

**Computer Science Department Technical Report
University of California
Los Angeles, CA 90024-1596**

**LOAD BALANCING ALGORITHMS IN A DISTRIBUTED
PROCESSING ENVIRONMENT**

Joseph Jacob Green

**October 1988
CSD-880087**



UNIVERSITY OF CALIFORNIA

Los Angeles

Load Balancing Algorithms in a Distributed Processing Environment

**A dissertation submitted in partial satisfaction of the
requirements for the degree Doctor of Philosophy
in Computer Science**


by

Joseph Jacob Green

1988



The dissertation of Joseph Jacob Green is approved.



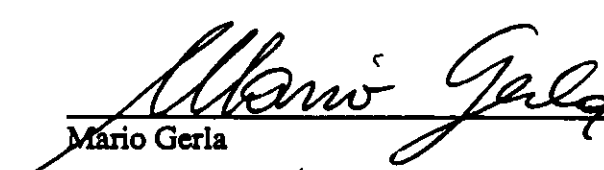
Glenn Graves



Bruce Rothschild



Eliezer Gafni



Mario Gerla



Leonard Kleinrock, Committee Chair

University of California, Los Angeles

1988

TABLE OF CONTENTS

	page
1 INTRODUCTION AND SURVEY	1
1.1) Introduction	1
1.2) Survey of the Literature	4
1.2.1) Static Load Balancing Algorithms	4
1.2.2) Quasi-Static Load Balancing Algorithms	6
1.2.3) Dynamic Load Balancing Algorithms	7
1.2.4) Routing with Dynamic Thresholds	9
1.2.5) Distributed Minimum Delay Routing in Computer Networks	11
1.3) Particular Papers of Interest	12
1.3.1) Dynamic Load Balancing in a Homogeneous Network	12
1.3.2) Two Centralized Load Balancing Algorithms	16
1.3.3) Two Distributed Load Balancing Algorithms	19
1.3.4) Queuing Systems and Boundary Value Problems	23
1.4) Motivation	27
1.5) Summary of Results	29
2 QUASI-STATIC NETWORKS	33
2.1) The Model	33
2.2) The Centralized Algorithm	36
2.3) The Distributed Algorithm	36
2.4) Summary	46
3 THRESHOLD NETWORKS	47
3.1) Model	48
3.2) Formulation of Problem	55
3.3) Distributed Algorithm	57
3.4) Simulation Results	58
3.5) Summary	61
4 LOAD SHARING IN BROADCAST NETWORKS	62
4.1) Introduction and Motivation	62
4.2) Preliminaries	63
4.2.1) Model	63
4.2.2) Performance Measures	64
4.2.3) Method for Load Sharing	66
4.2.4) Analysis of the Search Algorithm	69
4.3) Network Performance Without Load Sharing	71
4.3.1) Results for Uniform Distribution	72
4.3.2) Results for the Beta Distribution	76
4.3.3) Results for the General Distribution	78
4.4) Results for Broadcast Networks With Load Sharing	81
4.5) Communication Costs and Optimal Number of Pairs	93
4.6) Random Pairing	101
4.7) Conclusions	105
5 ANALYSIS OF TWO QUEUES WITH THRESHOLD	108
5.1) Model	108

5.2) Problem Formulation and Solution	111
5.2.1) System Equations	111
5.2.2) Performance Measures	117
5.2.3) Boundary Value Problem Formulation	124
5.2.4) Method of Solution	128
5.2.5) Proof of Uniqueness for $T=1$	137
5.2.6) The General Problem, T greater than 1	167
5.3) Discussion of Results	170
5.4) Conclusion	173
6 FUTURE RESEARCH	175
Appendix 1 DERIVATION OF M/G/1 DELAY WITH LOAD SHARING	177
Appendix 2 STUDY OF THE KERNEL EQUATION	180
References	183

LIST OF FIGURES

	page
(1.1): Throughput vs. Delay for Three Location Policies	15
(1.2): Probability of Idle Capacity with Work in the System	28
(2.1): Snapshot of a Load Balancing Network	38
(2.2): Possible Loop in Flow Tree	43
(3.1): Markov Chain for Node in Isolation	50
(3.2): Interarrival Process Simulation Results	59
(4.1): Model of a Broadcast Network	65
(4.2): Algorithm for Finding the Maximum Value	70
(4.3): Table of (No Load Sharing) Processing Delay and Simulation Results	75
(4.4): Beta Distribution Function	79
(4.5): Sample distribution After Pairing	87
(4.6): Table of Load Sharing Processing Delay and Simulation Results	91
(4.7): Approximation of Average Processing Time With Load Sharing	94
(4.8): Upper and Lower Bounds to $\nabla_K T, N = 500$	97
(4.9): Chernoff Bounds on $P[S_K/K \geq \gamma^*]$	105
(5.1): Two Node Threshold Queueing System	110
(5.2): State Diagram for the Threshold Queueing System	115
(5.3): Alternate Representation of the State Space	122
(5.4): Triangle Zone of States $0 \leq i + j < T - 1$	136
(5.5): Table of Winding Numbers for $B(z, w)$	140
(5.6): Image of $w(z)$ for $ z = 1$	151
(5.7): Zero of $B(z, w)$ for $\lambda_1 + \mu_1 < \alpha(1 - \mu_2 / (\lambda_2 + \alpha))$	158
(5.8): Example of Simply Connected Annulus	160
(5.9): Zero of $B(z, w)$ for $\lambda_1 + \mu_1 > \alpha(1 - \mu_2 / (\lambda_2 + \alpha))$	163

(5.10): Table of Values for $W_{ w =1}^{\#}(B(z,w))$ and $W_{ z =1}^{\#}(B^T(z,w))$	170
(5.11): Average Number in the system	172



ACKNOWLEDGEMENTS

I wish to express my gratitude to Leonard Kleinrock, the chairman of my dissertation committee, for providing enthusiasm, support, and advice during my graduate years. His guidance in the past few years was essential in helping me achieve my goals. I also wish to thank the other members of my committee, Eli Gafni, Mario Gerla, Glenn Graves, and Bruce Rothschild for their comments on earlier versions of my dissertation. In particular, I would like to thank Eli Gafni for offering keen insight to the problems I was working on.

My years at UCLA would not have been as fruitful, nor as enjoyable without the friendship and cooperation of my fellow graduate students. In particular, Yehuda Afek, Arthur Goldberg, and Chet Lanctot were especially close and I will value their friendship forever. I would like to thank the current PSL (Parallel Systems Lab) members for helping me, especially Lily Chien, and Willard Korfhage. At IBM Research, Richard Gail, Sid Hantler, and B. A. Taylor helped me unravel the complexities of complex analysis. They also provided a stimulating research environment during my stay at IBM. I would like to thank my parents, Morris and Charlotte Green, who provided me with support during my graduate years.

Finally, I dedicate this dissertation to my wife, Beth, and my son, Etan. During the darkest moments of PhD journey, they were a constant reminder of the joy of living.

VITA

March 16, 1959 born, New York City

1981 B.S.E.E., Cornell University, School of Engineering,
with distinction

1981 Electronics Engineer, Lear Siegler Astronics, Santa Monica

1981 UCLA School of Engineering Fellowship

1984 M.S., UCLA, SEAS/Computer Science

1982-1988 Post-Graduate Research Engineer
Advanced Teleprocessing Systems group
Leonard Kleinrock, Supervisor

PUBLICATIONS

"Analysis of a Time Stamp Queue", Master's Thesis, University of California,
Los Angeles, Computer Science department, 1984.

ABSTRACT OF THE DISSERTATION

Load Balancing Algorithms in a Distributed Processing Environment

by

Joseph Jacob Green

Doctor of Philosophy in Computer Science

University of California, Los Angeles, 1988

Professor Leonard Kleinrock, Chair

In this dissertation we study the issue of *Load Balancing* and *Load Sharing* in a *Distributed Processing Environment*. The models we work with are assumed to be static or quasi-static with respect to the arrival process of jobs. A reduction process is presented which transforms Load Balancing problems in such environments to data communications network routing problems. We provide efficient distributed algorithms (in terms of message complexity) for load balancing when the underlying communication network has a point to point architecture. In the case where the processing nodes communicate across a broadcast medium, we provide a simple yet efficient load sharing scheme that significantly reduces the average response time (delay) of a job in the network. In order to evaluate the average delay of a job in such an environment, we provide a new queueing model and analysis. The results of our work justify the notion that the major savings (in delay) are achieved with a small amount of effort (Load Sharing as opposed to Load Balancing). We also analyze a star topology where satellite nodes (workstations), connected to a central mainframe site, use a Threshold Load Balancing policy. The queueing problem (for this model) is reduced to a Boundary Value Problem and solved numerically for the case where the threshold is one.

We provide a proof (subject to a condition on the parameters) that the solution to our boundary value problem (and thus our queueing problem) is unique. The conditions for ergodicity of the queueing problem are derived in the proof of uniqueness of the solution. The reduction process (from a queueing problem to a Boundary Value problem), and the proof of uniqueness provide an interesting study of applying Boundary Value Analysis to Queueing problems.

CHAPTER 1

INTRODUCTION AND SURVEY

1.1) Introduction

In the last fifteen years there have been great advances in processor and communications hardware. Microprocessor and various support (memory management, communication handlers, etc.) chips have become faster, smaller, and much cheaper to produce. Similarly, data communications has also undergone a technological revolution. There now exists glass fibers pure enough to transmit Gigabits (10^9) of information per second, that can be manufactured at relatively low cost. Both of these technological revolutions have led researchers to create their own information revolution. The number of papers that propose new architectures and protocols for the large systems of the future has increased by leaps and bounds. This plethora of publications attempts to answer the question: Can we use current processor and communications technology to build super systems that are far more powerful than the large systems that we have today?

In this dissertation we focus on the issue of dividing up work among a number of communicating processors. The models we study all have individual job streams arriving to processors that are connected with a communication network. The idea is to distribute the work (jobs) in such a fashion as to reduce the average response time of a job.

In this chapter, we try to organize the relevant literature by defining some terms and presenting a survey. There are two terms in the literature that are used to describe different approaches. The terms *Load Balancing* and *Load Sharing*, are used by different authors to mean different things. In this work, we define *Load Balancing* to be a system that tries to minimize the response time (or time in system) for the average job (or customer). *Load Sharing* refers to systems that attempt to maximize the utilization (or minimize the idle time) of all the processors in the system. One simple case where there is a difference between the two approaches is when there are both very fast and very slow processors in the system. A *Load Sharing* algorithm would send some of the load to the slow processors thus decreasing their idle time. A *Load Balancing* algorithm might use the faster processors solely if the average response time would suffer by having jobs "stuck" in the slower processors. Almost all of the models dealing with *Load Sharing* algorithms assume a homogeneous network, that is, all of the processors run at the same speed.

Another important distinction is to classify approaches (or algorithms) by being *source initiated* or *server initiated*. Wang and Morris [Wang85] define *source initiated* algorithms to be those in which the source of a job decides, using some information at the source, where to send the job. *Server initiated* algorithms are those in which a server processor searches among the sources for the next job to process based on information that the server keeps.

The last definitions we now present explain *static*, *quasi-static*, and *dynamic* algorithms. In *static* algorithms, the different components of the system (e.g. servers, sources, communication nodes) base their decisions on pre-determined parameters (such as routing variables) that are not responsive to the current state of the network. *Quasi-static* algorithms also ignore the current state of the network but do adapt to

slowly changing characteristics (such as the arrival rates of jobs to the network) by tuning their decision variables. *Dynamic* algorithms attempt to adapt the network to quickly changing environments by keeping track of the current state of the system.

The reader might ask the question: Why bother with distributing the load among the processors? The answer has three parts a) efficiency b) availability and c) extensibility. The hope of exploring large multiprocessor systems is that there is a gain in performance for some such system at a lower cost than a super computer alternative. Assuming the latest machine (or theoretically possible machine) is able to compute at X *gigaflops* (10^9 floating point operations per second), the question is: Is it possible to design a system of N or more processors where the achievable processing speed of this system is at least X *gigaflops*? The answer is meaningful only if the total cost of N (or more) cheap processors plus communications is far cheaper than the one super machine alternative. Thus the multiprocessor approach is to find a more *efficient* solution (higher performance to cost ratio) than just buying the latest mainframe. By having a number of processors in the system, we can try to build in a degree of fault tolerance. That is, despite the eventual failure of one or more devices, the system will continue to function, albeit at a reduced level of performance. This increase in *availability* is crucial to many users of large systems. The success of the TANDEM Inc. line of systems is a direct result of the guaranteed availability that is built in. The last issue deals with the ability of the system to handle changes in the user requirements. After a system is in place the users might find more tasks to do on a computer along with larger amounts of input (e.g. larger data base files or finer meshes for finite point analysis). Mainframe manufacturers have been improving the performance of their machines and trying to maintain upward compatibility to satisfy their user population (at high cost). However, maintaining compatibility works against the increased performance in many cases. Also, the increase in performance may be too much or too little

for different users. A well designed multiprocessor system should offer *extensibility* (at low cost) by being able to add on processors as the users need them. Algorithms that distribute the load over a network should be insensitive (or lightly sensitive) to the number of processors in a network.

1.2) Survey of the Literature

In this section we present a survey of the load balancing and load sharing literature. The survey is intended to classify papers by the definitions that we presented earlier, and to summarize their main points and results. In some cases, we elucidate a particular model if it warrants special attention. The survey is organized on the basis of amount of information a node (or a component in the system) has when it makes decisions to send or search for new work. As a rule, an increase in the amount of information a node requires to make a decision requires an increase in the communication to obtain this information. There is also an issue that nodes might use outdated information to make decisions, thus degrading the performance of the system.

1.2.1) Static Load Balancing Algorithms

The simplest load balancing algorithms are source partition and server partition [Wang85]. In their source partition algorithm the sources are partitioned into groups, with each group being served by one processor. Assuming that the arrival process of each source is Poisson with intensity r_i , the group behaves as an *M/G/1* system. The jobs that arrive at the sources queue up at the server, so that the service policy is FCFS. Their server partition algorithms split the servers up into groups which are fed by one or more sources. If the arrival process is Poisson then the group behaves as an *M/G/K* system. The authors assumed, for simplicity, that the jobs have service times that are negatively exponentially distributed. The question remains how to parti-

tion the sources and the servers so that the minimum expected delay can be achieved. Assuming the workload is constant and known, two approaches have been used, namely, network flow theory and mathematical programming [Ston78, Chou82, Chu80], to optimize the system performance with respect to a particular measure. If the arrival rates vary then the partitions must be recast. Tantawi and Towsley [Tant85] derived a centralized algorithm to determine the source and server partition (we investigate this in greater depth later on). In [Ni81] an extra random parameter is used along with the partitions to distribute the jobs from sources to servers. Here as well, the probabilities must be chosen to optimize some measure (e.g. delay). Kleinrock [Klei84] studied the following static network design issue. Assume a central source of Poisson arrivals and a network of parallel chains of servers, with each chain having some number of servers (exponentially distributed service times). The total capacity of the servers in the network is C . Jobs entering the network choose one of the chains (concurrent processing of jobs) and then are processed in a pipeline manner in the chain (jobs are assumed to be composed of a number of tasks, each processed by a different server on the chain). Kleinrock found that the average delay in this system is greater than the average delay of a single machine with capacity C . This result, while at first is surprising, can be explained by looking at both systems at less than full load. A single machine uses all the capacity whenever there is work in the system. The distributed network may not be using all the capacity since a number of the chains might be idle when there are jobs being processed in other chains.

All the above algorithms utilize only known arrival rates and service time distributions. None of the servers or the sources know the state of the network. A disadvantage of this approach is that the network cannot respond to short term variations in the workload.

1.2.2) Quasi-Static Load Balancing Algorithms

In this class of algorithms, the network components track slowly changing parameters of the system and adapt the load balancing so that some measure is optimized. Although full network information (states of the system) is not communicated, some measurements are passed along. This extra information passing adds a load to the communication and processing burden already present. However, it is assumed that the information exchange is done infrequently and presents a negligible increase to the system load.

The major work in this class is that of Kurose and Singh [Kuro86]. This work is investigated in depth in section (1.3.3). Their algorithm is the result of adapting a model in mathematical economics to a model of distributed processing in computer networks. Consider the network to have *resources* and *agents* wishing to use the resources. The problem is to divide up the resources among the agents in an optimal (for some measure) manner. The solution to this problem involves calculating the marginal utility of each *agent*, and specifying how each *agent* reacts to his marginal utility and that of other *agents*. The problem with the analogy is that economics uses such models to show *how* utility is maximized in a real world (or close to a real world) setting. No consideration is given to achieving this optimization in an efficient manner. Adapting an algorithm from economics to distributed processing yields an unwieldy algorithm for computers. In particular, Kurose and Singh suggest that at each round of the algorithm each node (*agent*) sends every other node a distinct value. If the communication medium is a broadcast network then $O(n^2)$ messages are sent every round. In a point to point network the number of messages sent each round is $O(n^2)$ at best and $O(n^3)$ at worst (depending on the network topology). These communication costs are far from optimal and could overload certain links or nodes in the

system. Other algorithms that are used in packet routing networks are discussed in depth in section (1.2.5).

1.2.3) Dynamic Load Balancing Algorithms

These algorithms attempt to respond to short term variations in the workload. To do this, they require up to date information of the network (or part of it) to be maintained at each node. There has been no solid analysis done to show the effect of transferring large amounts of state information on network performance. However, some principles have been outlined by various authors.

A simple method which is called the state feedback method includes such algorithms as JSQ (Join the Shortest Queue) [Ni81, Chow79, Maju80]. Simple state information, such as queue length estimates, are broadcast periodically. Variants of this algorithm determine how far a node broadcasts its own information. There are two problems. The first is determining how often to update. If the update is done occasionally then the nodes are basing their decisions on possibly outdated information. On the other hand, frequent updating could congest the communications network. The second problem is determining who receives this information. In a large point to point network, broadcasting each node's state over the net could overload the communication links.

Chow and Kohler [Chow79] looked at a model of a central scheduler that sent arrivals to one of two queues. They used an analytic approach to compare three different dynamic policies. The first policy sends a new arrival to the shorter queue. The second policy attempts to minimize the delay of an arriving job. The third policy attempts to maximize the number of jobs processed in a period of time (e.g. between two arrival points). Using approximation techniques to solve a two dimensional Mar-

kov lattice, they found that for the model presented the third policy produced the best throughput versus delay curve (smallest delay for a given throughput). Ramakrishna [Rama83] also describes a system where a centralized switch uses the instantaneous queue lengths to determine where to send the next arriving job.

Eager et al [Eage84] describe a number of decentralized algorithms which use various amounts of information. The simplest model routes jobs to other nodes based on no information (static scheme). In the next algorithm, a node probes other nodes sequentially until a node is found with a queue length below a threshold T (a system parameter). The first such node found ends the probing and the job is sent to that node. If no such node is found after L probings the job remains with the current node. In the third algorithm, a node over the threshold will collect the queue lengths from L other nodes. The extra job(s) will be sent to the node with the shortest queue length only if it is below the threshold. Otherwise the job will remain at the current node. The result of the analysis is that the second policy yields a vast improvement over the first (in terms of average delay) even though little extra work is involved. The third policy provides very little improvement over the second even though it collects far more information. The authors conclude that a load balancing algorithm which collects little information is the best alternative in terms of the performance versus communication cost tradeoff. This paper is examined in greater detail in section (1.3.1).

Other systems use a form of bidding to distribute the load. Both DCS [Farb72] and CNET [Smit80] use versions of this algorithm. In *Normal* bidding a source which has work to send, broadcasts a request for bids to all servers (*source initiated* algorithm). Servers receiving the request may submit a bid based on their queue length, delay, etc. The requesting source then compares all the bids and selects the best (lowest) server to receive the work. In *Reversed* bidding an idle server will request

bids from sources (*server initiated* algorithm). The sources respond with bids that reflect their queue lengths or delay. The server may then select the job from the source yielding the best return (lower delay, etc.).

