loads of the system was selected. Let A be a subset of processors of an m*(M/M/1) system with homogeneous tasks and processors and $n_i(t)$ the number of tasks at processor i at time t then the *unbalance factor* of A at time t is defined as

$$UBF(A,t) \triangleq \begin{cases} \infty & (\Delta L(A,t) > 1) \wedge (\min_{j \in A}(n_j(t)) = 0) \\ \dfrac{\Delta L(A,t)}{\min_{j \in A}(n_j(t))} & (\Delta L(A,t) > 1) \wedge (\min_{j \in A}(nj(t)) > 0) \\ 0 & \text{otherwise} \end{cases} \quad (20)$$

where $\Delta L(A,t) \triangleq \max_{k,j \in A}(n_k(t) - n_j(t))$ is the *load-difference* of A at time t.

The above definition is based on a global point of view. All processors of A are considered as potential sources or targets for a migration operation. However the control element of a given processor evaluates the load distribution of the balancing region in order to decide whether to send out one of its tasks. Therefore an unbalance factor of the load of a given processor relative to its balancing region, is required. Let $BR_i(t)$ be the balancing region of processor i at time t and $m_{i,j}(t)$ be the number of tasks at processor j as known to processor i at time t, then the *relative unbalance factor* of i at time t is defined as

$$U\hat{B}F(i,t) \triangleq \begin{cases} \infty & (\hat{\Delta L}(i,t) > 1) \wedge (\min_{j \in BR_i(t)}(n_{i,j}(t)) = 0) \\ \dfrac{\hat{\Delta L}(i,t)}{\min_{J \in BR_i(t)}(n_{i,j}(t))} & (\hat{\Delta L}(i,t) > 1) \wedge (\min_{j \in BR_i(t)}(n_{i,j}(t)) > 0) \\ 0 & \text{otherwise} \end{cases} \quad (21)$$

where $\hat{\Delta L}(i,t) \triangleq \max_{k \in BR_i(t)}(m_{i,i}(t) - m_{i,k}(t))$ is the *relative load-difference* of i at time t. Note that only when a task is waiting for service at processor i and one of the processors of $BR_i(t)$ is idling, according to the information available to i,[5] $U\hat{B}F(i,t)$ becomes infinite.

[5] from this point on, unless stated explicitly otherwise, when the load distribution of $BR_i(t)$ is considered it is the distribution as known to processor i at time t.

In all other cases where no task is waiting at processor i or no processor is idle in $BR_i(t)$ the relative unbalance factor of i is finite. When the factor is zero, the load distribution is such that there is no processor whose queue length is smaller by two or more than the queue length of processor i, and thus a task transfer should not be executed.

According to the value of the relative unbalance factor of the source of a migration at the time it has been initiated, task transfers can be classified into two types - *last-minute* and *anticipatory* transfers.

2.3.1 Last-minute transfers

A transfer initiated when the $\hat{UBF}(i,t)$ is infinite will be classified as a *last-minute* transfer. Although the infinite value of the factor indicates that the balancing region is in a WI state, it is not always advantageous to initiate a transfer under such conditions. The beneficial effect of such a transfer depends on both the balancing distance between the two processors and the relative load-difference of the source processor.

2.3.2 Anticipatory Transfers

A transfer initiated by processor i when $\hat{UBF}(i,t)$ is greater than zero but finite will be called an *anticipatory* transfer. When $0 < \hat{UBF}(i,t) < \infty$ $BR_i(t)$ is not in a WI state. However, an instantaneous transfer of a task from i to the processor with the minimal number of tasks in $BR_i(t)$ decreases the probability that the region will reach a WI state in the future. A transfer initiated under such conditions can be considered as a preparative step taken to prevent the occurrence of a WI state. The advantage of such a transfer depends on the balance distance between the source and target of the transfer. The effect of the distance is not monotonic. When the distance is very small or too big the beneficial effect of an anticipatory transfer is limited. In the case of a big distance the load distribution of the system at the time the task arrives at the target might be considerably different than the distribution at the initiation time of the transfer. Consequently a decrease in the distance increases the power of such a transfer. However when the distance is very small there is no need for anticipatory transfers. Last-minute transfers are sufficient when the transfer time of a task is much smaller than its execution time. Therefore in such a case an increase in the distance will increase the beneficial effect of this type of transfer.

Chapter 3

The BTSQSS System

§3.1 Introduction

In order to be considered as a distributed system, a processing system should include at least *two* processors. Although two is the minimal number of processors, a 2*(M/M/1) system is sufficiently large to serve as a vehicle for studying the basic characteristics of the Load Balancing process. The interaction between the two processors due to the migration of tasks captures fundamental aspects of the LB phenomenon. By analyzing the performance of 2*(M/M/1) systems which are controlled by different LB algorithms, an insight of this phenomenon may be acquired. On the basis of the results obtained from such a performance study a range of ***acceptable*** balancing distances can be determined and the ***break-even point*** at which the overheads of the algorithm diminishes the advantages of the reduction in P_{wi} can be located. The manner in which such a system operates under various operating conditions, offers answers to questions like "can processor i take advantage of processor j when $BD_{i,j}$ is 2 ?", or "what happens when the communication activities require 10% of the systems processing capacity?". These answers may guide the design of LB algorithms for larger and more complex systems. The study of task migration in a two processor distributed system is the first step towards the development of an intuition for the LB phenomena.

This chapter presents a study of task migration criteria for 2*(M/M/1) distributed systems. An analysis of both the *transient* and *steady state* behaviour of the system and algorithms is included in the study. Since this study is the first step of an investigation of the LB process it was decided to use numerical methods (as opposed to discrete event simulation)

to derive the performance measures. It was assumed that by using stochastic models a better understanding of the dynamic properties of the migration phenomena may be obtained. The system is modeled by a multi dimensional *birth and death* process [Klei75]. The desire to use analytical methods for solving the performance models of the system motivated the selection of the communication system and the specification of the algorithm. Continuous simulation [numerical integration) is used for obtaining the time-dependent behaviour whereas the *steady state* performance models are solved by means of an iterative solution scheme. An analysis of the characteristics of a single task transfer, based on the system's transient behaviour, is presented.

§3.2 The BTSQSS System

The **Balanced Two Single Queue Single Server (BTSQSS)** system consists of two servers and a communication channel **(Fig. 3.1,).** The system is a 2*(M/M/1) distributed system with homogeneous users and processors. The channel interconnects the two queues and is capable of transferring one task at a time (half duplex link). The data rate of the channel is β *du/tu*, and the amount of data that has to be transferred when a task is migrated from one queue to the other, is a random variable with a negative exponential distribution and expectation γ^I. It is assumed that the expected number of *dus* that describe the results of a task (γ^O) is much smaller than γ^I and thus the transmission of the results hack to the entrance site is neglected.

The operation of the channel can either be controlled by the processors or be autonomous. In the first case the transmission of a task can be stopped in the middle and the system is thus defined as a ***atop*** system (S-BTSQSS), while in the latter case the system is defined as a **no-stop** system (NS-BTSQSS). In a no-stop system the initiation of a transfer will always result in a transfer of a task from one queue to the other. Regardless of the type of the system, the communication process may require processing capacity. It is assumed that when the channel is active the service rate of the processors, μ, is degraded by $\delta\%$. This degradation represents the processing overhead associated with the transfer of a task. The overhead is proportional to the duration of the transfer and is spread evenly along the transmission period. Therefore the service rate of each processor is $\mu(1 - \delta_f)$ during the channel busy periods, where $\delta_f = \frac{\delta}{100}$ is the degradation factor of the system.

The task distribution of the system at time t, $TD(t)$, is defined by the ordered pair $(n_1(t), n_2(t))$, where $n_1(t)$ and $n_2(t)$ are the number of tasks in the first and second queue

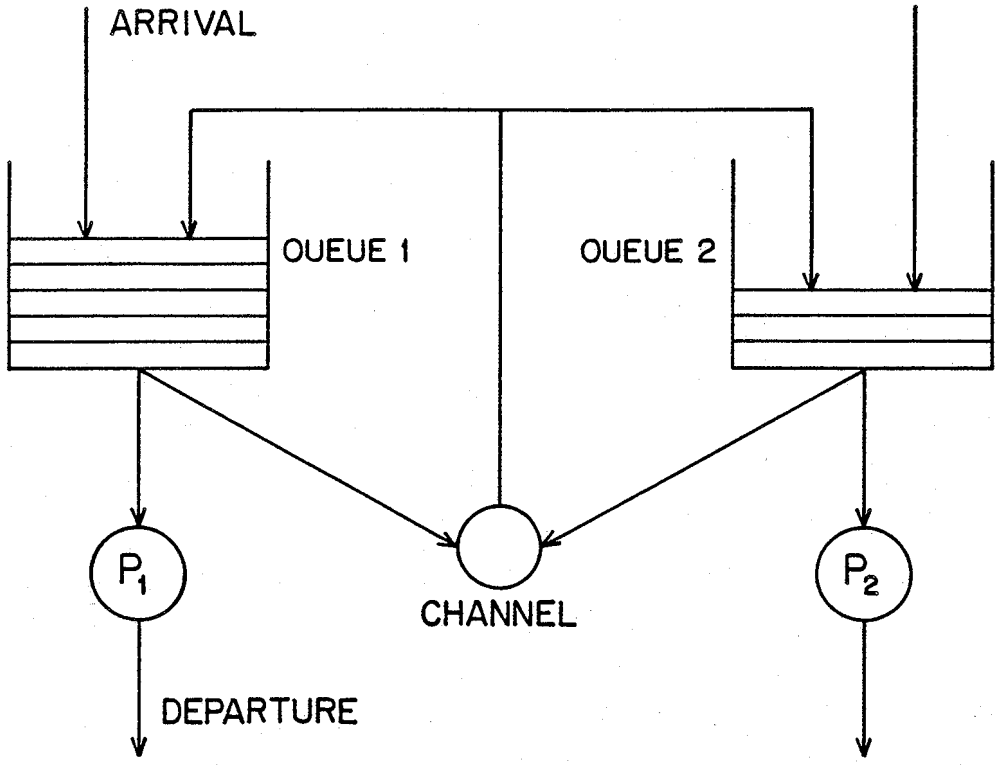

Figure 3.1. The BTSQSS system

respectivly. In an S-BTSQSS system a task which is being transferred remains in the queue of the source processor during the transmission period, whereas in an NS-BTSQSS system the task is removed from the source queue when the transfer is initiated. and is placed in the channel. The task will remain in the channel for the duration of the transmission.

The penalties associated with the LB process in S-BTSQSS systems are smaller than in equivalent no-stop systems. When the activity of the channel is controlled by the processors, the algorithm can intervene and stop a transfer in the middle. Thus a decision made at an earlier stage can be reconsidered and cancelled. Due to this capability, the number of 'wrong' transfers can be reduced. Although in most cases the communication system. is autonomous (most probably a reliable network), stop systems were considered in this study. Their performance was studied in order to evaluate the degree to which the ability to stop a transfer assists the task migration process. The results of the study give an indication of the conditions under which an attempt should be made to implement a stop system[1]

[1] In special purpose systems, control lines may be added to support such a facility.

because of the superiority of its performance.

The LB algorithms defined for the **BTSQSS** system do not include an information policy. Introducing the exchange of state-information into the model would have caused a considerable increase in its complexity and would have imposed the usage of simulation as a solution technique. In light of the motivation and focus of the study it was decided to assume that both processors are aware of the current task distribution of the system and thus there is no exchange of state-information in the model. Systems that utilize special control lines for controlling the migration process or system where the amount of state information is negligible relative to γ^I, meet this assumption.

§3.3 The AT algorithm

An anticipatory approach towards the load balancing process has influenced the definition of the Adaptive Threshold (AT) LB algorithm for **BTSQSS** systems. The migration criterion of the algorithm is based on a parameterized ***state dependent*** threshold. Inspite of its anticipatory nature the algorithm attempts to reduce the amount of communication capacity which it utilizes. The motivation for such efforts result from the assumption that the usage of the communication system has a ***price.*** Such an assumption is especially valid when the LB algorithm is not the sole user of communication resources. Since the system has only two processors each balancing region. includes both of them. The migration criterion of the algorithm is evaluated according to the current task distribution of the system. The 'load history' of the processors is not considered by the control element of the AT algorithm.

ALGORITHM AT ***(adaptive threshold]***

Control Law: Upon the arrival or departure of a task or when a task transfer terminates the control element is evoked at both servers. Processor i will initiate a transfer to its 'buddy' processor at time t if the channel is idle and the instantaneous task distribution meets the following criterion:

$$(\hat{\Delta L}(i,t) - 1 > L_p) \text{ A } (\hat{UBF}(i,t) > A_p)$$

where L_p and A_p are the *'last-minute'* and **anticipatory** parameters of the algorithm. If the system has a stop channel, the control element will stop a transmission in the middle whenever the system enters a state which does not fulfill the above criterion while the channel is busy.

The degree to which the algorithm initiates anticipatory transfers depends mainly on the value of its anticipatory parameter. A large positive value for A_p will prevent almost any transfer when the system is not in a WI state. Since $U\hat{B}F(i,t)$ is infinite when the system is in such a state, the initiation of 'last-minute' transfers is controlled by the second parameter, L_p. When $L_p = 0$ a migration will be initiated whenever $U\hat{B}F(i,t) = \infty$. However, when $L_p > 0$ a 'last-minute' transfer will be initiated only when at least l_p tasks wait for service at the non-idling processor.

Chow and Kohler in [Chow77] have suggested and analyzed an EB algorithm for an S-BTSQSS system. The algorithm defined there is a private case of the AT algorithm ($A_p = L_p = 0$). There where no penalties associated with the migration of a task in their model. It was assumed that the transmission process does not affect the service rate of the processors ($\delta = 0$) and that the processors possess full control of the channel activity (stop system). A simulation study of the expected turnaround time of a task as a function of the utilization of the servers and the transmission rate of the channel was presented in [Chow77].

§3.4 The Model

Markov chains are widely employed for modeling the performance of queueing systems. Most of the studies in the area of Queueing Theory are based on this type of stochastics process. When a queueing system does not meet the assumptions of a Markov Process a common approach for analyzing the system is first to *imbed* it on a Markov chain and only then solve it [Klei75].

The BTSQSS system with the AT load balancing algorithm forms a continuous multi-dimensional *markov chain*. The system and algorithm meet the assumption of this type of stochastic process since all the system state-time distributions are exponential (memoryless) and the decisions made by the algorithm are based only on the current state of the system. None of the system attributes are time dependent and therefore the chain which describes the behaviour of the system and algorithm, is homogeneous. Due to the exponential distribution of state-times - *inter-arrival, service* and *transfer* time - the probability of multiple events in a small time interval Δt is of the order of $O(\Delta t)$, and thus simultaneous events do not occur. Each of the chain events - *task arrival, task departure* and *transfer termination* - relate to a particular processor, Because of this property of the system and since the chain is homogeneous the stochastic model of the system meets the assumptions of a birth-and-death process,

The dimensionality of the process depends on the characteristics of the communication channel. A three dimensional **chain** is required in order to describe the behaviour of an **NS-BTSQSS** system, whereas a two dimensional state space is sufficient for describing a stop system. In order to simplify the notations and expressions used in the study it is assumed from this point on that service and transfer rates of the system are normalized according to the execution and data demands of a task respectively, i.e $\alpha = \gamma^I = 1$. The balancing distance of the two processors assuming the above normalization is $BD_{1,2} = \frac{\mu}{\beta}$.

341 The NS-BTSQSS Model

The ***state*** of the **NS-BTSQSS** at time t is described by the ordered triplet (n, m, k) where n and m are the number of **tasks** in the first and second queue respectivly, and k is the state of the channel. $k = 0$ means no transmission at time t and $k > 0$ indicates that a transmission from processor k to the other one is currently on its way.

Let $p(i, j, k, t)$ denote the probability that the state of an **NS-BTSQSS** system will be (i, j, k) at time t, then transition equations of the **NS-BTSQSS** model are the following:

$$
\begin{aligned}
\frac{\partial}{\partial t}p(i,j,0,t) = & -[2\lambda + \mu(e_i + e_j)]\,p(i,j,0,t) \\
& + (1 - z_{i,j})\{\mu\,[p(i+1,j,0,t) + p(i,j+1,0,t)] \\
& + \lambda\,[p(i-1,j,0,t) + p(i,j-1,0,t)] \\
& + \beta[p(i,j-1,1,t) + p(i-1,j,2,t)]\} \\
\frac{\partial}{\partial t}p(i,j,1,t) = & -[2\lambda + \mu(e_i + e_j)(1-\delta_f) + \beta]\,p(i,j,1,t) \\
& + \mu\{(1-\delta_f)[p(i,j+1,1,t) + p(i+1,j,1,t)] + u_{i+1,j}p(i+1,j+1,0,t)\} \\
& + \lambda\{p(i-1,j,1,t) + p(i,j-1,1,t) + u_{i+1,j}p(i,j,0,t)\} \\
& + \beta\,u_{i+1,j}\{p(i+1,j-1,1,t) + p(i,j,2,t)\} \\
\frac{\partial}{\partial t}p(i,j,2,t) = & \frac{\partial}{\partial t}p(j,i,1,t)
\end{aligned}
\tag{1}
$$

where

$$
u_{i,j} \triangleq \begin{cases} 1 & (i-j-1 > L_p \wedge (i-j > j\,A_p) \\ 0 & \text{otherwise} \end{cases}
$$

and $e_i \triangleq min(i, 1)$, $z_{i,j} \triangleq u_{i,j} + u_{j,i}$ and $p(-1, i, k, t) \triangleq p(i, -1, k, t) \triangleq 0$ for $i > 0$, $k = 0, 1, 2$. The state-transition-rate diagrams for the state $(i, j, 0)$ and $(i, j, 1)$ are presented in Fig. 3.2

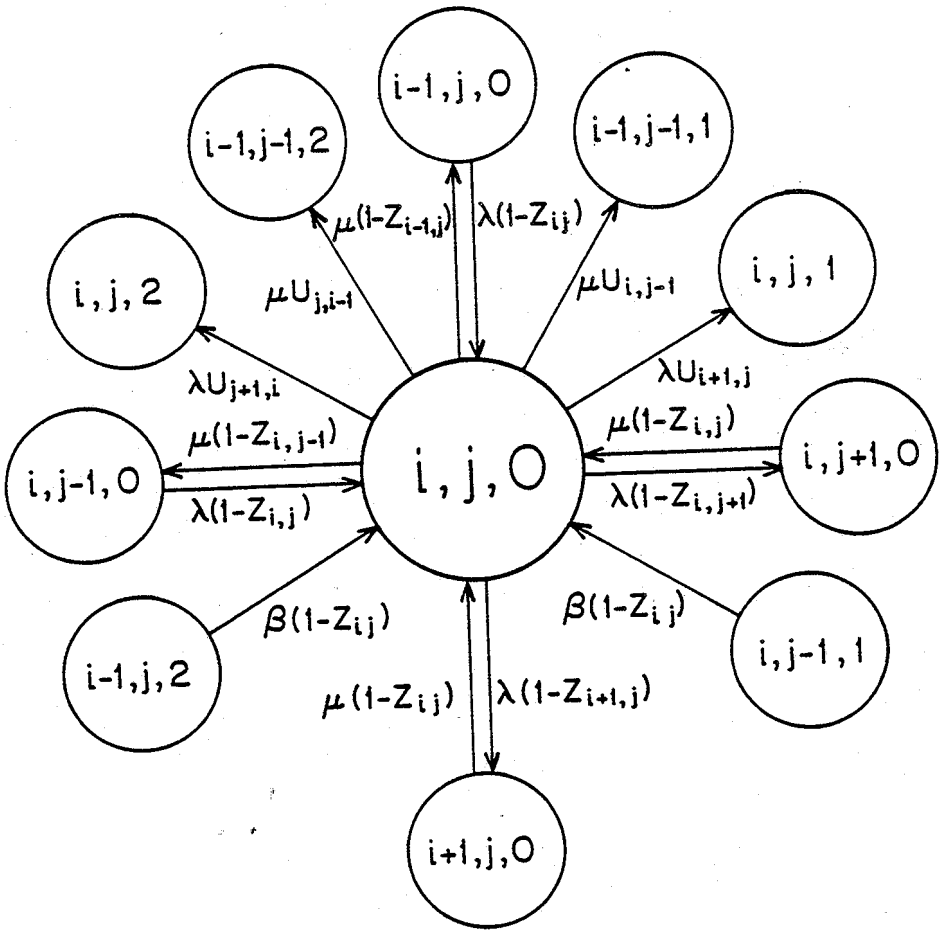

Figure 3.2. state-transition-rate diagram for $(i, j, 0)$ (NS-BTSQSS)

and Fig. 3.3 respectively. Note that for $(i, j, 1)$ there is a flow from $(i, j, 1)$ to $(i, j, 2)$ and vice versa. These flows represent the positive probability that a task will be migrated back, upon the termination of its previous migration.

3.4.2 The S-BTSQSS Model

Although the implementation of a stop system is more complex than that of a no-stop system, a two-dimensional state space is sufficient for modeling an S-BTSQSS system. The state of the channel in such a system can be derived at any instance t from $TD(t)$ and the migration criterion of the algorithm.

Let $p(i, j, t)$ denote the probability that $TD(t) = (i, j)$, then the transition equations of the NS-BTSQSS model are the following:

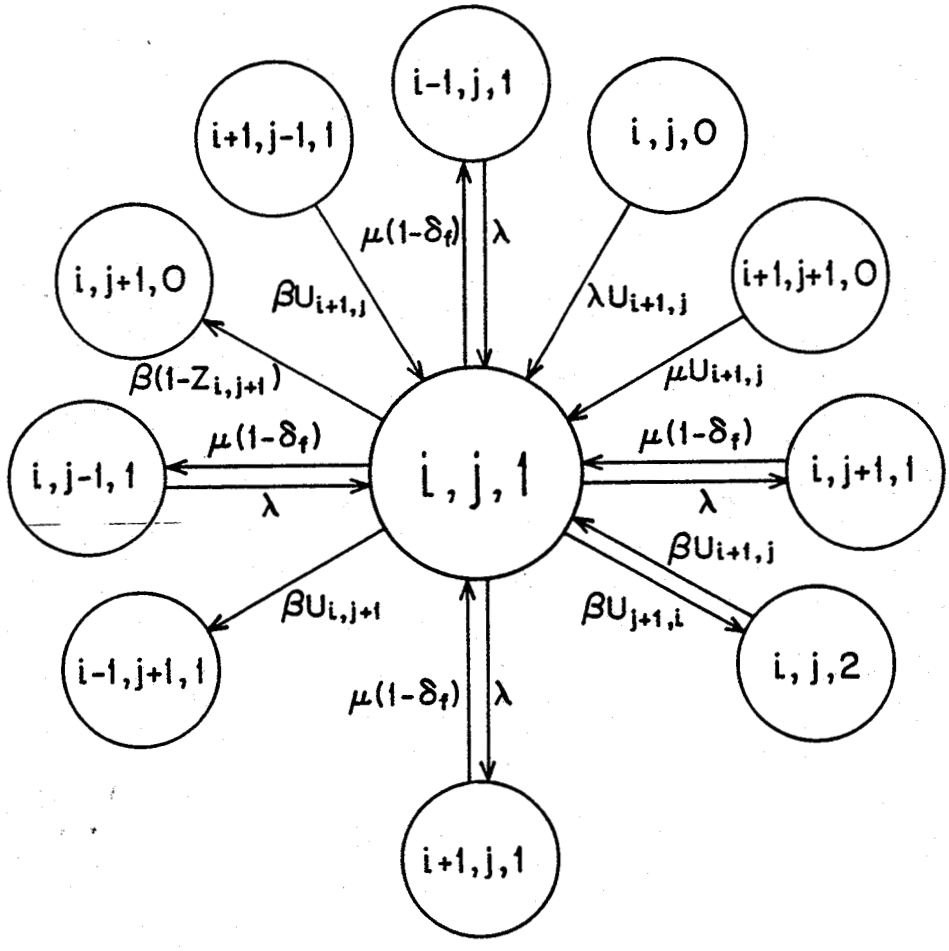

Figure 3.3. state-transition-rate diagram for $(i, j, 1)$ (NS-BTSQSS)

$$\begin{aligned}\frac{\partial}{\partial t}p(i,j,t) = &-(2\lambda + \mu(1-\delta_f z_{i,j}(e_i+e_j) + u_{i,j}\beta)\,p(i,j,t)\\ &+\mu\{(1-\delta_f\, z_{i+1,j})p(i+1,j,t) + (1-\delta_f\, z_{i,j+1})p(i,j+1,t)\}\\ &+\lambda\{p(i-1,j,t)+p(i,j-1,t)\}\\ &+\beta[u_{i+1,j-1}\,p(i+1,j-1,t) + u_{j+1,i-1}p(i-1,j+1,t)\}\end{aligned} \tag{2}$$

where $u_{i,j}$ and e_i are as defined in (1), and $p(-1,i,t) \triangleq p(j,-1,t) \triangleq 0$ for **all** $i,j > 0$. The state-transition-rate diagram for the S-BTSQSS model is presented in Fig. **3.4.**

$3.5 Price and Benefit of *a* Transfer

Whenever a control element of the LB algorithm is evoked the 'transmission dilemma' is faced and a decision whether to transfer a task has to be made. Since the LB algorithm aspires

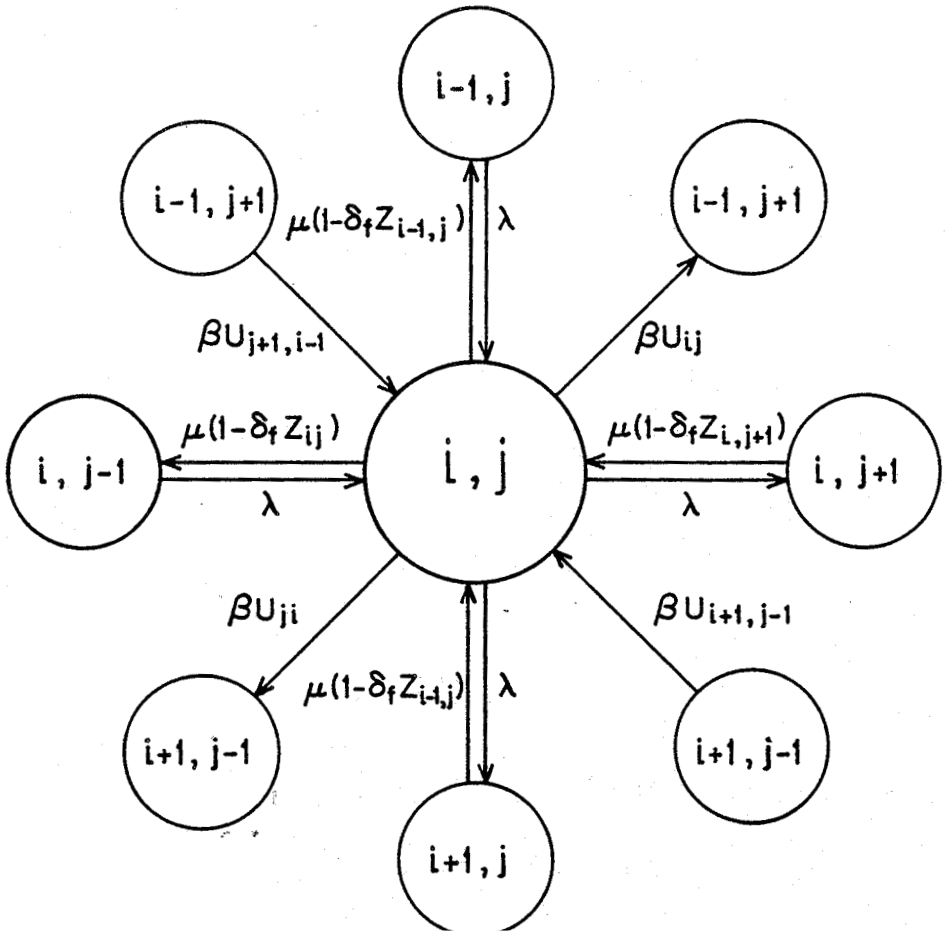

Figure 3.4. state–transition–rate **diagram (S-BTSQSS)**

to minimize the response time of the system, the algorithm is evaluated according to the net impact these decisions have on the expected queueing time of a task. This performance measure, which is a long-range steady-state measure, is affected by each individual migration. Therefore in order to establish an understanding for the relation between the control law of the algorithm and the performance of the system the properties of a transfer ought to be scrutinized. In this section a study of the 'price' and 'benefit' of a transfer are presented. The study is based on the transient behaviour of the **BTSQSS,** and all the performance measures were obtained from the differentialdifference equations of the model by means of continuous simulation.