In some network models nodes may act as sources for jobs as well as "sinks". The ADAPT [Peeb80] system provides each node with a "dipstick" that has minimum and maximum notches. If the workload at the node is below the minimum notch, the node broadcasts to other nodes that it is willing to accept additional work. If the workload is above the maximum notch then it will attempt to send incoming work to other nodes. In a similar system, POGOS [Case81], the author reported difficulties in keeping the system stable. Dipstick systems also suffer from excess communication when the load is high.

The last paper mentioned is that of Bryant and Finkel [Brya81]. They describe a *diffusion* algorithm in the ROSCOE system in which only neighboring nodes communicate with each other. A node that is congested randomly selects a neighbor and inquires about cooperating. If the neighbor responds positively the two nodes will be paired for a duration of time, and will share their load. Afterwards, the pair will break up and other cooperation ventures will be sought with other nodes. The system attempts to adjust the load so that extra work migrates from a region that is overloaded to a region that has more idle capacity. The network is point to point and communication is relatively light for large systems. The problem is that the migration rate is slow and not very well understood.

1.2.4) Routing with Dynamic Thresholds

In recent years (since 1977) there has been much interest in the area of dynamic control of multiple queues. This interest originated in the control of large packet

switching networks at the nodal level. Although controlling a large network optimally with a decentralized dynamic policy is still an unsolved problem, much progress has been made in optimally controlling a local environment.

The authors in [Ephr80] examined two models of two servers (negatively exponentially distributed service times) controlled by a central switch. In the first model, the switch has instantaneous knowledge of the number in each queue. The authors were able to prove (using a dynamic programming technique) that the optimal policy is to send an arrival to the shorter of the two queues. In the second model the switch has no current knowledge of the queue lengths; it only knows that at some previous point there were x_1 and x_2 customers in queue one and queue two respectively. If $x_1 = x_2$ then the authors can show the optimal policy is to alternate arrivals among the queues (round robin). If, however, $x_1 \neq x_2$ then the authors can prove that sending to the expected shorter queue is not optimal.

Hajek [Haje84] generalized the two queue model to include centralized plus individual arrival streams, defection mechanisms from one queue to the other, and a central server as well as individual servers for each queue. The result of his work was to show that the long run (infinite horizon) cost is convex with respect to the set of control variables. Thus there exists a *threshold* (or *switch over*) policy that minimizes the long run cost.

Other researchers [Nels85] have shown optimal control policies for models with one queue and a finite number of servers. In this model the servers are ordered by decreasing processor speed. The decision is when to send a waiting customer to an idle processor. If the processor is very slow then the customer's service time may be large leading to a higher average delay. Nelson formulated a policy which attempts to maximize the number of customers processed in the time between two external events

(arrival to the system, earthquakes, etc.). Note the similarity of this policy to the third policy in [Chow79]. The authors claim that the average delay of their policy, comes very close to the minimum delay. Kleinrock [Klei64] developed a similar model, but used a greedy policy in which each job attempted to minimize its delay. The authors in [Agra84] worked on the same model but without arrivals. The problem was to find a policy when to send the next waiting customer to an idle processor. The authors presented a *threshold* policy (i.e. send to the i^{th} processor if there are t_i or more customers waiting in the queue) and proved it to be optimal.

The distinguishing feature of these models is that there is a central queue or (more importantly) a central controller.

1.2.5) Distributed Minimum Delay Routing in Computer Networks

We include this topic since load balancing and packet (or message) switching are similar problems. In Chapter 2 we transform a well used load balancing model into a packet routing problem and adapt a known algorithm [Sega79] to the load balancing problem that converges to the minimum delay.

The authors in [Cant74, Frat73] presented an efficient centralized algorithm to route flow from many sources to many destinations. In 1977 Gallager [Gall77] presented a distributed algorithm that settled on a set of flows which minimized (within a small radius) the delay in a message routing network. The key contribution of the paper was to point out that global optimization could be achieved by each node individually minimizing the marginal delay over its own outgoing links. The algorithm (indeed the solution) was possible since the link delay was assumed to be a function of the flow in the link only, convex with respect to the flow, and locally computable. Segall [Sega79] presented another version which made the algorithm easier

to implement and understand. Bertsekas [Bert78] described a whole class of distributed minimum delay algorithms which included Gallager's algorithm as a special case.

We have now finished an overview of some topics that are used later in Chapters 2 and 3. The next section is an in-depth investigation of some pertinent literature. Following that, we explain the motivation for our work based on the previous papers in the field and some of the background topics discussed in this chapter.

1.3) Particular Papers of Interest

In this section we investigate a number of papers more closely. These papers provide much of the motivation for the models and algorithms we develop in Chapters 2, 3 and 4.

1.3.1) Dynamic Load Balancing in a Homogeneous Network

We briefly described in section (1.2.1) three policies presented and analyzed in [Eage84]. We now present the model in greater detail. The network consists of N identical nodes connected by a broadcast bus. All nodes have one server with the same service rate. The exogenous arrival rate is the same for all nodes. The cost of transferring a job from one node to another is reflected as a processing cost, not as a communication cost. The reason is that Lazowska [Lazo84] et al. found that the processor delay of packaging and unpacking the data far outweighs the communication delay. The homogeneity assumption is used for simplicity and, the authors claim, does not affect the conclusions drawn.

The authors point out that there are two aspects to Load Balancing algorithms, the *transfer* policy and the *location* policy. The *transfer* policy is used to decide if and when to send a job elsewhere for processing. The *location* policy is used to determine

the destination of an outgoing job. The *transfer* policy used here is the simple *threshold* policy. If there are more than T jobs in a node the processor activates the *location* policy to transmit the extra jobs to remote processors. Note that a processor cannot process one job while transmitting another. There are three *location* policies used that use either no, little, or greater amounts of information.

Random - The destination is chosen at random for each outgoing job. There is no need for any state information from any other node. However, the job may arrive to a node that is above the threshold, thus requiring further transmission to yet another node. Thrashing jobs is a problem and the authors show that this policy is inherently unstable. No matter what the system load is, eventually a state will be reached where all the processors are busy transmitting jobs rather than processing them. This instability property is akin to the inherent instability of the ALOHA access scheme. An extra parameter L is used to control the system. After a job migrates through L nodes it must be processed at the $L+1^{\text{st}}$ node, no matter how backlogged that server might be.

Threshold - The transmitting node tries to locate a suitable destination node by probing other nodes one at a time. If a node is found with $T-1$ jobs or less (adding another job would not place it above the threshold) then probing stops and the job is sent to that node. Otherwise, if the transmitting node probes L nodes unsuccessfully, then the job is processed locally. The *Threshold* policy tries to limit the communication overhead by sequentially probing rather than broadcasting. The authors found that a probe limit, L , of three to five provided almost as good performance as a probe limit of twenty.

Shortest - The transmitting node probes L random nodes requesting their queue lengths. The job is transferred to the node with the shortest queue length only

if that number is below the threshold. If none of the polled sites are below threshold then the job is processed locally. This policy uses the most information of all three policies. However, for the same system load (arrival rates and service rates) the parameter L need not change as the network size increases.

The analytic model for all three *location* policies relies on the Markovian nature of the problem. All distributions are exponential, and a node's behavior is determined by the number in its queue. The only variables to keep track of are the queue lengths in all N queues. The authors reduce the problem from an N -dimensional lattice to a one dimensional chain by making an explicit independence assumption. The arrival rate of jobs to a "typical" node is modified to reflect the probing and transfer of jobs from other nodes. The main conclusion of the paper is to show that a little information goes a long way in improving the performance, while a lot of information does not help that much more. The *random* policy shows a 40% decrease in the response time as the load (ρ) increases to 80%. This is a surprising result considering the *random* policy uses no information at all. The *threshold* policy performs as well or a little better than the *random* policy for the lower end of the load range ($\rho=0$ to .57). At the higher loads ($\rho= .8$ to .95), the *threshold* policy performs much better than the *random* policy. This is not so surprising in that at high loads the *random* policy is sending jobs from one overloaded node to another. The *thrashing* phenomenon for each job continues until the limit of L hops is reached or a suitable node is chanced upon. The major result of the paper is reflected in the response time curve for the *shortest* policy. While this policy collects far more information (approximately fifteen to seventeen nodes more) than the *threshold* policy, the average delay is about the same or a little less.

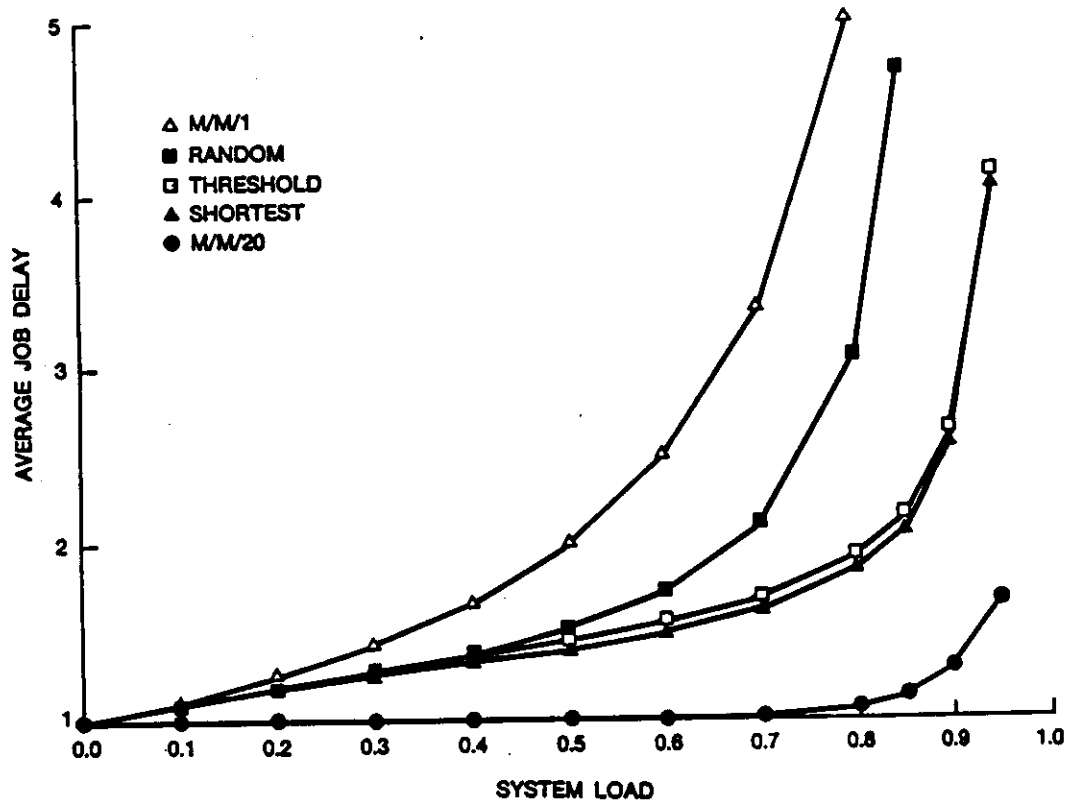


Figure 1.1
Throughput vs. Delay for Three Location Policies

The last issue discussed for now is the choice for the threshold T . In this model, all nodes have the same arrival and service processes. Thus the issue is to choose one threshold parameter for all nodes. Based on the analysis, the authors claim that a threshold of one, two, or three is the only decision for most values of the load ($\rho = 0$ to $.95$). Therefore, only a simple decision mechanism is needed for the nodes to choose the proper threshold. However, this conclusion can only be made in a homogeneous system. Also, the authors did not show the effect of the network size (N) or

the probe limit L on the threshold choice. We will discuss this issue in the Motivation section (1.4).

1.3.2) Two Centralized Load Balancing Algorithms

The first centralized algorithm we discuss is described in [Tant85]. The authors provide a model for a load balancing network where the communication as well as the processing costs are accounted for. The jobs are considered as flows in the network. Processing and transmission delays for the average job are functions of the flows through links and processors. The delay functions are assumed to be convex with respect to the flows which allows the authors to formulate a mathematical optimization problem that can be solved by finding a set of flows that satisfy the Kuhn Tucker conditions (derived from the constraints). A triangle inequality for the transmission delay function assumes that the communication delay from any node i to any other node j is smaller than the delay in going from i to j through any intermediate node k ($k \neq i, j$). The inequality forces each node to be in one of three sets as a solution to the optimization problem. The first set, *sources*, are those nodes that send out jobs to be processed remotely (as well as processing locally), but accept no jobs from any other nodes. The second set, *neutrals*, do not send out or accept any preempted jobs. *Neutrals* process their own internal jobs only. The last set, *sinks*, are those nodes which accept other jobs for processing (in addition to their own), but do not send out any jobs. A node may either be a *source*, a *neutral*, or a *sink* node. There are no nodes which accept and send out jobs, since any such node could be bypassed, thereby decreasing the overall delay by the triangle inequality assumption. The authors present a centralized algorithm which calculates the three sets given the parameters for each node. A sorting algorithm is also presented, which takes the current three nodal sets and varies their elements as a particular nodal or system parameter is varied. This

second algorithm allows network designers to watch what happens to the network as parameters change. The algorithms are efficient but centralized.

The second centralized algorithm includes a more general model than the one in [Tant85]. We briefly describe this model from the recent papers [Silv87, Silv84] and describe the solution method used. The network is modeled as a number of satellite sites [Gold83] connected together with a "communications network" (at the hub). Each site has a number of resources (CPU's, disks, programs, files, etc.) that are to be used by the incoming traffic. There are two types of traffic in each site. The *local* traffic is composed of "background" jobs that may run only at the site of their generation (i.e. they cannot be transmitted to another site). The other kind of traffic is *interactive* generated from the terminals connected to a site. This traffic may be routed to other sites to help relieve congestion. The *local* traffic is modeled as a closed chain, that is the number of *local* jobs at a site remains constant. In addition, the routing of *local* jobs within a site is fixed. The *interactive* traffic is modeled as an open chain (no restriction on the number of jobs in the system), and arrives at the sites according to a Poisson distribution. Although the *interactive* traffic may be moved from one site to another, the number of moves (or transmissions) is limited to one at most. In addition, some open chains of traffic can be constrained as to which sites they may be processed at.

The resources at a site are modeled as single server fixed rate, or infinite server service centers. The queue disciplines and service demands must satisfy the requirements for a product form network. The communications network is modeled as an infinite server service center.

The problem is to assign the flow of the open chains through the different sites such that the overall job delay is minimized. They construct a delay function that is convex in the set of open chain flows. The authors provide an algorithm, the Flow Deviation Load Balancing algorithm (FDLB), which finds the optimal assignment such that this delay is minimized. The solution method consists of the centralized version of the Flow Deviation Algorithm (FDA) [Cant74] together with Mean Value Analysis (MVA) [Lave83]. Given a starting (feasible) flow assignment, MVA is used to calculate the first two moments of the job delay. Then FDA is used to reappportion the flow assignment to reduce the delay. The method iterates until the average job delay converges.

The load balancing algorithm (FDLB) was applied to different settings. The results indicate a tremendous decrease in the average job delay when load balancing is used. The improvement (with load balancing) depends on the number of jobs in the closed (and fixed) chains, N_c . When N_c is small then FDA can deviate enough flow from the congested sites so as to reduce the average job delay significantly. As N_c gets large, the improvement due to the balancing is reduced since FDA is not allowed to reroute the *local* traffic.

Although the authors claim that the load balancing problem is more complicated than the routing problem, we show this not to be the case in Chapter 2. In fact, using the methodology in that chapter, we can reduce the problem presented in [Silv84] to a packet routing problem. In addition we can provide a distributed version of the FDLB algorithm if the conditions (on the delay function, listed in Chapter 2) are satisfied.

One important aspect of the FDLB algorithm, the computational complexity, is not discussed. For each iteration of the FDA, the subroutine MVA must be called to calculate the delay moments. When the number of chains (open and closed), *local* customers, and sites is considerable then the time for MVA computation is quite high. Multiplying the MVA complexity by the number of iterations of the FDA could result in a very long computation time for FDLB.

1.3.3) Two Distributed Load Balancing Algorithms

Kurose and Singh [Kuro86] worked on a load balancing network model that is similar to the one described in [Tant85], however, they [Kuro86] proceed to present a quasi-static *distributed* algorithm that minimizes the overall delay in the net. The distributed algorithm has its roots in the field of Mathematical Economics. The idea of the algorithm is to decrease the overall delay (towards the minimum) by having each node (in every iteration) adjust the flows to all other nodes based on the recipient's marginal utility. The flow adjustment phase is preceded by a round of information passing. We present the algorithm and then discuss its implementation.

Let f_{ij} be the amount of flow node i sends to node j . U_i is the utility (inverse delay) at node i , and

$\frac{\partial U_i}{\partial f_{ji}}$ is the marginal utility at node i due to the flow from node j . The marginal utility is the increase in the utility due to a small increment in the flow.

Repeat Forever:

- a. Each node i computes the marginal utility $\frac{\partial U_i}{\partial f_{ji}}$ (using the

current flow f_{ji}) and broadcasts this value to node j , $\forall j$.

- b. Each node i waits until it has received $\frac{\partial U_k}{\partial f_{ik}}$ from all k and then computes

$$\Delta f_{ik} = \begin{cases} \alpha_i \left[\frac{\partial U_k}{\partial f_{ik}} - \frac{1}{|K_i|} \sum_{k \in K_i} \frac{\partial U_k}{\partial f_{ik}} \right] & \forall k \in K_i \\ 0 & \forall k \notin K_i \end{cases}$$

where α_i is a step size (fixed) and K_i is the set of nodes to which i sends flow or to whom i will send flow since their marginal utility of receiving flow is greater than the average

$$\sum_{k \in K_i} \frac{\partial U_k}{\partial f_{ik}} / |K_i| \text{ (computed recursively).}$$

- c. each node i sends node k the new flow

$$f'_{ik} = f_{ik} + \Delta f_{ik}$$

Once each node i finds out the value of increasing or decreasing the flow to every other node k (start of step (b)) then node i computes how much to increase or decrease each flow by calculating the average marginal utility of sending flow from

node i , $\left[\sum_{k \in K_i} \frac{\partial U_k}{\partial f_{ik}} / |K_i| \right]$. Nodes that have a marginal utility greater than the average would receive an increase while those with a marginal utility below the average would receive a decrease. Those nodes not receiving any flow from i would receive

Δf_{ik} if their marginal utility is greater than the average, otherwise they would continue to receive no flow. The algorithm is efficient in that a node i may increase the flows to

more than one node in an iteration, speeding up the time for the algorithm to converge to the maximum utility. This aspect of the implementation is analogous to Bertsekas [Bert78] class of distributed algorithms. However, this algorithm is inefficient in the message complexity of the information collection phase. In a LAN the algorithm sends $O(n^2)$ messages in the information round (step (a)). In a point to point network the complexity varies with the diameter of the network. The number of messages transmitted can be as low as $O(n^2)$ or as high as $O(n^3)$.

Recently [Lee87] reported some results which are closely related to our algorithm presented in Chapter 3 but which were arrived at independently. We include a survey of this work and compare it with our own.

The load balancing environment is a star configured system with N satellite sites (workstations) and a central hub (mainframe). Each node i (satellite or the central server) has a Poisson arrival stream of jobs with rate λ_i . Jobs processed at node i have a service time that is exponentially distributed with service rate μ_i . The nodes have a communications server that is assumed to have a convex delay function in the flow that is transmitted through it.

There are three load balancing policies presented. Each of the policies uses a threshold in order to decide which of the incoming jobs (from the outside) should be transferred. Once a decision is made about a job (whether it will be processed locally or remotely), that decision cannot be changed. Jobs may be transmitted at most one time. A job that is transmitted, called a *remote* job joins the processor queue in the destination site. Jobs that are not transmitted, but remain at their originating site are called *local* jobs. For simplicity we describe the case where the central server only receives jobs and does not forward any to a satellite site.

The first policy has a priority discipline that favors *local* jobs over *remote* jobs. The second policy has the reverse priority discipline, while the third policy has no priority structure at all, i.e. all jobs queue up for the processor in a first come first serve manner.

Assuming the delay function at each node and in each communication link is convex, a non-linear optimization problem can be formulated, similar to the one presented in Chapter 3. Briefly, the problem is to minimize the average job delay (processing plus transmission time) by either working with the thresholds directly (an integer programming problem) or by relaxing the integer constraint on the threshold and working with the resulting flows. The threshold is set after determining the flow of jobs to be processed locally and remotely. The algorithms presented in [Lee87] are distributed and based on the work in [Gall77].

The authors also provide a heuristic algorithm for minimizing the average job delay by adjusting the thresholds up or down by one. In each iteration of the heuristic algorithm, a site compares its marginal delay with that of the central server. If the site's marginal delay is greater than the marginal delay at the central server then the threshold at that site is decreased by one, otherwise the threshold is increased by one. An extra slack variable can be used to prevent hysteresis. The idea is to equalize the marginal delays (thus minimizing the average job delay). The authors provide an example where the algorithm brings the average job delay close to the optimum, yet the thresholds keep on oscillating. There is no proof presented in this paper to show that the heuristic algorithm converges to the right value, or at all.

The development of the optimization problem and the distributed algorithm in [Lee87] is nearly identical to our own work in Chapter 3. However, the network configuration selected by Lee and Towsley is puzzling. If the network is configured as

a star then why use an iterative approach like the one presented in their paper? It is faster and far more efficient for all the satellites to send their actual delays to the central site where a centralized algorithm (such as FDA [Cant74]) can be run. The central server can easily determine the thresholds for every satellite node in the net and broadcast those values to each node. We develop the problem in Chapter 3 using a general point to point network. In our case, it is better to use a distributed algorithm since a centralized algorithm would require too much communication bandwidth to collect all the information at one site. We present a more detailed analysis in Chapter 3 (and for the broadcast case discussed in Chapter 4). We return to these papers in the motivation section (1.4).

1.3.4) Queueing Systems and Boundary Value Problems

In this section we focus on one paper [Fayo79] in which there is an elegant reduction of a queueing problem to a Riemann-Hilbert problem. After discussing the reduction technique used in this paper, we mention another technique developed by Cohen and Boxma [Cohe83].

A Riemann-Hilbert problem [Gakh66] can be formulated as follows: Assume there is a closed curve L in the complex plane t , and three real valued functions $a(t)$, $b(t)$, and $c(t)$ on L . Find a function $\Phi(t) = u(t) + i v(t)$ analytic in the interior of L , and continuous up to L , such that:

$$a(t)u(t) + b(t)v(t) = c(t) \quad t \in L$$

Let $G(t) = a(t) + i b(t)$ then the Riemann-Hilbert problem may be restated as follows: Find an analytic function $\Phi(t)$ (as before) such that:

$$\operatorname{Re}[\overline{G(t)}\Phi(t)] = c(t) \quad t \in L$$

where $\overline{G(t)}$ is the complex conjugate of $G(t)$.

The solution techniques for such problems are beyond the scope of this work. We refer the interested reader to [Gakh66] for a comprehensive treatment of the Riemann-Hilbert problem. However, we briefly mention some aspects of the solution that play a role when applying the Riemann-Hilbert problem to queueing problems.

The dimension of the solution space to a Riemann-Hilbert problem depends on the winding number or index, χ , of the function $G(t)$. We provide a precise definition of the winding number in Chapter 5. At this point we use an intuitive description. The winding number, χ , of $G(t)$ (on boundary L) is the number of times $G(t)$ wraps around the origin as t follows the contour L . If $\chi \geq 0$ then there exists a solution to the Riemann-Hilbert problem. In fact, the homogeneous problem has a $(\chi+1)$ dimensional solution space. On the other hand, when $\chi < 0$ then a solution need not exist; In this case, the problem may have a solution if $c(t)$ satisfies a certain set of $-\chi$ linear equations.

We now describe the queueing problem in [Fayo79] along with the reduction to a Riemann-Hilbert problem. Let there be two nodes (1 & 2) each having a server and a queue. Customers arrive to the nodes according to a Poisson distribution with rates λ_1 , and λ_2 (going to nodes 1 and 2, respectively). Both servers have negative exponential service rates. Server 1 (in node 1) serves customers with rate μ_1 when server 2 is busy, and with rate μ_1^* when server 2 is idle. Similarly, server 2 serves customers with rate μ_2 when server 1 is busy, and with rate μ_2^* when server 1 is idle. The difficulty in solving this problem is due to the coupling of one node's service rate to the state of the other node.

Assume that the system reaches equilibrium then the state of the two node system can be completely characterized by (n_1, n_2) where n_1 is the number of customers in node 1, and n_2 is the number in node 2. Let $P_{i,j}$ be the steady state probability that

the system is in state (i, j) . and define $P(z, w)$ to be the two dimensional generating function of the steady state probability distribution. Then the form of the equation obtained from transforming the steady state rate equations is (call it the system equation):

$$K(z, w)P(z, w) = a(z, w)P(0, w) + b(z, w)P(z, 0) + c(z, w)P(0, 0)$$

where $a(z, w)$, $b(z, w)$, and $c(z, w)$ are known functions, and $K(z, w)$ is given by:

$$K(z, w) = \lambda_1(1-z) + \mu_1(1-1/z) + \lambda_2(1-w) + \mu_2(1-1/w)$$

The functions $P(z, w)$, $P(z, 0)$, and $P(0, w)$ are unknown, and $P(0, 0)$ is the probability of an idle system.

When $K(z, w) = 0$, the right hand side of the system equation must vanish, otherwise $P(z, w)$ will not be analytic. Fayolle and Iasnogordski were able to derive another two equations from the system equation (when $K(z, w) = 0$) by extending the region of analyticity from $|z| \leq 1, |w| \leq 1$ to $|z| \leq \sqrt{\mu_1/\lambda_1}, |w| \leq \sqrt{\mu_2/\lambda_2}$. In particular, when $K(z, w) = 0$ and $|z| = \sqrt{\mu_1/\lambda_1}$ the authors showed that the imaginary part of w is zero (i.e. $\text{Im}(w) = 0$); Returning to the system equation we see that when $K(z, w) = 0$ we have

$$a(z, w)P(0, w) + b(z, w)P(z, 0) + c(z, w)P(0, 0) = 0$$

or

$$P(0, w) + \frac{b(z, w)}{a(z, w)}P(z, 0) = -\frac{c(z, w)}{a(z, w)}P(0, 0)$$

That is, when $|z| = \sqrt{\mu_1/\lambda_1}$, w is real and therefore so is $P(0, w)$. Taking the imaginary part of the above equation then yields:

$$\text{Im} \left[\frac{b(z, w)}{a(z, w)}P(z, 0) \right] = -\text{Im} \left[\frac{c(z, w)}{a(z, w)}P(0, 0) \right]$$

The above equation is in the same form as the Riemann-Hilbert problem presented at the beginning of this section. An equivalent equation for $P(0, w)$ can be derived by considering $|w| = \sqrt{\mu_2/\lambda_2}$. Solving both Riemann-Hilbert problems yields the functions $P(z, 0)$, and $P(0, w)$ subject to the normalization constant $P(0, 0)$ which is determined by evaluating $P(z, w) \Big|_{w, z=1} = 1$.

In order to carry out the reduction outlined above there are a few details left to check. First, we must guarantee that $P(z, 0)$ has no poles (i.e. is analytic) in the extended region of analyticity ($|z| \leq \sqrt{\mu_1/\lambda_1}$). Similarly, $P(0, w)$ must be shown to be analytic over $|w| \leq \sqrt{\mu_2/\lambda_2}$. In order for the system to be ergodic, the solution space to the Riemann-Hilbert problem must have a single dimension. Therefore, the winding number, χ , of $\overline{G(z)} = \frac{\overline{b(z, w)}}{a(z, w)}$ on the contour $|z| \leq \sqrt{\mu_1/\lambda_1}$ must be zero. The stability conditions of the system were derived as a result of proving that $\chi=0$.

Although the reduction technique provided by [Fayo79] is elegant, it is also limited. Specifically, the technique relies on the kernel function $K(z, w)=0$ to provide an analytic continuation of the system equation. However, this property is limited to simple state spaces and, hence, simple kernel functions. The problem we present in Chapter 5 has a kernel function that is much more difficult than the one analyzed in [Fayo79], and we cannot extend the region of analyticity for the unknown functions beyond the unit polydisk ($|z| \leq 1, |w| \leq 1$). Cohen and Boxma [Cohe83] developed a parameterization technique that can handle more sophisticated kernel functions than the one in [Fayo79], however, it cannot handle kernel functions where the mapping functions of the variables (z, w) are not simple curves. In Chapter 5 we develop a problem where neither of the two mappings, $z(|w|=1)$ and $w(|z|=1)$, are simple curves. To solve our boundary value problem we use the Splitting Technique developed by [Tay188]. This method is the most general solution method for these boundary value

problems to our knowledge. The Splitting Technique handles all kernel functions that have continuous mappings, $z(|w|=1)$ and $w(|z|=1)$ in the unit disk.

1.4) Motivation

What is so bad about running N machines independently of each other? Assume that no machine is overloaded and each is modeled as an $M/M/1$ queueing system. It is simple [Livn82] to derive the probability that there is at least one idle processor and at least one waiting job. Since each processor is independent and identical to all others (assuming homogeneity) with $Pr[\text{processor idle}] = P_0 = (1-\rho)$ we have $Pr[\geq 1 \text{ processor idle} ; \geq 1 \text{ job waiting}] =$

$$\sum_{k=1}^{N-1} \binom{N}{k} P_0^k (1-P_0)^{N-k} H^{N-k} = 1 - \rho^N + (1-\rho)^N [\rho^N - (1+\rho)^N]$$

where H^i is the probability that at least one job is waiting given that i processors are busy, is given by.

$$H^i = \frac{\sum_{j=1}^i \binom{i}{j} (1-P_0-P_1)^j (P_0(1-P_0))^{i-j}}{(1-P_0)^i} = \frac{(1-P_0)^i - P_1^i}{(1-P_0)^i}$$

$P_1 = P_0(1-P_0)$ is the probability of one in a system (job in the server only). The probability (of at least one system idle while a job is waiting in some other queue) as a function of ρ is shown in Figure (1.2) which was originally produced in [Livn82].

There are two aspects of the graph to note. The first is that with relatively few systems ($N=5$) there is already a high probability of wasted capacity over the expected range of ρ . The latter note is that as N gets larger the probability of wasted capacity goes to one for all loads (i.e. $Pr[\geq 1 \text{ processor idle} ; \geq 1 \text{ job waiting}] \approx 1$ for all ρ in the range $0 < \rho < 1$). The wasted capacity is the main motivation for load balancing al-

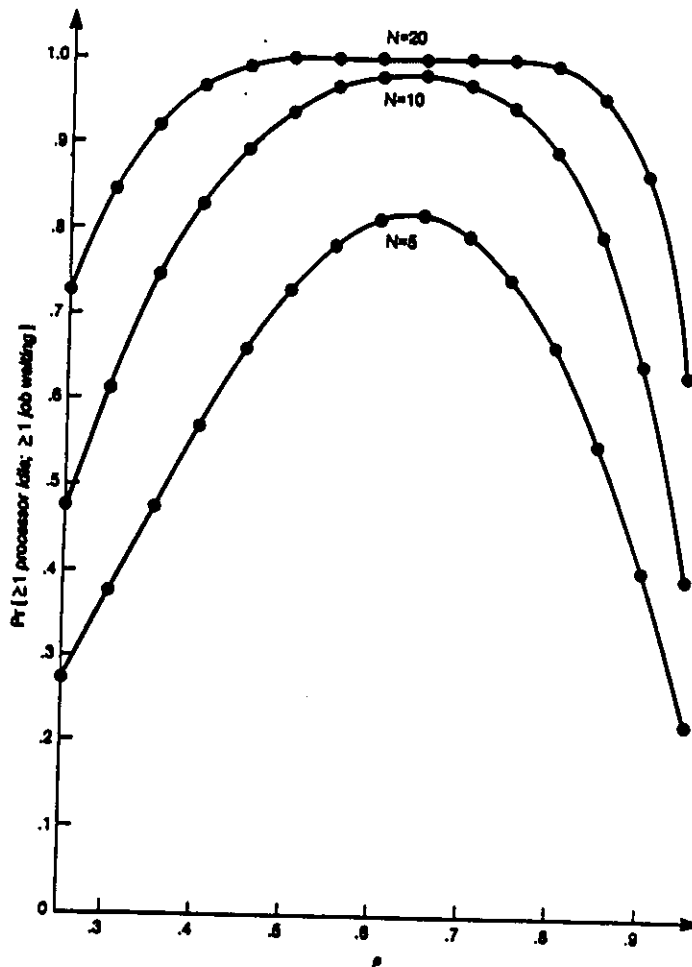


Figure 1.2
Probability of Idle Capacity with Work in the System

gorithms. The best situation we could hope for (in the homogeneous case) is an $M/M/N$ queueing system, while the worst (if we are not too stupid) is $M/M/1$. Since Load Balancing algorithms require some communication, the performance will fall somewhere in between $M/M/1$ and $M/M/N$. Eager et al [Eage84]. was able to show that simple schemes can improve the performance dramatically; however, the work was limited to showing some principles and did not derive algorithms for optimum performance in a more realistic (heterogeneous) setting. Tantawi et al [Tant85]. used a more realistic model but provided only a centralized algorithm. In addition, the au-

thors assumed a triangle inequality which might hold for broadcast nets but not for general networks. Thus in a real setting there will be intermediate types of nodes which both accept and send out remote jobs. Kurose and Singh [Kuro86] presented a distributed algorithm that is claimed to converge quickly; however it is extremely inefficient in the communication complexity. Furthermore, the nodes do not use any form of threshold which has been used in other models as a control variable.

1.5) Summary of Results

In this dissertation we study the design and analysis of load balancing (and sharing) algorithms in a distributed processing environment. The underlying communication networks connecting the processors are either general point to point or broadcast networks. We assume a static or quasi-static environment. That is, the exogenous arrival rates vary slowly and the load balancing (sharing) algorithms are able to *track* the input rates and adapt accordingly. Each node in the network has one or more processors and one or more communication links to its neighbors. Jobs that enter the network at node i can be processed at a processor in node i or at any other processor reachable from i . We assume that any files that are needed to process a job are replicated throughout the network, or that the files are short enough to be transmitted along with the job to another site. We assume further that the results of any job are short in length and that the time of transmission is negligible compared to processing and transmitting the job. Thus, we do not include the return time (sending the results back to the originating site) as a cost in any of our models.

In Chapter 2 we model the underlying network as a general point to point network. The strategy used is to minimize the average response time of a job in the network by transferring jobs from one processor to another by using routing variables. We detail a *distributed* load balancing algorithm based on Gallager's Minimum Delay

Distributed Routing Algorithm [Gall77] in which each node computes its own routing table based on the marginal delay at that node and at its neighbors. Our algorithm has a number of improvements over the one presented in [Kuro86]. We do not require an artificial approximation for the average delay at high utilization; Our algorithm in Chapter 2 incorporates the loop free property of Gallager's algorithm. (This is a critical feature for a load balancing application.)

The main result of Chapter 2 is that the processor load balancing problem in a quasi-static network can be reduced to a packet routing problem. Once the transformation (or reduction) is accomplished then all the results that were developed for packet routing (flow control, routing, etc.) can be applied to processor load balancing. Using our reduction method, a number of previous load balancing algorithms [Kuro86, Silv87, Silv84] are easily derived from the packet routing equivalent.

In Chapter 3 we modify the supporting network presented in Chapter 2. Here each node uses a threshold policy to control the flow of jobs to its own processors or to its neighbors. The contribution of this chapter is an optimal (in terms of the average response time of a job) distributed algorithm that lets each node determine the value of its threshold based on the marginal delay at that node and at its neighbors.

The model in Chapter 4 is a *broadcast* network of N homogeneous nodes. We present a new model in which the arrival rate of jobs at each node is drawn from a uniform distribution independently of the other nodes. We computed a tight approximation, \hat{T}_N to the average delay in such a network:

$$\hat{T}_N = \frac{2}{(1-\epsilon)^2} \left[\ln\left(\frac{1}{\epsilon}\right) - (1-\epsilon) \right]$$

where $[0, 1-\epsilon]$ is the range of the arrival rate to any node. We provide a simple, yet

efficient, algorithm to match up the K most heavily loaded users to the K most lightly loaded users in $O(K \log(L_{\max}))$ bits¹. Nodes that are paired share their pooled input rates equally. The effect of the matchings is to reduce the average network delay drastically by involving relatively few nodes. We compute a tight approximation for the average delay with load sharing:

$$\hat{T}_{N-2K} = 2 \left[\frac{(N+1)\ln(N/K) - (N-K+1)}{N-K+1} \right]$$

We then extend the model by including a communication cost for the load sharing traffic. By combining the average processing delay with the average communication delay we derive the optimal number of pairs in a network of N nodes to be $K^* = 2C/\mu$, where C is the capacity of the communications channel and $1/\mu$ is the average processing time of a job. We then compare the pairing scheme to a clustering policy where the nodes randomly select "buddies" to form clusters of size 2,3,..., K . We derive a bound on the tail of the distribution for the delay in a cluster with K nodes. This bound allows us to determine the probability of exceeding a particular delay value given K nodes in a cluster.

In Chapter 5 we return to the threshold policy used in chapter 3 and analyze a particular system. our intention is to model a central server system in which satellite workstations are connected to a central mainframe server. Our system consists of two nodes in tandem, in which each node has one processor. The first node uses a threshold, T , to determine when to transfer jobs to the second node, while the second node processes all incoming jobs. Even this simple queueing model is intractable using standard queueing theory techniques. We approach this problem numerically, by first transforming the queueing problem into a Boundary Value problem. Then we numeri-

¹ L_{\max} is the maximum average load any node in the network can experience and is equal to $1-\epsilon$.

cally construct two analytic functions $\Phi(z)$ and $\Psi(w)$ that are the generating transforms for the boundary states in the queueing problem. We prove that (under certain stability conditions) $\Phi(z)$ and $\Psi(w)$ are the unique solutions to the boundary value problem and hence, the probabilities obtained from these transforms are the unique solution to the queueing problem. For $T=1$, our proof is complete except for a restricted set of the parameter space $(\lambda_1, \mu_1, \lambda_2, \mu_2, \alpha)$. For the cases not covered, we conjecture that the solution is unique based on extensive numerical evidence. In our proof of uniqueness we derive two stability conditions (on the parameters) that makes the system of steady state equations ergodic.

CHAPTER 2

QUASI-STATIC NETWORKS

In this chapter we develop a simple static load balancing network model. Essentially, the load balancing problem is reduced to a message routing problem which has been solved already. The load balancing algorithm may either be centralized or distributed. The centralized version closely follows the Flow Deviation [Frat73] algorithm for minimum delay routing in a packet switched network. The alternative distributed algorithm closely resembles the algorithm presented by Gallager [Gall77] and modified by Segall [Sega79]. As opposed to the centralized algorithm which is static, the distributed algorithm allows the network to adapt to a slowly changing environment.

2.1) The Model

The network is modeled exactly as a packet switching network. The backbone of the network is a set of communication nodes that are interconnected by full duplex lines (of possibly different capacities). Processing nodes (CPU's) are attached to communication nodes by links as well. The network can be modeled as a graph G with a set N of communication nodes, and a set E of communication links. For the purpose of simplifying the presentation an extra vertex, V_OUT , is used to denote a fictitious destination. The processors in the network are represented as links connecting the communication nodes with the virtual destination, V_OUT . Thus if there are K processors attached to node i then E will include K separate edges from vertex i to V_OUT . Let $P(i)$ be the set of processors at node i .

It is assumed that the traffic (number of jobs in the network) over a long period of time is large enough so that a single job is an infinitesimal portion of the traffic. The model may then deal with the jobs in terms of flows (number of jobs per unit time) rather than number of jobs. Another assumption is that the traffic entering the network behaves as a Poisson source with total rate γ . Each communication node i in N has a separate Poisson arrival stream with rate γ_i ($\gamma = \sum_{i \in N} \gamma_i$). The processing times for jobs are drawn from the same distribution (no matter where the job originated) while the transmission times are drawn from another distribution. We assume that the processing times are independent of the transmission times.