By migrating a task from one queue to the other the LB process reduces the probability of a WI state, $P_{wi}(t)$, in the future. Fig. **3.5** presents $P_{wi}(i,j,t)$ which is defined as:

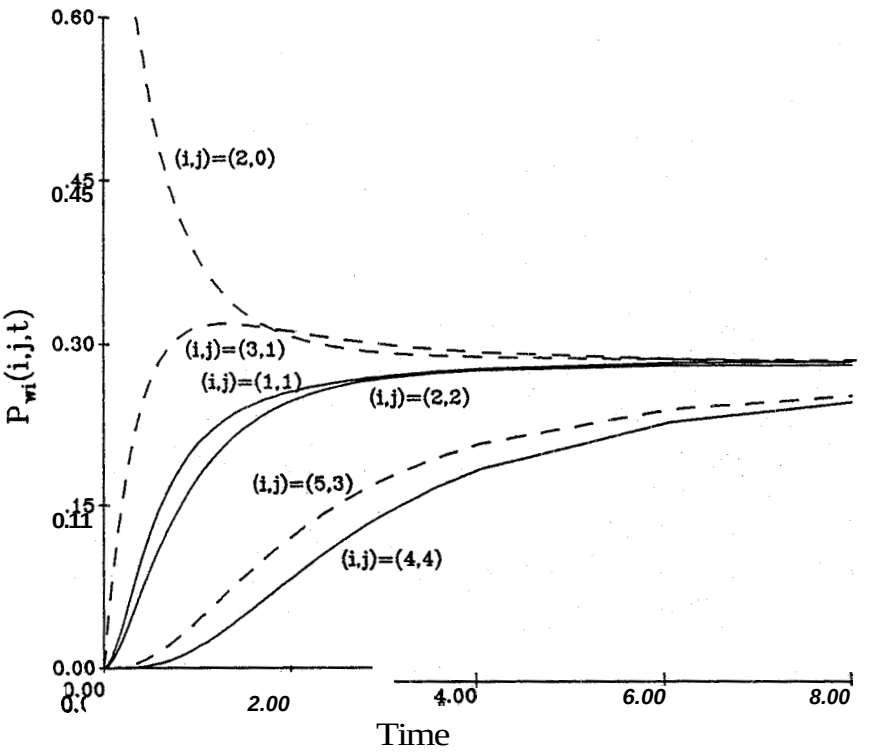

Figure 3.5. $P_{wi}(i,j,t)$ for a BTSQSS system with no migration ($\lambda = .8$)

$$P_{wi}(i,j,t) \triangleq P[\textit{of a WI state at time } t \mid TD(0) = (i,j) \textit{ and no transfers in the interval } (0,t)] \quad (3)$$

for different initial task distributions and with t as an argument. The curves of Fig 3.5 demonstrate the effect that a single task transfer has on $P_{wi}(t)$. Note that by changing the system task distribution from (3,1) to (2,2) the probability of a WI state is considerably reduced.

However, the migration of a task has penalties associated with it. One of them is an increase in the probability of a WI state during the transmission period (only in an NS-BTSQSS system). Fig. 3.6 presents $P_{wi}(2,0,t_t,t)$ which is defined as:

$$P_{wi}(i,j,t_t,t) \triangleq P[\textit{of a WI state at time } t \mid TD(0)=(i,j) \textit{ and a transfer was initiated at } t=0 \textit{ and terminated at } t_t] \quad (4)$$

for different values of t_t. From the figure one my conclude that when the transfer time is 'too long' a task transfer should not be initiated when TD(t)=(2,0) because the price of the transfer is higher than its benefits. These two figures demonstrate the dilemma which the load balancing process is faced with. For every $TD(t)$ the 'price' and the

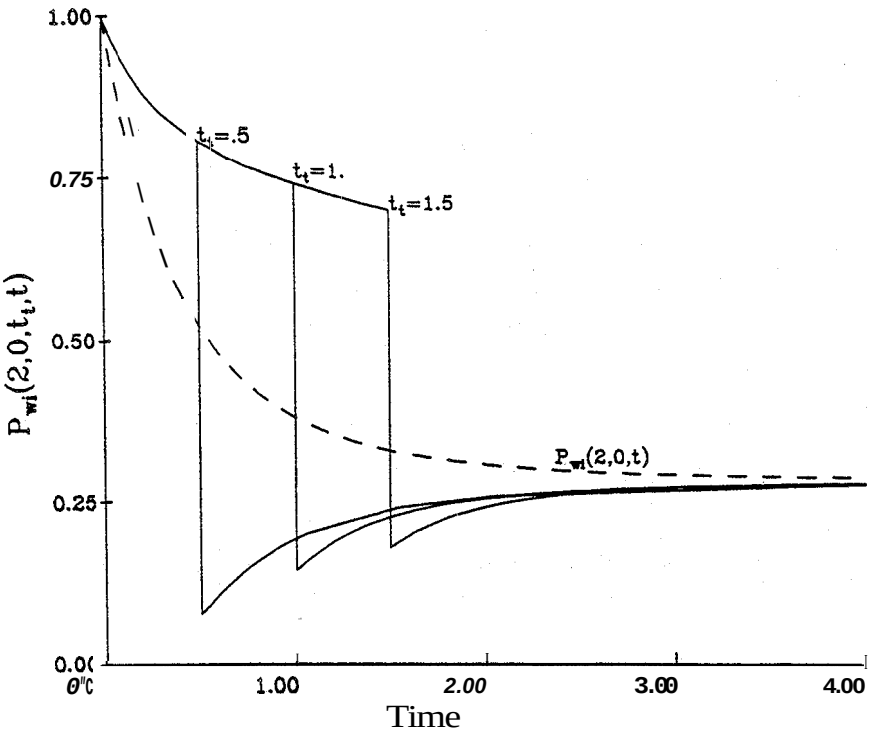

Figure 3.6. $P_{wi}(2,0,t_t,t)$ for an **NS-BTSQSS** system ($\lambda = .8$)

'benefit' of a transfer hake' to be evaluated and weighted one against the other. In order to assists this process two factors which quantitively define these two aspects of a transfer, were defined.

3.5.1 The Price of a Transfer

The transmission of a task may reduce the throughput of the system during the transmission period. This degradation is caused by either one or both of the following factors:

1. At least one task is being served when a transfer is initiated. Because of the reduction in the service rate caused by the transmission process this task will stay in the system a longer period then it would has stayed if the transfer was not initiated. The system time of other tasks that have been in the system when the transfer was initiated or that have arrived during the transfer period may be affected in a similar way.
2. In an NS-BTSQSS system a task that is being transmitted can not be served. Therefore it might happen that the server that initiated the transfer becomes idle while the task is still being transferred. In such a case the task could have been served during this period

if it had not been decided to migrate it. An NS-BTSQSS system might reach a state in which both processors are idle while a task is being transferred and thus can not be served by either one of them.

The *instantaneous throughput degradation factor*, $\mathrm{TDF}_{i,j}$ of a transfer initiated when $TD(0) = (i,j)$ is given by:

$$\mathrm{TDF}_{i,j} = \int_0^\infty (L_{i,j}(t) - \hat{L}_{i,j}(t)) f_{tr(i,j)}(t) dt \tag{5}$$

where:

$L_{i,j}(t)$ is the expectation of number of tasks in the system at time t, $(N(t))$, given that a transfer has been initiated at $t = 0$ when $TD(0) = (i,j)$ and has not terminated in $(0,t)$.

$\hat{L}_{i,j}(t)$ is the expectation of $N(t)$ given that $TD(0) = (i,j)$ and no transfer was initiated in $[0,t)$.

$f_{tr(i,j)}(t)$ is the probability density function (p.d.f) of the length of a transmission initiated at $t = 0$ when $TD(0) = (i,j)$.

The expressions for $L_{i,j}(t)$, $\hat{L}_{i,j}(t)$ and $f_{tr(i,j)}(t)$ are given in Appendix A. $\mathrm{TDF}_{i,j}$ will be considered as the 'price' of the transfer and will be used for deriving guidelines for the development of migration criteria.

3.5.2 The Success Factor of a Transfer

Tasks are migrated in order to reduce the queueing time of tasks that reside in the system and of those that will arrive at it in the future. The benefit of a transfer is its effect on the system's behaviour after it was successfully completed. No method has been found by which the contribution of a single transmission to the overall performance of the system can be evaluated. In the absence of such a measure the only way to evaluate the contribution of a transfer is to study its effect on the unbalance factor of the system.

Not all transmissions result in a reduction in this factor. In an S-BTSQSS system a transmission may be stopped in the middle and thus have no effect on the load distribution of the system. In an NS-BTSQSS a transfer may even cause an increase in the unbalance factor of the system when the load of the sender at the end of the transfer is smaller than the load of the receiver. The probability that a transmission initiated when $TD(t) = (i,j)$ will cause a reduction in the unbalance factor of the system, is defined as the *success factor*,

$SF_{i,j}$, of a transmission. The expression for this factor for the two types of systems is given in Appendix **A.**

Note that the probability that **a** transfer will cause an increase in the load-difference of the system should be included in the 'price' of the transfer, However, since **no** way was found to evaluate the effect of such a 'wrong' transfer on the future behaviour of the system it was not included in the definition of the 'price'.

3.5.3 Case Study

The impact of the initial conditions and the parameters of the **AT** algorithm on $TDF_{i,j}$ and $SF_{i,j}$ have been analized. The transition quation–for the probability functions included in the expressions for the two factors and the expressions themselves (see **A.4)** were solved by means of ***numerical integration.*** The time dependent model defined by these equations and expressions **was** translated into a C-SIMSCRIPT II.5[2] program.

Some of the results obtained from these solutions are given in Figures **3.7, 3.8** and **3.9.** Each figure consists of four graphs which present $TDF_{i,j}$ and $SF_{i,j}$, for both types of systems, with β as a parameter. The first two figures demonstrate the properties of two transmissions whose initial task distributions were $(2,0)$ and $(5,3)$ respectivly. The third graph presents the relation between.the A_p factor of the AT algorithm and the $TDF_{i,j}$ and $SF_{i,j}$ of **a** transfer which was initiated by a processor with 5 tasks in its queue. The number of tasks in the other queue is the maximal number which permits the initiation of a transmission for the given value of the anticipatory factor.

From these three graphs the following can be concluded

1. In a no-stop system with $\beta < 2$, a transfer should not be initiated by a processor with less than three tasks. $TDF_{2,0}$ for this type of systems is considerably high and $SF_{2,0}$ is less than **.6.** As can be seen from Fig. **3.8,** $TDF_{5,3}$ is relativly low and $SF_{5,3}$ is about **.7.** Therefore it is suggested that under such conditions a processor will. wait until more than two tasks will be waiting in its queue before shipping out one of them. Note that since $TDF_{2,0}$ is almost independent of δ, the cause for throughput reduction in this case is the increase in $P_{wi}(t)$ (see Fig. **3.6).** $TDF_{2,0}$ for $\delta = 0$ is almost the same as $TDF_{5,3}$ for $\delta = 10$.

---

[2] C-SIMSCRIPT [CACI76] is a combined (continuous & discrete event) simulation language. The continuous part of the language is **based on** the *Range - Kutta* [Rals65] method for numerical integration.

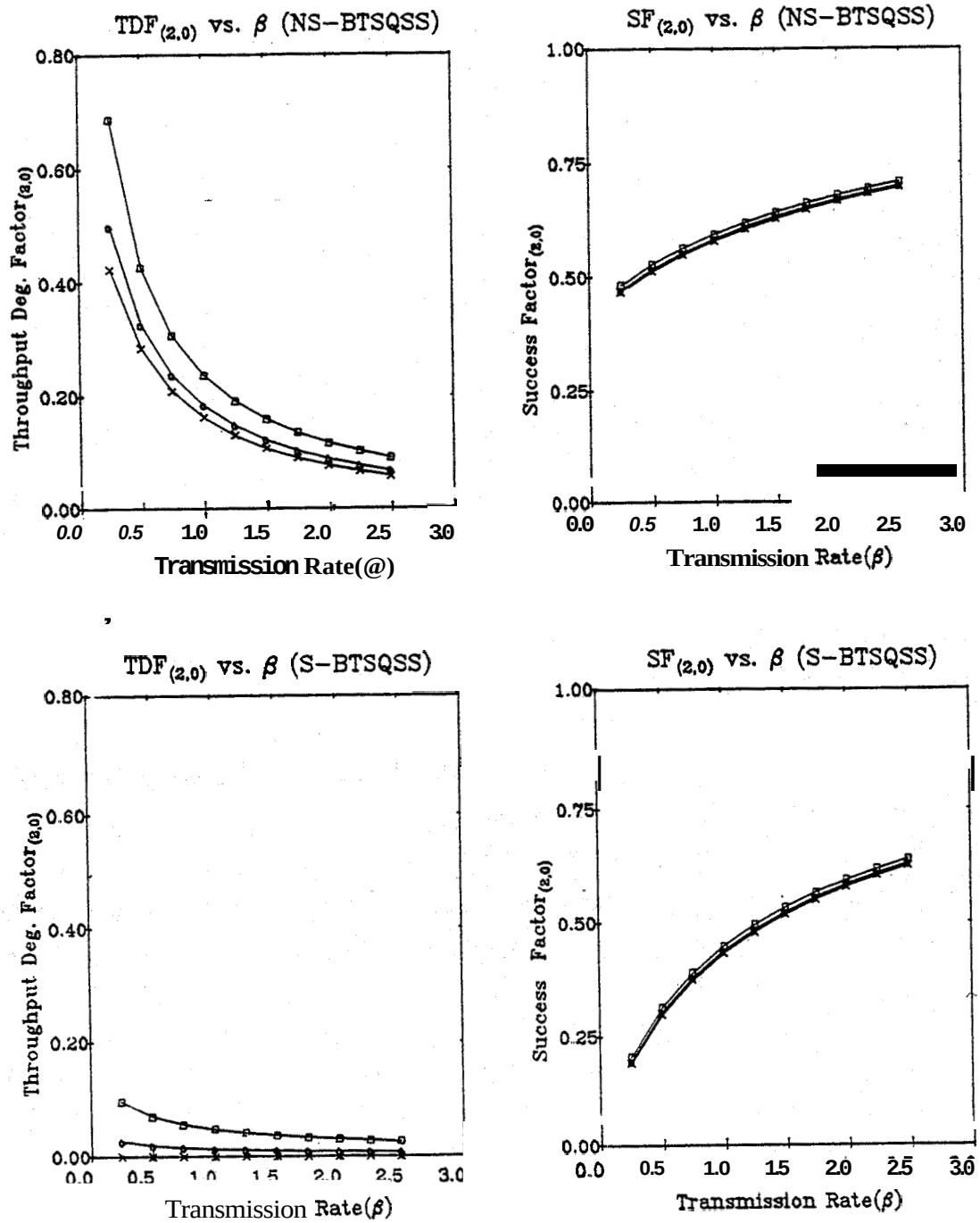

Figure 3.7. $\mathrm{TDF}_{i,j}$ and $\mathrm{SF}_{i,j}$ for $(i,j) = (2,0)$

Benefit. @ Price of Task Transfer for TD(0)=(5 ,3)

⟨ λ=.8 , A_i=L_i=0 ; × δ=0% ; o δ=3.% ; □ δ=10.% ⟩

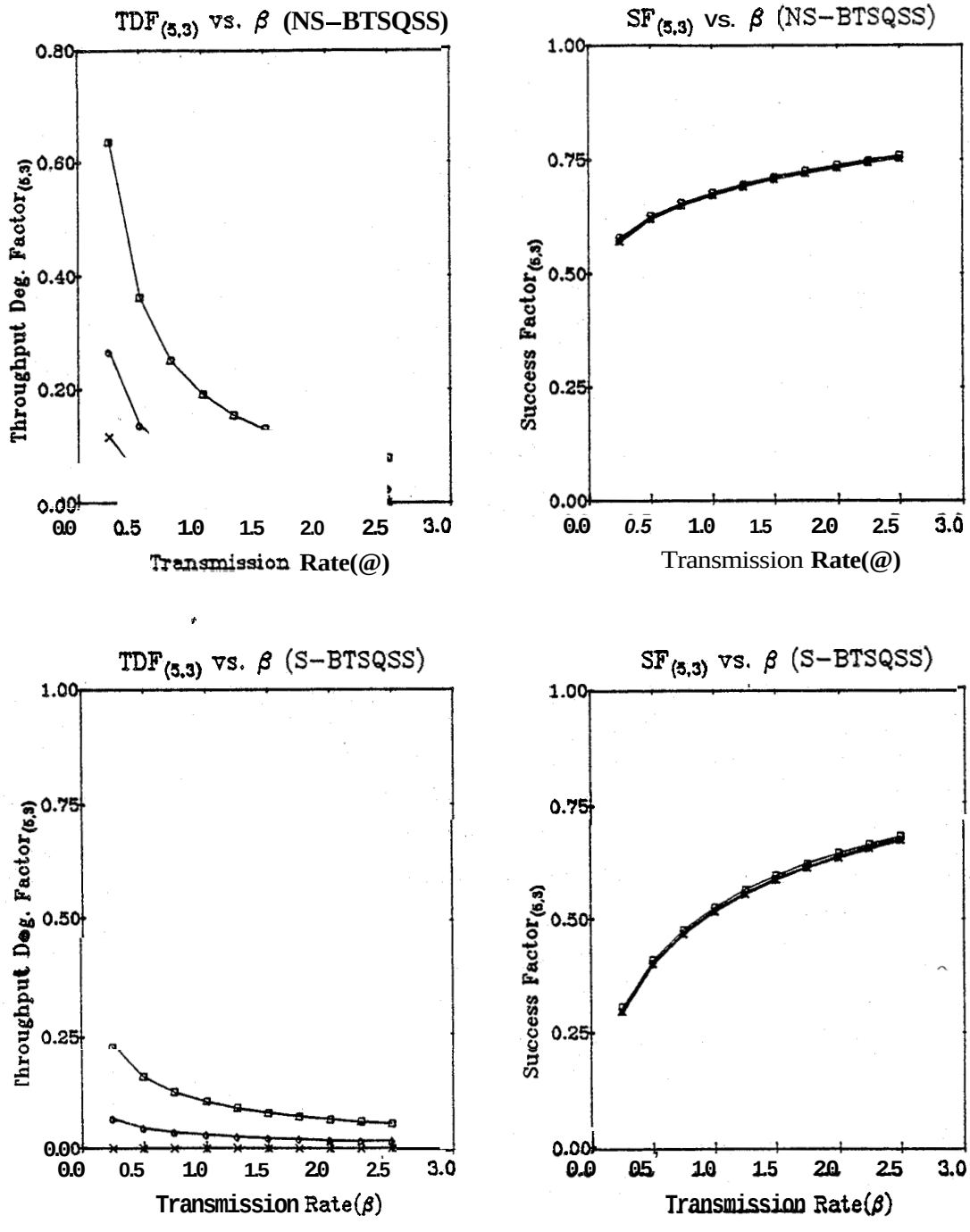

Figure 3.8. $\mathrm{TDF}_{i,j}$ and $\mathrm{SF}_{i,j}$ for $(i,j) = (3,5)$

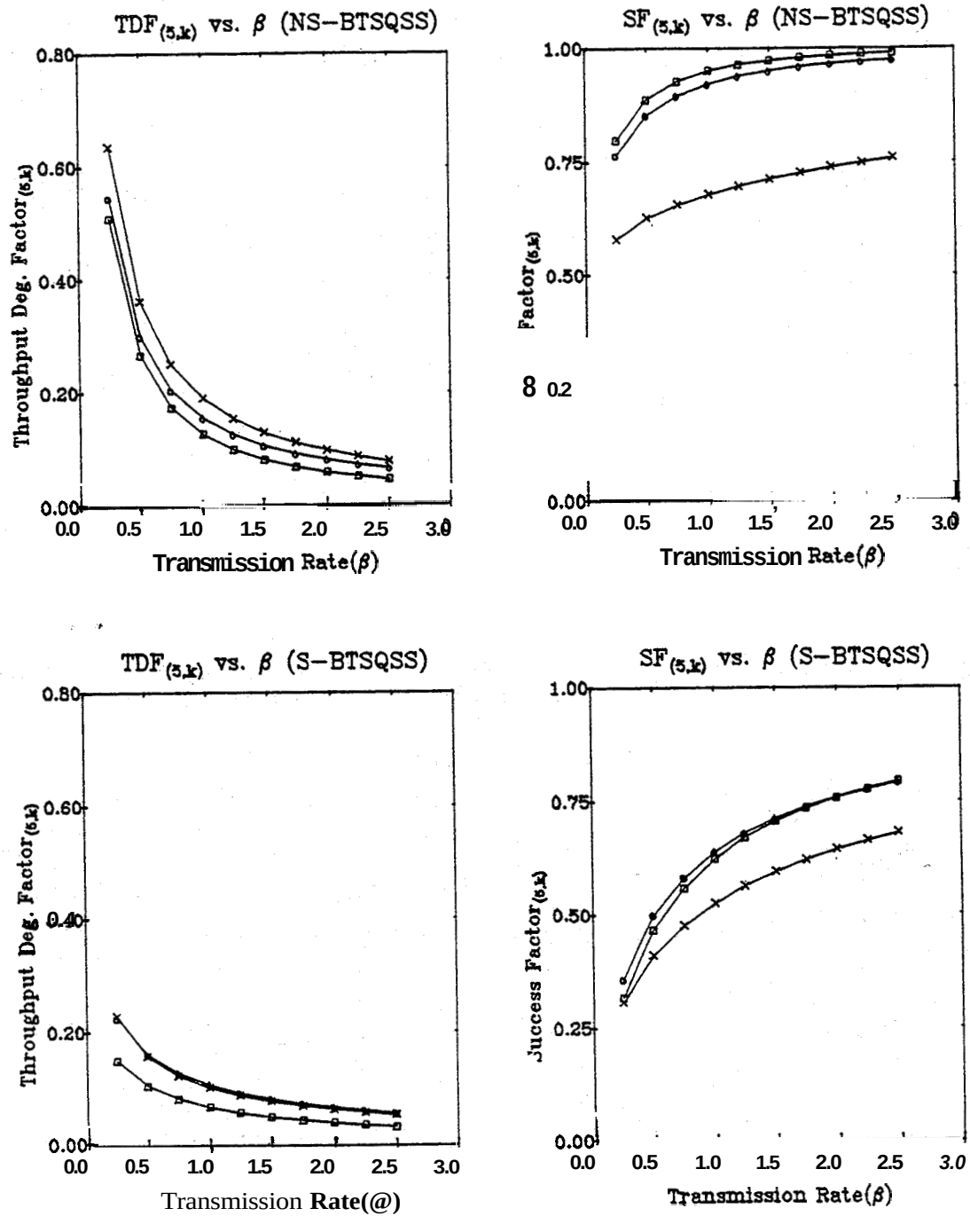

Figure **3.9.** $TDF_{i,j}$ and.$SF_{i,j}$ for $(i, j) = (5, k)$

models. The method is discussed in the coming section, and a performance study based on the performance measures that were obtained is presented in the following sections.

3.6.1 The Iterative Solution Method

The coefficients matrix of the linear transition equations which define the model falls into the category that is commonly solved by iterative methods. A number of such methods for general linear equations [Rals65] and transition equations [Gave76] have been developed. Brandwajn in [Bran79] has presented the "always converging scheme" for solving the balance equations of two-dimensional birth and death processes. As indicated by its name the method has an unconditional convergence and does not require normalization steps. Because of these properties and due to the low computational complexity of its iteration this method has been selected for solving the BTSQSS models.

The method has been extended for three-dimensional processes and has been implemented in PASCAL.[3] Table 3.1 presents the iterative step for the S-BTSQSS model. The conversion criterion used for both models is based on the difference between two consecutive iterations, of the value of fhe conditional probability of having n_1 tasks in the first server, given that there are n_2 tasks in the second, $p(n1 \mid n2)$.[4] The maximal value of the above difference for all feasible values of $n1$ and $n2$ for the iterations i and $i-1$ will be denoted by $dcon_i$. In the implementation of the two dimensional model the iterative process will terminate after the first iteration for which $dcon_i < 10^{-6}$. As a result of the non-monotonic behaviour of $dcon_i$ in the case of the three dimensional model it was decided to terminate the iterative process only when $dcon_i$ was smaller than 10^{-6} for the last 100 iterations. The thresholds used in the conversion test were selected empirically on the basis of a number of case studies.

Iterative solution schemes for a birth and death process assume a finite space-state. The value at which the state variables of the BTSQSS model are truncated affect the quality of the derived approximated solution. This truncation causes a degradation of the arrival rate due to customers' *rejection* and thus may distort the probability distributions. Since the size of the state-space affects the computation time of each iteration and their number, a space that causes a marginal degradation of the arrival rate and keeps the computation

---

[3] Two programs were written - one for each model,

[4] For the sake of this definition a task being transferred belongs *to* the source processor in both models.

$$p^k(i,j) = \{1+\varphi\{-[\lambda(e_i+e_j)+u_{j,i}\beta]p^k(i,j)\}^{-1}\{[1-\varphi[\mu(e_i+e_j)+u_{i,j}\beta]]p^{k-1}_{i,j}$$
$$+\varphi\{\lambda[p^k_{i-1,j}+p^k_{i,j-1}]$$
$$+\mu[(1-\delta_f z_{i+1,j})p^{k-1}_{i+1,j}+(1-\delta_f z_{i,j+1})p^{k-1}i,j+1]$$
$$+\beta[u_{i+1,j-1}p^{k-1}_{i+1,j-1}+u_{j+1,i-1}p^k_{i-1,j+1}]\}\}$$

where the superscript k denotes the value at the k^{th} iteration, $\varphi = [\max(\mu,\beta)]^{-1}$ and $p^k(-1,j) = p^k(j,-1) = p^k(N+1,j) = p^k(j,N+1) = 0$ for all $k > 0$ and $0 \leq j \leq N$

Table 3.2. The iteration step for an **S-BTSQSS** model

Attribute	N=20	N=25	N=30	N=32	N=35	N=37	N=48
W_q	4.62	4.95	5.00	5.03	5.06	5.07	5.08
P	0.90	0.90	0.90	0.90	0.90	0.90	0.90
P[reject]	$3\,10^{-3}$	$6\,10^{-4}$	$3\,10^{-4}$	$2\,10^{-4}$	$1\,10^{-4}$	$4\,10^{-5}$	$1\,10^{-5}$
# iter.	591	899	1029	1114	1239	1442	1676
time(sec)	17.0	45.7	64.2	78.6	104.1	157.0	239.7

$\beta = .5, A_p = L_p = 0$

Table 3.2. Attributes **of** the iterative method

time at a practical level, has to be selected. Table **3.2** presents the various attributes **of** the solution process with space size **as** an argument. Using the data presented in the table and information which was derived from similar case studies, a space-state of **35** has been selected for each processor. It was found that for all cases that where solved in the course of the study the rejection probability **was** less than 10^{-3}.

§3.7 Performance **study**

The expected normalized queueing time of a task in both types of systems and under various operating conditions are shown in Fig. **3.10** and **3.11.** The curves of the figures demonstrate the impact of the migration process on the response time of the system. The upper curves in the two figures represent the $\hat{W}_q$ of the system if no task migration takes place ($BD_{1,2} = \infty$). The lower bound for $\hat{W}_q$ is given by the curve for $BD_{1,2} = 0$. The two figures are a clear display of the significant gain that can be achieved by migrating tasks in a distributed system. Even when $\delta = 10$ and $BD_{1,2} = 2$ the LB algorithm considerably improves the performance of the system. Note that when $BD_{1,2} = 1.$, the response time of the system is almost as good as the response time of an M/M/2 system. Therefore when the expected transfer time

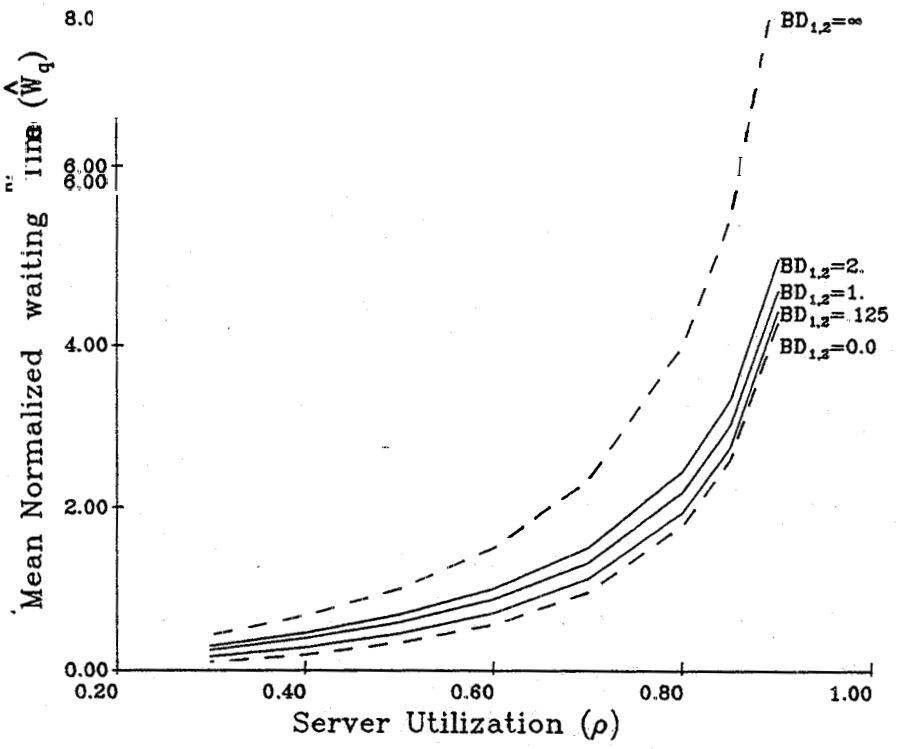

Figure 8.10. $\hat{W}_q$ *vs.* p for a **S-BTSQSS sytem** ($\delta = 0, A_p = L_p = 0$)

f a task is less then half, its expected execution time of the two servers can be considered **as** a single M/M/2 system. The relation between the balancing distance of the system and s performance, **as** demonstrated by the figures, lead to the conclusion that systems with alancing distances in range of .5 to 2., had to be used in the study of the migration criteria.