Let $f_{i,k}$ be the flow (of jobs) on link (i,k) ((i,k) in E), and let f_{i,P_j} be the flow from node i to V_OUT through processor P_j . The capacity for communication link (i,k) is denoted as $C_{i,k}$, and the processor capacities (in terms of operations or bits processed per second) are C_{i,P_j} . The flows then have the following constraints:

$$2.1a) f_{i,k} \geq 0; \forall (i,k) \in E$$

$$2.1b) f_{i,P_j} \geq 0; \forall i \text{ in } N; \forall P_j \in P(i)$$

$$2.1c) \sum_{k \in N} f_{ik} + \sum_{P_j \in P(i)} f_{i,P_j} = \gamma_i + \sum_{j \in N} f_{ji}$$

$$2.1d) f_{ik} \bar{b} < C_{ik}$$

$$2.1e) f_{i,P_j} \bar{c} < C_{i,P_j}$$

The third constraint is a statement of the conservation of flow through a node. The parameter \bar{b} , is the average number of bits per job, and \bar{c} is the average number of instruction cycles per job.

Let $D_{ik}(f_{ik})$ be the average delay on link (i,k) given that there is a flow f_{ik} over the link. We assume the following properties (as in Gallager) for the delay function:

- 2.1f) $D_{ik}(f_{ik})$ is a function only of the total flow in link (i,k) . The consequences of this assumption are discussed in [Gall77].
- 2.1g) $D_{ik}(f_{ik})$ is a non-negative continuous increasing function of the flow f_{ik} with continuous first and second derivatives.
- 2.1h) $D_{ik}(f_{ik})$ asymptotically approaches infinity as the flow f_{ik} approaches the capacity C_{ik} .
- 2.1i) $D_{ik}(f_{ik})$ is Convex up.

An example of $D_{ik}(f_{ik})$ given by Kleinrock [Klei75] is the expression for delay in an M/M/1 queueing system: $\frac{f_{ik}\bar{b}}{C_{ik} - f_{ik}\bar{b}}$. We can use this expression for delay if we make an assumption that job lengths (transmission time) are exponentially distributed. Analogously, $G_{i,P_j}(f_{i,P_j})$ is defined as the delay for processor P_j on node i . In the graph model, G_{i,P_j} is the delay across link i,P_j from node i to V_OUT. The same assumptions made above with respect to the transmission delay can be assumed for the processing delay as well. The total delay in the network, denoted by $H(\vec{f})$, is given by the sum of the processing and transmission delays over all the links in the graph.

$$\vec{H}(\vec{f}) = \sum_{(i,k) \in E} D_{ik}(f_{ik}) + \sum_{i \in NP_j} \sum_{P_j \in P(i)} G_{i,P_j}(f_{i,P_j})$$

The minimum delay in the network can be found by finding the set of flows, \vec{f} , that minimizes the expression $H(\vec{f})$. It is a straightforward exercise from [Gall77,

Sega79] to derive the following Kuhn-Tucker conditions that must be satisfied in order to find the flow f^* , where $H(f^*) = \min_{\forall \text{ feasible } f} H(f)$:

$$D'_{ik}(f^*_{ik}) + \alpha^*_k \begin{cases} = \alpha^*_i & f^*_{i,k} > 0 \\ \geq \alpha^*_i & f^*_{i,k} = 0 \end{cases} \quad (2.1)$$

$$G'_{i,P_j}(f^*_{i,P_j}) \begin{cases} = \alpha^*_i & f^*_{i,P_j} > 0 \\ \geq \alpha^*_i & f^*_{i,P_j} = 0 \end{cases} \quad (2.2)$$

The set α^*_i has the following intuitive meaning. Each α^*_i is the marginal delay at node i given the traffic flow at the node. If the input to node i (γ_i) is increased by an amount ϵ then the resulting increase to the delay is given by $\epsilon \alpha^*_i$.

2.2) The Centralized Algorithm

The total delay in the network was described in terms of sums of convex functions. Thus any algorithm that uses a downhill search technique will eventually locate the global minimum delay point. Since the load balancing problem was reduced to a multi-source single destination routing problem, we choose to use the Flow Deviation Algorithm [Frat73] to locate the minimum delay "balance". The algorithm is presented in [Klei76]

2.3) The Distributed Algorithm

We note here once more the similarity of load balancing to the packet routing problem in computer networks. We are able to take advantage of an existing algorithm [Sega79] with some minor modifications. Before the load balancing algorithm is presented, we will explain a few concepts used later on.

Based on the Kuhn Tucker conditions for optimality (Equations 2.1,2.2), the communication nodes will need to know three quantities. The first is the flow passing through the communication links ($f_{i,k}$) and the processors (f_{i,p_j}). These quantities can be estimated over a period of time. The algorithm will iterate slowly enough, and use small enough increments so that nodes should be able to get steady state estimates on the flow. The second quantity is the marginal delay on the links ($D'_{i,k}$). We assume the nodes know the value for the marginal delay for a particular value of the flow, $f_{i,k}$, either by computing directly from the function $D_{i,k}$ or by measuring the quantity directly on the link [Sega79]. The last quantity is the marginal processing delay G'_{i,p_j} . We assume the nodes are able to compute this function in the same manner as the marginal communication delay.

If we take a snapshot (see figure 2.1) of a load balancing network (or graph), there are some properties we expect to see, and others we hope not to see. One property that should be apparent is that the flows form a directed tree. That is, jobs start flowing from their origination nodes through a path of intermediate nodes until they arrive at the destination, V_OUT. However, along a given path of flow each of the nodes should be distinct. If a node i appears twice on a path p from origin j to destination V_OUT, then jobs flowing on path p incur extra delay circling around in a loop that includes node i . Formally, a node j that sends flow down a path j, i_1, i_2, \dots, i_m is upstream from all nodes i_x $x=1, 2, \dots, m$. If there are two nodes i, k ($i \neq k$) that are mutually upstream from each other then there exists a loop, otherwise the network is loop free. Unlike the original packet switching network, here we do not use real or adopted sons. Since all nodes are assumed to have flow arriving from the outside, there are no nodes with "preferred" sons that carry no flow. We mention in the algorithm explanation how looping is avoided.

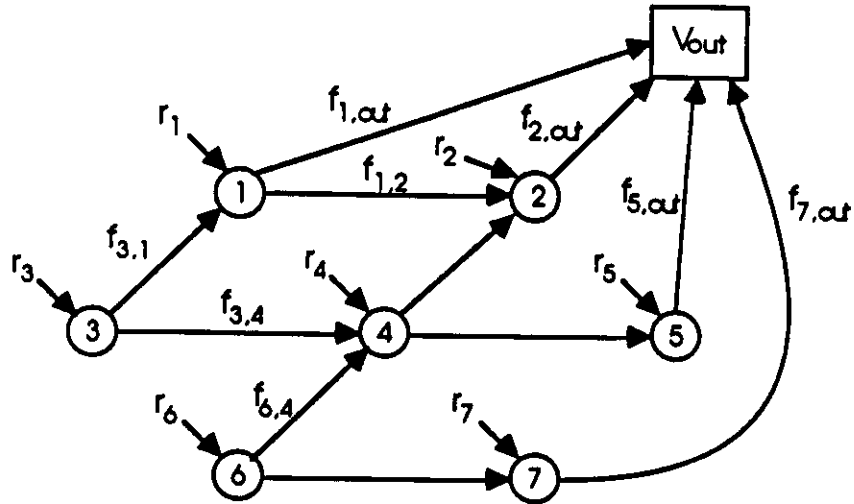


Figure 2.1
Snapshot of a Load Balancing Network

The algorithm works in iterations with each iteration consisting of two phases: information updating and rerouting. The information updating phase starts at the destination node V_OUT and ends at the leaves of the tree. During this phase nodes receive information regarding marginal delays from their downstream neighbors, compute the value of their own marginal delay (α_i) and send this value to their upstream neighbors. Once this phase has been completed at the leaves (nodes that only send out flow and do not receive any) the rerouting phase begins. In this phase each node i waits for its upstream neighbors to complete rerouting (increasing or decreasing) flow to i . When all of i 's upstream neighbors are finished, node i then reroutes the new flow $\sum_{j \in N} f_{j,i}^{new}$ based on the marginal delay values of its downstream neighbors. Basically,

node i will shift flow from downstream neighbors with larger marginal delays to those neighbors with smaller marginal delays. The rerouting phase (and the iteration) ends when the upstream neighbors of V_OUT finish their rerouting. The algorithm iteration ends at the same nodes at which it starts. This property insures that iterations do not overlap, that is, the n^{th} iteration will finish before the $n+1^{st}$ iteration starts. In both phases the algorithm follows the topology of the directed flow tree (from the root to the leaves and then back to the root), thus the loop free property of the algorithm is needed not only for efficiency but also for the correctness of the algorithm. We define a *Blocked* set of nodes to be all neighbors of a node i that i cannot start sending flow to in this iteration. If i were to start sending flow to a Blocked node then a loop might form.

The last note concerns the modifications to Segall's algorithm. Since the destination node, V_OUT , is only a virtual node, the responsibility of starting each iteration is with the upstream neighbors of V_OUT . That is, all communication nodes that send flow to their own processors (direct links to V_OUT) will initiate the next iteration. We still use the fact that $\alpha_{V_OUT} = 0$. In the discussion after the algorithm we elaborate further on this modification.

We now proceed to present the algorithm in detail. The notation f^n is the set of flows at the n^{th} iteration, α_i^n is the marginal delay at node i in the n^{th} iteration. A step size η will be used in the algorithm and explained later.

The algorithm is the same for every node i ($i=1,2,\dots,N$) in the network.

Phase I. (information update)

1. Wait for the α_j 's from each downstream neighbor j . If any downstream neighbor is blocked then node i declares itself to be

blocked. Let $NB(i,n)$ be the set of all downstream neighbors (plus any other neighbors that sent their α 's) that are not blocked in the n^{th} iteration.

2. Let $\alpha_i^n = \min_{k \in NB(i,n); P_j \in P(i)} \{[\alpha_k^n + D'_{i,k}], G'_{i,P_j}\}$
3. if for any downstream neighbor k , $\alpha_i^n \leq \alpha_k^n$ and $\eta[\alpha_k^n + D'_{i,k} - \alpha_i^n] < f_{i,k}^n$ then node i declares itself blocked.
4. Send α_i^n to all but the downstream neighbors. Also, notify them if node i is blocked.

Phase II. (Rerouting)

1. Wait for α 's from all the neighbors. Let

$$B(i,n) = \{j \mid j \text{ is not a blocked neighbor of node } i \wedge f_{i,j}^{n-1} > 0\}$$

For all neighbors $j \notin B(i,n)$ let $a_{i,j}^n = 0$ (deviate no new flow to those blocked neighbors).

For all neighbors $j \in B(i,n)$ let

$$a_{i,j}^n = [\alpha_j^n + D'_{i,j}] - \min_{m \in B(i,n); P_j \in P(i)} \{[\alpha_m^n + D'_{i,m}], G'_{i,P_j}\}$$

and

$$a_{i,P_j}^n = [G'_{i,P_j}] - \min_{m \in B(i,n); P_j \in P(i)} \{[\alpha_m^n + D'_{i,m}], G'_{i,P_j}\}$$

Choose a single processor that has the minimum marginal delay:

$$L_i^n = P_j \text{ where } a_{i,P_j}^n = 0$$

and $K_i^n = 0$ for all i . If no such processor exists then choose an outgoing link that has the minimum marginal delay.

$$K_i^n = k \text{ where } k \in B(i, n) \text{ and } (a_{i,k}^n = 0)$$

and $L_i^n = 0$ for all i .

2. Let $f_{i,k}^{n,1}$ be the flow on link (i,k) after all upstream neighbors of i have completed rerouting their flows. Let

$$\Delta_{i,k}^n = \min \left[f_{i,k}^{n,1}, \eta a_{i,k}^n \right]$$

$$\Delta_{i,P_j}^n = \min \left[f_{i,P_j}^{n,1}, \eta a_{i,P_j}^n \right]$$

3. Reroute flow away from links (or processors) that have higher marginal delays. If $L_i^n \neq 0$ then

$$f_{i,P_j}^{n,2} = \begin{cases} f_{i,P_j}^{n,1} - \Delta_{i,P_j}^n & P_j \neq L_i^n \\ f_{i,P_j}^{n,1} + \sum_{P_j \in P(i)} \Delta_{i,P_j}^n + \sum_{k \in N} \Delta_{i,k}^n & P_j = L_i^n \end{cases}$$

and

$$f_{i,k}^{n,2} = f_{i,k}^{n,1} - \Delta_{i,k}^n \quad \forall k \in N$$

otherwise (if the processors have higher marginal delays)

$$f_{i,k}^{n,2} = \begin{cases} f_{i,k}^{n,1} - \Delta_{i,k}^n & k \neq K_i^n \\ f_{i,k}^{n,1} + \sum_{P_j \in P(i)} \Delta_{i,P_j}^n + \sum_{k \in N} \Delta_{i,k}^n & k = K_i^n \end{cases}$$

and

$$f_{i,P_j}^{n,2} = f_{P_j}^{n,1} - \Delta_{P_j}^n \quad \forall P_j \in P(i)$$

4. New flow entering at node i is routed to downstream node or processor with minimum marginal delay. If $L_i^n \neq 0$ then

$$f_{i,P_j}^{n+1} = \begin{cases} f_{i,P_j}^{n,2} & P_j \neq L_i^n \\ f_{i,P_j}^{n,2} + \text{new flow} & P_j = L_i^n \end{cases}$$

and $f_{i,k}^{n+1} = f_{i,k}^{n,2}$ (for all $k \in N$). Otherwise $K_i^n \neq 0$

$$f_{i,k}^{n+1} = \begin{cases} f_{i,k}^{n,2} & k \neq K_i^n \\ f_{i,k}^{n,2} + \text{new flow} & k = K_i^n \end{cases}$$

and $f_{i,P_j}^{n+1} = f_{i,P_j}^{n,2}$ for all $P_j \in P(i)$.

5. The new set of downstream neighbors includes all nodes j with $f_{i,j}^{n+1} > 0$, in addition to the virtual node, V_OUT, if there exists at least one processor $P_j \in P(i)$ with flow $f_{i,P_j}^{n+1} > 0$.
6. Send α_i^n and blocking status to all downstream neighbors.

Initialization of the network is a simple matter in this algorithm. All nodes initially send their jobs to their own processors. If a node does not have its own processor then it may evenly divide its incoming flow over all of its outgoing links. The nodes may then start to measure their respective marginal delays and proceed with the above algorithm. We assume the existence of a flow control policy that prevents a short term saturation problem on a processor or link, especially in the initialization phase.

The algorithm identifies situations where loops may occur and acts to avoid them by controlling the path of the flow (see figure below).

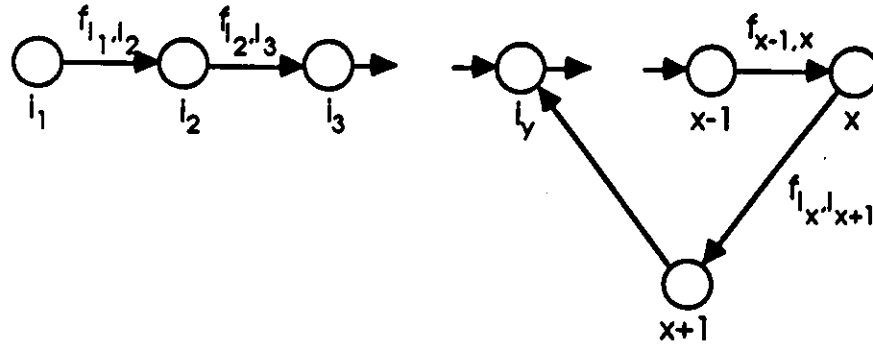


Figure (2.2)
Possible Loop in Flow Tree

Assume there exists a path p of distinct nodes i_1, \dots, i_m where i_1 is the originating node and i_m is the upstream neighbor of V_OUT. A loop may occur if a downstream neighbor of i_x ($x=1, \dots, m-1$) decides to send flow to i_x or to any node i_y upstream from i_x . Node i_{x+1} may decide to send flow to i_y ($y=1, \dots, x$) if the marginal delay at i_{x+1} (α_{x+1}) is greater or equal to the marginal delay at i_y (α_y). If the flow $f_{i_x, i_{x+1}}$ is greater than the amount i_x can divert away from i_{x+1} , $\left[\eta[\alpha_{i_{x+1}} + D'_{i_x, i_{x+1}} - \alpha_{i_x}] \right]$, there will exist a loop of flow from i_x to i_{x+1} through i_y and back to i_x . The algorithm detects this situation in step (I.3) and blocks i_x and all nodes upstream (i_1, \dots, i_{x-1}) in a blocked set. The algorithm in step (II.1) will prohibit node i_{x+1} to any member of a blocked set thus preventing a loop from forming. In the ensuing iterations i_x will continue to divert flow away from i_{x+1} (assuming $\alpha_{i_{x+1}} \geq \alpha_{i_x}$ still holds) until there is no more and i_{x+1} is no longer a downstream neighbor of i_x . Only then will nodes i_1, \dots, i_x be removed from the blocked set and i_{x+1} will be allowed to send flow to some i_y .

$(y=1, \dots, x)$.

There are two differences between our algorithm and Segall's original algorithm. The first change involves the different functions used for processor delays. In step (I.2) and throughout phase II each node compares and diverts flow based on the marginal delays of communication and of processing. However, processors act as communication links to the virtual destination thus G can be considered a delay function for that link. The second difference is in the way iterations of the algorithm are synchronized. In Segall's algorithm the destination acts as a controller. When the upstream neighbors finish rerouting flow they send their α 's to the destination node signaling an end to the iteration. The destination node may then broadcast a message ($\alpha=0$) to its upstream neighbors starting the next iteration. In this algorithm the destination is a virtual node, so that synchronizing the iterations is left to the upstream neighbors of V_OUT . Assume that one node (or a set of nodes) wakes up and starts broadcasting its marginal delay to its neighbors. Eventually this broadcast wake up call propagates throughout the network since the graph G is assumed to be connected. When the broadcast reaches the leaves of the path tree the first phase will be completed and the second phase will begin. Different iterations cannot overlap since each node i is held up from rerouting (Phase II) until all upstream neighbors have sent node i the value of their marginal delays. Thus all nodes reroute flow for the same iteration in the same phase.

The step size η is used to deviate an amount of flow during each iteration such that the algorithm eventually converges to the minimum delay. Letting M be an upper bound to the value of D'' (the second derivative of the delay with respect to a flow), Gallager and Segall have shown [Gall77] that with a step size

$$\eta = \left[2MN^5 \right]^{-1}$$

any rerouting strictly decreases the delay in the network. Greater values for the step size that guarantee convergence and speed up the descent of the network delay may exist. The authors in [Bert84] presents a method of dynamically changing the step size at each node that helps speed up the convergence. However, there exist cases where the dynamic behavior of the step size causes the algorithm to diverge.

The proof of correctness (and convergence) for the algorithm can be found in [Sega79] and will not be repeated here.

In phase 2 flow is decremented from all links and processors with high marginal delays and rerouted to the processor or communication link with the minimum marginal delay. Bertsekas [Bert78] suggested an approach where flow is rerouted from links with α 's higher than some quantity α' to links with α 's lower than α' . In Gallager's algorithm the value of α' is the minimum of the marginal link delay plus the marginal delay of a neighbor over all the neighbors. A simple value for α' is

$$\alpha' = \frac{\sum_{k \in B(i,n)} \alpha_k^n + D'_{i,k}}{|B(i,n)|}$$

That is, α' is the average marginal delay of sending an increment of flow over all the downstream neighbors (that are not blocked). A node i then decrements $\Delta_{i,j}$ from all downstream nodes j whose value $\alpha_j^n + D'_{i,j} > \alpha'$ and reroutes this flow (plus all new flow) to all downstream neighbors $l \in B(i,n)$ whose value $\alpha_l^n + D'_{i,l} < \alpha'$. The rerouted (plus new) flow should be proportionally distributed, that is

$$f_{i,l}^{n,2} = f_{i,l}^{n,1} + \left[1 - \frac{\alpha_l^n + D'_{i,l}}{\alpha'} \right] \sum_{\alpha_j + D'_{i,j} > \alpha'} \Delta_{i,j}$$

so that nodes with a very low value receive more of the rerouted flow than nodes with

higher marginal delays.

2.4) Summary

In this chapter we presented a model (previously used) of a load balancing network, and a distributed algorithm for achieving the minimum delay. The algorithm is efficient in its message complexity. In every round each node sends one message of constant length to each of its neighbors. Thus the message complexity is $O(|E|)$ where $|E|$ is the number of links in the net. Although the number of links can be as high as N^2 , most nets have a bounded number, M , of links per node. The distributed algorithm of [Kuro86] has a message complexity of $O(N^2)$ at best, and $O(N^3)$ at worst. In addition, the algorithm in this work uses a loop avoidance mechanism, while the algorithm in [Kuro86] does not. We can reduce the number of iterations needed till convergence with the modification that flow can be increased on more than one link.

Our algorithm can be easily extended to include site constrained jobs. The virtual destination of all jobs is still V_OUT, however, we introduce chains of open traffic that are allowed only specific routes in the network. Each chain is routed through a path of links and finally through a processor that is allowed to process that class of traffic.

CHAPTER 3

THRESHOLD NETWORKS

In the previous chapter we examined a quasi-static load balancing network and algorithm. The communication nodes route jobs either to their own processors or to other communication nodes using probabilities that are updated every so often. That is, when a job arrives to node i (from another node or from the outside), the job will be routed to an outgoing link (i,k) with probability $\phi_{i,k}$ or to a processor P_j with probability ϕ_{P_j} . The sum of the routing probabilities $\sum_{k \in N} \phi_{i,k} + \sum_{P_j \in P(i)} \phi_{P_j} = 1$. The flow across a link (i,k) is just $\phi_{i,k} f_i$ where f_i is the total flow passing through node i . The distributed algorithm presented in the last chapter minimizes the overall network delay by having each node determine its local set of ϕ 's. The problem with routing probabilities is that jobs may get sent off to remote locations when the local processor is idle or momentarily underutilized. Although the network average delay may be minimized, the variance of the delay may be quite high. In this chapter we develop a new approach to load balancing that attempts to correct the short term problems of routing probabilities yet maintains the minimum delay in the long run. We present a communication node with a *threshold* policy in a network setting. Although the model developed here is ideal for simplicity, more complicated models could be developed so long as the delay function retains convexity.

First, the model will be presented and analyzed in terms of the given parameters. Next, we will develop an optimization problem and show a solution to it. The model includes an independence assumption regarding the arrival process to each

node. We include simulation results to verify the assumption.

3.1) Model

The network consists of a set N of nodes, and a set E of links connecting the nodes. At each node i there is an outside arrival process which is Poisson with intensity γ_i . There are also arrivals to the nodes from other nodes. Jobs arriving to node i join one queue where they wait for service. Each node i has two servers, a processor, and a communications server. The processor is used to perform the work the job brings in. Whenever the processor becomes idle it will select the next job at the head of the queue (if there is one waiting, otherwise it will wait for the next arrival) and process the job. Each job has a service time that is negatively exponentially distributed with an average time $1/\mu$ cycles per job. The processor at node i works at a constant rate of R_i cycles per second. Combining the processor rate and the average service time yields an equivalent model of a negatively exponentially distributed server with average rate $\mu_i = \mu R_i$ jobs per second. The communications server is used to transmit excess jobs to remote nodes in the network. If there are T_i jobs waiting in the queue, the communications server (when idle) will take the next available customer in the queue and start to transmit him to another node. If the number in the system drops below T_i+1 then the communications server stops transmitting. If there was a customer in transmission at the time, it is transferred to the (idle) processor. The transmission time for a job is also negatively exponentially distributed with mean $1/\alpha$ bits per job. The communications server at node i transmits at a constant rate C_i bits per second. We can model the communications server as a negatively exponentially distributed server with rate $\alpha_i = \alpha C_i$ jobs per second. We have made two assumptions in our analysis. The first is that job transmission times are redrawn from the same distribution every time a job is transmitted over two or more hops. The second is that consecutive job

arrivals have transmission times that are independent of one another (independence assumption [Klei64]). A further assumption is that job transmission times and job service times are independent of each other.

In summary, the network is modeled as a set of independent $G/M/2$ queueing nodes with server rates μ_i and α_i . The first server operates whenever there are customers in the system. The second server operates whenever there are T_i+1 or more customers in the system. A customer in the second server may be preempted and placed in the first server if the number in the system drops below T_i+1 before his service (transmission) is complete. The arrival process to a node consists of an external Poisson process and a general arrival process from other nodes. Each node may be connected to a number of different nodes. A customer in transmission may be sent from i to j with probability $P_{i,j}$.

The output process from a communications server is not a simple one. The overflow process described here is even more complicated than the process described and analyzed by Cinlar and Disney [Cin67]. In a typical network, a node has an exogenous arrival process plus a bounded number of incoming links. If we assume that the main source of jobs is from the outside, or that the number of incoming links is great enough, then the arrival process behaves approximately as a Poisson process. This assumption implies that each node behaves as if it were in isolation with the same arrival intensity as in the network. In other words we assume a Jacksonian network of threshold queues. It can be shown for small networks (two nodes) that the Jackson model does not satisfy the system steady state equations, so that our last assumption is quite bold. In section (3.5) we verify this assumption with our simulation results.

The network probability distribution can be expressed in terms of the individual nodal distributions. The equilibrium probability distribution for the number in the system can be expressed as:

$$P[\vec{N}] = B^{-1} \prod_{i=1}^{|N|} P_i[n_i] \quad (3.1)$$

where $\vec{N} = (n_1, n_2, \dots, n_N)$ and $P_i[n_i]$ is the steady state distribution of having n_i jobs at node i in isolation. The constant B^{-1} is a normalizing constant. Calculating the steady state distribution for any node in isolation is a simple matter. The only variable that needs to be accounted for is the number in the system. What the behavior of the node is or whether there is a job in transmission or not is solely determined by the number in the system. Below we have included a figure of the birth death chain that describes the single node system behavior.

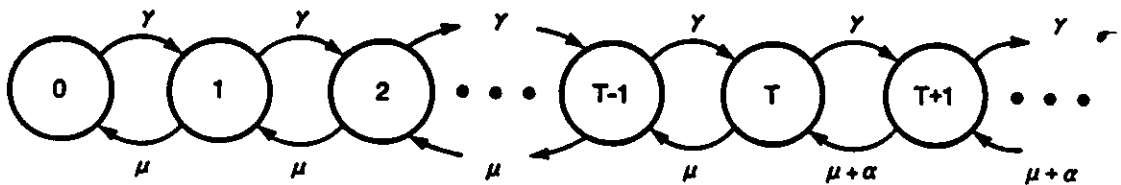


Figure (3.1)
Markov Chain for Node in Isolation

The arrival rate is λ no matter what the nodal state is. The service rate is μ when the second server is off, and $\mu + \alpha$ when there are more than T in the system. The equilibrium probability of k in the (single node) system is:

$$P_k = \begin{cases} P_0 \left(\frac{\lambda}{\mu} \right)^k & \text{if } k < T \\ P_0 \left(\frac{\lambda}{\mu} \right)^T \left(\frac{\lambda}{\mu + \alpha} \right)^{k-T} & \text{if } k \geq T \end{cases} \quad (3.2)$$

where

$$P_0 = \left[\sum_{k=0}^{T-1} \left(\frac{\lambda}{\mu} \right)^k + \left(\frac{\lambda}{\mu} \right)^T \sum_{k=T}^{\infty} \left(\frac{\lambda}{\mu + \alpha} \right)^{k-T} \right]^{-1} = \left[\frac{1 - \left(\frac{\lambda}{\mu} \right)^T}{1 - \frac{\lambda}{\mu}} + \frac{\left(\frac{\lambda}{\mu} \right)^T}{1 - \frac{\lambda}{\mu + \alpha}} \right]^{-1} \quad (3.3)$$

The total arrival rate to a node i , λ_i , is composed of exogenous arrivals (at rate γ_i), and overflow arrivals from neighboring nodes. The overflow rate from a node can be calculated by considering the long term flow of jobs that are communicated, not processed. Over a long period of time the flow of jobs that node j transmits is:

$$f_j = \sum_{k=T+1}^{\infty} \alpha_j P_k^j = \frac{\alpha_j \left(\frac{\lambda_j}{\mu_j} \right)^{T_j} P_0^j \left(\frac{\lambda_j}{\mu_j + \alpha_j} \right)}{1 - \frac{\lambda_j}{\mu_j + \alpha_j}} \quad (3.4)$$

A node j transmits jobs only when there are T_j+1 or more jobs in the system. In each state $k \geq T_j+1$ the transmission rate of jobs is a constant α_j . A job that is transmitted will be sent to a neighboring node l with probability $p_{j,l}$ ($\sum_{l \in N} p_{j,l} = 1$), so that the resulting flow of jobs from j to l is expressed as

$$f_{j,l} = f_j p_{j,l} \quad (3.5)$$

The total input rate to j is given by

$$\lambda_j = \gamma_j + \sum_{i \in N-j} f_{i,j} \quad (3.6)$$

The conservation of flow at each node j requires that

$$\sum_{k \in N} f_{j,k} + f_{j,OUT} = \gamma_j + \sum_{i \in N} f_{i,j} \quad (3.7)$$

The average number in the network \bar{M} is the sum of the individual averages (\bar{m}_i) for each node, $\sum_{i \in N} \bar{m}_i$. Since the input rate to the net is fixed we see (by Little's result) that minimizing the average number in the net is equivalent to minimizing the average response time of the jobs. The average number at a node is easily calculated for this model using the probability distributions in (3.2) and (3.3). The average number is expressed as a function of the threshold, the arrival and service rates.

$$\bar{m}_i = \frac{\frac{x - Tx^T + x^{T+1}(T-1)}{(1-x)^2} + \frac{x^T(y + T(1-y))}{(1-y)^2}}{\left[\frac{1-x^T}{1-x} + \frac{x^T}{1-y} \right]} \quad (3.8)$$

where $x = \frac{\lambda_i}{\mu_i}$, $y = \frac{\lambda_i}{\mu_i + \alpha_i}$, and T is the threshold at node i . The problem with the expression is that the only control variable the node can use is the threshold, T . The difficulty lies with the behavior of \bar{m} as T varies. When $T=1$ (the lowest T can be) the node behaves as an $M/M/2$ system with nonequal server rates. When $T \rightarrow \infty$ the system behaves exactly like an $M/M/1$ system. The average number in the system then starts from a finite value and increases asymptotically to another finite value as T increases. Unfortunately, \bar{m}_i is not a convex function with respect to the control variable T . In order to handle any optimization problem we need a convex function. We now express the system average in terms of the flows leaving the node (through the processor and communication links). From equation (3.4) we have the expression for the flow of jobs transmitted from a node. Using the same reasoning we can calculate the flow of jobs, β , processed at a node.

$$\beta = \sum_{k=1}^{\infty} \mu P_k = \begin{cases} \mu P_0 \left[\frac{\frac{\lambda}{\mu} \left(\frac{\lambda}{\mu} \right)^T}{1 - \frac{\lambda}{\mu}} + \frac{\left(\frac{\lambda}{\mu} \right)^T}{1 - \left(\frac{\lambda}{\mu + \alpha} \right)} \right] & \lambda \neq \mu \\ \mu P_0 \left(T + \frac{\mu}{\alpha} \right) & \lambda = \mu \end{cases} \quad (3.9)$$

The ratio of β/f , when $\lambda \neq \mu$, is given by:

$$\beta/f = \frac{\mu \left[\frac{\frac{\lambda}{\mu} \left(\frac{\lambda}{\mu} \right)^T}{1 - \frac{\lambda}{\mu}} + \frac{\left(\frac{\lambda}{\mu} \right)^T}{1 - \left(\frac{\lambda}{\mu + \alpha} \right)} \right]}{\frac{\alpha \left(\frac{\lambda}{\mu} \right)^T \left(\frac{\lambda}{\mu + \alpha} \right)}{1 - \left(\frac{\lambda}{\mu + \alpha} \right)}} \quad (3.10a)$$

and, when $\lambda = \mu$, is:

$$\beta/f = T + \frac{\mu}{\alpha} \quad (3.10b)$$

Isolating for T yields:

$$T = \begin{cases} \frac{\log \left[\frac{C}{C-1 + \frac{\alpha\beta\lambda}{\mu f(\mu+\alpha)}} \right]}{\log \left[\frac{\lambda}{\mu} \right]} & \lambda \neq \mu \\ \frac{\beta}{f} - \frac{\mu}{\alpha} & \lambda = \mu \end{cases} \quad (3.11)$$

where

$$C = \frac{1 - \frac{\lambda}{\mu+\alpha}}{1 - \frac{\lambda}{\mu}} \quad (3.12)$$

Replacing T in equation (3.8) with (3.11) and (3.12) we now have an expression for the average number in a node as a function of f and β . We denote the delay function for node j to be

$$D_j(f_j, \beta_j) = \frac{\bar{m}_j(f_j, \beta_j)}{\lambda_j}$$

and the system delay (response time) to be

$$D(\vec{f}, \vec{\beta}) = \sum_{j \in N} (f_j + \beta_j) D_j(f_j, \beta_j)$$

We were unable to prove that the average delay in system for a network node is convex with respect to the flow variables β and f . We assume that the delay function is convex in the next section which discusses the problem formulation and optimization.

3.2) Formulation of Problem

The load balancing problem (LBP) can now be formally presented:

LBP: Minimize $D(\vec{f}, \vec{\beta})$ subject to the following constraints.

1. $\sum_{k \in N} f_{j,k} + \beta_j = \gamma_j + \sum_{i \in N} f_{i,j}$
2. $\beta_j \geq 0$
3. $f_j \geq 0$
4. $\beta_j < \mu_j$
5. $f_j < \alpha_j$

The last two constraints are not necessary since the delay function approaches infinity when those constraints are violated (and the step size of flow is small enough so that we will not overshoot the capacity constraints).

Although the threshold T was presented as a discrete parameter, we will assume it to be continuous for simplicity. Since D_j is a function of only f_j and β_j , we can find the minimum of $D(\vec{f}, \vec{\beta})$ using Lagrangian multipliers $(\phi_j, \psi_j, \eta_{j,k})$. Let

$$G(\vec{f}, \vec{\beta}, \phi, \psi, \eta) = D(\vec{f}, \vec{\beta}) - \sum_{j \in N} \phi_j \left[\sum_{k \in N} f_{j,k} + \beta_j - \sum_{i \in N} f_{i,j} - \gamma_j \right] + \sum_{j \in N} \sum_{k \in N} \eta_{j,k} f_{j,k} + \sum_{j \in N} \psi_j \beta_j$$

The optimal solution satisfies the following Kuhn Tucker conditions:

$$\frac{\partial G}{\partial f_{j,k}} = \frac{\partial D_j \left[\sum_{l \in N} f_{j,l}, \beta_j \right]}{\partial f_{j,k}} - \phi_j + \phi_k + \eta_{j,k} = 0 \quad (3.13)$$

$$\frac{\partial G}{\partial \beta_{j,k}} = \frac{\partial D_j(f_j, \beta_j)}{\partial \beta_j} - \phi_j + \psi_j = 0 \quad (3.14)$$

$$f_{j,k} \geq 0 \quad \eta_{j,k} f_{j,k} = 0 \quad (3.15)$$

$$\beta_j \geq 0 \quad \psi_j \beta_j = 0 \quad (3.16)$$

From (3.15) we see that either flow $f_{j,k}$ is zero or positive yielding:

$$\frac{\partial D_j \left[\sum_{l \in N} f_{j,l}, \beta_j \right]}{\partial f_{j,k}} + \phi_k \begin{cases} = \phi_j & f_{j,k} > 0 \\ > \phi_j & f_{j,k} = 0 \end{cases} \quad (3.17)$$

From (3.16) either flow β_j is zero or positive which yields:

$$\frac{\partial D_j(f_j, \beta_j)}{\partial \beta_j} \begin{cases} = \phi_j & \beta_j > 0 \\ > \phi_j & \beta_j = 0 \end{cases} \quad (3.18)$$

Equations (3.17) and (3.18) are a compact form of the Kuhn Tucker conditions. As in Chapter 2, ϕ_j is the marginal delay at node j . That is, if the input to j , γ_j , is increased by ϵ , then the resulting increase in the network delay will be $\epsilon \phi_j$. An increase in the input at j will cause an incremental increase in the processor delay $\left[\frac{\partial D_j(f_j, \beta_j)}{\partial \beta_j} \right]$ and an incremental increase in the communication delay $\left[\frac{\partial D_j \left[\sum_{l \in N} f_{j,l}, \beta_j \right]}{\partial f_{j,k}} + \phi_k \right]$ for each node k that j sends flow to. Equations (3.16) and (3.17) specify that the optimum point (f^*, β^*) has been achieved when the cost of sending an incremental amount of flow to all downstream neighbors is equal and no more than the cost of sending flow to a non-downstream neighbor. The cost of sending more flow to a processor should be equal to the cost of sending incremental flow to a downstream neighbor.

3.3) Distributed Algorithm

The network can now be viewed as a set of communication nodes each with its own threshold, processor, and communication links. The problem is for each node to regulate the flows to its own processors and to other nodes by increasing or decreasing its "reservoir level" . By raising (increasing the threshold) or lowering (decreasing the threshold) the dam, each node controls the reservoir of jobs that it holds. If the downstream neighbors of a node j are becoming congested, then j could increase the threshold, retaining more jobs for its own processors, and decreasing the flow it sends downstream. If, on the other hand, j 's own processors are getting congested then j could lower the threshold (dam) and pass on more of its load to downstream neighbors.

The distributed algorithm for this model is similar to the one presented in Chapter 2. Both problems deal with local regulation of the flow. One difference with the previous algorithm is that in this case after a node decides on what the outgoing flows should be (rerouting), the node has to decide on a threshold and routing probabilities that will achieve those flows.

Another difference with the previous model deals with synchronizing the iterations. Here, all nodes have links to the virtual node (V_OUT) through their own processor. Thus the responsibility of synchronizing the phases is with those nodes whose thresholds are set to infinity (i.e. they do not send flow to any downstream neighbor). Initially, all nodes may send their input flows to their own processors. This may overload the processors for which $\gamma > \mu$. We assume that the node will drop the excess input (using flow control techniques) until it is able to start sending some flow to downstream neighbors. The initial value of the threshold for each node is infinity, and the marginal delay, ϕ , is the marginal processor delay. The changes to the earlier algo-

rithm are during the rerouting steps (phase 2). We now list the new step (2.3') that comes right after step (2.3):

(2.3'):

- a. $f_i^{n+1} = \sum_{k \in N} f_{i,k}^{n+1}$
- b. $\beta^{n+1} = f_{i,P_j}^{n+1}$
- c. Calculate and set T using equations (3.11), (3.12), f_i^{n+1} , and β^{n+1}

$$\text{let } P_{i,k}^{n+1} = \frac{f_{i,k}^{n+1}}{f_i^{n+1}}$$

3.4) Simulation Results

We ran simulations of the arrival process at a network node with a different number of input sources. Figure (3.2) includes a table of the simulation results compared to what a true Poisson process with the same intensity would yield. Each input source was a threshold queue with Poisson input $\gamma_i = 1$ and a service rate $\mu_i = .75$, and a transmission rate $\alpha_i = .5$. The threshold was varied from 1 to 5. We ran simulations with different numbers of input sources for each threshold value. The arrival process to a network node consists of the overflow process from each input source plus the exogenous Poisson arrival process (with rate γ). In our simulations we set γ to 0 so as to study the overflow processes more carefully. We found that by superimposing a Poisson process on the overflow processes, the correlation coefficient further decreased thus making the total arrival process more "Poisson".