3.7.1 Channel Utilization

It was shown in **2.2.1** that the load balancing process may require a high rate of task transfers. By limiting the rate at which anticipatory transfers are initiated this rate can be significantly reduced. However, such a reduction may cause an increase in the P_{wi} and thus an increa in the response time of the system. The degree to which the **AT** algorithm initiates such transfers is controlled by the A_p parameter.

The effect of the value of A_p on the utilization of the channel and $\hat{W}_q$ for a stop system with $\delta = 0$ is shown in **Fig. 3.12** and Table **3.3.** Note that at the price of a small increase in the expected queueing time of a task, the utilization of the channel can be considerably reduced. For highly utilized systems an increase in the arrival rate causes a

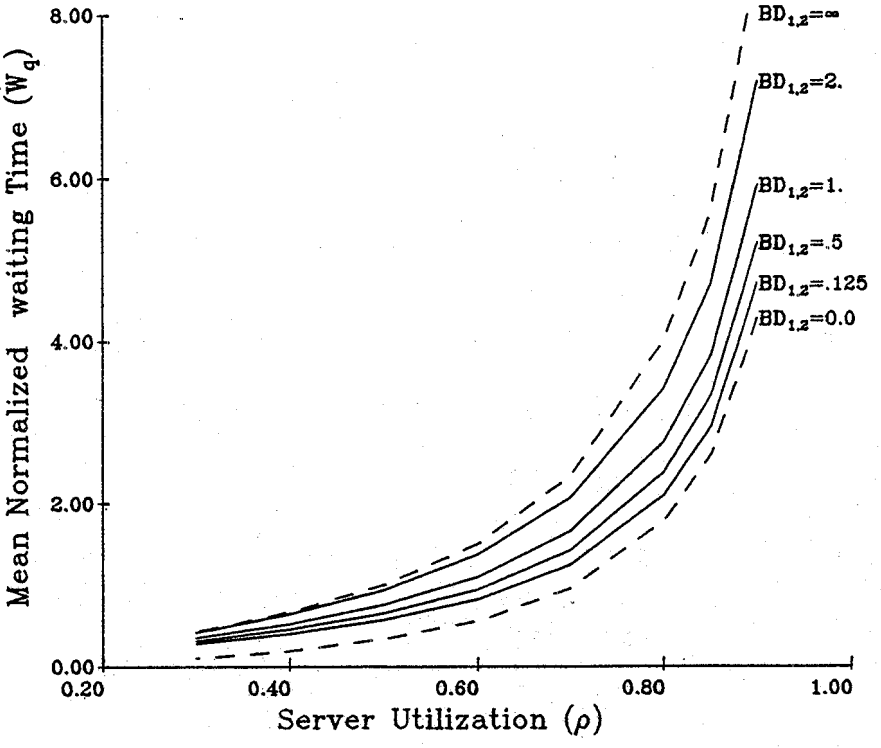

Figure 3.11. $\hat{W}_q$ *vs.* ρ for an NS-BTSQSS sytem ($\delta = 10, A_p = L_p = 1$)

A_p	$\rho = 9$	$\rho = 85$	$\rho = 8$	$\rho = 7$	$\rho = 6$	$\rho = 5$	$\rho = 4$	$\rho = 3$
0.	4.67	3.04	2.19	1.32	0.87	0.59	0.39	0.25
0.5	4.69	3.04	2.19	1.32	0.87	0.59	0.39	0.25
1.0	4.75	3.09	2.22	1.33	0.87	0.59	0.39	0.25
1.5	4.78	3.12	2.24	1.34	0.88	0.59	0.39	0.25
2.0	4.87	3.19	2.30	1.38	0.90	0.60	0.40	0.25
2.5	4.90	3.21	2.31	1.38	0.90	0.60	0.40	0.25
3.0	4.97	3.27	2.36	1.41	0.92	0.61	0.40	0.25

Table 3.3. $\hat{W}_q$ of S-BTSQSS system ($\delta = 0, BD_{1,2} = 1., L_p = 0$)

decrease in the channel utilization when $A_p > 0$. This property of the AT algorithm might be critical when the transmission process reduces the service rate of the processors ($\delta > 0$). The optimal value for A_p depends on the ***cost function*** which is assigned to the waiting time of a task and the usage of the channel.

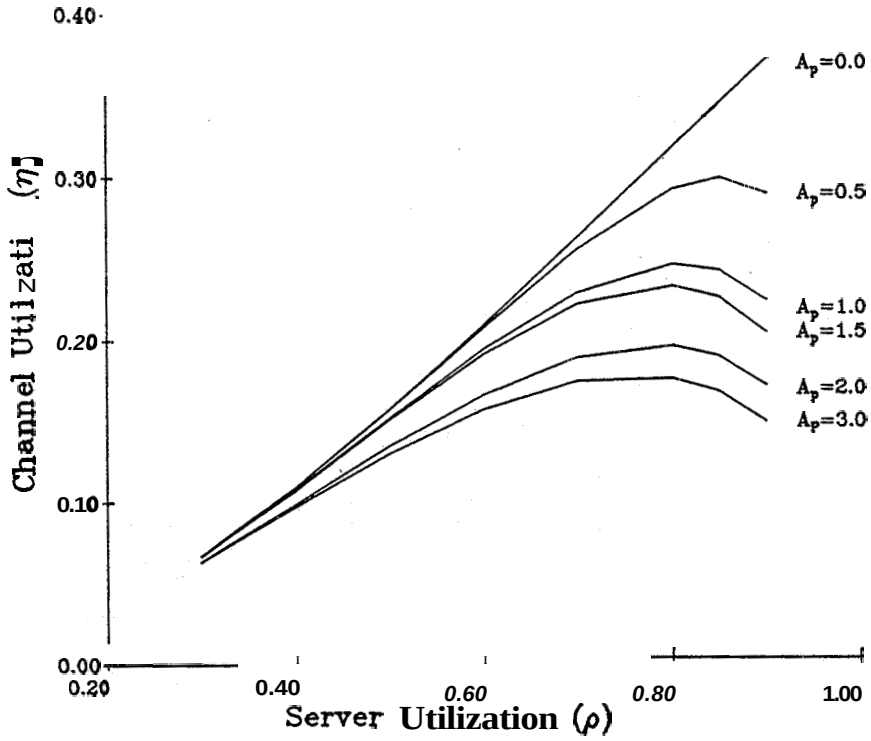

Figure 3.12. Channel utilization ***vs*** ρ for S-BTSQSS system ($\delta = 0, L_p = 0, BD_{1,2} = 1$)

3.7.2 The Migration Criterion

The effect of the migration policy on the normalized expected queueing time of a task, $\hat{W}_q$, is demonstrated by **Fig. 3.13, 3.14** and **3.15.** The figures present $\hat{W}_q$ of the two systems with A_p and L_p as parameters for different server utilization and service degradation factors. The values presented in these figures indicate the strong impact that the values assigned to A_p and L_p have on the expected turnaround time of a task. Note that for no-stop systems with $\rho = .9$ and $\delta = 10\%$, $\hat{W}_q$ is greater than that of an equivalent M/M/1 system if the **'wrong'** migration criterion, ($A_p = 0, L_p = 0$), is selected, whereas the $\hat{W}_q$ of the same system with the 'correct' criterion ($A_p = 2$) will be **30%** smaller than in an M/M/1.

For most values of A_p and L_p a stop system has a smaller $\hat{W}_q$ then an equivalent no-stop system. In some cases the difference between the performance of the two systems with the same migration criterion may be considerably large. However, when each of the systems is controlled by a migration policy that is best suited to the system's attributes, the difference will, in most cases, be less than **5%** . From the above observation it follows that the ability of the server to control the operation of the channel does not considerably improve the efficiency of the balancing process although the penalties associated with the transmission of

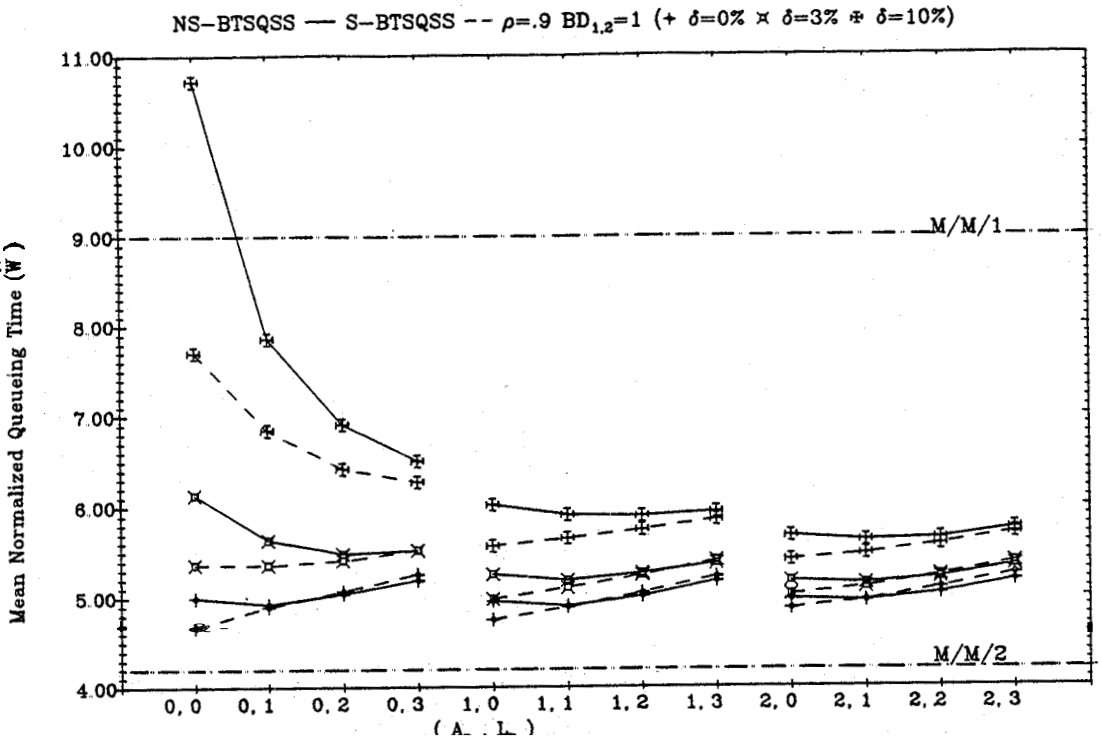

Figure 3.13. $\hat{W}_q$ *vs.* Algorithm Parameters ($d_{1,2} = 1, \rho = .9$)

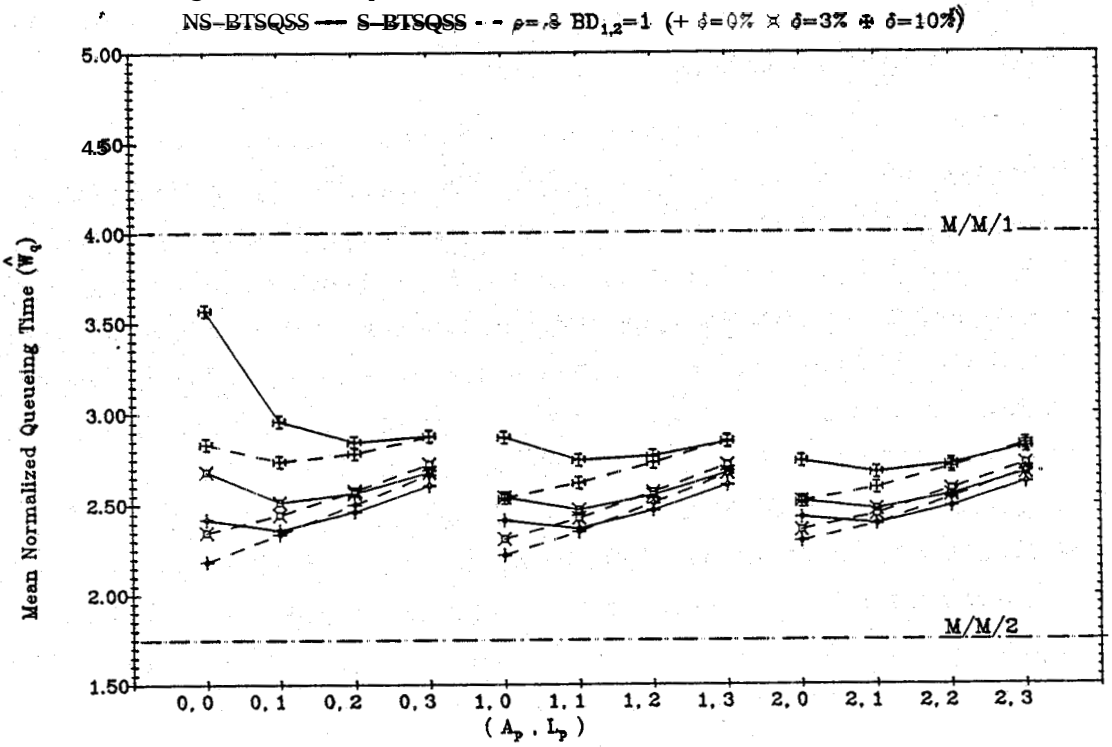

Figure 3.14. $\hat{W}_q$ *vs.* Algorithm Parameters ($d_{1,2} = 1, \rho = .8$)

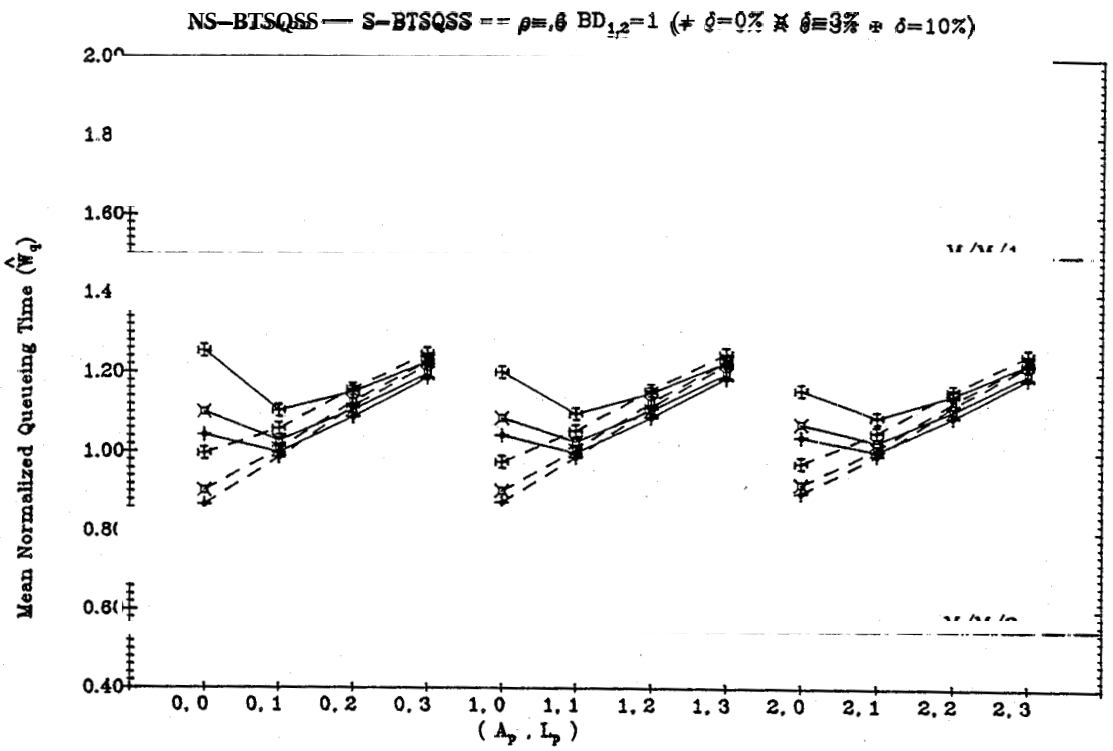

Figure 3.15. W_q vs. Algorithm Parameters ($d_{1,2} = 1, \rho = .6$)

a task are higher in an **NS-BTSQSS** system.

Results obtained for systems with different balancing distances are presented in Figs. **3.16** and **3.17.** These two figures together with Fig. **3.1.4** demonstrate the relation between the balancing distance between the two processors and the manner in which a migration policy affects the performance of the system. Note that even when the distance is **2** (the expected transmission time of a task is twice *as* long as it's expected service time) and when $\delta = 10\%$, as in Fig **3.16,** the $\hat{W}_q$ **is** reduced by **20%** relative to an M/M/1 system.

From all the results presented above the following guidelines for the selection of a migration criterion can be concluded

1. When $BD_{i,j} \leq 1$ by selecting $A_p = 1$ the changes in $\hat{W}_q$ due to an increase in δ (up to 10%) can be kept low. For all the cases that where analyzed it was found that this difference can be kept below **10%.**
2. In a stop system 'last-minute' transfers should be initiated regardless of the queue size of the initiator. In all cases (except when A_p is too **small** Fig. 3.13) assigning a non-zero value to L_p caused an increase in
3. For all no-stop systems with $BD_{1,2} \geq .5$, the last-minute parameter should be greater than zero. In such systems a 'last-minute' transfer should be initiated **only** when there

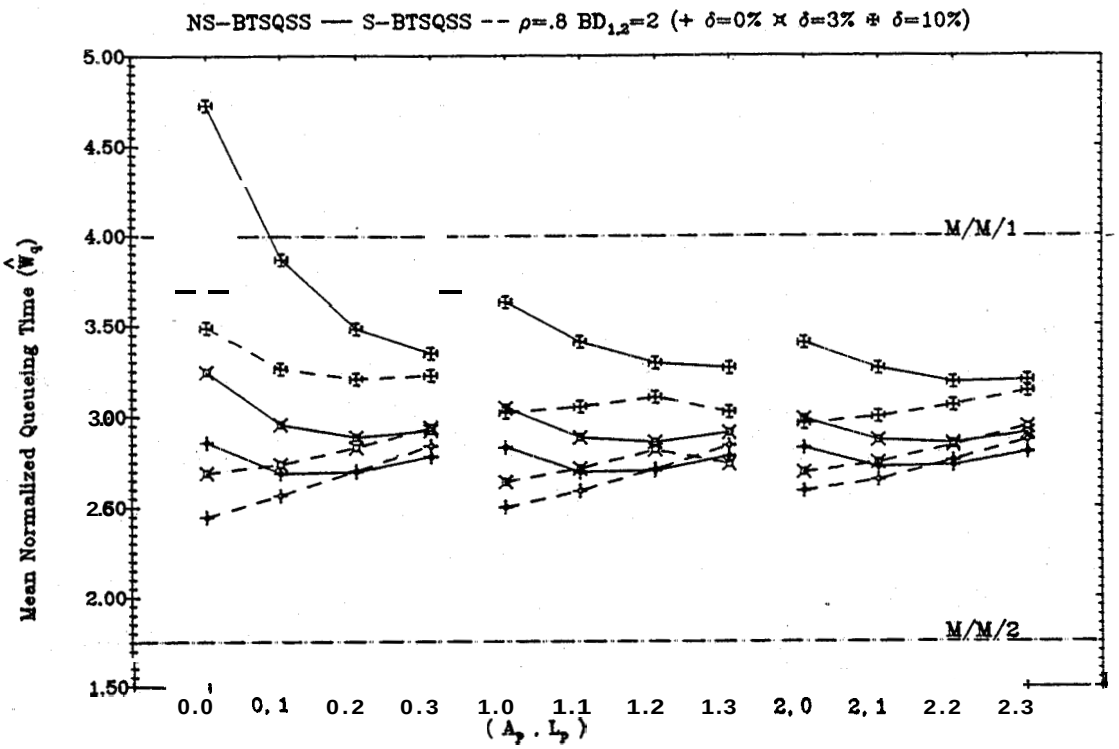

Figure 8.16. $\hat{W}_q$ *vs.* Algorithm Parameters $(d_{1,2} = 2, \rho = .8)$

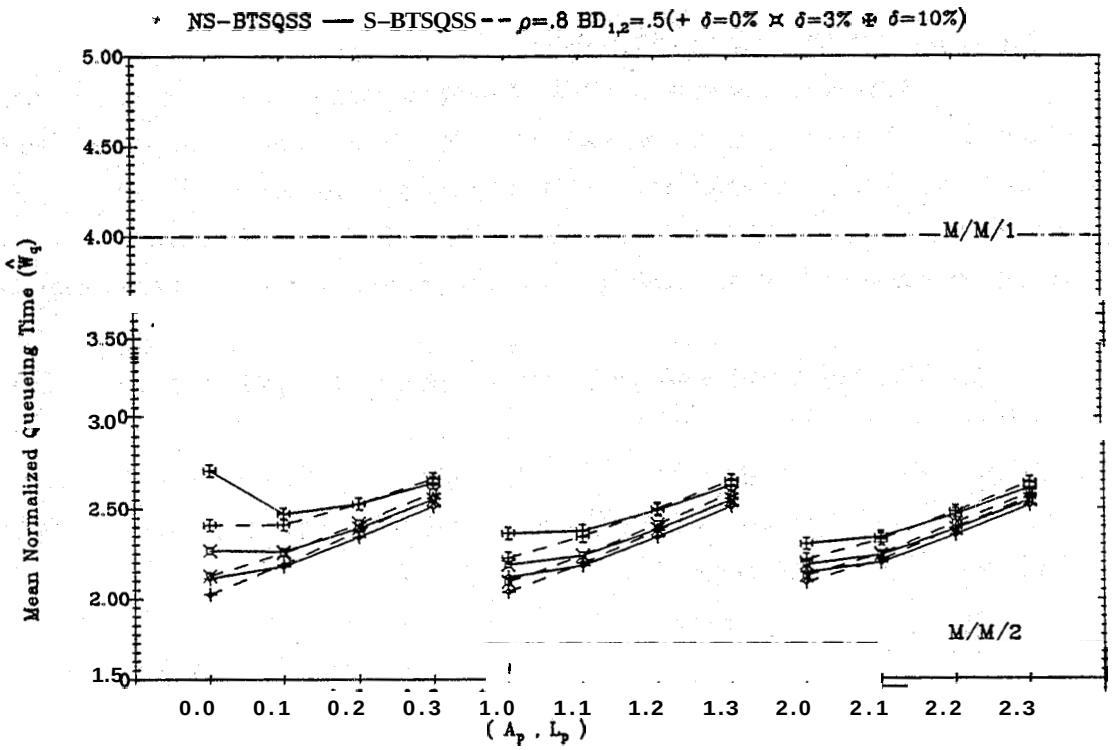

Figure 8.17. $\hat{W}_q$ *vs.* Algorithm Parameters $(d_{1,2} = .5, \rho = .8)$

are more than two tasks in the system.

4. In systems where $\delta \geq 3$ and $BD_{1,2} \geq .5$, 'anticipatory transfers' should be initiated only when the load-difference of the system is large compared to the total number of tasks in the system. Therefore the anticipatory parameter for such systems has to be non-zero.

Chapter 4

Broadcast Distributed Systems

§4.1 Introduction

Broadcast communication subnets are widely used for interconnecting processing elements. A broadcast subnet consists of a single communication channel which is shared by all the switching elements of the system. An arbitration mechanism, centralized or distributed, is required for resolving conflicts when two or more stations attempt to transmit simultaneously. Although cities or even continents are linked together by broadcast channels [Abra77], in most cases the elements that communicate via a broadcast medium are less than a few kilometers apart. The processing units are usually located in the same room, same building or at the same institute and thus establish a *local* network. The increasing demand for *office automation* and distributed processing motivated the development of various protocols and high bandwidth channels for this type of networks. High communication capacity is a distinguishing feature of local networks. The bit rate of their communication channels is in the range of .1–30 $Mbit/sec$. An erlier version of these algorithms has been presented in [Livn82].

In this chapter three load balancing algorithms for broadcast m*(M/M/1) systems are defined and the results of a simulation study of their performance is presented.

§4.2 The Broadcast Model

The model describes a broadcast m*(M/M/1) distributed system with homogeneous processors and users (Fig. 4.1). The communication subnet of the model is a passive broadcast

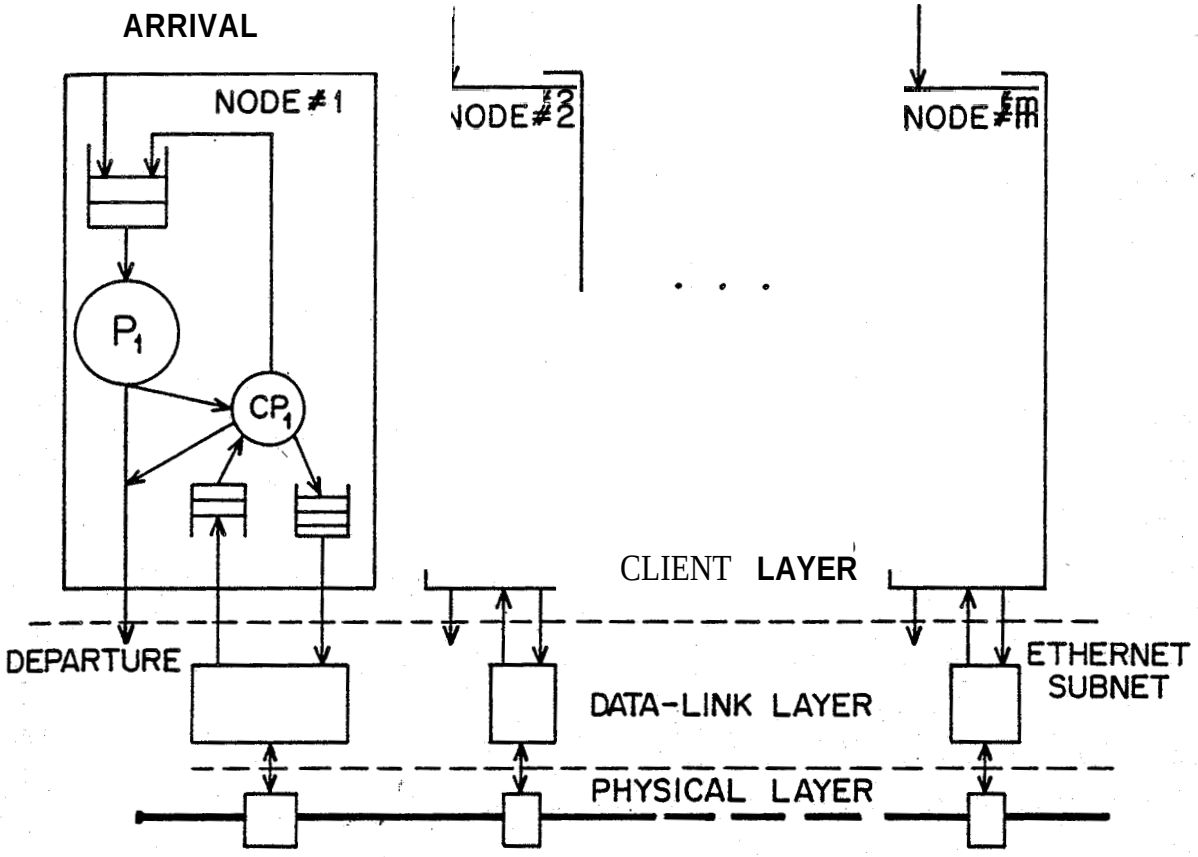

Figure 4.1. The Broadcast m*(M/M/1) model

medium with the ETHERNET [Metc76] communication protocol. The communication activities of a node are controlled either by the processor or by a dedicated *Communication Processor,* CP, that serves as a ***front-end*** for the node. Each node maintains an input and an output message queue in which arriving and departing messages are placed respectivly. It is assumed that the buffering space available at each node is unlimited.