T	S	Arrival Rate		Variance		Correlation
		Simulation	Expected	Simulation	Poisson	
1	1	.3479	.3478	10.62	8.27	.0301
	2	.6975	.6957	2.36	2.07	.0337
	3	1.041	1.044	1.02	.918	.0331
	4	1.386	1.391	.5601	.5166	.027
	5	1.738	1.739	.3524	.3306	.0246
	6	2.084	2.087	.2433	.2296	.0266
2	1	.3159	.3168	16.7	9.96	.0458
	2	.6324	.6337	3.38	2.49	.0608
	3	.9542	.9505	1.34	1.11	.0538
	4	1.269	1.267	.717	.6226	.0535
	5	1.59	1.584	.4465	.3985	.0460
	6	1.889	1.901	.3114	.2767	.0402
	7	2.215	2.218	.2213	.2033	.0381
3	1	.2967	.297	23.3	11.3	.0503
	2	.5944	.594	4.33	2.83	.0786
	3	.8947	.891	1.68	1.26	.0813
	4	1.195	1.188	.8592	.7086	.072
	5	1.49	1.485	.5313	.4535	.0615
	6	1.775	1.782	.3658	.3149	.0519
	7	2.085	2.079	.2578	.2314	.053
4	1	.2821	.2837	30.8	12.4	.0532
	2	.5661	.5673	5.31	3.11	.0864
	3	.855	.851	1.98	1.38	.0864
	4	1.13	1.135	1.027	.7768	.085
	5	1.41	1.418	.611	.4971	.075
	6	1.698	1.702	.4105	.3452	.0644
	7	1.977	1.986	.2945	.2536	.0564
	8	2.262	2.269	.2197	.1942	.0448
5	1	.2728	.2744	37.4	13.3	.0512
	2	.552	.5488	6.2	3.32	.091
	3	.8266	.8233	2.19	1.48	.1
	4	1.104	1.098	1.115	.8299	.0891
	5	1.382	1.372	.6531	.5312	.0739
	6	1.646	1.647	.4461	.3689	.0729
	7	1.922	1.921	.3157	.271	.0614
	8	2.199	2.195	.2366	.2075	.0576
	9	2.482	2.47	.1834	.1639	.0468

Figure (3.2)
 Interarrival Process Simulation Results
 $\gamma_i=1$, $\mu_i=.75$, $\alpha_i=.5$, 200,000 arrivals per simulation run
 T= Threshold; S= Number of Sources

The expected arrival rate was calculated using equation (3.4). The simulation arrival rates are within 1% of the expected arrival rates for each case in figure (3.2). The variance of the simulated interarrival process is consistently higher than the variance of a pure exponentially distributed interarrival process (corresponding to the Poisson arrival process). This observation is expected since the overflow processes from the input sources are much more bursty than the Poisson process. Each input source cycles through a period consisting of a phase where the transmitter is turned on and sending jobs at a rate α_i , and an off phase where no jobs are transmitted. The length of time of the off phase depends on the parameters γ_i , μ_i , α_i , and the threshold, T . In particular, holding γ_i , μ_i , and α_i constant and varying only the threshold, we see that the simulation variance (for a given number of input sources) is closer (both relatively and absolutely) to the pure (exponentially distributed) process variance when the threshold is lower. When T is smaller, the "off" phase is smaller making the overflow process less bursty, hence the smaller variance. For any particular threshold T , the variance of the simulated process approaches (both relatively and absolutely) the assumed process as the number of input sources increases. This last observation is also no surprise since it is well known that the superposition of N independent renewal processes tends towards a Poisson process for large N (Palm-Khinchin Theorem).

In the last column of figure (3.2) we listed the correlation coefficients of the interarrival process for each simulation. We observed that the correlation never exceeded 10% (which is small). Furthermore, for a particular threshold T , the correlation coefficient decreases when the number of sources grows large. Again, this effect is a result of the Palm-Khinchin Theorem. We noticed that the correlation coefficient increases with the number of input sources, N , when N is small. We cannot explain this phenomena although we note it. We also observed that for a particular number of input sources the correlation coefficient increases with the threshold. The increase is

due to the longer "off" phases. If the last interarrival period was long the next one is likely to be short (since the input source is now "on"). This likelihood increases if the "off" periods increase in length relative to the interarrival time during an "on" period.

The result of all these simulations is that in heavily loaded networks (where load balancing is needed) the independence assumption is valid when the number of input sources is greater than 4 or 5. The variance of the simulations differed by .1 or less from the assumed process variance at those points. We did not run simulations for thresholds greater than 5, however, there is evidence [Eage84] that the source nodes in a heavily loaded network will have thresholds of 4 or less.

3.5) Summary

We have formulated a new model of a load balancing network that includes nodes with thresholds. The nodes use their threshold values instead of routing variables to control the flows. We used the system delay and the constraints on the flows to derive the Kuhn Tucker conditions for optimality. We modified the algorithm in Chapter 2 so that the nodes could settle on the optimum flows in a distributed manner. Each node sets its own threshold so that flow is routed to the links or processor with minimum marginal delay.



CHAPTER 4

LOAD SHARING IN BROADCAST NETWORKS

4.1) Introduction and Motivation

The previous two chapters dealt with load balancing algorithms in point to point networks. The networks serve as the communication basis for a *distributed processing* environment. Jobs that arrive to a node in the network may be processed at that site or at other sites (nodes) in order to reduce the expected job response time. The algorithms presented earlier reroute flow (of jobs) incrementally based on local information that is computed, measured, or received from neighboring nodes. As the algorithms iterate, the objective function (namely, the average delay of a job in the network) approaches, or converges to the global minimum. How would such iterative algorithms perform in a broadcast environment? That is, assuming the nodes were connected by a broadcast medium, these algorithms are not nearly as efficient in terms of time and message complexity as are other methods in this medium. In each iteration, every node broadcasts one message (containing the marginal delay at that node), yielding an $O(N)$ time and message (per iteration) complexity, where N is the number of network nodes. However, quite a few iterations would be needed until the delay converged to a point close to the optimum. A far quicker solution is for each node to broadcast its input (N messages), and for each node to carry out the same centralized optimization algorithm to find the matching (sources to sinks) that would minimize the network delay. This is the approach in [Ston78] (far more efficient in time and messages than the algorithm in [Kuro86]). If the inputs to the network change, then

this broadcast matching has to be done every time some node's input varies significantly. The cost of this approach is $O(N)$ time and messages each time the network needs to rebalance.

In this chapter, we draw on some intuition discussed in the first chapter. That is, a little load sharing goes a long way. Rather than try to balance the entire network, we relieve the high delay at the most heavily utilized machines by sharing their (processing) load with some underutilized machines. Although this idea appears in a number of places, we provide a new approach that matches up pairs of nodes. The model we develop is completely new along with the expected delay analysis. The results obtained from the analysis describe the response time as a function of the amount of load sharing. Our results show that significant reductions in the average processing time of a job (over the whole network) can be achieved by pairing up a fraction of the total number of nodes. The amount of work needed to pair up the nodes is minimal compared to the flooding mechanisms found in other approaches.

4.2) Preliminaries

4.2.1) Model

In this chapter we use a broadcast medium for communication among N processors. Each processor (or node) in the network receives work from an external user population, and (possibly) from other nodes in the network. The external input to node i (from the users) arrives as separate jobs. The external job stream into node i has a Poisson distribution with rate γ_i jobs per second. All of the processors in the network are identical; in particular, they all have the same speed. All the jobs arriving to the network are assumed to have the same service time distribution with an average service time equal to $\frac{1}{\mu}$ seconds. For simplicity, the service time distribution is assumed

to be a negative exponential so that each node can be modeled as an $M/M/1$ service station with (external) arrival rate γ_i , and service rate μ . All of the nodes have the ability to communicate with each other (i.e. send and receive jobs). The communication bus is modeled as an $M/M/1$ service node with an arrival rate of λ_{N+1} jobs per second and average service time of \bar{x} seconds per job. The capacity of the bus is C jobs per second. The input rate of traffic to the bus depends on the load sharing algorithm while the service rate depends on the average transmission time of a job and the bus bandwidth. The network is modeled (see figure (4.1)) as N $M/M/1$ queues feeding a communications server with a fraction of their external inputs, $r_i\gamma_i$. The rest of the external input, $(1-r_i)\gamma_i$ is processed by the node and "exits" the network after completion. The output from the communications server is fed back to the individual nodes. The fractions r_i ($0 \leq r_i \leq 1$, $\sum_i r_i = 1$) divide the shared traffic among the nodes. The object of this chapter is to determine the optimum value of these fractions.

4.2.2) Performance Measures

We wish to provide an analytic expression for an objective measure of network performance with and without processor load sharing. Before any load sharing is done we are concerned with a network of N independent queues. The objective function used is the sum of average delays of all processing nodes weighted by the fraction of the total traffic flow that each node sees. Let T be the average delay of a job in the network, then T can be written [Klei76] as

$$T(\vec{\lambda}) = \sum_{i=1}^N \frac{\lambda_i}{\gamma} T_i(\lambda_i) \quad (4.1)$$

where $\vec{\lambda} = (\lambda_1, \lambda_2, \dots, \lambda_N)$, $\gamma = \sum_{i=1}^N \gamma_i$ is the total input to the network, and $T_i(\lambda_i)$ is the average delay at node i (from section 4.2.1):

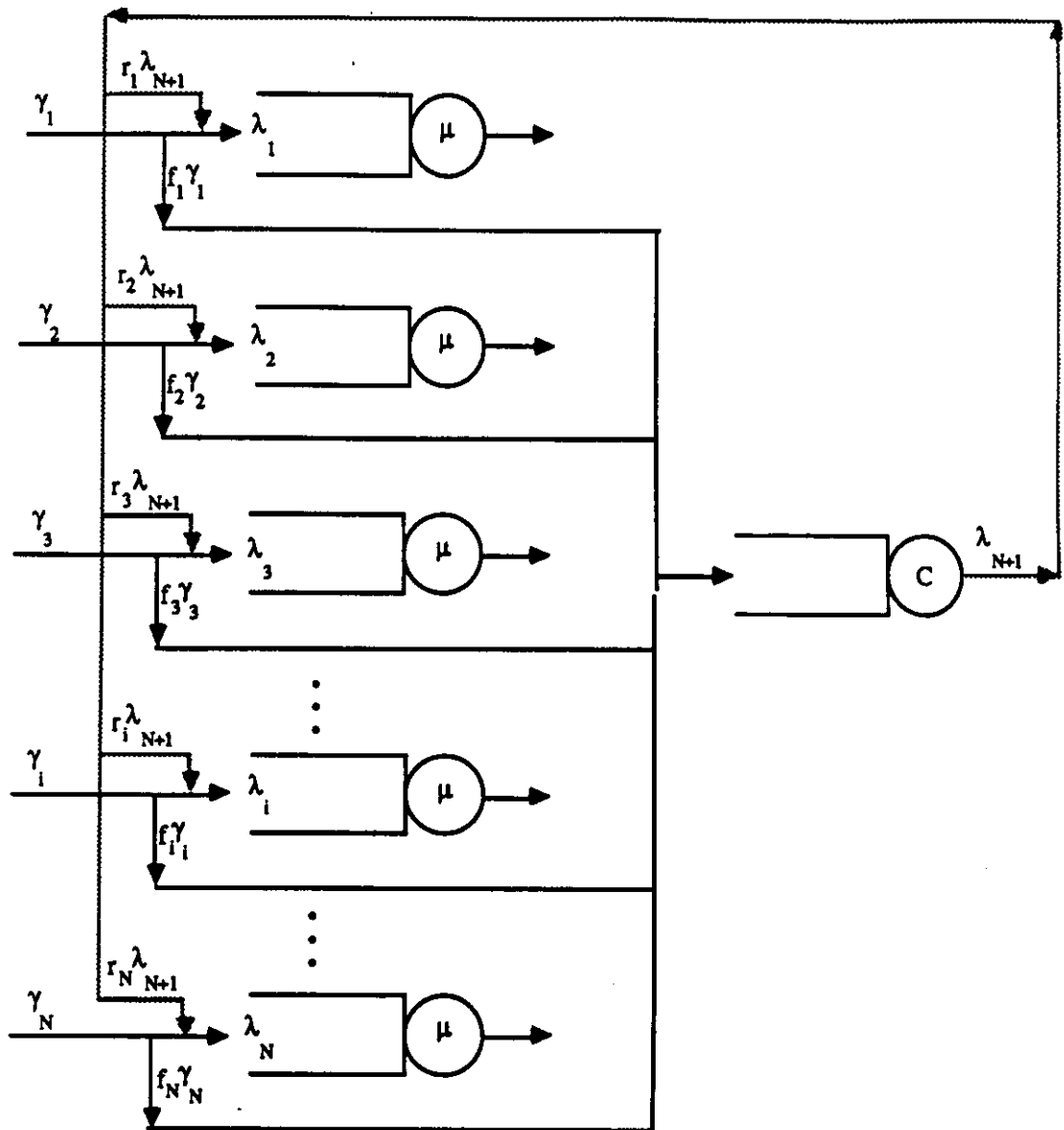


Figure (4.1)
Model of a Broadcast Network

$$T_i(\lambda_i) = \frac{\frac{1}{\mu}}{1 - \frac{\lambda_i}{\mu}} \quad (4.2)$$

At no loss of generality, we assume $\mu=1$ (at all nodes) for the remainder of this work.

After the network redistributes the traffic there are two components to the average job delay. The first component is the processing delay (expressed in the previous equation) with input rate λ_i , the total flow to the processor at node i . The second component is the communication delay experienced by the jobs that are sent from one node to another for remote processing. The amount of traffic that is communicated through the bus is:

$$\lambda_{N+1} = \frac{1}{2} \sum_{i=1}^N |\gamma_i - \lambda_i| \quad (4.3)$$

The difference $|\gamma_i - \lambda_i|$ is the amount of job traffic (sent or received) by node i . The factor of $1/2$ avoids double counting the same traffic that is sent and received. The complete expression for the average delay after load sharing is:

$$T(\vec{\lambda}) = \sum_{i=1}^N \frac{\lambda_i}{\gamma} T_i(\lambda_i) + \frac{\lambda_{N+1}}{\gamma} D(\lambda_{N+1}) \quad (4.4)$$

Where $D(\lambda_{N+1})$ is the average delay for the communication server as a function of the flow through it, λ_{N+1} .

4.2.3) Method for Load Sharing

In this section we describe a method of acquiring information and distributing loads in an efficient manner by taking advantage of the broadcast medium. The objective is not to completely balance the load, but to address the major imbalances in the network. We assume that each node is capable of estimating its average load over a

reasonable period of time. However, due to the randomness of the external input, some nodes may require more time than other nodes to obtain a more accurate estimate of their input rate. For this reason we do not want to use an algorithm (such as TDMA) in which every node must participate (perhaps broadcasting inaccurate information, or none at all). On the other hand, if many nodes wish to participate in a new round of load sharing, we would like to transmit a minimum of information. The amount of information transmitted is examined later in this section and in section (4.5).

The idea is to have a series of rounds in which information is obtained (by participating nodes) and traffic is divided up. In each round the network nodes identify two nodes, namely, the most heavily loaded node and least loaded node. In our model we assume that the average delay function (of the load) is identical at each node. By specifying the load at a node, the delay is also specified. The two nodes identified pair up by evenly dividing their combined loads. That is, the heavily loaded node transmits a fraction of its load to the other node in the pair such that each node has the same amount of traffic to process. The amount of new network traffic (due to this round) is added to the present amount of communication traffic (known at each node, or at a network leader). If the expected decrease in the job processing time outweighs the extra communication delay incurred by the most recent pairing then the algorithm iterates and a new pair is found, otherwise the latest pair breaks up and the load sharing algorithm halts. After running the load sharing algorithm, each node will either send traffic to another node, receive traffic from another node, or not participate and process its external input only.

Finding the minimum and maximum values in a broadcast environment can be done efficiently in terms of the number of bits transmitted. It is not the purpose of this

work to present a new algorithm to accomplish the search. Instead we outline a method that is a distributed version of the binary search. We assume that a large number of nodes wish to participate and that there is a maximum load that any node reports, L_{MAX} .

The search for the greatest (least) load in the network is essentially a search for the maximum (minimum) number in the range $[0, L_{MAX}]$. This problem is similar to finding the packet with the earliest arrival time in a broadcast network [Cape79]. A node or set of nodes that wish to initiate the algorithm (due to varying input rates) synchronize the network by broadcasting a start message over the bus. The start message is followed by a variable number of slots, in which the node with the maximum is determined, followed by the transmission of the maximum value. Each slot serves as a test on a particular subset of the range. A node that is participating in the algorithm broadcasts a one in the current slot if its value is within the range being searched, or remains silent if the value is out of range. All participating nodes listen on the bus for broadcasts from each other. There are three possible results to a test on the current slot. The idle result is when no node broadcasts a one. The successful result is when only one node broadcasts a one, and the collision result is when two or more nodes broadcast a one. We assume that the nodes have circuitry to detect when two or more nodes are broadcasting simultaneously. The algorithm proceeds to the next slot based on the result of the test on the current slot. If the result was a success then we have found the node with the maximum value. The search stops and this distinguished node then broadcasts the actual value over the bus. If the test result was a collision then there are at least two candidates for the maximum. To continue the search, only nodes that broadcast a one (involved in the collision) continue. All other nodes are not candidates and do not participate any more. The current range $[x, y]$ is then halved and the next slot is a test of the top half $[\frac{y+x}{2}, y]$. For the last case (idle

result) assume that the range of the current slot is $[\frac{y+x}{2}, y]$. The idle result means that the search found no candidate in the upper half of some previous range $[x, y]$, so that the search must continue in the lower half. The next slot is then a test of the range $[x, \frac{y+x}{2}]$. The first slot (following the start message) starts the search over the range $[0, L_{MAX}]$. The detailed algorithm running at each node i in the network is presented in figure (4.2). The search algorithm uses a broadcast and listen primitive that is assumed to be provided at a lower level protocol. The primitive is called with a one if a broadcast as well as listening is desired, as opposed to passing a zero for just listening.

The same search procedure, with one modification, can be used to locate the minimum value as well. Instead of first searching the top half a range, the algorithm would first search the lower half. A further refinement is to first search for the first K maxima and minima. This would require the search process to backup and retest ranges that had yielded collision results earlier.

When the searching and pairing phases are finished then the heavier loaded nodes involved in the recent pairings siphon off and transmit the fraction of traffic (calculated earlier) to their buddy nodes. The communication traffic is reduced when a node involved in a pairing notices a change in its external input. If the change is significant then the node suspends the pairing, notifies the network of the drop in communication traffic, and initiates a new round of the load sharing algorithm.

4.2.4) Analysis of the Search Algorithm

Every iteration of the search algorithm cuts the range to be explored by half. At worst $\log(L_{MAX})$ iterations are needed to locate the maximum (we assume that each node has a distinct load value). Each iteration is one message slot with a length of one

```

SEARCH (LOAD:real);

vars
    X,Y: real; /* bottom and top markers respectively */
    RESULT: (SUCCESS, IDLE, COLLISION);
    STATE: (LOW, HIGH, DONE, OUT);

begin
synchronize broadcast network for search;
Y:= MAXLOAD;
X:=0;
RESULT:= BDCST&LSTN(1);
if (RESULT=SUCCESS)
    then STATE:= DONE
    ELSE
        if (LOAD < (X+Y)/2)
            then STATE:= LOW
            else STATE:= HIGH;

while (STATE ^= DONE or OUT) do

    case STATE of

        LOW: Y=(X+Y)/2;
            RESULT:= BDCST&LSTN(0);
            if (RESULT=SUCCESS or COLLISION)
                then STATE:= OUT
                else if (LOAD >= (X+Y)/2)
                    then STATE:= HIGH;

        HIGH: X=(X+Y)/2;
            RESULT:= BDCST&LSTN(1);
            if (RESULT=SUCCESS)
                then STATE:= DONE
                else if (LOAD < (X+Y)/2)
                    then STATE:= LOW;

    end /* case */
end; /* SEARCH */

```

Algorithm for Finding the Maximum Value
Figure (4.2)

bit. Assuming that we would like to locate the first K maxima and minima (K is discussed in section (4.5)), then the time and message complexity is $O(K \log(L_{MAX}))$ bits. The nodes with the maximum and minimum values also broadcast their values which

adds another $O(K \log(L_{MAX}))$ bits for a total that is of the same order. An alternative search procedure is to use TDMA, where each node broadcasts its value if the node has not seen a larger value broadcast, or nothing if it has heard a larger value. In the worst case, a large fraction of the network broadcast their values yielding a time and message complexity of $O(N \log(L_{MAX}))$ bits, where $O(\log(L_{MAX}))$ bits are needed for each broadcast. We will see later that K is of order \sqrt{N} so that the search procedure outlined in the previous section is more efficient than the TDMA solution by a factor of $O(\sqrt{N} \log(L_{MAX}))$ bits.

A cost that was not considered above is the amount of time and bits it requires to synchronize the network. This cost depends upon the size of the starting set of nodes and the lower level protocols that resolve collisions. This cost, however, is inherent in any load sharing algorithm that operates asynchronously (i.e. when needed).

4.3) Network Performance Without Load Sharing

In this section the average delay for a job is derived for a network with no load sharing. We wish to evaluate the network performance without specifying a particular assignment of external input rates, γ_i , to the nodes (i.e. $\lambda_i = \gamma_i$). Therefore, we assume that the vector of inputs, $\vec{\gamma} = (\gamma_1, \gamma_2, \dots, \gamma_N)$ is a vector of random variables, γ , with a distribution $F_{\gamma}(\vec{\gamma}) = P[\gamma \leq \vec{\gamma}]$. For any assignment, $\vec{\gamma}$, to the r.v.'s γ there is an expected delay $T(\vec{\gamma})$. The performance measure that is derived in this section is the expectation of $T(\gamma)$:

$$E(T(\vec{\gamma})) = \int_{\gamma} T(\vec{\gamma}) dF_{\gamma}(\vec{\gamma}) \quad (4.5)$$

We assume that the nodal input rates are independently drawn from the same distribution. Each node is modeled as an M/M/1 system with service rate $\mu=1$. To keep each node stable (without load sharing) the input rate cannot exceed $1-\epsilon$ for some small

$\epsilon > 0$. Without this limitation the expected delay at each node would diverge to infinity making it impossible to derive a meaningful expression for the delay in the network. Thus we assume the input rates to be distributed over the domain $[0, 1-\epsilon]$. In the next section where we consider load sharing, we can do away with ϵ . In the following two subsections we look at the cases for different input rate distributions.

4.3.1) Results for Uniform Distribution

The simplest case is for the uniform distribution of processing load, that is,

$$F_{\gamma}(\vec{\gamma}) = \prod_{i=1}^N F_{\gamma_i}(\gamma_i) = \prod_{i=1}^N \frac{\gamma_i}{1-\epsilon} \quad (4.6)$$

where γ_i is the random variable for the input rate to node i . The density function for the input rate at node i

$$f(\gamma_i) = dF_{\gamma_i}(\gamma_i) = \frac{d\gamma_i}{1-\epsilon} \quad (4.7)$$

is defined only over the domain $0 \leq \gamma_i \leq 1-\epsilon$. The expected value of $T(\vec{\gamma})$, which is denoted by \bar{T} , can be written using equations (4.5), (4.7), and (4.2):

$$\bar{T} = \int_{\gamma_1} \dots \int_{\gamma_N} \left[\sum_{i=1}^N \frac{\gamma_i}{\gamma} \frac{1}{1-\gamma_i} \right] \prod_{j=1}^N \frac{d\gamma_j}{1-\epsilon} \quad (4.8)$$

The expectation integral is intractable due to the $\gamma = \sum_{i=1}^N \gamma_i$ term in the denominator. We approximate the integral by removing γ from the inside of the integral and replacing it with the expected value of the total input rate to the network, $\bar{\gamma}$. This is an excellent approximation for large N , which is verified later in this section when we present the simulation results. This expected value is the sum of the expected input rates at each node, $\bar{\gamma}_i = \frac{(1-\epsilon)}{2}$. The approximate expected delay, which is denoted by \hat{T}_N , is:

$$\hat{T}_N = \frac{2}{N(1-\epsilon)} \left[\int_{\gamma_1} \cdots \int_{\gamma_{N-1}} \prod_{j=1}^{N-1} \frac{d\gamma_j}{1-\epsilon} \left[\int_{\gamma_N} \left(\frac{\gamma_N}{1-\gamma_N} + \sum_{i=1}^{N-1} \frac{\gamma_i}{1-\gamma_i} \right) \frac{d\gamma_N}{1-\epsilon} \right] \right] \quad (4.9)$$

The expression has been rearranged to isolate the delay and uncondition the expression on one random variable, γ_N . Let M_N denote the last term (inside the innermost brackets):

$$M_N = \int_0^{1-\epsilon} \left(\frac{\gamma_N}{1-\gamma_N} + \sum_{i=1}^{N-1} \frac{\gamma_i}{1-\gamma_i} \right) \frac{d\gamma_N}{1-\epsilon} \quad (4.10)$$

so that

$$\hat{T}_N = \frac{2}{N(1-\epsilon)} \left[\int_{\gamma_1} \cdots \int_{\gamma_{N-1}} \prod_{j=1}^{N-1} \frac{d\gamma_j}{1-\epsilon} M_N \right] \quad (4.11)$$

Solving equation (4.10) for M_N , by substituting $y=1-\gamma_N$ yields

$$M_N = \frac{1}{1-\epsilon} \int_{\epsilon}^1 \frac{dy}{y} + \sum_{i=1}^{N-1} \frac{\gamma_i}{1-\gamma_i} = \left[\frac{\ln(\frac{1}{\epsilon})}{1-\epsilon} - 1 \right] + \sum_{i=1}^{N-1} \frac{\gamma_i}{1-\gamma_i}$$

We define M to be M_1 , which is:

$$M = M_1 = \left[\frac{\ln(\frac{1}{\epsilon})}{1-\epsilon} - 1 \right] \quad (4.12)$$

The expected contribution of node N 's delay to the average job delay is summarized in the parenthesized term on the right. As $\epsilon \rightarrow 0$ the expected delay mushrooms to infinity as expected. By substituting the expression for M_N into (4.11) we have

$$\hat{T}_N = \frac{2}{N(1-\epsilon)} \left[\frac{\ln(1/\epsilon)}{1-\epsilon} - 1 + \int_{\gamma_1} \cdots \int_{\gamma_{N-1}} \prod_{j=1}^{N-1} \frac{d\gamma_j}{1-\epsilon} \sum_{i=1}^{N-1} \frac{\gamma_i}{1-\gamma_i} \right] \quad (4.13)$$

There is a simple recursive relationship between \hat{T}_N and \hat{T}_m (for $0 < m \leq N-1$) if we let

$$R_m = \int \cdots \int \prod_{\gamma_j=1}^m \frac{d\gamma_j}{1-\epsilon} \sum_{i=1}^m \frac{\gamma_i}{1-\gamma_i}$$

and rewrite equation (4.13) so that

$$\hat{T}_N = \frac{2}{N(1-\epsilon)} R_N = \frac{2}{N(1-\epsilon)} \left[\left[\frac{\ln(\frac{1}{\epsilon})}{1-\epsilon} - 1 \right] + R_{N-1} \right] \quad (4.14)$$

The solution to the above recursion is

$$R_m = Mm$$

for $(0 \leq m \leq N)$ and the closed form approximation to the expected delay with no load sharing is:

$$\hat{T}_N = \frac{2}{N(1-\epsilon)} R_N = \frac{2}{N(1-\epsilon)} MN = \frac{2}{(1-\epsilon)^2} \left[\ln(\frac{1}{\epsilon}) - (1-\epsilon) \right] \quad (4.15)$$

It is surprising to note that the rightmost equality in the above expression is independent of the number of nodes, N . The omission of N is due to the approximation in equation (4.9) where the sum of input rates was pulled out of the denominator and replaced by the expected sum, $\bar{\gamma}$. The approximation gives the same results as the M/M/1 queue when considering ϵ at 0 or 1.

$$\lim_{\epsilon \rightarrow 1} \hat{T}_N = \lim_{\epsilon \rightarrow 1} \frac{2 \left[1 - \frac{1}{\epsilon} \right]}{-2(1-\epsilon)} = \lim_{\epsilon \rightarrow 1} \frac{1}{\epsilon} = 1$$

We have used L'hospital's rule to obtain the third equality in the above equation. The average delay of 1 agrees with the average delay of an M/M/1 queue with $\lambda=0$ and $\mu=1$. The approximation tends to infinity (like the M/M/1 average delay) when $\epsilon \rightarrow 0$ (when $\lambda \rightarrow \mu$). In the table in figure (4.3), we show how the approximation \hat{T}_N compares to simulation results for different N and ϵ . Each simulation sample consisted of independently drawing N values from a uniform distribution (over $[0, 1-\epsilon]$) The sam-

ple values were used to compute the average job delay, T_N using the M/M/1 delay equation. The results of the simulation show that the approximation, \hat{T}_N upper bounds the expected delay and improves as N gets larger. For small N , the approximation and simulation results differ by 15% . However, as N gets larger (i.e. $N=500$) the difference shrinks to 5.5% . The difference is the result of pulling γ out of the denominator of the expectation integral and replacing it with $\bar{\gamma}$. For large N we note that this approximation is validated by the simulation.

ϵ	N	\hat{T}_N	Simulation	# samples
10^{-2}	10	7.377	7.09	40,000
	50		7.32	40,000
	100		7.35	40,000
	500		7.37	40,000
10^{-3}	10	11.841	11.02	40,000
	50		11.69	40,000
	100		11.74	40,000
	500		11.81	40,000
10^{-4}	10	16.424	15.32	40,000
	50		16.23	40,000
	100		16.20	40,000
	500		16.26	40,000
10^{-5}	10	21.026	20.99	40,000
	50		20.77	40,000
	100		20.79	40,000
	500		20.82	40,000
10^{-6}	10	25.631	22.45	40,000
	50		23.52	40,000
	100		23.26	40,000
	500		24.39	40,000

Comparison of (No Load Sharing) Processing Delay Results with Simulation
Figure (4.3)

The same type of delay analysis can be applied to other systems as well. The waiting time for an M/G/1 system depends on the first two moments of the service distribution, but only on the first moment of the arrival process. From [Klei75], the average system time for M/G/1 is:

$$T(\gamma) = 1 + \frac{\gamma(1+C_b^2)}{2(1-\gamma)}$$

where C_b^2 is the coefficient of variation (standard deviation divided by the mean) of the service distribution. The average delay for a network of M/G/1 nodes (same service distribution at each node) is:

$$\bar{T}_{M/G/1} = \int_{\gamma_1} \cdots \int_{\gamma_N} \sum_{i=1}^N \frac{\gamma_i}{\gamma} \left[1 + \frac{\gamma(1+C_b^2)}{2(1-\gamma)} \right] \prod_{j=1}^N f(\gamma_j) d\gamma_j$$

Using the same methodology as for the M/M/1 case we obtain:

$$\hat{T}_{M/G/1} = \frac{2}{1-\epsilon} \left[\left[\frac{\ln(\frac{1}{\epsilon})}{1-\epsilon} - 1 \right] + \frac{C_b^2-1}{2} \left[\frac{\ln(\frac{1}{\epsilon})}{1-\epsilon} - \frac{1}{2}(3-\epsilon) \right] \right]$$

For the case of the exponentially distributed service times ($C_b^2=1$) the above result is equal to equation (4.15).

The delay expression for M/G/1 models depends only on the first moment of the arrival process. However, the delay expressions for G/M/1 and G/G/1 systems rely on the first as well as higher moments. Our methodology, which is well suited for M/G/1 systems, is questionable for G/M1 and G/G/1 models. It is not clear how the higher order moments of the arrival process are affected when the first moment is selected using a probability distribution. For Poisson arrivals, the second moment is related to the square of the first; however, this is not true for other distributions.

4.3.2) Results for the Beta Distribution

In this section the input rate to a network node is drawn from a Beta distribution, that is:

$$f(\gamma_i) = A \gamma_i^b (1-\gamma_i)^c \quad 0 \leq \gamma_i \leq 1 \quad (4.16)$$

where A is the normalization constant given by

$$A = \frac{\Gamma(b+c+2)}{\Gamma(b+1)\Gamma(c+1)} \quad (4.17)$$

The parameters b, c of the distribution can be any two real numbers. The notation $\Gamma(x)$ denotes the Gamma function applied to x [Abra72]. In the case where x is an integer, then $\Gamma(x) = (x-1)!$

There are two reasons we include the Beta distribution in our analysis. First, many unimodal distributions can be approximated by changing the parameters b , and c . In Figure (4.4) we show some sample Beta distributions with different parameter values. Second, the distribution has no mass for the case $\gamma_i = 1$ (for nonzero c) thus eliminating the need for the ϵ correction. The network delay approximation \hat{T}_N , is

$$\hat{T}_N = \frac{1}{N\bar{\gamma}} \int \cdots \int \sum_{\gamma_N=1}^N \frac{\gamma_i}{1-\gamma_i} \prod_{j=1}^N \left[A \gamma_j^b (1-\gamma_j)^c d\gamma_j \right]$$

where

$$\bar{\gamma} = \int_0^1 A \gamma^b (1-\gamma)^c d\gamma = \frac{b+1}{b+c+2}$$

As in the previous section, we isolate the integral for the N^{th} term.

$$\begin{aligned} M &= \int_0^1 \left[\frac{\gamma_N}{1-\gamma_N} + \sum_{i=1}^{N-1} \frac{\gamma_j}{1-\gamma_j} \right] A \gamma_N^b (1-\gamma_N)^c d\gamma_N \\ &= \sum_{i=1}^{N-1} \frac{\gamma_i}{1-\gamma_i} + A \int_0^1 \gamma_N^{b+1} (1-\gamma_N)^{c-1} d\gamma_N \end{aligned}$$

However, the integral on the right hand side is the inverse of the normalization constant, A' , for a Beta distribution with parameters $b'=b+1$ and $c'=c-1$. From equation (4.17):

$$A' = \frac{\Gamma(b'+c'+2)}{\Gamma(b'+1)\Gamma(c'+1)} = \frac{\Gamma(b+c+2)}{\Gamma(b+2)\Gamma(c)}$$

so that M is now:

$$\begin{aligned} M &= \sum_{i=1}^{N-1} \frac{\gamma_i}{1-\gamma_i} + \frac{\Gamma(b+c+2)}{\Gamma(b+1)\Gamma(c+1)} \frac{\Gamma(b+2)\Gamma(c)}{\Gamma(b+c+2)} \\ &= \sum_{i=1}^{N-1} \frac{\gamma_i}{1-\gamma_i} + \frac{b+1}{c} \end{aligned}$$

We used the following property of the Gamma function [Abra72] to simplify

$$\frac{\Gamma(c)}{\Gamma(c+1)} \text{ and } \frac{\Gamma(b+2)}{\Gamma(b+1)}, \text{ that is,}$$

$$\Gamma(x+1) = x\Gamma(x)$$

After substitution of M the approximate average processing delay expression is:

$$\hat{T}_N = \frac{(b+c+2)}{N(b+1)} \int_{\gamma_1} \cdots \int_{\gamma_{N-1}} \prod_{j=1}^{N-1} f(\gamma_j) \left[\sum_{i=1}^{N-1} \frac{\gamma_i}{1-\gamma_i} + \frac{b+1}{c} \right]$$

which is solved recursively (as in the last section) to obtain:

$$\hat{T}_N = \frac{b+c+2}{N(b+1)} \frac{N(b+1)}{c} = \frac{b+c+2}{c} \quad (4.18)$$

The parameter c is equivalent to the correction factor ϵ used earlier. As c approaches 0, the distribution is zero for all $0 \leq x < 1$, with an impulse of 1 at $x=1$. The average arrival rate then equals (or approaches) the server rate, saturating the server and causing an infinite delay.

4.3.3) Results for the General Distribution

The approximation equation (4.9) can be adjusted for a general distribution for γ_i defined over the domain $[0, 1-\epsilon]$. The network input rate, γ , is removed from the expectation integral and replaced with the expected value, $\bar{\gamma}$. The quantity M (equa-

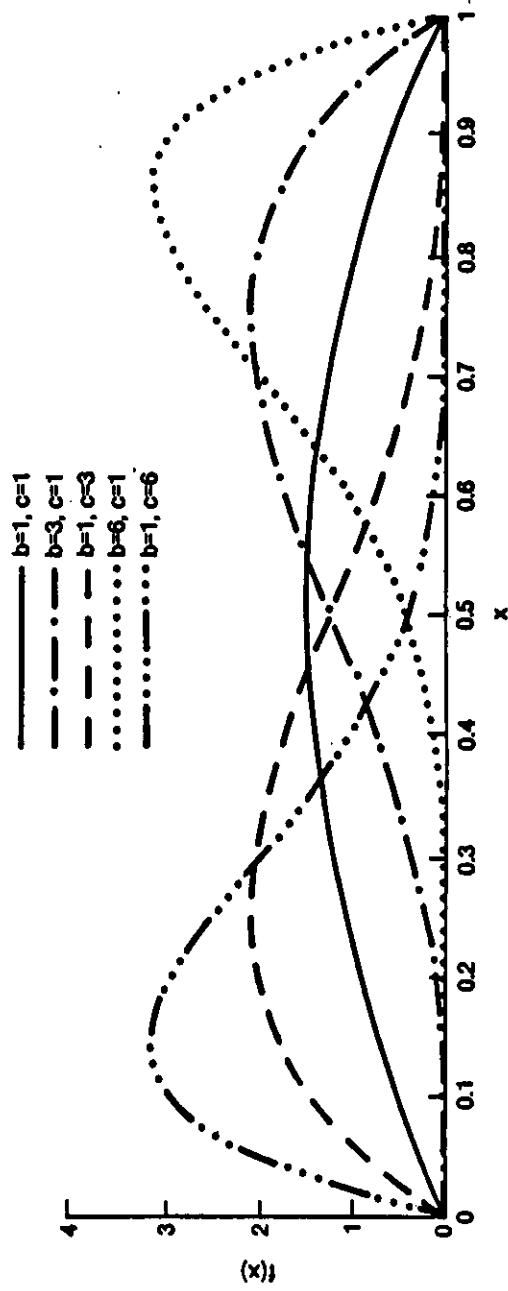


Figure (4.4)
Beta Distribution Function

tion (4.11)) is then calculated for the distribution given. That is, for general $f(\gamma_i)$

$$M = \int_0^{1-\epsilon} \left[\sum_{i=1}^{N-1} \frac{\gamma_i}{1-\gamma_i} + \frac{\gamma_N}{1-\gamma_N} \right] f(\gamma_N) d\gamma_N$$

$$= \sum_{i=1}^{N-1} \frac{\gamma_i}{1-\gamma_i} + \int_0^{1-\epsilon} \sum_{l=1}^{\infty} [\gamma_N]^l f(\gamma_N) d\gamma_N$$

where we have expanded the M/M/1 delay expression at node N into a power series. By moving the integral inside the summation, we see that the M term is composed of the average delays at the other nodes plus the sum of the moments of γ_N .

$$M = \sum_{i=1}^{N-1} \frac{\gamma_i}{1-\gamma_i} + \sum_{l=1}^{\infty} \overline{\gamma_N^l} \quad (4.19)$$

Replacing the expression for M into the expression for \hat{T}_N , and recognizing the same recursive relationship seen in the previous section, we have

$$\hat{T}_N = \frac{1}{N\bar{\gamma}} N \left[\sum_{l=1}^{\infty} \overline{\gamma^l} \right] = 1 + \frac{\overline{\gamma^2}}{\bar{\gamma}} + \frac{\overline{\gamma^3}}{\bar{\gamma}} + \dots \quad (4.20)$$

where $\overline{\gamma^l}$ is the l^{th} moment of the general distribution.