The processing time required for the transmission or reception of a balancing message is determined by the OVerHead parameter of the node, $OVH_i = (ovh_1^i, ovh_2^i, ovh_3^i)$. **The** processing time, at node i, required by a message whose transmission time is T ms is; $ovh_k^i + T\ ovh_3^i\ ms$, where $k = 1$ for a ***data message*** and $k = 2$ for a **control *message.*** A data message is a message that carries a description of a task or the results of its execution. The information needed for controlling the migration process is exchanged via control messages.

Each node has a front-end boolean parameter, FE_i, which determines whether the node has a CP. In a node that has no CP, $FE_i = 0$, the arrival or departure of a message

interruptes the nodal processing element. If the processor is not occupied by another message the current task is ***preempted*** and is ***resumed*** only after all processing demands made by the communication protocol or balancing algorithm are fulfilled. However, when a front-end processor is part of the node, $FE_i = 1$, messages are served by this processor and none of the processing capacity of the node is utilized by the balancing process.

4.2.1 ETHERNET PROTOCOL

The communication protocol selected for the broadcast subnet is the **ETHERNET. The ETHERNET is based** on a ***Carrier Sense Multiple Access with Collision Detection*** (CSMA-CD) access method and was first described by Metcalfe and Boggs in [Metc76]. After being in use for several years the protocol was recently given a detailed specification [Digi80] which establishes a ***standard*** for the protocol. The specification gives a detailed definition of the ***Physical and Data-Link*** layers of the protocol which are the lowest two layers of the ***Open Systems Interconnection*** (OSI) reference model [Tane81]. Several vendors have developed hardware to realize the **ETHERNET** [Hind82], [Elli82] and a considerable amount of effort has been devoted to the analysis of its performance [Stuc83].

The Physical Layer is a coaxial cable with *base-band* signalling. Its bit rate is $10 Mbit/sec$ and up to 1024 stations which are less than $2.5 Km$ apart can be interconnected by one ETHERNET network. The Data Link Layer senses the state of the cable by means of the ***currier sense*** signal. This signal is controled by the Physical Layer and indicates whether a packet is or is not present on the cable. Each Data Link *defers* transmission as long as the carrier is set and during an *interframe spacing* gap of $9.6 \mu sec$ after the carrier drops. Simultaneous transmissions by two or more stations cause a *collision* which is detected by the Physical Layer. Following the detection of a collision the ***collision detected*** signal is set by the Physical layer for each station that has participated in the collision. The contending Data Link Layers respond to the setting of this signal by transmitting a 48 bit *jam* message and by suspending any further transmission attempts for a *backoff* period. The duration of this period is determined according the ***binary exponential backoff*** scheme.

§4.3 Load Balancing Algorithms for Broadcast Systems

From the point of view of the load balancing process, broadcast communication system have two advantageous properties. The first one is the ***uniformity*** of balancing distance and the

second is the *message broadcast* property. The time required for transferring a message from one node to the other via a broadcast system is the same[1] for all pairs of nodes. Therefore the balancing distance between all nodes is equal to the balancing diameter of the system. Due to the uniformity of distance, all nodes whose load is the same are equal-priority candidates for receiving a waiting task. The control element of the LB algorithm resident at the node has to consider only the load distribution of the balancing region, which may include the entire set of processors. A large nodal balancing region provides the control law a broad view of the instantaneous system load and thus improves the ability of the algorithm to minimize the probability of a WI state.

The message broadcast facility of this type of communication system supports the information component of the algorithm in providing *global* and *updated* information about the instantaneous load distribution of the system to the control element. By sending one message a node can inform the entire system about its current state or to describe a balancing dicision it has made. The information policy is free of *touting* and *flow-control* considerations. However the broadcast property is actually a *double-edge sword* because it indicates that the system has only a single communication resource. A broadcast communication system can not transfer a number of messages simultaneously.[2] Simultaneous transmission attempts by any subset of station leads to contention and thus to queueing delays that increase the turnaround time of a message.

Using the above analysis of the characteristics of a broadcast system as a guideline three different load balancing algorithms for this type of system have been developed. An attempt was made to encapture in the three algorithms the various aspects of the task migration phenomena in broadcast systems. The algorithms differ in their information and task migration policies and each of them represents a different approach to the balancing problem. All the LB algorithms defined are for broadcast m*(M/M/1) with uniform processors and users.

4.3.1. BST Algorithm

The Broadcast STatus (BST) load balancing algorithm is a natural extention of the task

---

[1] propagation delays are negligible for such systems and thus the geographical location does *not* affect the transmission time.

[2] A number of messages can be transferred over the same medium if frequency modulation techniques are used. However in such a case the communication system is no more a single system.

migration criterion of the **BTSQSS** system (see Chap. **3.).** The algorithm is *anticipatory* hits nature, and has a *liberal* information policy that provides each node with a complete picture of the instantaneous load distribution of the system. The control-law of the algorithm aspires to keep the unbalance factor of the system below a given value. This value is a parameter of the algorithm and can be adapted to the properties of the system. The algorithm takes advantage of both the uniformity **of** the balancing distance and the broadcast facility **of** the communication network.

ALGORITHM BST (*Broadcast* every *change in state*)

Information Policy: Whenever the length of its queue changes the node broadcasts **a** *status* message that describes the new size of the queue. Each node records the information it receives via this messages in its ***Load Distribution,*** LD_i, vector. The information stored **in** this vector describes the instantaneous load distribution of the system **as** seen by node i. The node updates the element in LD_i which describes its current load, ld_i^i, upon the successful completion **of** the transmission of a status messag.

Balancing Region: The balancing region of a node is the entire system.

Control Law: Whenever $LD_i = (ld_1^i, \ldots, ld_m^i)$ is updated and the output message queue is empty the nodal LB control element is invoked. The algorithm will initiate the migration of a task from node i to node j if all the following conditions are fulfilled

1. $(ld_i^i > ld_k^i) \vee (ldf = ld_k^i \wedge i \geq k)$ $\quad for \;\; 0 < k \leq m.$ $\quad (maximal)$

   i has to be the node with the longest queue. When two or more have the maximal load, ties are broken according to node numbers.

2. $ld_j^i \leq Id);$ $\quad for \;\; 0 < k \leq m.$ $\quad (minimal)$

   j has to be the node with the shortest queue. When more than one node **has** a minimal number of waiting tasks the selection of the target node for the migrated task is done randomly.

3. $\hat{UBF}(i,t) > BT$ $\quad (threehold)$

   the unalance factor of $BR_i(t)$ has to be greater than BT which is a parameter of the algorithm.

Note that since the control element of the BST algorithm will consider a task migration **only** after all previous messages have been transferred, **all** the nodes **will base** their migration decisions **on** the same information. The BST algorithm **utilizes** the broadcast media as a means for synchronizing the distributed control process **as far as** the information on the instantaneous load distribution is concerned. The messages are analized by **all** the nodes in the same order and therefore at each stage all nodes have the same picture of *the* system load.

4.3.2 The BID Algorithm

The **Broadcast when IDle (BID)** load **balancing algorithm** is based **on** *Last Minute* task transfers. The control element of the algorithm is invoked **only** when one of the systems' nodes becomes idle. The information policy of the **BID** algorithm is less *liberal* than **the** previous one and utilizes three types of messages. like the BST algorithm this **algorithm** takes advantage of the two advantageous properties **of** the broadcast medium.

ALGORITHM BID (*broadcast when idle)*

Information Policy: A node broadcasts an *idle* message whenever it enters an idle **state.** Following the transmission **of** such a message the node receives *reservation* messages to **whom** it replies with *accept/reject* messages.

Control Law: The control element *of* the **BID** algorithm consists of the two **components** - the *loaded* and the *idle* component. The first one is invoked when **an** idle **message arrives** and consists of the following steps (for node i whose queue length is n_i):

i. If $n_i > 1$ go to step ii, else terminate the algorithm.
ii. Wait Dn_i^{-1} units of time. D is a parameter **of** the algorithm and its value depends **on** the characteristics of the communication medium (proporgation **and** round-trip delays).
iii. Send a reservation message to the node that **has** been declared idle.
iv. Wait for a reply message from the idling node.
v. If the reply **is** an accept message and $n_i > 1$, initiate a task transfer to the node which has accepted the reservation, else terminate the algorithm.

The purpose of the *state-dependent time-out* period in step ii is to give nodes with more tasks higher priority in sending reservation messages and thus to give them a better chance to migrate a **task** to the idle node. Note however, that due to **the** backoff algorithm *of* the ETHERNET it is not guarantied that the node whose message has arrived **first** will be transmitted first.

The *idle* component of the control law is invoked whenever a reservation message arrives at the node that has broadcasted an idle message. .If the node is still idle and no previous reservation has been accepted, an accept message is sent as a response to the reservation request. In all other cases a reject message is transmitted.

Balancing Region: The Balancing Region of node i at time t, $BR_i(t)$, includes all those nodes that have sent an idle message ,i has sent them a reservation message and they have not yet replied.

433 The PID Algorithm

Unlike the two previous algorithms the **Poll when IDle (PID)** load balancing algorithm does not utilize the broadcast capability of the communication system. The algorithm is based on a ***polling*** strategy and its migration criteria initiates only 'last-minute' transfers. The Polling procedure takes advantage of the uniformity of the balancing distance of the system.

ALGORITHM PID(*Poll when idle*)

Information Policy: An idling node sends ***request*** messages by which it notifies the nudes to whom the messages are directed that it is willing to receive a task. The node receives as a reply a data message which contains the description of a task or an *empty* message.

Control Law: Two components constitute the control element of the PID algorithm - the ***poll*** and *reply* components. As a node enters an idle state it invokes the poll component that consist of the following sequence of steps:

i. Randomly select a set of R nodes $(a_1, ..., a_R)$, and set the counter j to 1. R is a parameter of the algorithm that determines the size of the Polled set.

ii. Send a request message to node a_j and wait for a reply message.

iii. Receive the reply message. Node a_j will either send back a task or an empty reply.

iv. If the node is still idle and $j < R$, increment j and go to step ii., else terminate the polling.

The reply element of the control law is executed when a request message arrives. If the node has more than one task in its queue, one of the waiting tasks is migrated to the node that has sent the request. An empty message is sent back if no task is waiting in the queue.

channel transmission rate (β)	*10 Mbit/sec*
slot time	*51.2 μsec*
transmission time of control message	*102.4 μsec*
expected task service time($\frac{\alpha}{\mu}$)	*50 msec*
BT parameter of STB algorithm [1]	*2.1*
D parameter of BID algorithm	*1 msec*
R parameter of PID algorithm [2]	*5*
balancing distance($BD_{i,j}$)	*.025,.05,.1,.2*
overhead (OVH) [3]	*$BD_{i,j} = .025$ means $\gamma^I \approx 1.5$ Kbyte (0.,0.,0.)*
simulation length $m > 8$	*40 sec*
simulation length $m \leq 8$	*80 sec*

[1] *except in 4.3.1,* [2] *except in 4.3.1,* [3] *except for 4.3.3*

Table 4.1. Simulation parameters for ***study*** of broadcast m*(M/M/1) ***systems***

4.4 simulation Study

A simulator for the broadcast m*(M/M/1) ***system*** **has** been developed using the **DISS** simulation language (see chapter **6).** The system **is** mapped onto a ***star*** topology **with the ETHERNET** subnet as a center **(Fig. 4.2).** Each of the **system** nodes is modeled by a DISS process and there are three types **of** such processes **- one** for each algorithm. The ETHERNET node is a realizationof the ***ETHERNET subnet model,*** **as is** described in **[Mel83b],** and includes both the Physical and Data-Link Layers of the protocol. Due to **this** mapping none of **the** communication protocol elements is included in the nodal process. Therefore by replacing the central node of the Simulator, broadcast m*(M/M/1) distributed systems with different communication suhnets can be simulated; In the **coming** subsections a performance study of the three algorithms will be presented. The study focuses on the effect which the three algorithms have on the expected normalized queueing time of a task, $\hat{W}_q$. Table **4.1** lists the numerical values of the simulation parameters. In all the cases it **was** assumed that the amount **of** data **units** needed to describe the results of a task is equal to the **amount** needed to define the task, i.e. $\gamma^I = \gamma^O$.

4.4.1 Algorithmic Parameters

The behaviour of both the BST and PID algorithm is determined by the values of their parameters. The selection of the values to be assigned to the BT and R parameters should

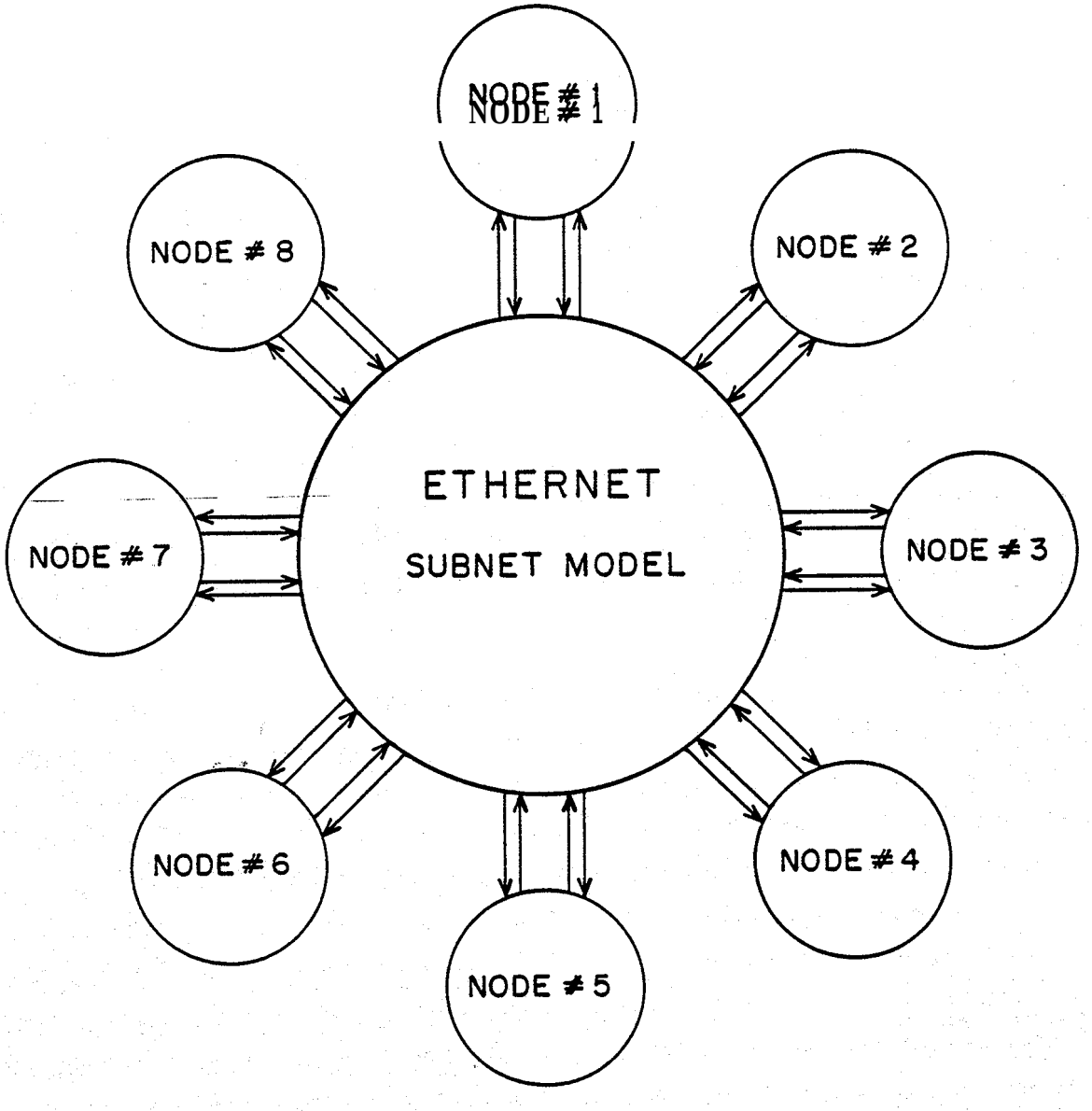

Figure 4.2. The Directed Multigraph Presentation of the Model

be guided by the properties of the system for which the algorithms are intended. The main factors to be considered are the balancing distance between the system processors. and their number.

Fig. 4.3 presents $\hat{W}_q$ and channel utilization (η) for systems with different balancing distances and number of processors (m) with BT as parameter. Note that an increase in BT, i.e less anticipatory transfers, causes a decrease in $\hat{W}_q$ in most of the cases. This improvement in the response time of the system is due to the reduction in the number of 'unnecessary'

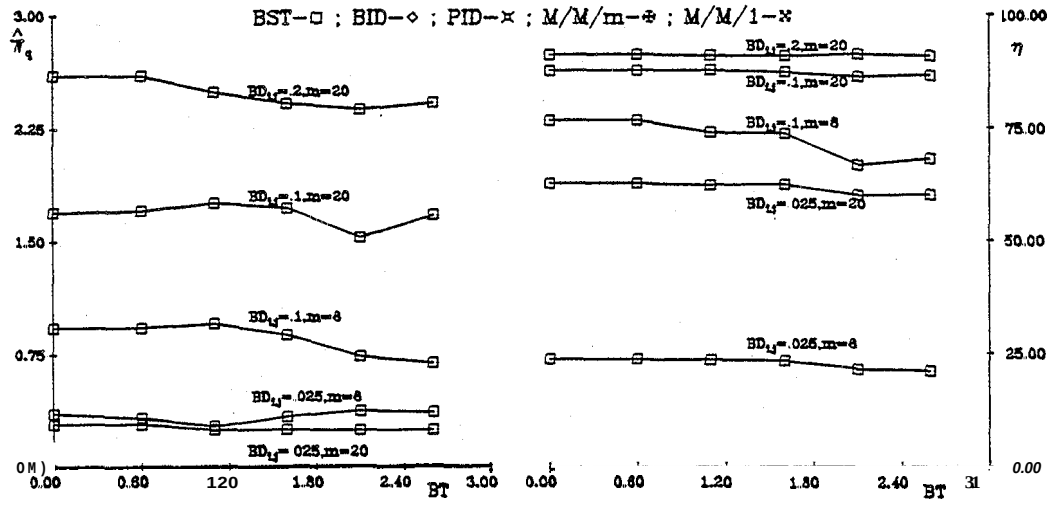

Figure 4.8. $\hat{W}_q$ and η vs. BT for **BST** ($\rho = .8$)

transfers. In a highly utilized **ETHERNET** a small reduction in the load has a significant effect on the turnaround time of a message. But there is a point where any further reduction in the rate of anticipatory transfers causes an increase $\hat{W}_q$. The location of this 'turning point' is system dependent.

The communication activity of the PID algorithm can be easily controlled by the value of the R parameter: Fig. 4.4 shows how η and $\hat{W}_q$ depend on the size of the polling set. When the balancing distance is small ($BD_{i,j} = .025$) $\hat{W}_q$ is a monotonic decreasing function of R. However, when $BD_{i,j} = .2$ and $R > 5$ an increase in the size of the set causes a degradation in the performance of the system. Note that even when $R = 3$, $\hat{W}_q$ is reduced due to the balancing process from 4 to3 1.7 for $BD_{i,j} = .2$ and to .75 for $BD_{i,j} = .025$.

4.4.2 Number of Nodes

The $\hat{W}_q$ of an M/M/m queueing system with a task arrival rate of $m\lambda$ is a monotonic decreasing function of m (see 2.2). Although the addition of another server increases the rate at which tasks arrive at the system the supplemental node decreases the expected queueing time of a task.

The effect of the number of nodes, m, on the $\hat{W}_q$ of a broadcast m*(M/M/1) system is demonstrated by Fig. 4.5, 4.6 ,4.7 and 4.8. The figures present the $\hat{W}_q$ of the

---

[3] 4 is the expected normalized queueing time of a task in an M/M/1 system with a utilization of .8.

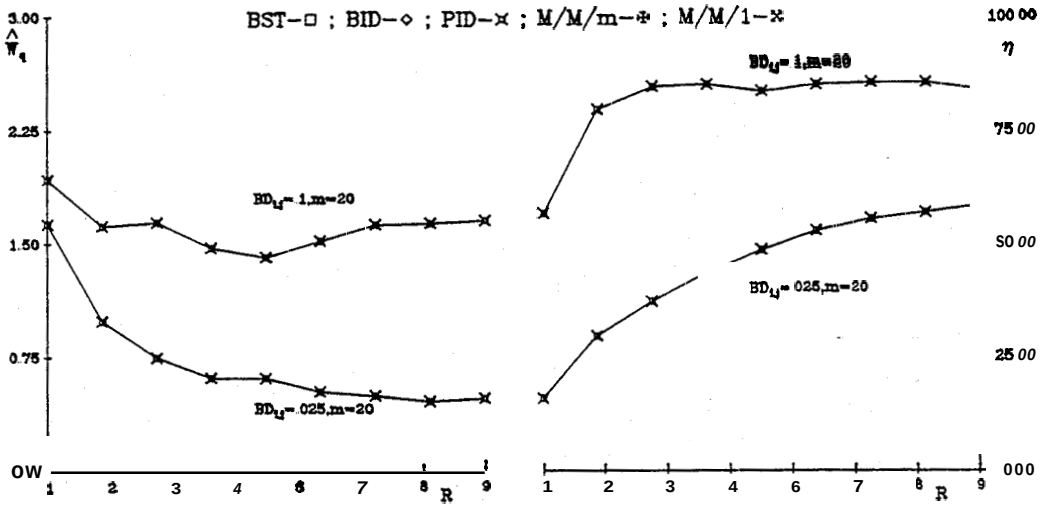

Figure 4.4. $\hat{W}_q$ and η **vs.** R for **PID** ($p = .8$)

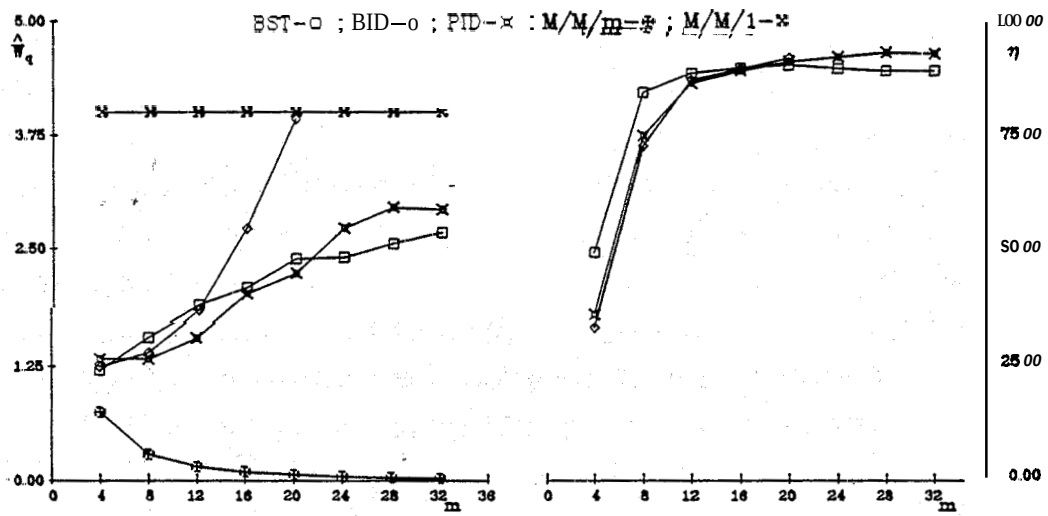

Figure 4.5. $\hat{W}_q$ and η vs. m ($BD_{i,j} = .2$, $\rho = .8$)

three algorithms for four different balancing distances $(BD_{i,j})$. In all the **cases** the balanced system has a considerably better $\hat{W}_q$ than the unbalanced **system,** M/M/1.

For a system with $BDi,j = .2$ the expected waiting time of a task is decreased by at least **70%.** The degree to which the balancing algorithm approaches the optimal $\hat{W}_q$ of an m server system (M/M/m) depends both on the balancing distance of the system and on the number of nodes, The curves show that an increase in the number of nodes in a balanced distributed system has two counteracting effects. **On** the one hand, it improves the probability that a waiting task will be transferred to an idle server, as in an M/M/m system.

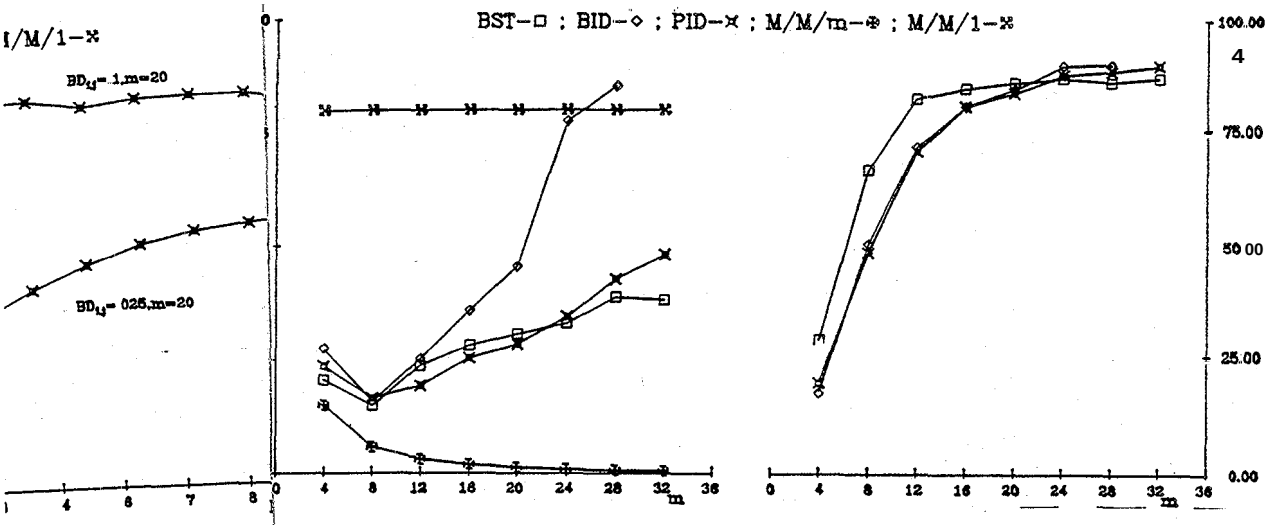

Figure *4.6.* $\hat{W}_q$ and η vs. m $(BD_{i,j} = .1, \rho = .8)$

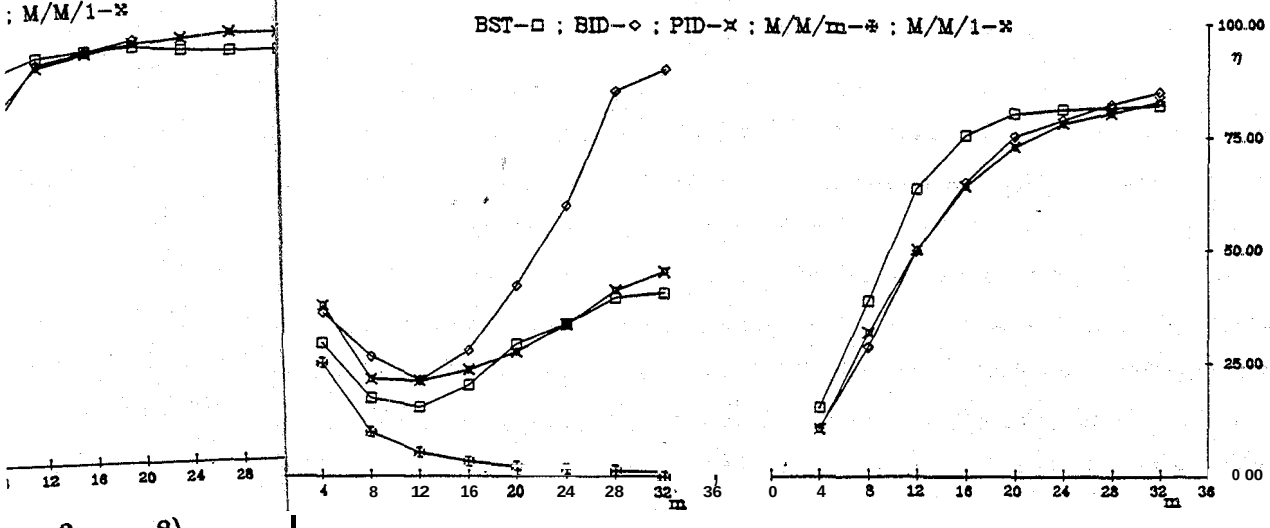

Figure 4.7. $\hat{W}_q$ and η vs. m $(BD_{i,j} = .05, \rho = .8)$

other hand, it raises the utilization of the communication channel. **Higher** channel causes a slow-down in the balancing process resulting from an increase in message delays. The net result of these two effects will determine whether the increase in es, does not affect,or deteriorates the expected turnaround time of a task. **Every** reaches a point, m_m, at which an addition of another server will came **an** increase he value of m_m depends on the algorithm and balancing rate of the system. Note cases when m is less than the m_m of the **BST** algorithm the $\hat{W}_q$ of this algorithm is st. After it reaches its minimal value the $\hat{W}_q$ of the STB algorithm increases until

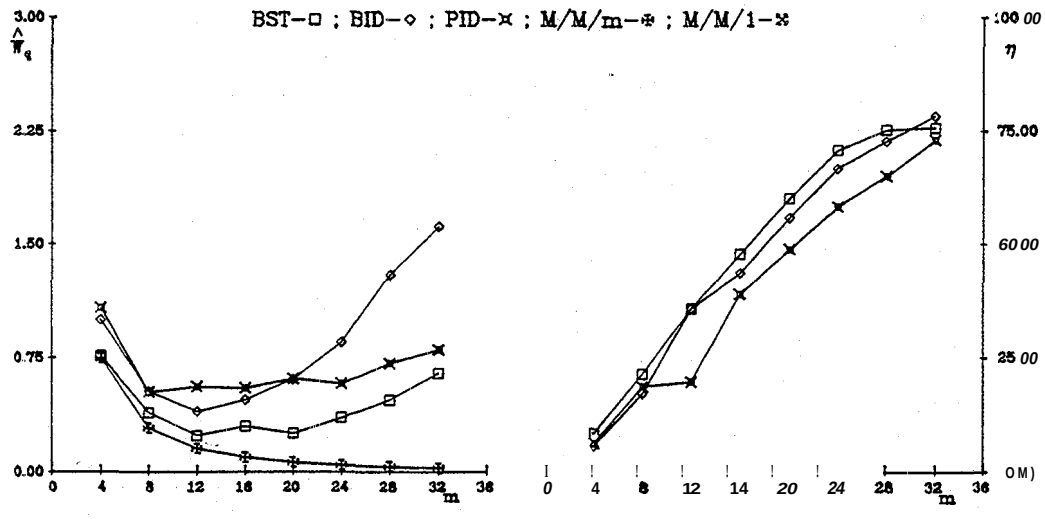

Figure 4.8. $\hat{W}_q$ and η vs. m ($BD_{i,j} = .025, \rho = .8$)

it becomes greater than the $\hat{W}_q$ of the PID algorithm. The degradation in the performance of the STB algorithm is caused by the increase in transmission delays.

Both the BST and BID algorithms are sensitive to changes in the transmission delay of a message. By waiting for all previous messages to be transferred before a new migration is considered, The BST algorithm adaptes its activity dynamically to the load on the channel. Therefore even when the channel is highly utilized the algorithm succeeds to enhance the response time of the system. Whereas the BID algorithm may reach a point where it becomes a cause for performance degradation as a result of its attempts to make a reservation at any idling node reaches a point where it turns to be a cause of performance degradation.

The PID algorithm is less sensitive to the utilization of the channel. As demonstrated by the four curves of $\hat{W}_q$ for this algorithm, there is a wide range of m values for which the algorithm has almost the same performance. The 'hand shaking' mechanism of this algorithm minimizes the number of 'wrong transfers'. For low and moderated channel utilizations the PID and BID algorithm have similar $\hat{W}_q$.

4.4.3 Processing Overhead

In all previous cases it was assumed that the load balancing process has no processing overheads. Table 4.2 demonstrates how processing overheads affect the $\hat{W}_q$ of the BST and PID algorithm for three different arrival rates.

$\frac{\lambda\alpha}{\mu}$		OVH-0,0,0	OVH-1,0,0	OVH-1,.5,0	OVH-5,0,0	OVH-5,.5,0
.85	PID	.9	1.18	1.43	2.32	2.58
	BST	.79	1.24	∞	3.02	∞
.70	PID	.46	.60	.68	1.08	1.20
	BST	.35	.53	4.04	1.30	∞
.60	PID	.34	.42	.47	.57	.85
	BST	.19	.29	1.23	.77	2.7

Table 4.2. $\hat{W}_q$ for Different arrival rates ($FE_i = 0, m = 16, BD_{i,j} = .05$)

$\lambda\alpha$		OVH-0,0,0	OVH-1,0,0	OVH-I,.5,0	OVH-5,0,0	OVH-5,.5,0
.85	PID	.9	I.02	1.02	1.64	1.5
	BST	.79	.88	.85	1.10	1.18
.70	PID	.46	.57	.59	.75	.77
	BST	.35	.39	.48	.61	.67
.60	PID	.34	.41	.42	.58	.60
	BST	.19	.24	.28	.43	45

Table 4.3. $\hat{W}_q$ for Different arrival rates ($FE_i = 1, m = 16, BD_{i,j} = .05$)

An increase in the task arrival rate, λ, has opposite effects on the activity of the BST and PID algorithms. The information exchange activity of the BST algorithm increases due to an increase in λ, since more tasks per time unit mean more state changes. The BST algorithm broadcasts at least 2λ control messages per station per time Unit. The PID algorithm reacts in an opposite manner to an increase in λ. Due to the increase in server utilization, the length of the nodal busy periods increases and thus the rate at which the polling element of the algorithm is invoked, decreases. Nevertheless, the expected number of stations which a node has to poll each time, decreases, as well, due to the increase in ρ. There is a higher probability that the first nodes to be polled are willing to migrate a task.

The values presented in Table 4.2 reflect these characteristics of the two algorithms. Note that the effect of the increase in the information exchange activity of the BST algorithm is amplified because all the control messages are broadcast messages. So that every time a message is transmitted every processor has to devote some processing capacity in order to receive and decode the message.

Table 4.3 demonstrates the advantageous effect with the addition of a communication processor to every node. The $\hat{W}_q$ of the BST algorithm is considerably improved due to the increase in the processing power of the node. As far as the PID algorithm is con-

cerned, this increase does not significantly improve its performance. Since in most of the cases this algorithm utilizes processing capacities when the node is idling, the reduction in the utilization of the processor does not reduce its $\hat{W}_q$.

Chapter 5

Store **and** Forward Systems

§5.1 Introduction

Following the development of the need for ***resource sharing*** and ***teleprocessing*** at the early 1960s stand-alone computers were integrated into computer networks at the end of that decade [Robe70],[Puzi73]. Most of these networks are operational ever since and some of them have grown subsequently.[1] The communication subnets of these networks consist of switching elements, commonly called IMPs,[2] that are connected in pairs by cables or leased telephone lines to form a ***point to point*** subnet. These subnets embodied a ***mesh-connected back-bone*** network with a ***packet switching*** strategy [Tane81]. The success of the ***time-sharing*** approach in sharing the resources of one computer among many users and the ***burstly*** nature of the inter computer traffic aspired the selection of this switching strategy. The ***peer*** interconnection approach used in these subnets allowed flexibility in their operation mode and topology.

Packet switching imposes a ***store-and-Forward*** pattern on the transmission process of a message in a point-to-point subnet. When a message has to be transmitted from a source IMP to a destination IMP which is not directly connected to the source the message has to go via intermediate switching elements. At each intermediate IMP the message has to be stored until the desired link is free, and only then forwarded.

---

[1] The ARPANET interconnects today more than 100 computers which are spread over half of the globe.

[2] IMP (*Interface Message Processor*) is the name of the switching elements of the ARPANET.

The data rate of the communication lines used in store-and-forward subnets is in most cases, less than .1 *Mbit/sec*. Although attempts are made to utilize fiber-optic technology in point-to-point subnets there is no such operational subnet yet and all commercial networks are based on leased lines. Because of the low bandwidth of the communication lines and the requirement for intermediate buffering space, store-and-forward protocols require procedures for *routing* management, for *buffer allocation* and for the detection or prevention of *deadlocks*. Due to the complexity of these issues and the urgent need for efficient protocols for point-to-point networks, a considerable amount of effort has been devoted to the development and analysis of protocols for this type of networks [Klei80], [Schw80]. Various protocols have been implemented and their characteristics analyzed. Most of these protocols utilize complex flow control algorithms that require a high level of control activities [MaQu80].

One of the main issues associated with the design of point-to-point subnets is their topology. This chapter focuses on the interdependency between the topology of the system and the load balancing phenomena. An LB algorithms for store-and-forward m*(M/M/1) distributed systems with homogeneous processors and users is defined. The results of a simulation study of the performance of this algorithm for systems with different topologies are presented. These results shed light on the way load balancing consideration should affect the topology selection procedure.

5.1.1 The Store and Forward Model

The topology of the communication subnet of the store and forward model is defined by a *regular*[3] graph, $G = (V, E)$. The nodes of the graph, V, are switching processors and the *edges*, E, are *full-duplex* communication links. Every processor of the m*(M/M/1) system is attached to a different IMP Fig. 5.1. The IMP and the processor share the same data storage facilities. The processing elements and users of the system are homogeneous and all links have the same data rate, β *du/tu*. Each link has a queue in which messages that are waiting to be transmitted are placed. It is assumed that each node has an unlimited buffering space.

In the view of the motivation of this study and the characteristics of the model a *'simple'* communication protocol for the store and forward m*(M/M/1) model has been selected. The protocol implements a *message switching* strategy which does not impose any limitation on the size of the transmitted data blocks. The routing scheme of the protocol

[3] The *Graph Theory* terminology used in this chapter followes the definitions given in [Tane81] (Chap. 2.).

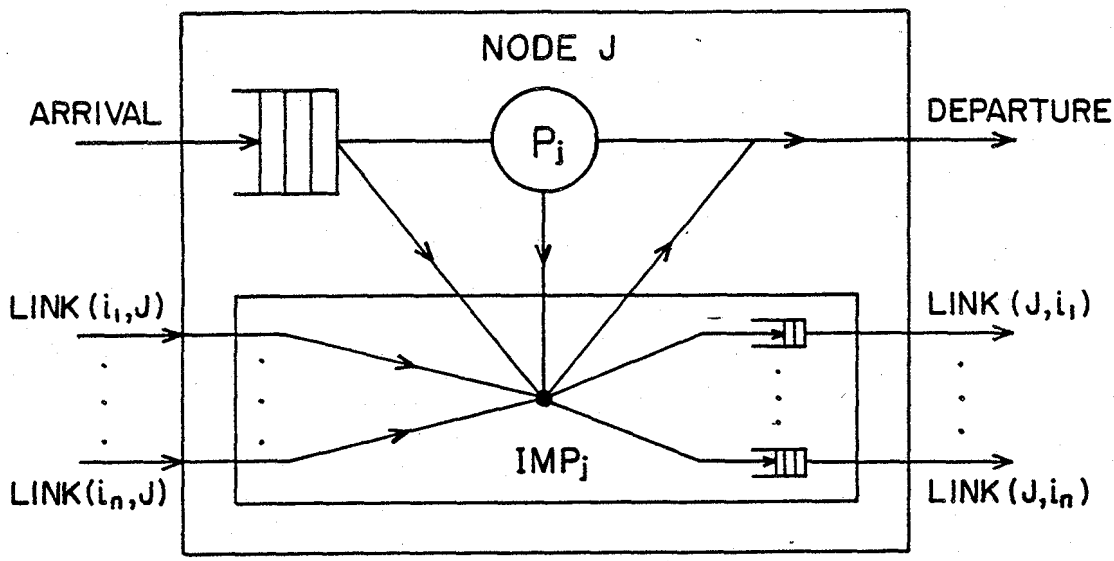

Figure 5.1. A nade of the store-and-forward m*(M/M/1) system

is *static.* Messages are routed along the shortest *pass,* the *geodesic,* between the source and the destination. When several geodesics exist between two nodes one of them is selected at the time the system is established. The messages are transmitted according to an FCFS discipline. Due to the unlimited buffering space available at each node no buffer reservation or deadlock detection/prevention mechanisms are required. Transient messages are directed upon arrival at the input queue of their next link. It is assumed that the processing time of the protocol is much smaller than the transmission time of a message and thus negligible.

§5.2 Load Balancing Algorithms

Unless the graph G = (E, V) that defines the interconnection scheme of the model is a complete graph, the balancing distance between the processors in a store and forward system is not the same for all pairs of nodes. The balancing distance between processor i and j of the system is $d_{i,j}\gamma^I \frac{\mu}{\beta\alpha}$, with, $d_{i,j}$ being the length of the geodesic between the two processors in G. The non-uniformity of the balancing distances together with the lack of a broadcast facility in this type of system imposes *locality* on the load balancing process. The difficulty in maintaining a global up-to-date picture of the instantaneous load distribution of the system-at every node and the need to consider balancing distances, prohibit the establishment of large balancing regions. Unlike broadcast systems, not all the processors with the same load have

equal priority as candidates for being a target for a migration operation. Those processors which are closer, have higher priority. A natural criterion for including a processor in a balancing region will be its balancing distance from the owner of the region. The region of a processor will, most probobly, include only those nodes which are *adjacent* to the processor. In such a case the control element which resides at node i has to consider only the load distribution of $BR_i(t)$ and is free of topological considerations.

An algorithm for store-and-forward m*(M/M/1) systems with homogeneous users, processors and subnet[4] is defined in this section. The algorithm is based on an adjacent balancing region. Performance models of systems with various topologies are defined and solved.

521 The HO1 Algorithm

The control law of the **Hop One** (HO1) algorithm aspires to keep the load distribution of $BR_i(t)$ balanced at all times. The algorithm is anticipatory in its nature and a user might be migrated several times before being executed.

ALGORITHM. HO1(*migrate one hop*)

Information Policy: The load of a processor is $n_i + rc_i$ where, n_i is the length of its queue and rc_i is the *reservation* counter. This counter is incremented when a reservation is accepted and decremented upon the arrival of the task for which the reservation has been made. Whenever the load of the node changes the processor sends all its adjacent processors a *status* message that describes the new load. Each node records the information it obtains from these messages in a *Load Distribution* vector, LD_i. Before a task is migrated from one processor to the other a *reservation* message is sent to the target processor.

Balancing Region: The balancing region of node i at time t consists of the following processors:

$$BR_i(t) \triangleq \{j \in V \mid (i,j) \in E \text{ and no data message at input queue of } (i,j) \text{ at time } t\}.$$

The above definition for balancing region is motivated by the desire to prevent unsuccessful task transfers (see 3.3). In order to minimize the probability that due

[4] A point-to-point subnet is defined as homogeneous if G is a *regular* graph and all its links have the *same* capacity.

to long communication queueing time such a transfer will take place, an adjacent node to which the link is currently occupied by a data message is not included in the balancing region. Note that the load of a node is defined according to its current state and does not reflect its ability to 'fan-out' additional load. This ability depends on the load of the nodes which constitute the balancing region of the node. The algorithm is motivated by a 'one step at a time' approach and therefore does not consider the ability of the target node to ship the migrated task one step further.

Control Law: the control element of the HO1 algorithm is invoked whenever a status or reservation message arrives. Processor i with $LD_i = (ld_1^i, \ldots, ld_m^i)$ will initiate a user transfer to processor $j \in BR_i(t)$ when all the following conditions hold true:

1. $n_i > 2$ *(waiting)*

   Due to the high throughput degradation factor of last minute transfers initiated when only one task is waiting in systems with long balancing distances (see 3.5.3), the algorithm initiates transfers only when at least two tasks are waiting for service.

2. $ld_k^i \geq ld_j^i$ *for all* $k \in BR_i(t)$ *(minimal)*

   j has to be the node with the minimal load in the balancing region of i. When several processors have the minimal load, one of them is randomly selected.

3. $rc_i + n_i - ld_j^i > 1$ *(threshold)*

   The load-difference of the $BR_i(t)$ has to be greater than one.

§5.3 Effect of Interconnection Scheme

The expected turnaround time of a task in a store-and-forward m*(M/M/1) system depends on the expected queueing length of the processors and the length of the geodesic between the execution and entrance sites of the task. The expected number of tasks in a queue depends on the ability of the node at which the queue resides to distribute its load among the other processors of the system. This ability depends on the structure of the *Distance* Tree, $DT_i =$

$(\tilde{E}, V), \tilde{E} \subseteq E$, of the processor. DT_i is a *spanning* **free** of G such that i **is** its **root** and for all $j \in V$, $d_{i,j} = \tilde{d}_{i,j}$, where $d_{i,j}$ and $\tilde{d}_{i,j}$ are the length of the geodesic between i and j in G and DT_i respectivly. **A** quantitative description **of** the ability of the processor to 'fan-out' its load can be given by means of the following function

$$RL_i(n) = \sum_{j=0}^{n-2} (n-j) U_j^i$$

where U_j^i **is** the number of nodes at the j^{th} level of DT_i.[5] The value of $RL_i(n)$ is the minimal number of tasks that ought to be in the system **so** that a **load** distribution with $n_i = n$ will be a **no-migrate** distribution. The system is defined to be in a no-migrate distribution if a) the queues of all **links** are empty b) $(|\, n_i - n_j \,| \leq 1 \vee n_i \leq 2)$ for **all** $(i,j) \in E$. **A** processor with n **tasks** at its queue would 'prefer' that the system would be in a no-migrate state **only** if $n_j \geq n-1$ for $j = i, j \in V$, i. e. that $RL_i(n) = n + (n-1)(|\, V \,| - 1)$ for all n. The function RL_i will have the above **form** only if the *out* **degree** of server i is $|\, V \,| - 1$ which means that the graph G has to be a complete graph.

§5.4 Simulation Study

The **DISS** (see chapter **6)** methodology and simulation language have been used for developing a simulator for the store-and-forward m*(M/M/1) system with the HO1 LB algorithm. Each of the model nodes describes the behaviour **of** a processor and its adjacent **IMP.** The specific properties of DISS have made the construction **of** simulators for systems with different toplogies, an easy task. In the course of this simulation study five different system topologies have been analyzed. These topologies and the corresponding nodal distance-trees are shown in Fig. **5.2.** Note that for **all** the topologies which were selected all the nodes have the same distance tree. In Table **5.1** the values of the $RL_i(n)$ function for the different topologies are displayed. The table demonstrates the interdependency between the values of the function **and** the interconnection scheme of the communication subsystem.

In this study it **was** assumed that the length of the control messages **is** much smaller than that of the data messages. Consequently, the transmission time of these messages was neglected and a control message is considered to be transferred instantaneously. However, the communication link has to be free when such a transfer **is** executed. One time unit,

[5] **The root of the** tree **is** *at* **level zero.**

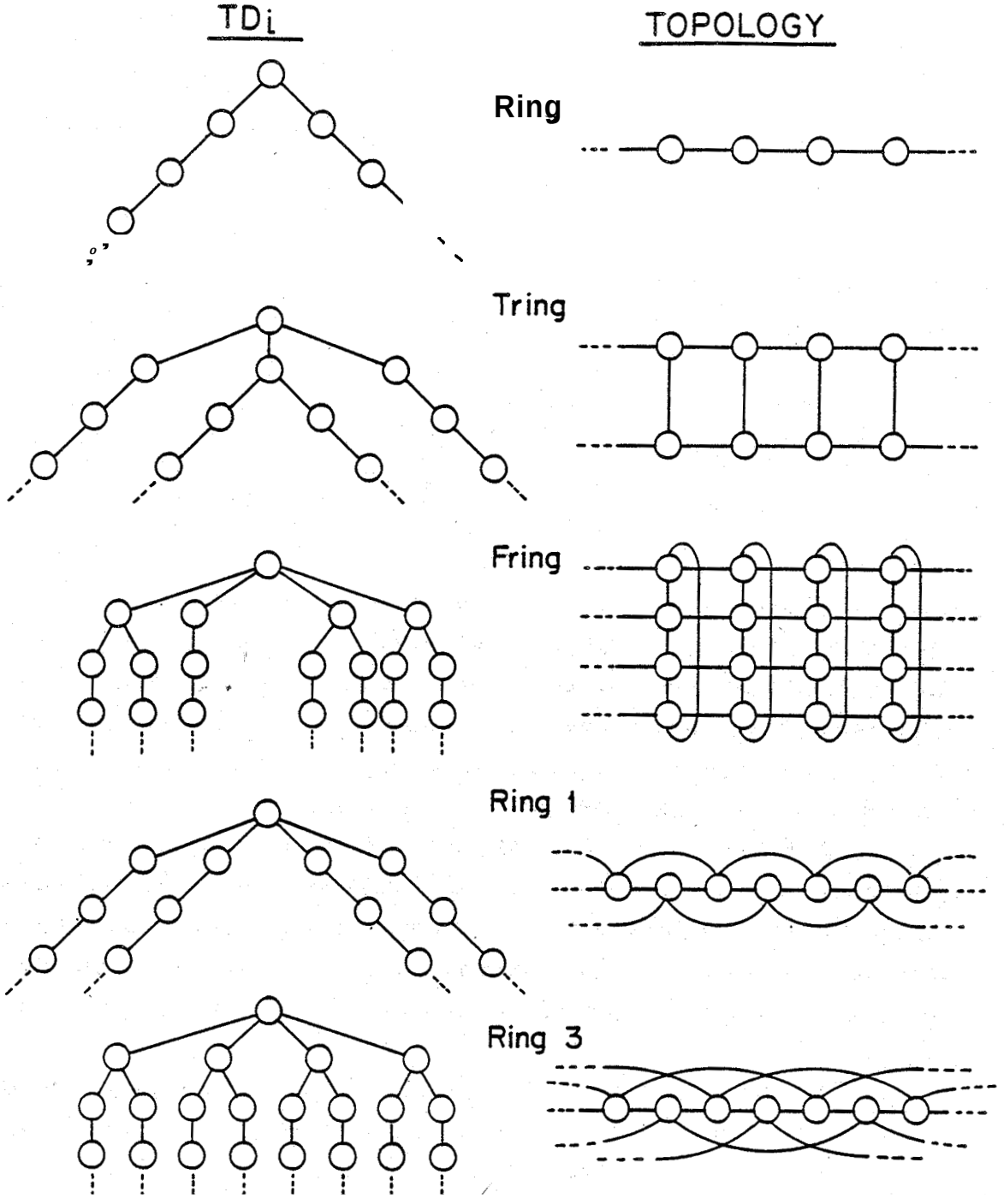

Figure **5.2.** The toplogies and **their** distance-trees

topology	$n=3$	$n=4$	$n=5$	$n=6$	$n=7$	$n=8$
Ring	7	14	23	34	47	62
Tring	9	21	37	57	81	109
Fring	11	24	41	62	87	114
Ring1	11	30	56	88	108	132
Ring3	11	32	59	87	111	135

Table 5.1. $RL_i(n)$ for the different toplogies (m=24)

γ^I, γ^O	$m=4$	$m=8$	$m=12$	$m=24$
.1 , 0.	24	1.4	1.2	1.2
.1 , 1.	3.3	2.6	2.5	2.6
1. , 0.	3.9	2.6	2.5	2.5
1. , 1.	3.9	3.2	3.1	3.2

Table 5.2. $W_{,,}$ for Rings of different sizes ($\lambda = .9$)

tu, in the simulation runs was equal to the expected execution time of a task ($\tilde{\mu}^{-1}$) and the length of each run was 2500 *tus.* The various systems were analyzed under two different task-arrival patterns - the Hom9 and Lo19 patterns. According to the Hom9 pattern, the system is homogenebusly loaded and $\lambda = .9$, i.e the utilization of all servers is .9. When the tasks arrive according to the Lo19 pattern, the system is not homogeneously loaded. The arrival rate of processor 1 is 3.2 and the arrival rates for all the others is .8. Note that the sum of all arrival rates for the two patterns is the same.

The manner in which the number of processors affect the ability of the HO1 algorithm to reduce the response time of the system is demonstrated by Table 5.2. For all four combinations of γ^I and γ^O that where simulated when m goes from 4 to 8, $\hat{W}_q$ improves considerably. However, any further increase in the system's size does *not* affect $\hat{W}_q$. Note that even in a 4*(M/M/1) system with $\gamma^I = \gamma^O = 1, \hat{W}_q$ is reduced due to the LB process from 9 to[6] 3.9.

The results presented in Table 5.3 show that when a store-and-forward m*(M/M/1) system is homogeneously loaded, a change in the interconnection scheme of the communication subnet has only a marginal effect on $\hat{W}_q$. Although the distance tree of a node defined by the Ring3 topology has better properties than the one defined by the Ring1 and the Fring

[6] 9 is the $\hat{W}_q$ of an M/M/1 system with a utilization of .9.

γ^I , γ^O	Ring1	Fring	Ring3
.1 , 0.	.70	.69	.62
.1; 1.	1.67	1.69	1.67
1. , 0.	2.20	2.17	2.16
1. , 1.	2.94	2.96	2.94

Table 5.3. $\hat{W}_q$ for toplogies with 4 m links ($\lambda = .9$)

topologies, these differences in the structure of TD_i do not affect $\hat{W}_q$ when tasks arrive according to the Hom9 pattern. However, when the arrival pattern of the tasks is Lo19 these differences do affect $\hat{W}_q$. Table 5.4 clearly displays the interdependency between the $RL_i(n)$ function of a topology and the performance of the system. When the arrival rate of one node is higher than the arrival rate of the other, the structure of its distance tree plays an important role in determining $\hat{W}_q$. The table presents the expected normalized queueing time of a task which has arrived at node 1 , $\hat{W}_q$ (1), and $\hat{W}_q$ for different values of γ^I and γ^O.

The results displayed at the second row of the Table 5.4 demonstrate an important aspect of the HO1 algorithm. When γ is small (.1) the algorithm may migrate a task several times and thus the execution site of a task might be several hops away from its entrance site. Therefore if γ^O is not negligible the task might be considerably delayed on its way back to the node at which it had entered the system. In the case represented by the second row of Table 5.4, tasks are transferred without any delay 'down the stream' to their execution site but are queued up on their way back. Note that due to this behavior of the HO1 algorithm the response time of the system is better for $\gamma = \gamma^O = 1.$ than for $\gamma^I = .1, \gamma^O = 1.$ This property of the algorithm is also demonstrated by Table 5.5. This table shows the inter-dependency between the number of communication links and the performance of the system for the HL19 arrival pattern. When each node has only two links the system is 'choked-up' under such conditions unless γ is small and γ^O is negligible. However, when the system has three or four links per node it can serve all tasks and even provide them with better response time than an M/M/1 system which is busy 90% of the time.

	Ring		Fring		Ring3	
γ^I, γ^O	$\hat{W}_q$ (1)	$\hat{W}_q$	$\hat{W}_q$ (1)	$\hat{W}_q$	$\hat{W}_q$ (1)	$\hat{W}_q$
.1 , 0.	1.0	.75	.80	.69	.77	.63
. 1 >1.	18.2	4.4	34.0	7.0	12.5	3.3
1. , 0.	3.2	2.3	2.9	2.2	2.8	2.2
1. ,. 1.	5.1	3.1	5.1	3.2	4.75	3.2

Table 5.4. $\hat{W}_q$ and $\hat{W}_q$ **(1)** for **toplogies** with $4m$ links $(\lambda_1 = 3.2, \lambda_i = .8\ 1 < i \leq m)$

	Ring		Tring		Ring3	
γ^I, γ^O	$\hat{W}_q$ (1)	$\hat{W}_q$	$\hat{W}_q$ (1)	$\hat{W}_q$	$\hat{W}_q$ (1)	$\hat{W}_q$
.1 , 0.	2.5	1.4	1.2	.80	.77	.69
.1 , 1.	∞		60.	11.3	12.5	3.3
1. , 0.	∞		4.2	2.5	2.8	2.2
1. , 1.	∞		9.4	3.9	4.75	3.2

Table 5.5. $\hat{W}_q$ and $\hat{W}_q$ **(1)** for **different numbers of** links $(\lambda_1 = 3.2, \lambda_i = .8\ 1 < i \leq m)$

Chapter 6

DISS

§6.1 Introduction

Performance prediction is an essential step in the process of system design and system upgrading. When different alternatives are examined by a designer or manager their relative performance may constitute a cardinal argument for regarding one as superior to the other. In order to predict the performance of a non-existing system under an estimated workload, a *performance model* of the devised system has to be defined and solved for the foreseen load. A quantitative description of the desired *performance measures is* then derived from the solution. The characteristics of the model and the performance measures considered, determine whether *analysis* can be used as a solution scheme or whether *simulation* is the only means by which the measures can be derived. A considerable amount of effort has been devoted to the study of analytic solution schemes for performance models and several methods for solving *queueing network* models have been developed and implemented [Bask75],[Chan80]. Although these methods are remarkably general and useful in system modeling, there are many interesting models that do not meet their assumptions and thus have no known traceable numerical solution. In order to release some of the constrains of these methods, *approximation* schemes for solving performance models have been developed [Saue80]. The main shortcoming of this approach is its inability to bound the error in the results.

Because of the limited scope of numerical solution schemes, simulation, in spite of its drawbacks, is widely used for predicting system performance. Simulation is a technique

which can predict the characteristics of a model by following the *state* changes it undergoes over a period of time. When simulated, the evolution of the model under stimuli that model its inputs is observed, and the desired behavioural measures are derived. In most cases simulation is a repetitive process and is, thus, executed, most likely, by a computer. The computer simulates the model according to a behavioural description, a *simulator*, written in a programming language. Languages that address themselves to such descriptions are called simulation languages.

Simulation is computationally expensive and requires a considerable amount of programming effort. In order to assist the programmer in writing simulation programs, a number of simulation languages have been designed. The various languages differ in their programming approach and simulation strategy. In this chapter the **Distributed System Simulation (DISS)** approach for modeling and simulating distributed systems is defined. DISS views both the model and the simulation program as modular structures which consist of *self-contained* building blocks. The language is a *macro* extention of the **SIMSCRIPT II.5** simulation language and implements a process interaction simulation strategy.

6.1.1 Motivation

The continuous growth in the size and complexity of Distributed Processing Systems (DPS) increases the need for efficient methods for predicting their performance. The complexity of these systems and the variety of services they provide prohibit the usage of intuition as a design tool. In most cases analytical solution schemes can not be used for solving the performance models of this type of system, mainly because of the strong interdependency between the components of the system [Wong78]. Due to this interdependency, the models do not satisfy the *local balance* property and thus have no *product-form* solution [Chan77]. Interdependency between system elements is a cohesive attribute of DPS because of the *cooperative* nature of its elements [Ensl81]. Therefore performance prediction of a DPS almost always entails a simulation study of the system.

Distributed processing systems consist of *loosely coupled autonomous* elements which endow this type of system with the qualities of *modularity* and *extensibility* [Ensl81]. Since the replacement of a component or the integration of a new one is a simple operation in a DPS an analysis of the impact of topological changes on the systems' performance will undoubtedly be included in a performance study of such a system. Therefore it is desired that the simulation program used in such a study will also be modular and extensible. The

efficiency of the study will depend on the degree to which internal logical and structural changes in one module impose modification on others. In a simulation program where one module has direct access to a variable of another module (*tightly coupled* modules) or a single module may imperativly schedule an event for another one (*non-autonomous* modules) a local change in one element may effect other modules. Changes of this type may require a major modification of the entire program unless it is composed of loosely coupled autonomous modules.

The importance of modularity and extensibility of simulation programs for the study of performance issues of DPS and the lack of a simulation language that provides means for building such programs motivated the development of the DISS methodology for modeling Distributed Processing Systems and the design and implementation of the DISS language. DISS provides the modeler and the programmer with a comprehensive approach to modeling and simulating this type of system. Although the development of the methodology and the language was guided by the characteristics of DPS, the world view of DISS is also applicable to simulation studies of other types of systems.

6.1.2 The World View of DISS

DISS is based on a *comprehensive* view of the two components of a simulation study - the ***model*** and the ***simulator.*** Each of these components is considered as a network of loosely coupled and autonomous modules that interact via a well-defined interface. Two modules of a model or simulator are ***loosely*** coupled when they can exchange information but there is no direct access from one module to the variables of the other. The information is exchanged via a 'mail box' that can be accessed by the two coupled modules but which does not constitute an integral part of either one of them. Receiving information and sending out information is an input/output operation for such modules and is executed via *ports*.

In a discrete event environment an element is *autonomous* if no other element can ***imperatively*** schedule an event for it. Autonomy does not mean isolation: an autonomous module interacts with the other modules by receiving events scheduled by them. Yet such a module exercises control over the events it is willing to accept at a given instance by means of an ***interrogative*** mechanism. The loose coupling and autonomy of the modules guaranty that a change in the structure or logic of one module does not impose changes on other modules and thus, endow the model or simulation program with modularity.

Each module of the network is a Discrete Event System **(DEVS)** described **as** a model or presented **as** a component of a simulation program. Discrete Event Systems were first formally defined by Zeigler [Zeig76]. This definition has been extended ***so*** that a DEVS can be specified **as** an autonomous module and can be loosely coupled with other DEVSs (see **A.2** for a formal definition of a DEVS specification). **According** to the world view of **DISS** ***input*** and ***output*** ports, ***masks*** **(for** external events) and ***input*** variables are cohesive attributes of a DEVS and thus should be part of its specification. Since a DEVS exchanges information with others through ports, each external event or input variable is associated with an input port, and an output variable with an output port.

By interconnecting an output port **of** one DEVS with an input port **of** another, individual systems can be integrated into a network. Such a connection is created via a mapping process from the output variables **of** the source port, onto the input variables and external events of the target. The topology and interconnection scheme of a DEVS network can be represented by a ***directed multigraph*** where the nodes are DEVS and the **arcs** are the output port to input port connections.

When a model, described by a network **of** DEVS specification, is mapped to a **DISS** program its structure is preserved. **A** DEVS is realized by a **DISS process** and the interconnection by a **DISS** *arc*. It is not **only** the graph presentation of the model that is preserved by the **DISS** program but also the autonomy of its modules and the ***looseness*** of the interconnections. Consequently the program is endowed with the modularity and extensibility of the modeled system.

The **DISS** methodology imposes a structural similarity between the system, the model and the simulation program, which assists the designer, the modeler and the programmer in communicating with each other. Effective communication is an important aspect of a simulation study. The ability to relate changes in the specification of the system to the model and the extent to which the modeler can become acquaintance with the realization of his model, are major factors in determining the efficiency and quality of a simulation study. The **DISS** methodology and simulation language assist programmers and modelers to learn about each others work. In addition, it efficiently supports sharing models and simulation programs. Because a **DISS** process is self-contained, libraries consisting of various **DISS** realizations of DEVS models can be gradually constructed. Members of such libraries **can** then be selected for incorporation in simulators of different systems.

6.1.3 Simulation Languages

A considerable amount of programming effort is required for constructing a simulator for a system. In order to simplify the routine tasks associated with such a process and in order to assist the programmer in writing and debugging the simulator, a number of simulation languages have been designed. The design of these languages was motivated not only by the need for programming convenience but also by the desire to *articulate* the modeling concepts [Kivi67]. Therefore the design of a simulation language is based on both a programming approach and a modeling philosophy. The modeling phase of the simulation study is mainly effected by the simulation strategy of the language. Various strategies of this kind - *event scheduling, activity scanning* and **process** *interaction* [Zeig76] - are implemented by the different languages. Each of these approaches to discrete event simulation imposes a different structure on the model and on the simulator. Most of the simulation languages are *supersets* of general purpose language, like FORTRAN, PL/1 and **ALGOL,** and therefore combine the flexibility and richness of the base language with the special-purpose features needed to simulate discrete event systems.

The first steps in the design of specialized computer simulation languages were made during the latter part of the 1950-s. The first discrete simulation languages - **GASP** [Prit69], **SIMSCRIPT** [Dims64], GPSS [Gree72] and others (see [Teic66]) were introduced in the early 1960-s. Most of these languages where event oriented and had no facilities for nested declarations of variables or program structures. A behavioral description of a discrete event system given in an event oriented language is composed of a set of subroutines, each of which describes the activity of an event. Individual events may be related one to another, like all the internal events of a DEVS. However none of the above languages provides means by which a structural binding between events can be established. Therefore in all these event oriented languages modules of related events can not be constructed.

The progress in programming methodology which was brought about by the introduction of ALGOL60, together with the continuous increase in the usage of simulation as a means for solving performance models, motivated the design of **process** oriented languages [Fran78]. Several languages based on this simulation strategy - **SIMULA** [Dahl66], **ASPOL** [MacD73], **SIMSCRIPT II.5**[1] [Russ83]- were introduced during the later 1960-s and the early 1970-s. Both **SIMULA** and ASPOL are *block structure* languages and provide facilities for

[1] SIMSCRIPT II.5 processes were added to the language in 1975.

nested-scope definitions.

A process is a dynamic entity that consists of a set of related activities. Each activity is associated with an event and defines the change of states the system undergoes because of that event. Upon the completion of the execution of such an activity the process is *suspended* until a future resumption. This resumption represents the occurrence of an event, whether internal or external. Once resumed the process proceeds its execution from the point at which it has been suspended.

The entire process represents a sequence of events and thus can be considered as a realization of a DEVS. However, due to the limitation of the above process-oriented languages, there are many DEVS specifications that can not be realized by a single process. In all these languages different statements have to be used when the process is suspended until an internal event occurs or when it is suspended untill the occurrence of an external event. Therefote a process can not be suspended until either an external or internal event will occur. Nevertheless the *bold* , *wait*[2] or *work* and *suspend*[3] mechanisms of these languages do not support a process suspension until the first out of a number of internal events will occur. Therefore only DEVS that do not posess simultaneous internal delays can be realized by one process in the aboye process-oriented langauges.

ASPOL is the only simulation language that provides means by which a process can autonomously control the set of external events that it is willing to accept - the *wait(e)* statement. But the language does not provide a mechanism to be used following a *wait(e)* statment for locating the event which caused the resumption. In all other languages a process has no control whatsoever on the external event that may resume it.

SIMSCRIPT II.5 is a general purpose language with a rich varity of data structures and features that support discrete event simulation. The language has served the author in many simulation studies and is widely available and commonly used. These properties of the language together with the acquaintes with its internal structures[4] turned SIMSCRIPT II.5 into a natural candidate for constituting a base language for the simulation language that: will support and complement the DISS modeling methodology.

---

[2] In SIMULA and ASPOL

[3] In SIMSCRIPT II.5

[4] All the internal structures of the timing mechanism of the language are accessable

§6.2 Modeling with DISS

Modeling is the second step which a simulation study **involves,** and **is** the least well understood step of such a study. **A** system can be modeled only after its salient components **and** interactions have been isolated by means of an analysis of the system and of the *experimental frame* of the the study. Modeling can be defined **as**

> ***the process of developing an internal representation and set of transformation rules which con be ured to predict the behaviour and relationships between the set of entities composing the system*** [Fran77].

The internal presentation, formulated by means **of** ***state variables*** and the ***transformational rules,*** is an abstract description of the behaviour of the system. The **DISS** modeling methodology views the abstraction of the behaviour of a **DPS as** a **two** stage procedure. First, the system is mapped to ***a*** directed multigraph where each node is a **DEVS.** Then each node **is** modeled individually. The former stage **of** the modeling process is mainly a ***structural*** abstraction of the system, whereas at the latter the behaviour of the system elements is modeled. Every node **of** the graph presentation of the system stands for one[5] or more **of** the system elements, and is a well defined autonomous area ***of*** activity. Every system element, including communication channels is related to a **DEVS** whose input/output variables and ports are defined at this stage. The selection of the mapping scheme depends on both the structure **of** the system **and** the requirements **of** the experimental frame [Mel83b].

6.2.1 Nodal Interconnection

The interconnection scheme **of** the multigraph represents the coupling between the modules of the model. **A** directed arc that goes from the source node to another node, the target, represents the ability of the source to transfer information to the target. The data transferred describes a change in the internal state of the source that the target might be interested in. These changes are reflected by corresponding changes in the output variables **of** the source so that they can been ***'seen'*** from the outside. **A DEVS** receives external information via input ports. Thus in order for the target node to become aware of a change that has taken place at the source, either an input variable **of** the target has to be modified or an external event **has** to be scheduled for the target.

---

[5] **Fractions of elements can be also mapped to different nodes but in such a case it is suggested to consider each part as an element of the system.**

An *arc* of a **DEVS** network is a mapping from the output variables of an output port of the source, onto the input variables and external events of an input port of the target node. The mapping depends on the way the target node reacts to changes in the output variables of the source. When the target might consider such a change as an event, i.e. the node might be waiting for it, the output variable should be related to an external event at the input port. However, if the value of the output variable has an effect only on the internal state transition function of the target DEVS, the mapping is to an input variable.

A mapping from an output variable to an external event establishes an ***infer-node event***. Such an event is triggered by the output function of the source when the variable is assigned a value and appears as an external event at the input port of the target. An inter-node event represents an ***active*** transfer of information that is based on *an* attempt made by the source to ***alert*** the target. By relating an output variable to an input variable an ***infer-node state variable*** is established. The variable is written by the source and read by the state transition function of the target. Such a variable represents a ***passive*** exchange of data. The first opportunity at which the target will be able to use the information that was stored in the variable will be at the time of occurrence of the following event.

6.2.2 The Node

Once the mapping of the system onto a directed multigraph has been completed, the modeling process may proceed to its second stage - the ***nodd abstraction***. At this stage a Discrete Event System specification is established for each of the nodes that constitute the multigraph. Since the input/output structures of the nodes have been defined at the previous stage each node can be modeled individually. However similar nodes will most Iikely, be treated in the same manner and thus their **DEVS** specification is defined simultaneously. Throughout the modeling process the node is considered as the atomic building block of the model. Although it is the smallest component of the mode, a node is self-contained in both its logic and structure and therefore has an autonomous existence.

The internal presentation of a **DEVS** is given by its state variables. There are ***two*** types of state variables - the ***piecewise constant*** and ***countdown clock*** variables [Zeig76]. The variables of the first type describe the current state of the system, whereas the clock times at which internal events are scheduled to occur, are given by the second type. These times are internally determined and controlled by the node. Each countdown variable is associated with a different event and thus the set of these variables represents the internal events of the

DEVS. Scheduling such an event means an assignment of a positive value to a countdown clock. Once the variable is assigned it will decay linearly, as a function of time, until it reaches zero. Precisely when it reaches zero, the internal event associated with the variable occurs.

The piecewise constant variables change only when an internal or external event takes place. Such an event means that the system undergoes a change in state. As a result of a state transition, the state, the output and the mask variables of the system may change. The changes in these variables due to a particular event are defined by the state transition, the output and the masking functions (see A.2 for a detailed description of the elements of a DEVS specification structure). Although each of these functions is defined formally as a single relation, it is actually composed of several functions, each of which defines the reaction of the DEVS to a given event. All the functions associated with the same event form the *activity* of the event. The activity is a mapping into the range $S^M \times \vec{K}^M \times \vec{OP}^M$ and is a natural way to describe the transformational rules of the node.

Some of the DEVS specification of the various nodes obtained at this stage may be similar or even identical. In order to limit the number of realizations required, the specifications may be grouped into disjoint sets such that every set represents one *type* of DEVS. All members of the same set will be considered as different *instances of* the same type of DEVS and each node will get its individual characteristics by means of input parameters.

§6.3 Simulating With DISS

DISS is a high level simulation language which is a *macro extention* of the SIMSCRIPT 'II.5 simulation language. The extention is based on the define *to* mean and substitute mechanisms of the base language. All the routines that support the DISS language have been written in SIMSCRIPT II.5. As a result DISS is compatible to the same systems as SIMSCRPT II.5.[6] The desire to save the programming effort which the implementation of a preprocessor entails, motivated the macro extention approach. However, due to this approach the statements of DISS had to be structured according to the syntactical constrains of the base language.

The DISS language was designed to provide a tool for building modular discrete event simulation programs with well-defined interfaces between their modules. The design

[6] SIMSCRIPT 115 is available for CDC, Honeywell, IBM, NCR, PRIME, UNIVAC and VAX computers.

was guided by the idea that the smallest self-contained element of a discrete event model is a DEVS, and thus the simulation program should be constructed as a network of modules, each of which is a realization of a **DEVS.** Every module should be an autonomous and self-contained unit that interacts with the other units by means of a well-defined mechanism.

The **DISS** simulation language is process-oriented. A simulator written in **DISS** consists of a **preamble,** the ***Executive Manager*** of the experiment and a set of process descriptions. Each process is a realization of a **DEVS** specification type. The simulator is viewed as a directed multigraph whose nodes are instances of these processes. The language provides tools for the description of the behaviour of a DEVS by a self-contained autonomous process. The statements and data structures of the language constitute a mechanism for synchronizing the activities of the processes. This mechanism is based on an *arc* structure that interconnects the processes. Each arc is capable of capturing the inter-node events and variables defined by the output to input mapping which is represented by the arc.

All process types are named in the **preamble** of the **DISS** program **dong** with the external events and the inter-node state variables of the simulator. The inter-node state variables defined for the needs of the simulator are appended to the **DISS** arc structure. All the definitions of global variables and data structures should be included in the **preamble.** In most cases only those structures that are passed from one node to the other will be declared in the **preamble.** All other data structures that are needed for describing the model are the state-variables of the individual DEVSs and are defined locally by the processes.

The world view of **DISS** imposes a network structure for the simulator and defines the mechanism by which the nodes interact. Therefore the language can provide various services that are common to these types of simulators. These services reduce the amount of code needed for a behavioural description of the model and thus enable the the designer of the simulator to concentrate on the particular needs of his model.

In the following sections the salient properties and features of the **DISS** language will be described. This chapter is not intended to serve as a user guide or manual for the language. All the information needed for writing and running **DISS** programs can be found in the **DISS** user guide [Mel83a]

6.3.1 The Executive Manager

The execution of a **DISS** program is considered as a simulation EXPERIMENT that may consist of several runs. An experiment is characterized by the topology of the simulated

model, whereas the node type and the actual values of their input parameters specify a run. The experiment is established and controlled by the Executive Manager process of the simulator. All the activities of the Executive Manager are associated with the management of the simulation experiment and are not related to the logic of the model. Both the topology and the characteristics of the nodes are given as input data to the simulator. The Executive Manager reads in a weighted neighbor list which represents the directed multigraph and the types of the various nodes. The input parameter values for each individual node are read in by the node itself.

The course of a run is directed by the Executive Manager by means of control events. By scheduling such an event for a particular node the node is alerted imperatively and the activity of the given event is executed. Control events may be used for simulating faulty elements, for obtaining status reports at selected instances and for terminating the run.

Fig. 6.1 presents the basic structure of the Executive Manager. The process consists of two main elements - the *experiment* and the run *manager*. The main task of the first element is to establish all the data structures required for nodal interaction as determined by the topology of the experiment. This element is executed only once and thus most of the data structures which it establishes are permanent. The process instances that represent the nodes are activated by the run manager that is executed once for each run. Once activated, the processes can be controlled via control events. Such an event is also used for terminating the run and consequently eliminates all process instances. Before proceeding to the next run the manager must release all structures that were established by the run. Most of the tasks to be accomplished by these two elements are executed by the powerful DISS statements **init.the.network**, **init.the.nodes** and **terminate.run**

6.3.2 Wait Until Event

The realization of a Discrete Event System by a DISS process is based on the unique **wait.until.event** scheduling mechanism of the language. The mechanism enables a process to wait until one out of a dynamically selected set of events takes place. Once resumed, the process undergoes a two-phase decoding procedure at the end of which the activity area of the event that caused the resumption is reached and executed. The process manages the set of acceptable events by means of a masking system. With the exception of Control events, there is no way by which an event can cause a resumption of a process when the process

```
process exec.manager
                    ''experiment manager

init.the.network
:
while run.v<==runs.v, do
                    ''run manager

 init.the.nodes
 :
 terminate.the.run
 loop ''of run.v<==runs.v
                    ''experiment termination
 end ''of process exec.manager
```

Figure *6.1*. Structure of Executive Manager Process

considers this event unacceptable. The **wait.until.event** approach defines an *interrogative* scheduling mechanism that supports the autonomy of the process.

This scheduling mechanism is composed of the **following** three elements:

1. **Management of internal events -** Each countdown **clock of** the DEVS specification is related to an internal event in the process realization of the system. These events are defined, scheduled and manipulated internally by the process. **DISS** provides a wide variety of statements for managing internal events. Statements for scheduling, cancelling, updating, suspending and resuming such events are part **of** the language **and** assist the implementation **of** complex scheduling algorithms.
2. **Inter node alerts -** The mapping from an output variable to an input variable **or** external event is executed implicitly at the source process **of** the arc. When an output variable is mapped to an external event at the target process, the **set.alert** statement **is** used for assigning a new value to the variable. The statement , when executed, informs the scheduling system of **DISS** that an inter-node event **has** occured and thus an attempt is made to alert the target process. The number **of** the output port, the new value **of** the output variable and the name of the external event are the attributes by which the source process identifies an inter-node event and are thus part **of** the syntax **of** the **set.alert** statement.
3. Masking - Whenever an event occures the scheduler of **DISS** consults the process **masks**

in order to determine whether the process is waiting for this particular event. If the mask is set, the event will be placed in a pending state until its mask is reset. The **mask, unmask** and **mask.priority** statements provide a flexible tool for managing the masks of a node. A mask is defined per event and per input port, and can be controlled according to the priority levels of the ports.

The internal structure of the process is strongly affected by the properties of the **wait.until.event** scheduling mechanism. The cyclic execution of the wait statement and the two phase decoding procedure of events imposes a well-defined structure OR the process. A typical **DISS** process structure is shown in **Fig. 6.2.** Note that because of the structural isolation of the event activities, the simulator is endowed with a second level of modularity. An activity can be easily removed and replaced by another. This quality of the nodal structure assists the programmer in merging two processes into one.

Each event, internal, external or control, has two attributes associated with it - a ***value*** and ***port number.*** For an external event the first attribute holds the value of the output variable of the source node that is mapped to this event. The second attribute is the number of the event input port. The attributes of an internal event are assigned when the event is scheduled and can be used €or binding an event to a given port or entity.

6.3.3 Allocation of Nodal Data-Structures

The degree to which a process can be self-contained depends mainly on its ability to declare variable and data structures locally. When a process uses global variables their definition becomes a part of the process although it is not included in the process. A **DISS** process can locally define a wide variety of data structures. In addition to variables and arrays,[7] a **DISS** process can locally define **sets**, ***random variables,*** and ***statistics recording probes.*** The three latter structures are defined by the **establish** statement. This statement, in addition to its declarative role, leads to the establishment of an instance, represented by a ***temporary entity,*** of the structure. The name of the structure, as given in the statement, serves as a pointer to this entity.

Two types of sets - ***fifo*** and ***ranked*** by ***high value*** - can be defined by the **establish** statement. All the set operations associated with these sets are executed by using SIMSCRIPT II.5 statements.

[7] In SIMSCRIPT II.5 only **variables and arrays** can **be defined locally.**

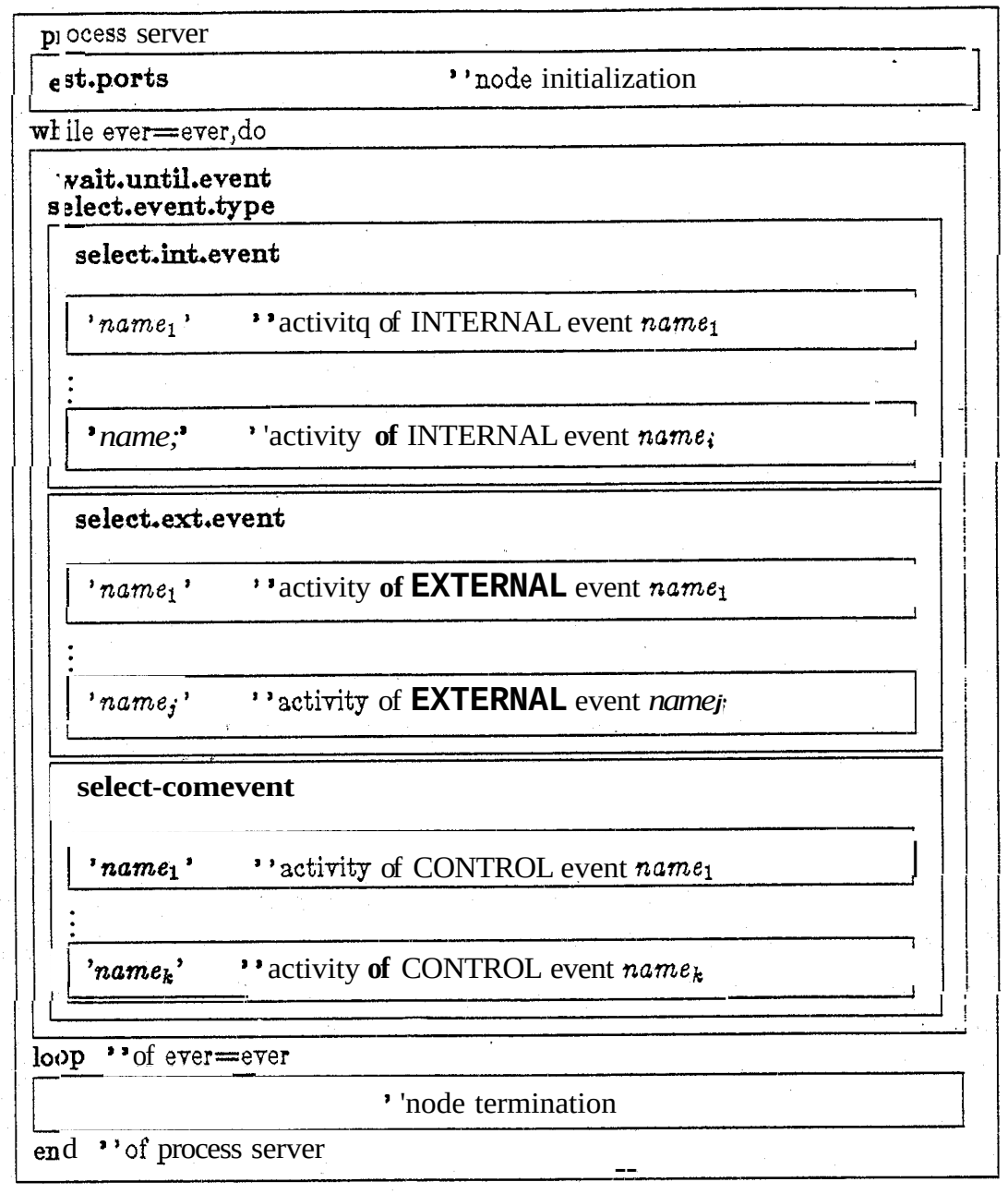

Figure 6.2. Typical Process Structure

By establishing a probe the process builds a tool for sampling a phenomenon and for obtaining a set of statistical quantities for the sample. The probe structure together with the **measure** statement constitute a mechanism for deriving the ***average, standard deviation, maximum,*** and ***minimum*** of a selected `measure`. The statistical computation method used for deriving these quantities is determined by the type assigned to the probe when established.

Two methods are available - the *tally* and *accumulate* methods [Russ83]. In the second method a sample is weighted by the duration that measured phenomena remained unchanged with the value that was sampled.

Due to the statistical properties of simulation experiments, confidence intervals are widely used for evaluating the quality of the results obtained from simulation runs. In order to facilitate this evaluation process, a ***batch*** probe has been included in DISS. The probe operates according to the *batch* ***mean*** method for statistical analysis[Gord78]. The **size** of each batch into **which** the sample is divided is controlled by a global variable. The confidence interval for a selected level of a batch probe is retrieved by using the **conf.int** function. In order to evaluate the randomness of the batch means **DISS** provides a function for computing the auto-correlation of these means[Law 79].

6.3.4 Tracing and Debugging

In the design of the **DISS** language special attention has been devoted to the development of tracing and debugging utilities. Traces are a useful tool for relating the behaviour of the simulator to the specification **of** the model. By following the sequence of state changes the simulator undergoes, one can decide whether the implementation follows the transition rules of the model. Due to the complexity and size ***of*** DPS simulators **it is** difficult to follow their activities over a period of time, and to see how their various elements interact. **A** considerable amount of data is required for describing the behaviour of such a simulator and thus the handling of trace information might be an involved process.

The trace reporting utility of **DISS** is based on the **snap** statement and is controlled by input parameters. At those program locations where a report is desired a **snap** statement has to be inserted. The statement generates a report - a ***four*** letter literal and two integer values - that is displayed in columns on the output device (see example in Appendix C). Each column is associated with a node so that by following the data presented at a given column the activities of a selected node can be traced. The lines of the tracing report are related to simulation time, in increasing order. An attempt is made to place as many reports **as** possible on the same line.

The four letter literal identifies a tracing report and relates it to the activity that caused the report. By means of input parameters ***a*** subset of reports can be selected OF excluded according to their literals so that only reports that belong to this subset will be

displayed. This facility together with a global tracing level mechanism[8] constitute a flexible tracing utility.

The debugging utility of **DISS** assist the programmer in isolating the cause of a run-time error. The utility is based on a detailed event report that describes the current state of the simulator. The report details the attributes of the *active* event and all *immediate, pending* and *scheduled* events. This information complements the data provided by the standard trace-back report of SIMSCRIPT II.5 when a run-time error occurs. The status report can be invoked by the programmer at selected locations by using the **report.events** statement. The amount of data presented is controlled by two global **DISS** variables.

[8] **DISS has a global variable named TRACE.L whose value changes in time according to a pattern controlled by input parameters.**

Chapter 7

Conclusions and Directions for Further Research

Since the early days of mankind the primary motivation for the establishment of communities has been the idea that by being part of an organized group the capabilities of an individual are improved. The great progress in the area of inter-computer communication led to the development of means by which *stand-alone* processing sub-systems can be integrated into ***multi-computer*** 'communities'. The major object of this investigation has been to define methods by which a processing sub-system which belongs to such a community can take advantage of the other members of this community in order to enhance its response time and at the same time to assist the others in achieving the same goal. By doing so, a 'small' sub-system can provide the services of a 'large' one.

§7.1 Conclusions

The results obtained from the study of the LB algorithms which were defined in this thesis have demonstrated the ability of the task migration process to reduce the response time of a DPS. In the opening analysis it was shown that in a multi-resource system which does not employ an LB mechanism, there is a high probability that a task will be waiting for service while at the same time a server which is capable of serving it, is idling. The different load balancing algorithms which were defined, establish a set of distributed decision processes which DPSs of various kinds may use in order to take advantage of the multiplicity of their resources. The performance measures obtained in the study point at the ability of the task migration process to reduce the systems' response time even when the communication and the processing overheads associated with this process are none trivial. This ability indicates that

task migration is a 'practical' approach and thus should be part of any distributed processing environment

Reduction in the expected waiting time of a task due to load balancing is one of the benefits which a number of stand-alone systems may achive by establishing a multi-computer community. Although in some cases a larger community does not mean better performance, an individual subsystem can improve the quality of services it provides by joining a multi-computer environment. However, in order to achieve the desired improvement, the LB algorithm has to be adjusted to the size as well as to the other properties of the system. The taxonomy of load balancing which was presented in chapter 2 was used throughout the study for describing and characterizingthe different LB algorithms that were discussed in the thesis. In the various case-studies which were analyzed in the course of this investigation it was shown that under a given set of operating conditions and for a system with given characteristics, different balancing algorithms might have opposing effects on. the system's performance. Nevertheless, when a 'wrong' migration criterion is selected or a too 'liberal' information policy is employed, the LB process may become a cause of performance degradation.

It was shown that for a broadcast DPS, higher resource multiplicity does not necessarily result in better response time. Each of the LB algorithms which were defined for this type of systems reaches a point at which an increase in the number of servers decreases the performance of the system. Therefore when a number of processing systems is given it might be better, as far as the $\hat{W}_q$ is concerned, to assemble them into two or more multi-resource systems than to integrate them into one system. It was found that when an LB algorithm which utilizes the broadcast capabilities of the communication subnet is used, a front-end communication processor has to be attatched to each system. The processing capacity which this type of message requires should not be taken from the main processor.

Since the addition of a node to a store-and-forward system means an increase in the number of communication links, an increase in the size of such a system does not cause a degradation in the performance of the system. However, the manner in which the sub-systems which constitute such a system are interconnected, i.e. the topology of the communication subnet, does effect the ability of the LB process to enhance the system response time. This interdependency between the topology and the behaviour of the LB process should be taken into consideration when a store-and-forward system is designed.

The strong interdependence between the DPS characteristics and the preformance of the LB algorithm demonstrate the importance of performance prediction as a design tool for

such decision processess. It was found that when simulation is used as a solution method for a DPS model the DISS methodology and simulation language assist the design, the realization and the execution of the simulation study. The modularity of the simulators and the utilities provided by DISS have enhanced the construction process of the simulators which were needed for this thesis, These advantages of the DISS approach have been demonstrated by a number of other studies which have analyzed various aspects of the performance of DPSs [Levi82],[Camp83],[Kant83].

§7.2 Directions for Further Research

All the studies of the load balancing problem in DPS systems have considered the task migration process as an isolated phenomenon. Now as a better understanding of the properties of this process has been acquired, the interaction between the load balancing process and other phenomena associated with DPSs should be investigated. Methods for incorporating load balancing considerations into distributed database management systems and distributed computing mechanisms, should be developed and their performance studied. The information which has been accumulated on the basic characteristics of load balancing algorithms should be used as a basis for studies which focus on more specific aspects of the problem.

Another area for research should be to try to develop a framework for a comparative evaluation of control processes for DPS. The dependency between the behaviour of these algorithms and the system parameters deters from any attempt to select the ultimately 'best' algorithm. For distributed routing, concurrency control and LB algorithms there is no absolute answer to the question '*is algorithm* A better *then* B ?' (see [Mcqu80] and [Gall82]). Therefore a systematic scheme together with a set of well defined criteria for evaluating such algorithms has to be established.

Performance prediction will be the main tool such a scheme would employ. More effort has to be devoted to the development of numerical, iterative and simulation methods for solving performance models of DPSs. An attempt should be made to use advanced iterative solution schemes for solving multi-dimensional birth and death processes. The framwork for modeling and simulating DPSs that has been defined by DISS is not yet a complete structure. Additional utilities should be added to the language and a better understanding of the process in which a system specification is transformed into a model should be acquired.

Appendix A

§A.1 TR for LookAhead policy

Assume an M/M/2-like system with a 'look ahead' migration policy and let $A = \{P_1, P_2\}$ be the set of the system processors. Due to the migration policy and the properties of an M/M/m-like system, $\Delta L(A,t)$ can be one or zero and the probability that the system will be in a WI state is zero. A transfer will be initiated whenever the $\Delta L(A,t)$ is one, and a task has arrived at the longer queue or departed from the shorter one. Thus the transfer rate of this policy is given by

$$TR_1 = (\lambda + \tilde{\mu})\, P[\Delta L(A,t) = 1] - \tilde{\mu} P[\textit{1 task in the system}] \tag{1}$$

Since $P_{w,i}$ is zero, the above equation can be rewritten as

$$TR_1 = (\lambda + \tilde{\mu}) \sum_{i=0}^{\infty} P_{2i+1} - \tilde{\mu} P_1 \tag{2}$$

where P_i is the probability of having i tasks in *an* M/M/2 system. Replacing P_i by $P_0 2\rho^i$ [Klei75] and factorizing the following can be derived

$$TR_1 = 2 P_0 \rho [(\lambda + \tilde{\mu}) \sum_{i=0}^{\infty} \rho^i - \tilde{\mu}] \tag{3}$$

By using the equation for the sum of a geometric series the final equation for TR_1 is obtained:

$$TR_1 = 2\tilde{\mu} P_0 \frac{\rho^2}{(1-\rho)} \tag{4}$$

§A.2 TR for 'Trouble Shooting' Policy

Assume an M/M/2-like system which is controlled by the 'trouble shooting' migration policy. In such a system a task is transferred from one queue to another whenever one of the following events occurs:

E1 A task has arrived at a non-empty queue when there is only one task in the system.

E2 A server has completed the service of a task, no other tasks are waiting in its queue and there are two or more tasks in the system.

Therefore the transfer rate of the system is given by

$$TR_2 = \lambda P[\textit{one task in the system}] + \tilde{\mu} P[\textit{one task in one server and two or more in the other}] \quad (1)$$

By defining

$$P_{i,j} \triangleq P[n_1 = i \textit{ and } n_2 = j] \quad (2)$$

and

$$\hat{P}_i = \sum_{j>i} (P_{i,j} + P_{j,i} \quad (3)$$

Eq. (1) can be rewritten as

$$TR2 = \lambda \hat{P}_0 + \tilde{\mu} \hat{P}_1 \quad (4)$$

and the following set of equations can be derived

$$\lambda \hat{P}_i = \tilde{\mu}[\hat{P}_{i+1} + 2P_{i+1,i+1} \qquad i \geq 1 \quad (5)$$

Summing Eq. (5) for all applicable i yields

$$\lambda \sum_{i=1}^{\infty} \hat{P}_i = \tilde{\mu} \left(\sum_{i=2}^{\infty} \hat{P}_i + 2 \sum_{i=2} P_{i,i} \right) \quad (6)$$

Since $\sum_{i=0}^{\infty} \hat{P}_i + P_{0,0} = 1$ and by factorizing the following equation for $\hat{P}_1$ can be derived

$$\hat{P}_1 = (1-\rho)(1 - \hat{P}_0 - P_{0,0}) + 2 \sum_{i=2}^{\infty} P_{i,i} \quad (7)$$

In an M/M/2-like system in which no tasks are transferred, all task distributions with the same total number of tasks are equally probable. Therefore because of the properties of the migration policy of the system the following inequalities can be concluded

$$P_{i,i} \geq \frac{i}{(2i-1)} P[2i \textit{ tasks in the system}] \qquad i \geq 1 \quad (8)$$

$$P_{i,1} \leq \frac{1}{i} P[i+1 \textit{ tasks in the system}] \qquad i \geq 1 \quad (9)$$

$$\hat{P}_0 = \frac{1}{2}[P_{0,1} + P_{1,0}] \quad (10)$$

From (8),(9),(10) and since P_{wi} is zero the following lower bound for $\hat{P}_i$ is obtained

$$\hat{P}_1 \geq (1-\rho)(1-p_1-P_0) + 2\sum_{i=2}^{\infty} \frac{1}{(2i-1)} P_{2i} \tag{11}$$

where P_i is the probability of having i tasks in an equivalent M/M/2 system. From **(1) and (11)** and by replacing P_i by $2P_0\,\rho^i$ the following lower bound for TR_2 **is** derived

$$TR_2 \geq \tilde{\mu}[1-\rho+P_0(1+\rho)[\rho \ln\left(\frac{1+\rho}{1-\rho}\right)-1]] \tag{12}$$

On the other hand from the definition of $\hat{P}_i$ and Eq. **(9)** it follows that

$$= 2\sum_{i=2}^{\infty} P_{i,1} \leq 2\sum_{i-2}^{\infty} \$Pi + 1 \tag{13}$$

By the same reasoning used for deriving Eq. **(12)** the following upper bound for TR_2 is derived from **(5)** and Eq. **(13)**

$$TR_2 \leq \tilde{\mu}\, 2P_0\rho[\rho - 2[\rho + \ln(1-\rho)]] \tag{13}$$

§A.3 $TDF_{i,j}$ for S-BTSQSS systems

Assume two independent M/M/1 systems. the service rate of each system is $(1-\delta_f)\tilde{\mu}$ and task arrive at each queue at **a** rate of λ tasks per time unit. Let $q_1(t)$ and $q_2(t)$ be their queue sizes at time t respectively and let $\hat{p}_{i,j}(k,l,t)$ be defined as

$$\begin{aligned}\hat{p}_{i,j}(l,k,t) \triangleq P[(q_1(t),q_2(t)) = (k,l) \in U \mid (q_1(0),q_2(0) = (i,j) \in U \text{ and} \\ (q_1(x),q_2(x)) \in U \text{ for } x \in (0,t)]\end{aligned} \tag{1}$$

where $U = \{(n,m) \mid n,m \geq 0 \wedge (n-m-1 > L_p) \wedge (n-m > m\,A_p)\}$
From the properties of an M/M/1 system it follows that

$$\begin{aligned}\frac{\partial}{\partial t}\hat{p}_{i,j}(l,k,t) = &-[2\lambda + (1-\delta_f)\tilde{\mu}(e_k+e_l)]\hat{p}_{i,j}(l,k,t) \\ &+ (1-\delta_f)\tilde{\mu}[\hat{p}_{i,j}(k+1,l,t) + \hat{p}_{i,j}(k,l+1,t)] \qquad \text{for all } k,l \in U \\ &+ \lambda[\hat{p}_{i,j}(k-1,l,t) + \hat{p}_{i,j}(k,l-1,t)]\end{aligned} \tag{2}$$

where $\hat{p}_{i,j}(n,m,t) \triangleq 0$ for $n,m \notin U$.

From the definition of the AT algorithm for a S-BTSQSS system it can be shown that for such a system

$$\hat{p}_{Tar(i,j)}(t) \triangleq P[\textit{transfer initiated at t=0 with TD(0)=(i,j) will terminate in } (t,t+\Delta t) \mid \textit{it had not terminated in } [0,t]] = \left(\beta + \frac{(1-\delta_f)\tilde{\mu}\hat{p}_{Tde(i,j)}(t) + \lambda\hat{p}_{Tar(i,j)}(t)}{\hat{p}_{ub(i,j)}(t)}\right)\Delta t \tag{3}$$

with

$$\hat{p}_{ub(i,j)}(t) \triangleq \sum_{(k,l)\in U} \hat{p}_{i,j}(k,l,t)$$

$$\hat{p}_{Tar(i,j)}(t) \triangleq \sum_{(k,l)\in V} \hat{p}_{i,j}(k,l,t)$$

$$\hat{p}_{Tde(i,j)}(t) \triangleq \sum_{(k,l)\in W} \hat{p}_{i,j}(k,l,t)$$

$$V \triangleq \{(n,m) \mid (n,m) \in U \wedge (n,m+1) \notin U\}$$

$$W \triangleq \{(n,m) \mid (n,m) \in U \wedge (n-1,m) \notin U\}$$

From (1) and (3) the following expression for $L_{i,j}(t)$ in a S-BTSQSS system can be derived

$$\begin{aligned} L_{i,j}(t) = \{p_{Tar(i,j)}(t)\}^{-1}\Bigg(&\beta \sum_{l,k\in U} (l+k)\hat{p}_{i,j}(k,l,t) \\ &+ \{\hat{p}_{ub(i,j)}(t)\}^{-1}\Big((1-\delta_f)\tilde{\mu}p_{Tde(i,j)}(t) \sum_{(k,l)\in W} (k+l)\hat{p}_{i,j}(k,l,t) \\ &+ \lambda p_{Tar(i,j)}(t) \sum_{(k,l)\in V} (k+l)\hat{p}_{i,j}(k,l,t)\Big)\Bigg) \end{aligned} \tag{4}$$

By similar reasoning it can be shown that for S-BTSQSS systems the p.d.f of the length of a transfer which was initiated at $t=0$ with $TD(0) = i,j$, $f_{tr(i,j)}(t)$, is given by

$$f_{tr(i,j)}(t) = e^{-\beta t}\{\beta p_{ub(i,j)}(t)(1-\delta_f)\tilde{\mu} \sum_{(k,l)\in W} \hat{p}_{i,j}(k,l,t) + \lambda \sum_{(k,l)\in V} \hat{p}_{i,j}(k,l,t)\} \tag{5}$$

Since in the definition of $\hat{L}_{i,j}(t)$ it is assumed that transfers do not terminate in the interval $[0,t]$, $\hat{L}_{i,j}(t)$ is given by:

$$\hat{L}_{i,j}(t) = \sum_{k,l\geq 0} (k+l)p_i(k,t)p_j(l,t) \tag{6}$$

with $p_i(k,t)$ being the probability that an M/M/1 system with service rate $\tilde{\mu}$ and arrival rate λ will have k customers at time t given that there were i customers at $t=0$. According to the definition given in 3.5.1 by computing (4), (5) and (6) the throughput degradation factor of a transmission in an S-BTSQSS system can be derived.

§A.4 $TDF_{i,j}$ for NS-BTSQSS Systems

The computation of the throughput degradation factor for a NS-BTSQSS is much simpler than in the case of a stop system. Assume an M/M/1 system with service rate $(1-\delta_f)$ and arrival rate λ and let $\tilde{p}_i(k,t)$ be the probability that the system will have k customers at time t given that there were i customers at $t=0$ then it follows that NS-BTSQSS $L_{i,j}(t)$ is given by

$$L_{i,j}(t) = \sum_{k,l\geq 0} (k+l)\tilde{p}_{i-1}(k,t)\tilde{p}_j(l,t) + 1 \tag{1}$$

The p.d.f of the duration of a transfer for such a system is the following:

$$f_{tr(i,j)}(t) = \beta e^{-\beta t} \tag{2}$$

$\hat{L}_{i,j}(t)$ is the same as for the S-BTSQSS system and thus from (1) (2) and Eq. (5) in the previous section $TDF_{i,j}$ for a nostop system can be derived.

§A.5 SF_{ij} for S-BTSQSS system

A transfer of duration t will not be stopped in the middle if during the transfer period the task distribution meet the criterion of the migration policy. The probability that this will happen for a transfer that was initiated at $t=0$ with $DT(0)=(i,j)$ is:

$$P[\textit{transfer of length } t \textit{ was not stopped in the middle} \mid TD(0)=(i,j)] = 1 - \int_0^t p_{ub(i,j)}(t)dt \tag{1}$$

Since the duration of a transfer has a negative exponential distribution the following expression for $SF_{i,j}$ can be obtained:

$$SF_{i,j} = \int_0^\infty \beta e^{-\beta t}\left(1 - \int_0^t p_{ub(i,j)}(x)dx\right)dt \tag{3}$$

Appendix B-DEVS Specification

§B.1 DEVS specification

A definition of a *Discrete event system specification* is presented in this section. The definition extends the specification structure as has been defined by Ziegler in [Zeig78], so that *masks* and *input/output* ports are included in it. The world view of DISS that considers DEVS as autonomous elements that can be loosely coupled one to the other, motivated this extention. This extended structure provides the means for describing the behaviour of autonomous DEVS. Due to the port structures included in the specification definitions of DEVS network should not include elements from the state sets of the individual systems. The definition of the network then reflects the looseness of the coupling between its components.

Definition: A DEVS specification is a structure[1]

$$M = < \vec{\mathrm{IP}}^M, S^M, \vec{\mathrm{OP}}^M, \vec{K}^M, \delta^M, \xi^M, \lambda^M, t >$$

where

$\vec{\mathrm{IP}}^M = \{\mathrm{IP}_1^M, \ldots, \mathrm{IP}_n^M\}$ is a structure - the input ports structure. Each of the indivdual input ports is a structure:

$$\mathrm{IP}_i^M = \{X_i^M, I_i^M\}$$

with the first element of the structure being the set of external events of the input port a:

$$X_i^M = (\chi_{i_1}^M, \ldots, \chi_{i_l}^M)$$

and the second element being the set of input variables of port i:

$$I_i^M = (\beta_{i_1}^M, \ldots, \beta_{i_l}^M)$$

S^M is a set - the set of sequential states. S^M is the cross product of the range of the state variables, $\alpha_1, \ldots, \alpha_k$.

$\vec{\mathrm{OP}}^M = \{\mathrm{OP}_1^M, \ldots, \mathrm{OP}_m^M\}$ is a structure - the output port structure. Each output port is a set of output variables:

$$\mathrm{OP}_i^M = (\gamma_{i_1}^M, \ldots, \gamma_{i_j}^M)$$

---

[1] An attempt was made in the definition to use the same notation used in [Zeig76] as much as possible.

$\vec{K}^M = \{K_1^M, \ldots, K_m^M\}$ is a structure - the mask structure, The dements of the structure are sets of masks

$$K_i^M = (\kappa_{i_1}^M, \ldots, \kappa_{i_j}^M)$$

^{M} is the mask of the external event $\chi_{i_j}^M$. The masks serve as a means by which the system communicates with its surrounding. By setting a mask the system declares whether it consideres a given change in the state of the universe around it as an event. Therefore an external event may occur at time t only if its mask is set at that time.

δ^M is a function - the quasitransition function. Let $Q^M = \{(s, e) \mid s \in S^M, 0 \leq e \leq \tilde{t}(s)\}$ be the state space of the system and $\mathbf{3} \notin \bigcup X_i^M$ a symbol that denotes the 'nonevent' then δ^M is a mapping:

$$\delta^M : Q^M \times (\bigcup X_i^M \cup \{\Phi\}) \times \text{INPUTS} \rightarrow S^M$$

where INPUTS is the cross product of the range of all input variables and $\tilde{t}(s)$ is the duration of state s when no external events occur.

ξ^M is a function - the masking function. The masking function maps from $Q^M \times INPUTS$ onto $\vec{K}^M$.

λ^M is a function - the output function which maps from $Q^M \times INPUTS$ onto $\vec{OP}^M$

$\tilde{t}$ is a function - the time advance function. $\tilde{t}$ is a mapping from S^M into to the nonnegative reds:

$$\tilde{t} : S^M \rightarrow R_{o,\infty}^+$$

The value of $\tilde{t}(s)$ is defined to be $\min\{\sigma_i\}$ where $\{\sigma_i\}$ is the set of countdown-clock state variables of the system M. Therefore $\tilde{t}(s)$ can be interpreted as the duration of state s when the system is isolated (no external events).

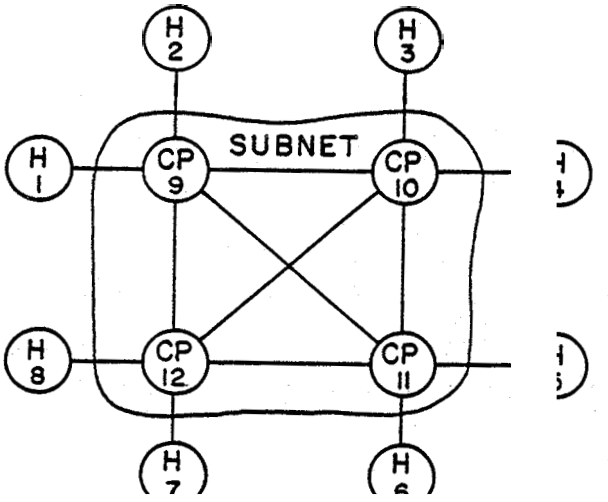

Figure *C.1.* The Distributed System

Appendix C-Example

Assume that a simulation study of the distributed processing system presented in Fig C.1. has to be performed. In this example the main elements of the modeling phase of the study are described and the listing of a simulator (written in DISS) that realizes the model, is presented. The system consists of a number of hosts that are interconnected by a message switching store-and-forward communication system. The subnet is made up of communication processors, cp, that are connected by full-duplex communication lines. Each cp has a finite buffer space in which the messages are stored. Therefore the communication protocol must perform a 'space reservation' step before a message is transmitted. Each host receives an independent stream of tasks. Every task is assigned an execution site at which it will be served. This assignment is performed by the resource allocation algorithm of the distributed system. The task departs from the system via the host of entry into the system,

§C.1 Model Definition

C.1.1 Structural Abstraction

C.1.1.1 Mapping to a Directed Graph

The elements of the above system may be grouped into nodes in a number of ways three of which are listed below:

1. Each element of the system defines a node. The model will include two types of nodes,
2. The **host** and the **cp** are grouped into one node so that the model has only one type of node. An input parameter will determine whether the node is a host, a **cp** or both.
3. A **host** defines one type of node whereas all the communication processors of the network are grouped into a second node type. This second node will represent the entire network. In this case the topology of the network will be represented internally by this node.

The selection of a mapping scheme depends strongly on the experimental frame of the study (see [Mel83b] for a detailed discussion). Each of the above schemes can be considered as being the best in keeping with the requirements of different studies. One scheme may be more modular whereas another may have a more efficientimplementation. A detailed analysis of the above schemes is beyond the scope of this example. For the purpose of this example it will be assumed that the first scheme has been selected.

C.1.1.2 Arc definition

The inter-node state variables of the model are the following:

INV1 - buffer.full indicates the state of the message buffer of a **cp.**

INV2 - **wait** will be set whenever the node wants to transfer a message along the arc and the buffer of the target node is full.

The inter-node events of the model are the following:

INE1 - start.trans form a **host** or a **cp** to a **cp.** Indicates that the source node has started to send data into a buffer at the target.

INE2 - **end.trans** between every pair of interconnected nodes. The Occurrence of this event indicates that the last data unit of the message has arrived at the buffer.

INE3 - **buffer.avail** this event takes place when a **cp** whose buffer state has changed from full to available, assigns a buffer to a node which is in a wait state. Such an event may be caused only by a **cp** but should be accepted by both types.

Note that due to this approach to the inter-nodal information exchange the **cp** is given full autonomy in allocating available buffer space. The algorithm used by the **cp** is transparent to the node that sends the message. In this example it is assumed that each **cp** supports a dedicated buffer space for each input port. By means of the buffer.full variable and

the **buffer.avail** event it can select which **waiting** node will be given a buffer .space that has become available. In addition to the dedicated buffers, a number of spare buffers are provided, and determined as an input parameter. Before each node can be implemented as a process the Internal Events of each node have to be defined.

C.1.2 Behavioural Abstraction

Only the internal events of the **host** will be listed here. All the other details of the behavioural .description of the two DEVS can be derived from the listing of the simulator which is appended to the example. *All* the reserved names of DISS appear in the listings in capital laters.

C.1.2.1 The **host**

The **host** node includes the following Internal Events:

HIV1 - **end.message** This event represents the **delay** associated with the transfer of a message.

HIV2 - **end.task** The end of the execution period of a task is represented by this event.

HIV3 - **task.arrival** The arrival procedure of the **tasks** is modeled by this event. The arrival of one task causes the scheduling of the next arrival.

§C.2 The Simulator

C.2.1 The Preamble

```
309  ''preamble for point-to-point simulator
310
311
312
313      DECLARE end.trans        X_EVENT(1)
314      DECLARE buffer.avail     X_EVENT(2)
315      DECLARE start.trans      X_EVENT(3)
316
317      DECLARE termination      C_EVENT(1)
318
319    proceeeee include boat,  communication.processor
320  ''========
321
322    temporary entities
323  ''================
```

```
324
325     every ARC has a buffer.full,  a wait
328
327       define buffer.full , wait  as integer variables
328
329     every task may belong to a DISS.SET and has a id,
330       a destination, a length, a arrival.time, an entrance.site, a buf.n
331       and an  exec.time
333
333       define destination , id, task.counter, entrance.site, buf.no
334         ae integer variables
335       define arrival.time, length, exec.time aa double variables
338
337     define n.hosts as integer variables
338 end '' of point-to-point simulator preamble
```

C.2.2 The Host

```
 1  process host                              ''22:50:41   83/07/29
 2     DECLARE end.message      I.EVENT(1)
 3     DECLARE end.execution    I.EVENT(2)
 4     DECLARE arrival          I.EVENT(3)
 5
 8     ESTABLISH taskq      TO.BE FIFO.SET
 7     ESTABLISH oatq       TO.BE FIFO.SET
 8     ESTABLISH sys.time   TO.BE T.PROBE
 9
10     define i, arr.seed, exec.seed, trans.seed, task.counter as integer va
11     let arr.seed   □  NODE.V
12     let exec.seed  =  N.NODE+NODE.V
13     let trans.seed =  2*N.NODE+NODE.V
14     define int.arrival, low.exec, high-exec, lor.trans, high.trans
15        as double variables
16
17     read int.arrival                  start new c u d
18     read low.exec, high.exec, low.trans, high.trans   start new card
19     write BODE.V,int.arrival,low.exec,high.exec,lor.trans,high.trans as
20        i 3,5 d(6,1), /, /
21
22     EST.PORTS
23
24     define port.util as 1-dim integer array
25     reserve port.util(*) aa OUT.DEGREE(NODE.V)
28     for i = 1 to OUT.DEGREE(NODE.V) by 2, do
27        ESTABLISH port.util(i) TO.BE A.PROBE
28     loop
29     SET.TIMER exponential.f(int.arrival,arr.seed) FOR.E arrival
30
3f     while ever = ever , do
32        WAIT.UNTIL.EVENT
33        SELECT.EVENT.TYPE
34           SELECT.EXT.EVENT
36
36           'end.trans'    ''a message has arrived
```

```
36          'end.trans'     ''a message has arrived
37  ''**>'      if TRACE.L>2 SXAP "eota",0,id(VALUE.E) always
38              if entrance.site(VALUE.E)= NODE.V ''a local task has arrived
39  ''==>''        MEASURE time.v-arrival.time(VALUE.E) WITH.PROBE sys.time
40              destroy the task called VALUE.E
41              else      ''an external task has arrived
42                 file the VALUE.E in SET(taskq)
43                 if n.SET(taskq) = 1  ''start to execute the new task
44                    SET.TIMER exec.time(VALUE.E) FOR.E end.execution
46                 always    ''of n.SET(taskq)
46              always    ''of  entrance.site(VALUE.E)
47              cycle     ''of ever = ever
48
49          'buffer.avail'       ''a wanted cp has an available buffer
60
61              subtract 1 from PORT.E  ''switch to transmission port
62              let wait(OUT.ARC(PORT.E)) = 0 ''reset inter node variable
53  ''**>''     if TRACE.L>2 SNAP "bfav",destination(f.SET(outq)),PORT.E   a
64              go to beg.trans   ''intiate a message transmission
66
56          SELECT.INT.EVENT
67
68          'end.message'  ''end of transmission delay
69              remove first task from SET(outq)
60  ''==>''     MEASURE 0 WITH.PROBE port.util(PORT.E)
61              SET.ALERT PORT.E FOR.E end.trans, task  ''an inter node even
62  ''**>''     if TRACE.L>2 SXAP "eott",destination(task),id(task)  always
63              if   n.SET(outq) > 0   go ta beg.trans    always
64              cycle  ''of ever = ever
66
66
67          'end.execution'   ''end of taak execution
68
69              remove the first task from SET(taskq)
70  ''**>''     if TRACE.L>2 SXAP "eote",0,id(task)  always
71              if n.SET(taskq) ↓=0     ''more tasks to process
72                 SET.TIMER exec.time(f.SET(taskq)) FOR.E end.execution
73              always     ''of n.SET(taskq)
74              if entrance.site(task) = NODE.V  ''was it a local task
75  ''==>''        MEASURE time.v-arrival.time(task) WITH.PROBE sys.time
76              destroy the task
77              else      ''return task to arr.host
78                 let destination(task) = entrance.site(task)
79                 file task in SET(outq)
80                 if   n.SET(outq) = 1 go to beg.trans   always
81              always   '' of entrance.site(task)
82              cycle     ''of ever = ever
83
84          'arrival'
86
86              SET.TIMER exponential.f(int.arrival.. arr.need) FOR.E arrival
87              create a ''new'' task
88              let arrival.time(task) = time.v
```

```
89              add 1 to task.counter
90              let id(task) = NODE.V*100000 + task.counter
91              let exec.time(task) = uniform.f(low.exec,high.exec,exec.seed
92              let entrance.site(task) = NODE.V
93              let destination(task)=randi.f(1,n.hosts,arr.seed)
94  ''**>''     if TRACE.L>2 SNAP "tarr",destination(task),id(task) always
96              if destination(task) = NODE.V  ''execute the task locaiy
96                 file taek in SET(taskq)
97                 if n.SET(taskq) = 1  ''is it the only task
98                    SET.TIMER exec.time(task) FOR.E end.execution
99                 always  ''of n.SET(taskq)
100             else  ''execute task at a remote host
101                let length(task) = uniform.f(low.trans,high.trans,trans.s
ioa                file taek in SET(outq)
103                if n.SET(outq) = 1 go to beg.trans always
104             always  ''of  destination(task)
105             cycle   ''of ever = ever
106
107          'beg.trans'   ''try to intiate a message transmission
108
109             let PORT.E = SEL.PORT(destination(f.SET(outq)))
110             if buffer.full(IN.ARC(PORT.E)) = 0  ""the target buffer in av
111                SET.TIMER length(f.SET(outq))  FOR.E end.message, 0, port
112 ''==>''        MEASURE 1 WITH.PROBE port.util(PORT.E)
113                SET.ALERT PORT.E FOR.E start.trans  ''inter node event
114             elee  ''buffer ie not available
115                let wait(OUT.ARC(PORT.E)) = 1 ''set inter node variable
116             always  ''of buffer.full(IN.ARC(PORT.E))
117             cycle    ''ever = ever
118
119         SELECT.CON.EVENT
120
121         'termination'  ''print statistics and terminate
122            write NODE.V, AVG.P(sys.time) as  i 4.d (10,3)
123            for i = 1 to OUT.DEGREE(NODE.V) by 2
124               write NUM.P(port.util(i))/2,AVG.P(port.util(i)) as i 5,d(7
126               leave
126
127     loop
128     for each taek in SET(outq)  , do
129        remove the task from SET(outq)  ''and''  deetroy the task loop
130     for each taek in SET(taskq)  , do
131        remove the taek from SET(taskq)  ''and''  destroy the task  loop
132
133     DISPOSE.NOFE
134 end ''of procese host
```

C.2.3 The CP

```
1  process communication.processor
2     DECLARE end.message        I.EVENT(1)
```

```
4     define ever as an integer variable
6     define i, oatb. des.port, rund.buff, spb.ctr, act.message, available
8        as integer variables
7     define baf.vec, out.port, port.util as 1-dim integer arrays
8
9     EST.PORTS
10
11    read outb                    start new card
12    write NODE.V, oatb as i 3,b 37,i 2,/,/
13    reserve out.port(*), port.util(*) ae OUT.DEGREE(NODE.V)
14    reserve buf.vec(*) as in.degree(NODE.V) • outb
15    for i = 1 to OUT.DEGREE(NODE.V) by 2, do
16       ESTABLISH out.port(i)  TO.BE FIFO.SET
17       ESTABLISH port.util(i) TO.BE.A.PROBE
18    loop
19    let rund.buff = 2
20
21    while ever = ever, do
22       WAIT.UNTIL.EVENT
23       SELECT.EVENT.TYPE
24          SELECT.EXT.EVENT
26
26          'start.trans'  ''a neighbor has intiated a transfer
27             let buffer.full(OUT.ARC(PORT.E)) = 1 ''set inter node variab
28    ''**>''  if TRACE.L>2 SNAP "sota", PORT.E, 0  always
29             cycle  ''of ever = ever
30
31          'end.trans'    ''a message has arrived
32    ''**>''  if TRACE.L>2 SNAP "eota", 0,id(VALUE.E) always
33             if spb.ctr < outb  ''there is a free s - buffer
34                let.buf.no(VALUE.E) = 0
36                add 1 to spb.ctr  ''move the message to a 8-buffer
36                let.buffer.full(OUT.ARC(PORT.E)) = 0
37                if wait(IN.ARC(PORT.E)) = 1      ''the source has another
38                   SET.ALERT PORT.E+1 FOR.E buffer.avail   ''inter node e
39                always  ''of buffer.full
40             else    ""no e-buffer is avialble
41                let buf.no(VALUE.E) = PORT.E+1
42                let buf.vec(PORT.E+1) = VALUE.E
43             always  ''of sbp.ctr
44             let PORT.E = SEL.PORT(destination(VALUE.E))
46             file VALUE.E in SET(out.port(PORT.E))
48             if n.SET(out.port(PORT.E)) = 1 go to beg.trans  always
47             cycle
48
49          'buffer.avail' ''input buffer via output PORT.E is avail.
50
51             subtract 1 from PORT.E
62    ''**>''  if TRACE.L >2 SNAP "bufa", PORT.E,0    always
63             let wait(OUT.ARC(PORT.E)) = 0
64             go to beg.trans
55
66          SELECT.INT.EVENT
67
```

```
58            'end.message'
59
60  ''**>''      if TRACE.L > 2 SNAP 'entt', 0,id(VALUE.E) always
61  ''==>''      MEASURE 0 WITH.PROBE port.util(PORT.E)
62               remove the VALUE.E from SET(out.port(PORT.E))
63               SET.ALERT PORT.E FOR.E end.trans, VALUE.E
64               let available = 0
66               if buf.no(VALUE.E) = 0 ''it is located in a spare buffer
66                  subtract 1 from spb.ctr
67                  for i = 0 te in.degree(NODE.V)-2 by 2 with
88                    buf.vec(mod.f(rund.buff+i,in.degree(NODE.V))+2) ↓= 0
69                    find the first casa if found
70                      let rund.buff=mod.f(rund.buff+i,in.degree(NODE.V))+2
71                      add 1 to spb.ctr
72                      let buf.no(buf.vec(rund.buff)) = 0
73                      let buf.vec(rund.buff) = 0
74                      let buffer.full(OUT.ARC(rand.buff-1)) = 0
76                      let available = rund.buff
78                    always
77               else ''in the in.port buffers
78                  let buffer.full(OUT.ARC(buf.no(VALUE.E)-1))=0
79                  let buf.vec(buf.no(VALUE.E)) = 0
80                  let available = buf.no(VALUE.E)
81               always
82               if available ↓=0 and wait(IN.ARC(available-1)) = 1
83                   SET.ALERT available FOR.E  buffer.avail
84               always
86             , if SET(out.port(PORT.E)) is empty cycle  always
86
87            'beg.trans'
88               if buffer.full(IN.ARC(PORT.E))=0
89                  let act.message = f.SET(out.port(PORT.E))
90  ''**>''         if TRACE.L>2 SNAP "bgtr",0,id(act.message)  always
91  ''==>''         MEASURE 1 VITH.PROBE port.util(PORT.E)
92                  SET.TIMER length(act.message) FOR.E
93                     end.message, act.message, PORT.E
94                  if routing.matrix(NODE.V,destination(act.message))
95                           ↓= destination(act.message)
96                     SET.ALERT PORT.E FOR.E start.trans
97                  always
98               else
99                  let wait(OUT.ARC(PORT.E)) = ■
100              always
101              cycle
102
103           SELECT.CON.EVENT
104
105           'termination'
106              write NODE.V as i 3,s 11
107              for i = 1 to OUT.DEGREE(NODE.V) by 2
108                  write NUM.P(port.atil(i))/2,AVG.P(port.util(i)) as i 5.
109              write as /./
110              leave
```

```
111
112         loop
113         for i = 1 to OUT.DEGREE(NODE.V) by 2
114            for each task in SET(out.port(i)) , do
115               remove task from SET(out.port(i)) ''and''destroy the task loc
116         DISPOSE.NODE
117   end ''of process communication.processor
```

C.2.4 The Executive Manager

```
 1  process to EXEC.MANAGER
 2     define i , j , node  as integer variables
 3
 4
 6     IHIT.THE.NETWORK
 8
 7
 8     PRINT.NEIGHBOR.LIST 6 , 1
 9
10     read  n.hosts , SIM.TIME                start new card
11     INIT.TRACING
12     INIT.SEEDS 100 , 0
13
14     write as 'node parameters*,/        for i = 1 to 15 write as "=" write
15     write as  'node i.a.t l.x.t h.x.t 1.t.t h.t..tepr input output*,/.
16               * no.  sec:  sec:  sec:  sec:  sec: buf ports ports*,/
17     for i = 1 to 53 write as "=" write as /
18
19     INIT.THE.NODES
20
21     work SIM.TIME units
22
23     for node = 1 to N.NODE    SET.CONTROL node RR.E termination
24
26     write ae "results",/     or i = 1 to 6 write ae "=" write as /,/
26     write as  *node avg t    port no. 1  port no. 3  port no. 5",
27               "  port no. 7  port no. 9",/,
28               " no. exec tim.  msgs  util.  msgs  util.  msgs  util.",
29               " msgs  util.  msgs  util.",/
30     for i = 1 to 80 write as "="  write as /,/
31
32     TERMINATE.RUN
33
34  end     '' of process EXEC.MANAGER
```

C.2.5 Example of Output

```
point-to-point network lor example
                                   neighbor list (to (from , weight, count))
   1 (  9, 2,   1.0)
   2 (  9, 2,   1.0)
   3 ( 10, 2,   1.0)
   4 ( 10, 2,   1.0)
   6 ( 11, 2.   1.0)
   6 ( 11, 2,   1.0)
   7 ( 12, 2,   1.0)
   8 ( 12, 2,   1.0)
   9 (  1, 2,   1.0)  (  2, 2,   1.0)  ( 10, 2,   1.0)  ( 11, 2,   1.0)  ( 12, 2,   1.0)
  10 (  3, 2,   1.0)  (  4, 2   1.0)  (  9, 2   1.0)  ( 11, 2,   1.0)  ( 12, 2   1.0)
  11 (  6, 2,   1.0)  (  6, 2,   1.0)  (  9, 2   1.0)  ( 10, 2,   1.0)  ( 12, 2,   1.0)
  12 (  7, 2,   1.0)  (  8, 2,   1.0)  (  9, 2,   1.0)  ( 10, a   1.0)  ( 11, 2,   1.0)
nods parameters
===============
node i.a.t l.x.t h.x.t l.t.t h.t.t spr input output
 no.   sec.   sec.   sec.   sec.   sec. buf ports ports
=============================================================
  1  26.0     1.0   20.0     .1    2.0
  2  26.0     1.0   20.0     .1    2.0
  3  26.0     1.0   20.0     .1    2.0
  4  26.0     1.0   20.0     .1    2.0
  6  26.0     1.0   20.0     .1    2.0
  8  26.0     1.0   20.0     .1    2.0
  7  26.0     1.0   20.0     .1    2.0
  8  26.0     1.0   20.0     .1    2.0
  9                                    1
 10                                    1
 11                                    1
 12                                    1
results
=======
node  avg t     port no. 1  port no. 3  port no. 5  port no. 7  port no. 9
 no. exec tim. msgs  util. msgs  util. msgs  util. msgs  util. msgs  util.
=============================================================================
   1    18.519   37  .0678
   2    19.291   38  .0656
   3    18.248   30  .0595
   4    18.797   36  .0634
   5    19.849   34  .0464
   6    18.913   35  .0599
   7    18.818   41  .0712
   8    19.313   32  .0659
   9             37  .0678   37  .0655   19  .0361   21  .0291   28  .0559
  10             29  .0591   36  .0615   19  .0361   18  .0311   18  .0367
  11             34  .0464   35  .0595   21  .0291   18  .0311   20  .0311
  12             42  .0736   32  .0659   28  .0559   17  .0344   20  .0311
```

C.2.6 Example of Tracing Report

file- 1 run # 1 distributed system simulator 07/29/83 20:56:15

```
time        1        2        3        4        5        6        7        8        9       10       11       12
==========+========+========+========+========+========+========+========+========+========+========+========+
600.4048                                                                  tarr(  1, 800017)                sota(  3,      0)
600.6496          eote(  0, 800016)                                               sota(  3,      0)
500.8781                                                                                           entt(  0, 500013)
500.8781                                                                                    sota(  0, 500013)
600.8781                                                                                    bgtr(  0, 500013)
501.0846                                                                  eott(  1, 800017)                sota(  0, 800017)
601.0846                                                                               sota(  9,      0)       bgtr(  0, 800017)
501.2622                          bfav(  3, 300018)                                                 entt(  0, 400018)
501.2622                                                                                            sota(  3,      0)
501.4144                                                                                            entt(  0, 400019)
501.4144                                                                                  sota(  0, 400019)
501.4144                                                                                  bgtr(  0, 400019)
601.4769                                                                  eote(  0, 800016)
601.6106          eott(  8, 800015)                                               sota(  0, 800015)
601.6106                                                                          bgtr(  0, 800015)      sota(  5,      0)
501.5848                tarr(  6, 300019)                                                  sota(  1,      0)
601.7244                                                                  sota(  0, 800017)       entt(  0, 800017)
502.3815                                                                                    entt(  0, 500013)
602.4716                                                                  entt(  0, 800015)       sota(  0, 800015)
602.4716                                                                                            bgtr(  0, 800015)
602.8082                          eott(  3, 300018)                                         sota(  0, 300018)
602.8082                                                                                    bgtr(  0, 300018)
603.0218                                                                  entt(  0, 400019)
603.0218                                                                  bgtr(  0, 800017)
603.1229                tarr(  ti, 300019)                                                 sota(  0, 300019)
608.1229                                                                                    bgtr(  0, 300019)
603.1229                                                                                             sota(  7,      0)
603.4324                                                                                             entt(  0, 800015)
603.8817                                                                  entt(  0, 800017)
503.9641                                                                                    entt(  0, 300018)
504.4324                                                        eote(  0, 600011)                    sota(  1,      0)
504.8611                                                                                    entt(  0, 300019)
604.8811                                                                                             sota(  0, 300019)
604.8811                                                                                             bgtr(  0, 300019)
604.9890                                                          eote(  0, 100019)                  sota(  3,      0)
606.9181                                                        eott(  6, 600011)                    sota(  0, 600011)
606.9181                                                                                             bgtr(  0, 600011)
606.9181                                                                                    sota(  9,      0)
608.1992                                                                                    entt(  0, 300019)
608.2182           eote(  0. 200021)
608.4010 tarr(  2, 100022)                                                        sota(  1,      0)
508.8840                                                                  eott(  1, 100019)                sota(  0, 100019)
508.8840                                                                               sota(  9,      0)       bgtr(  0, 100019)
608.9870 eott(  2, 100022)                                                             sota(  0, 100022)
608.9870                                                                               bgtr(  0, 100022)
507.3998                                                                                                    entt(  0, 600011)
607.3998                                                                                             sota(  0, 600011)
507.3998                                                                                             bgtr(  0, 600011)
607.6330                                                                               entt(  0, 100022)
608.3789                                                                               sota(  0, 100019)       entt(  0, 100019)
608.3789                                                                               bgtr(  0, 100019)
608.7833 eote(  0, 400019)                                                             sota(  1,      0)
608.8136                                                                                            entt(  0, 600011)
610.0739                                                                               entt(  0, 100019)
610.3807 eott(  4, 400019)                                                             sota(  0, 400019)
610.3807                                                                               bgtr(  0, 400019)
610.3807                                                                                     sota(  5,      0)
610.7470                                                  eote(  0, 300019)                          sota(  3,      0)
```

bfav - buffer available bgtr - begin transfer bufa - buffer available enta - end of task arrival
eota - e.o.task.arrival aota - e.o.task execution eott - e.o.task transfer sota - s.o.task arrival
tarr - task arrival

Refferences

[Abra77] Abramson, N: *"The Throughput of Packet Broadcasting Channels,"* IEEE Trans. Comm., Vol. COM-25,pp. 117-128, January 1977.

[Bask75] Baskett F.,Chandy K. M., Muntz R. and Palacios F.: *"Open, Closed and Mixed Networks of Queues with Different Classes of Customers,"* Journal of the ACM, Vol. 22, No. 2, pp. 148-160, April 1975.

[Bran79] Bradwajn A. : *"An Iterative Solution of Two - Dimensional Birth and Death Processes,"* Operations Research, Vol. 27, No. 3, pp. 595-605, May 1979,

[Brya81] Bryant R. M. and Finkle R. A., "A *Stable* Distributed *Scheduling Algorithm,"* Second International Conference on Distributed Computing Systems, Paris, Prance, April 1981, pp. 314 - 323.

[CACI78] *"Continuous Simulation and Combined Simulation* in *SIMSCRIPT II.5.,"* C.A.C.I, 1978.

[Camp83] Campeanu D. : *"Performance Evaluation of a Parallel File* Sorting *Algorithm,"* Msc. Thesis, Weizmann Institute of Science, Rehovot, Isreal, November 1983.

[Chan77] Chandy K. M., Howard J. H. and Towsley D. F.: *"Product-form and Local Balance in Queueing Networks,"* Journal of the ACM, Vol. 24, No. 2 pp. 250-263, April 1977.

[Chan80] Chandy K. M. and Sauer C. H: *"Computational Algorithms for Product-form Queueing Networks,"* Communication of the ACM, Vol. 23, No. 10, pp. 573-583, October 1980.

[Chow77] Chow Y. C. and Kohler W. H. : *"Dynamic Load Balancing in Homogeneous Two-Processor Distributed Systems,"* International Symposium on Computer Performance, Modeling, Measurement and Evaluation, Yorktown Heights, New york, pp. 39-52, August 1977.

[Coff73] Coffman E. G. and Denning P. J.: *"Operating Systems Theory,'* Prentice-Hall 1973.

[Dahl66] Dahl O. J. and Nygraard K. : *"SIMULA - An ALGOL - Based Simulation Language,"* Communication of the ACM, Vol. 9, No. 9, pp. 671-678, September 1966.

[Digi80] Digital , Intel and Xerox: *"The ETHERNET,"* September 1980.

[Dims64] Dimsdale B. and Markowitz H. M.: *"A Description of the SIMSCRIPT Language,"* IBM Sys. J., Vol. 3, No. 1, pp. 57-67, January 1964.

[Echh78] Echhouse R. H. and Stankovice J. A. : *"Issues in Distributed Processing. - An Overview of Two Workshops,"* Computer, January 1978, pp. 22 - 26.

[Elli82] Elliot D., Kopec S.: *"One Chip Carries Out Ethernet protocol,"* Electronic Design, pp. 121-128, October 1982.

[Ensl78] Enslow P. H. : *"What is a Distributed Data Processing System?"* Computer, Vol. 11, No. 1, pp. 13-21, January 1978.

[Ensl81] Enslow P.H. and Saponas T.G.: *"Distributed and Decentralized Control in Fully Distributed Processing Systems,"* Final Technical Report, GIT-ICS-81/02, Georgia Institute of Technology, School of Information and Computer Science, Atlanta, Georgia 30332, 1981.

[Fayo80] Fayolle G., King P. J. B. and Mitrani I.: *"The Solution of Certain Two - Dimensional Markov Models,"* The 7-th IFIP International Symposium on Computer Performance, Modeling, Measurements and Evaluation, pp. 283-289, may 1980.

[Herz75] Herzog U., Woo 1. and Chandy M. : *"Solution of Queueing Problems by a Recursive Technique,"* IBM J. Res. Development, Vol. 19, No. 3, May 1975.

[Fran77] Franta W.: *"A Process View of Simulation,"* North-Holland, New York, 1977.

[Gall82] Baller B. I, :. *"Concurrency Control Performance Issues,"* Phd thesis, University of Toronto, September 1982.

[Gave76] Gaver D. P. and Humfeld G. : *"Multitype Multiprogramming Models,"* Acta Informatica 7, pp. 111-121, 1976.

[Gord78] Gordon G. : *"System Simulation,"* Prentice-Hall, Inc. Englewood cliffs, NJ., 1978.

[Gree72] Greenberg S. : *"GPSS Primer,"* John Wiley & Sons, Inc, New York, 1972.

[Hindi82] Hindin H.J.: *"Dual-Chip Sets Forge Vital Link for Ethernet Local-Network Scheme,"* Electronics, pp. 89-91, October 1982.

[Jens78] Jensen E. D. : *"The Honeywell Experimental Distributed Processor-An Overview,"* Computer, Vol 11, No. 1, pp. 28-38, January 1978.

[Kan83] Kantor M. : *"Simulation and Performance Evaluation of Basic 2PL Algorithm,"* Msc. Thesis, Weizmann Institute of Science, Rehovot, Isreal, September 1983.

[Kivi67] Kiviat P. J.: *"Development of Discrete Digital Simulation Languages,"* Simulation, Vol VIII, No. 2, February 1967.

[Klei75] Mleinrock L. : *"Queueing Systems Vol. 1 : Theory,"* Wiley, New York, 1975.

[Klei80] Mleinrock L. and Gerla M.: *"Flow Control: A Comparative Survey,"* IEEE Trans. Comm., Vol. COM-28, No. 4, pp. 553-574, April 1980.

[Krat80] Kartzer A. and Hammerstrom D. : *"A Study of Leveling,"* Proceedings of COMPCON80, pp. 647-654, September 1980.

[Law 79] Law A. M. and Carson, J. S. : "A *Sequential Procedure for Determining the Length of a Steady-State Sirnulation,"* Operations Research, Vol. 27, No, 2, pp. 1011-1025, Sept.-Oct. 1979.

[Levy82] Levy E. : *"The Simulation Model for a Concurrency Control Algorithm of a Distributed, Multiple Copy Database," Msc.* Thesis, Weizmann Institute of Science, Rehovot, Isreal, May 1982.

[Livn82] Livny M. and Melman M. : *"Laud Balancing in Homogeneous Broadcast Distributed Systems,"* in Proc. of Computer Network Performance Symposium, College Park, Maryland. April 1932, pp. 47-55.

[Metc75] Metcalfe R. M. and Boggos D. R. : *"Ethernet: Distributed Packet Switching for Local Computer Networks,"* Communication of the ACM, Vol. 19, No. 7, pp. 395-404, July 1976.

[MacD73] MacDougall M. H. and MacAlpine J. S. : *"Computer System Simulation with Aspol ,"* Proceedings Symposium on the Simulation of Computer Systems, pp. 92-103, June 1973.

[Mcqu80] McQuillan J. M., Richer I. and E. G. Rosen *"The* New *Routing Algorithm for the ARPANET,"* IEEE Trans. Comm., Vol. COM-28, No. 5,pp. 711-719, May 1980.

[Mel83a] Melman M. and Livny M. : *"DISS User Guide,"* Weizmann Institute of Science, Rehovot, Israel.(preliminary printing). June 1983.

[Mel83b] Melman M. and Livny M. : *"Simulation Models of the Ethernet Protocol," in* Proc. of Summer Computer Simulation Conference, Vancouver, B.C., July 1983

[Ni 81] Ni L. M. and Abni K. : *"Nonpreemptive Load Balancing in a Class of Local Networks,"* Proceedings of the 1931 Computer Networking Symposium, December 1981, pp. 113 - 118.

[Prit69] Pritsker A, A. B. and *Kiviat P.* : *"Simulation with GASP II,"* Prentice-Hall, Inc. Englewood Cliffs, NJ, 1969

[Puzi73] Puzin L.: *"Presentation and Major Design Aspects of the CYCLADES Computer Network,'* in *Proc.* 3rd ACM-IEEE Commun. Symp. Tampa. Fl, pp. 80-87, November 1973.

[Rals65] Ralston A. : *"A First Course in Numerical Analysis,"* McGraw-Hill, 1965.

[Robe70] Roberts L. G. and Wessler B. D.: *"Computer Network Development to Achieve*

Resource Sharing," in 1970 AFIPS Conf. Proc (SJCC), Vol. 36, pp. 543-549.

[Schw80] Schwartz M. and Stern T. E. : *"Routing Techniques Used in Computer Communication Networks,"* IEEE Trans, Comm., Vol. COM-28, No. 4, pp. 539-552, April 1980.

[Saue80] Sauer C. H. and Chandy K. M.: *"Approximate Solution of Queueing Models,"* Computer, Vol. 13, No. 4, pp. 25-32, April 1980.

[Scho78] Schoute F. C. and McQuillan J. M. : "A *Comparison of Information Policies for Minimum Delay Routing Algorithms,"* IEEE Transactions on Communications, Vol. Com-26, pp. 1266-1271, August 1978.

[Ston77] Stone H. S. : *"Multiprocesser Scheduling with the Aid of Network Flaw Algorithms,"* IEEE Trans. *of* Software Eng., Vol. SE-3, No. 1, January 1977, pp. 85-93.

[Stuc83] Stuck, B. W.: *"Calculating the Maximum Mean Data Rate in Local Area Networks,"* Computer, Vol.16, Number 5, pp. 72-76, May 1983.

[Russ83] Russell E. D., Editor : *"SIMSCRIPT II.5 Programming Language,"* C.A.C. Inc., Los Angeles, Ca, 1983.

[Tane81] Tanenbaum, A. S. : *"Computer Networks,"* Prentice-Hall, Englewood Cliffs, N.J. 1981.

[Teic66] Teichroew D. and Lubin J. F. : *"Computer Simulation - Discussion of the Technique and Comparison of Languages,"* Communication *of* the ACM, Vol. 9, No. 10, pp. 723-741, October 1966.

[Wong78] Wong J. W. : *"Queueing Network Modeling of Computer Communication Networks,"* ACM Computing Surveys, Vol. 10, No. 3, pp. 344-351, September 1978.

[Zeig76] Zeigler B. P.: *"Theory of Modelling and Simulation,"* John Wiley, New York, 1976.

א ב ס ט ר ק ט

רבוי המשאבים במערכות מבוזרות ואי תלותם של משאבים אלה זה בזה עושים את שיטת העברת המשימות לאמצעי מתאים להשגת שפור בזמן התגובה של המערכת. אולם עכובים בתקשורת הכרוכים בבצוע תהליך זה ותקורתו, מעמידים בספק את יעילותן של שיטות אזון העמסים בהעלאת רמת הבצועים של מערכות מסוג זה.

בתיזה זו נחקרת בעית אזון העמסים במערכות מבוזרות ומוצגת הערכה השוואתית של מספר אלגוריתמים בביצוע תהליך זה. השיטה שעליה מתבסס מחקר זה מושתתת על התפיסה של בעית אזון העמסים, בדומה לבעיות רבות אחרות הקשורות למערכות מבוזרות, כבעיה דו-מימדית. המימד הראשון כולל מדדים המגדירים את האלגוריתם עצמו ואילו המימד השני מייצג את מאפייני המערכת המבוזרת. הבנת התלות הקימת בין בצוע האלגוריתם לבין תכונות המערכת חיונית לבירור מהותו של תהליך אזון העמסים. בעבודה מוצגים ומנותחים מספר אלגוריתמים חדשים לאיזון עמסים במערכות מבוזרות בעלות משטרי שדור שונים. ניתנות הגדרות ומובאים פתרונות למודלים המציגים ביצועים של אלגוריתמים שונים ומערכות שונות. התוצאות המסוכמות במחקר זה מורות בבירור כי גם כאשר העכובים בתקשורת והתקורה של תהליך אזון העמסים אינם זניחים, יכול השמוש בשטה זו להביא לשפור נכר בזמן התגובה של המערכת.

קיימות גישות שונות להגדרה ולפתרון של מודלים המציגים בצועים של אלגוריתמים. הגישות הננקטות בעבודה זו הן אנליזה או סימולציה בהתאם למורכבות המודל ולדרגת הפירוט הנדרשת. מובאת שיטה לסימולציה של מערכות מבוזרות וזו משמשת לגיבוש מדדים ביצועיים. המודלים ותכנית הסימולציה שנבנו בשטה זו משקפים את אי התלות שבין האלמנטים במערכת ואת קשר "הצימוד הרופף" ביניהם. כפועל יוצא מכך משקפים המודלים הללו את תכונת המודולריות של המערכת המבוזרת.