As a check, we can calculate the approximation for the uniform distribution. The l^{th} moment is given by

$$\overline{x^l} = \int_0^{1-\epsilon} \frac{x^l}{(1-\epsilon)} dx = \frac{(1-\epsilon)^{l+1}}{l+1}$$

The approximation, \hat{T}_N is then written as:

$$\hat{T}_N = \frac{1}{\bar{x}} \sum_{l=1}^{\infty} \overline{x^l} = \frac{2}{(1-\epsilon)^2} \sum_{l=1}^{\infty} \frac{(1-\epsilon)^{l+1}}{l+1}$$

$$\begin{aligned}
&= \frac{2}{(1-\epsilon)^2} \left[\sum_{l=0}^{\infty} \frac{(1-\epsilon)^{l+1}}{l+1} - (1-\epsilon) \right] \\
&= \frac{2}{(1-\epsilon)^2} \left[\frac{\ln\left(\frac{1}{\epsilon}\right)}{(1-\epsilon)} - 1 \right]
\end{aligned}$$

which agrees with the result in eq. (4.15). The calculation above uses the series expansion for the natural log, that is, $\ln(1-x) = -\sum_{l=1}^{\infty} \frac{x^l}{l}$.

Although the results derived in the last three sections do not include any load sharing, there are two reasons for presenting the results. The first is to present and justify our approximation approach. The approach derives the expected job delay in a network with input rates that are drawn from a common distribution. The second reason is to compare the network performance with and without load sharing. In the next section, we extend our approach to obtain closed form expressions for the delay in a load sharing environment.

4.4) Results for Broadcast Networks With Load Sharing

The method (and algorithm) for load sharing was described in section (4.2). In brief, the network matches up the node with the maximum delay with the node that has the minimum delay. This pairing process continues for K iterations until the communication delay incurred from the last pairing exceeds the amount of time saved in the processing delay due to the benefits of load sharing. In this section we obtain an expression for the expected processing delay given K pairings ($K=1,2,\dots,n/2$). We use the uniform distribution for the nodal input rates for the remainder of this chapter. Although we were able to find an expression for \hat{T}_N using a beta distribution, we could not do so in the case of load sharing. This will become apparent when we derive the

results using the uniform distribution.

As the algorithm proceeds and pairs up more nodes, the processing delay can be written as the sum of two components. The first component is the delay contributed by the nodes that have not been paired up yet. The second component involves the nodes that are already sharing their loads. If the number of network nodes, N , is very large then we can expect that the maxima identified through the K^{th} round had values of load very close to $1-\epsilon$, while the minima had values close to 0. Each pairing yields two nodes each having input rates that are roughly $\frac{1}{2}$, and average delays of 2 seconds. This second component is weighted by $2K/N$, the fraction of nodes paired up. For small K , the weighted second component is close to 0, and completely dominated by the weighted first component. For the next few sections we are concerned only with small K and therefore we ignore the second component entirely. The problem for this section is to obtain an expression for the first component.

After K pairings the network will have $N-2K$ remaining unpaired nodes with input rates distributed uniformly over $[0, x_K]$, where x_K is the value of the K^{th} maximum. Our first attempt at calculating the new network delay is to use the value x_K in our previous delay expression (4.15) as an upper bound to the range, and then to uncondition that delay based on the density for the K^{th} maximum, $f_{K:N}(x)$. First we calculate the cumulative distribution of the K^{th} maximum.

$$\begin{aligned}
 F_{K:N}(x) &= P(X_{K:N} \leq x) \\
 &= \sum_{i=N-K+1}^N \binom{N}{i} \left[\frac{x}{1-\epsilon} \right]^i \left[1 - \frac{x}{1-\epsilon} \right]^{N-i}
 \end{aligned} \tag{4.21}$$

This equation results from observing that the K^{th} maximum, $X_{K:N}$, will obey $X_{K:N} \leq x$ only if at least $N-K+1$ random variables have values that are less than or equal to x .

The probability that i uniformly distributed r.v.'s ($N-K+1 \leq i \leq N$) each have values $\leq x$ is $\left(\frac{x}{1-\epsilon}\right)^i \left(1 - \frac{x}{1-\epsilon}\right)^{N-i}$ multiplied by the number of ways of choosing i out of N random variables. The approximation, \hat{T}_N , can be modified by changing the range of the integrals from $[0, 1-\epsilon]$ to $[0, X_{K:N}]$.

$$\hat{T}_{N|X_{K:N}} = \frac{2}{X_{K:N}} \left[\frac{\ln \left[\frac{1}{1-X_{K:N}} \right]}{X_{K:N}} - 1 \right] \quad (4.22)$$

For large N , and small K , the values of the first K minima will be small enough that the lower bound of the range will be kept at 0. The new approximation is expressed by unconditioning $\hat{T}_{N|X_K}$ using the density for the K^{th} maximum (derivative of equation (4.21)):

$$\hat{T}_{N-2K} = \int_0^{1-\epsilon} \frac{2}{x} \left[\frac{\ln \left[\frac{1}{1-x} \right]}{x} - 1 \right] \sum_{i=N-K+1}^N \binom{N}{i} \left(\frac{x}{1-\epsilon} \right)^{i-1} \left(1 - \frac{x}{1-\epsilon} \right)^{N-i-1} \left[i - \frac{Nx}{1-\epsilon} \right] \frac{dx}{1-\epsilon} \quad (4.23)$$

The above expression is intractable due to the presence of the logarithm and hypergeometric terms. Instead of solving the integral exactly, we now derive a number of approximations that yield successively better results when compared to simulations. The first approximation to \hat{T}_{N-2K} is to assume that the input rates to the $N-2K$ nodes are uniformly distributed over the domain $[\mu_{N-K+1:N}, \mu_{K:N}]$, where $\mu_{R:N}$ is the expected value of the R^{th} maximum out of N random variables. The mean value for $\mu_{R:N}$ can be easily derived or found in [Davi70]

$$\mu_{R:N} = \int_0^{1-\epsilon} x dF_{R:N}(x) dx = \frac{N-R+1}{N+1} (1-\epsilon) \quad (4.24)$$

Thus, the expected network input rate for $N-2K$ nodes is

$$\bar{\gamma} = (N-2K) \frac{[\mu_{K:N} - \mu_{N-K+1:N}]}{2} = \frac{(N-2K)(1-\epsilon)}{2}$$

The probability density for the input rate to a node is uniform over the new domain $[\mu_{N-K+1:N}, \mu_{K:N}]$

$$\frac{d\gamma_i}{\mu_{K:N} - \mu_{N-K+1:N}} = \frac{d\gamma_i}{1-\epsilon} \left[\frac{N+1}{N-2K+1} \right] \quad (4.25)$$

Using the new expressions for the integration bounds, the density, and the network input rate in equation (4.9) we have $\hat{T}_{N-2K}^{(1)}$, our first approximation to \hat{T}_{N-2K} :

$$\hat{T}_{N-2K}^{(1)} = \frac{2}{(1-\epsilon)} \left[\frac{\ln \left[\frac{N+1-K(1-\epsilon)}{(N+1)\epsilon + K(1-\epsilon)} \right]}{\left[\frac{N-2K+1}{N+1} \right] (1-\epsilon)} - 1 \right] \quad (4.26)$$

In contrast to equation (4.15), the approximation here depends on N and also K . Even as N gets very large the logarithmic term, which dominates the expression, remains dependent on N , and K . In the table in figure (4.6) we show values of $\hat{T}_{N-2K}^{(1)}$ for some representative values of N, K , and $\epsilon=10^{-6}$. The table includes simulation results for the same parameter values. In each simulation sample, the first K maxima and minima (out of N r.v.'s) were identified and paired. The simulation delay includes the delay incurred by locally processed jobs, as well as shared jobs. Note that differences between simulation results and $\hat{T}_{N-2K}^{(1)}$ now differ by $\approx 10\%$ even as N gets very large ($N=500$). Another problem is that the approximation is no longer an upper bound to the simulation results as was the case in section (4.3.1). The reason for the poor behavior is that

using $\mu_{N-K+1:N}$, $\mu_{K:N}$ for the integration bounds is too aggressive. There is a small, but not insignificant, probability that one of the remaining $N-2K$ nodes has a value that is larger than $\mu_{K:N}$. This value will be a large component of the delay calculated by the simulation. However, the approximation, $\hat{T}_{N-2K}^{(1)}$, does not consider this case at all, thus underestimating the expected delay. Herein lies the solution as well! Let us again consider the remaining $N-2K$ random variables. Instead of using a uniform distribution, as before, the r.v.'s will be drawn from a new distribution that gives decreasing weight to the upper end of the domain where the K^{th} maximum is expected to be. The domain of the new distribution will be $[0, 1-\epsilon]$. At the low end, γ_i will be distributed uniformly, however, at the high end the density will drop off towards 0 reflecting the probability that γ_i is greater than $\mu_{K:N}$.

$$f_{\text{new}}(x) = G^{-1} f(x) P[\mu_{K:N} > x]$$

where G^{-1} is a normalizing constant. Indeed, when x is small then $P[\mu_{K:N} > x]$ is nearly one, and x is distributed as before. As x increases, the probability $P[\mu_{K:N} > x]$ decreases slightly until a point x_0 where $P[\mu_{K:N} > x]$ drops precipitously. Thus, γ_i is distributed even at the high end of the domain, but with a much smaller probability than at the lower and mid range. The probability, $P[\mu_{K:N} > x]$ can be determined from equation (4.21) and the complete expression for the new distribution is:

$$f_{\text{new}}(x) = \frac{\frac{dx}{1-\epsilon} \sum_{i=K}^N \binom{N}{i} \left[\frac{x}{1-\epsilon} \right]^{N-i} \left[1 - \frac{x}{1-\epsilon} \right]^i}{\int_0^{1-\epsilon} \sum_{j=K}^N \binom{N}{j} \left[\frac{x}{1-\epsilon} \right]^{N-j} \left[1 - \frac{x}{1-\epsilon} \right]^j \frac{dx}{1-\epsilon}} \quad (4.27)$$

The above distribution reflects the pairing of nodes by decreasing the probability mass at the high end of the range. Our new expression does not take into account the nodes with the minimum values that were removed (by the pairing). A more realistic expression would also include a decrease in the probability mass near the low end of the dis-

tribution range. However, the low end does not contribute much at all to the delay, so that neglecting it does not affect the approximation much. In figure (4.5) we have plotted the new distribution in equation (4.27) for some values of K , the number of pairings. It is evident from the figure that the distribution falls off earlier for larger K .

Note that the equation in the denominator of equation (4.27) is equal to $\frac{N-K+1}{N+1}$ for any distribution $f(x)$. The result is easy to see after changing the order of summation and integration, and recognizing the normalization constant for the Beta function, viz.,

$$\begin{aligned}
 \int_0^{1-\epsilon} f(x) P[\mu_{K:N} > x] &= \int_x^1 f(x) \sum_{j=K}^N \binom{N}{j} F(x)^{N-j} (1-F(x))^j dx \\
 &= \sum_{j=K}^N \binom{N}{j} \int_x^1 F(x)^{N-j} (1-F(x))^j dF(x) \\
 &= \sum_{j=K}^N \binom{N}{j} \left[\frac{\Gamma(N-j+1)\Gamma(j+1)}{\Gamma(N+2)} \right] \\
 &= \frac{N-K+1}{N+1}
 \end{aligned} \tag{4.28}$$

To derive the last equality we used the relationship $\Gamma(I+1) = I!$ when I is an integer. Now let us modify equation (4.9) taking into account the new distribution to obtain $\hat{T}_{N-2K}^{(2)}$, our second approximation to \hat{T}_{N-2K} .

$$\hat{T}_{N-2K}^{(2)} = \frac{1}{\gamma_{N-2K}} \left[\int_0^{1-\epsilon} \cdots \int_0^{1-\epsilon_{N-2K}} \prod_{i=1}^{N-2K} f_{new}(x_i) \sum_{j=1}^{N-2K} \frac{x_j}{1-x_j} \right] \tag{4.29}$$

Following the technique we have used before, terms relating to the last node (r.v.) are moved to the right to be computed first.

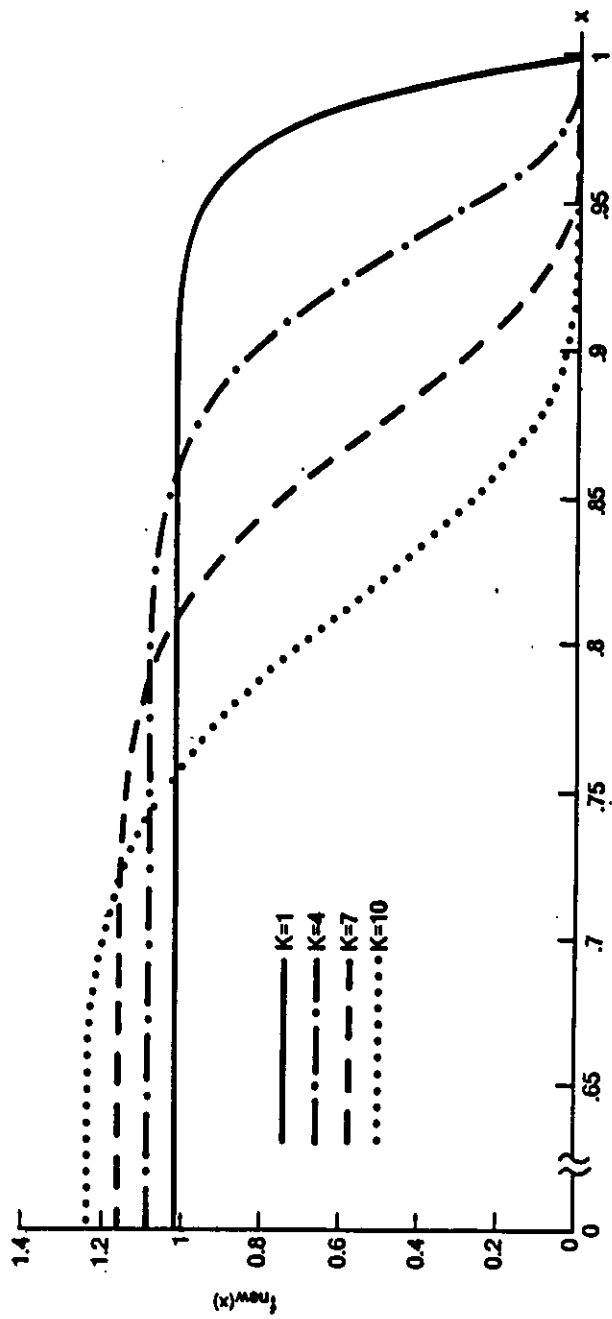


Figure (4.5)
Sample Distribution After Pairing

$$\hat{T}_{N-2K}^{(2)} = \frac{1}{\gamma_{N-2K}} \left[\int_0^{1-\epsilon} \cdots \int_0^{1-\epsilon_{N-2K-1}} \prod_{i=1}^{N-2K-1} f_{new}(x_i) M_{N-2K} \right] \quad (4.30)$$

where M_{N-2K} is the rightmost integral.

$$\begin{aligned} M_{N-2K} &= \int_0^{1-\epsilon} \left[\sum_{j=1}^{N-2K-1} \frac{x_j}{1-x_j} + \frac{x_{N-2K}}{1-x_{N-2K}} \right] f_{new}(x_{N-2K}) \\ &= \sum_{j=1}^{N-2K-1} \frac{x_j}{1-x_j} + \int_0^{1-\epsilon} \left[\frac{x}{1-x} \right] \frac{\sum_{i=K}^N \binom{N}{i} \left[\frac{x}{1-\epsilon} \right]^{N-i} \left[1 - \frac{x}{1-\epsilon} \right]^i \frac{dx}{1-\epsilon}}{\frac{N-K+1}{N+1}} \end{aligned}$$

At this point, the artificial parameter ϵ can be done away with. There will be no problem of infinite delays as $x \rightarrow 1$ since the denominator of the delay will cancel with one of the geometric terms in the numerator. The integral and summation can be exchanged in order of evaluation. We then solve the integral by parts over the domain $[0, 1]$ which yields:

$$\begin{aligned} M_{N-2K} &= \sum_{j=1}^{N-2K-1} \frac{x_j}{1-x_j} + \frac{\sum_{i=K}^N \binom{N}{i} \frac{(i-1)! (N-i+1)!}{(N+1)!}}{\frac{N-K+1}{N+1}} \\ &= \sum_{j=1}^{N-2K-1} \frac{x_j}{1-x_j} + \frac{\sum_{i=K}^N \frac{N-i+1}{i}}{N-K+1} \\ &= \left[\frac{1}{N-K+1} \right] \left[(N+1) \sum_{i=K}^N \frac{1}{i} - (N-K+1) \right] \quad (4.31) \end{aligned}$$

The expression for M_{N-2K} is always positive, which can be proven by showing the bracketed term above is always positive as follows.

$$\begin{aligned}
(N+1) \sum_{i=K}^N \frac{1}{i} &\geq (N+1) \sum_{i=K}^N \frac{1}{N} \\
&= \frac{(N+1)(N-K+1)}{N} \\
&> N-K+1
\end{aligned}$$

The expected network input rate, $\bar{\gamma}_{N-2K}$, is now

$$\bar{\gamma}_{N-2K} = (N-2K) \int_0^1 x f_{new}(x) = \frac{(N-2K)}{2} \left[\frac{N-2K+2}{N+2} \right] \quad (4.32)$$

By placing $\bar{\gamma}_{N-2K}$ (equation (4.32)), and M_{N-2K} (equation (4.31)) into equation (4.30), and recognizing the recursive structure as before, we obtain:

$$\hat{T}_{N-2K}^{(2)} = \left[\frac{2(N+2)}{N-2K+2} \right] \frac{\left[(N+1) \sum_{i=K}^N 1/i - (N-K+1) \right]}{N-K+1} \quad (4.33)$$

The above approximation can be simplified (and made even more accurate, surprisingly) by using the old expression for the expected input rate (i.e. $\bar{\gamma}_{N-2K} = (N-2K)/2$, for $\epsilon=0$) rather than the expression in equation (4.32); this yields our third approximation, $\hat{T}_{N-2K}^{(3)}$, to \hat{T}_{N-2K} .

$$\hat{T}_{N-2K}^{(3)} = \frac{2 \left[(N+1) \sum_{i=K}^N 1/i - (N-K+1) \right]}{N-K+1} \quad (4.34)$$

Figure (4.6) compares the last two approximations with $\hat{T}_{N-2K}^{(1)}$ and the simulation results. The results show that $\hat{T}_{N-2K}^{(2)}$, and $\hat{T}_{N-2K}^{(3)}$ are much better approximations than $\hat{T}_{N-2K}^{(1)}$, and they also bound the simulation results from above. The last approximation, $\hat{T}_{N-2K}^{(3)}$, is much tighter than $\hat{T}_{N-2K}^{(2)}$. Indeed, $\hat{T}_{N-2K}^{(3)}$ differs from the simulation

results by less than 1% when $N=500$ and $K=1,2,3,4$. This last observation is surprising considering that $\hat{T}_{N-2K}^{(2)}$ uses the new distribution $f_{new}(x)$ to calculate $\bar{\gamma}_{N-2K}$, whereas $\hat{T}_{N-2K}^{(3)}$ uses the old (uniform) distribution. The reason is that the network input rate (used for weighting the nodal delay) does not change after the nodes pair up for load sharing; i.e. the input rate, $\bar{\gamma}$ remains at $N/2$. $\hat{T}_{N-2K}^{(3)}$ is consistent with $\bar{\gamma} = N/2$, whereas $\hat{T}_{N-2K}^{(2)}$ is not.

The above approximations do not include the average delay at the $2K$ nodes that are sharing load. If we scale $\hat{T}_{N-2K}^{(3)}$ by $(N-2K)/N$ (i.e. $\bar{\gamma}=N/2$ not $(N-2K)/2$) and add the weighted delay at the load sharing nodes, $2 \left[\frac{2K}{N} \right]$ the result is, our fourth and final approximation to \hat{T}_{N-2K} :

$$\hat{T}_{N-2K}^{(4)} = \frac{4K}{N} + \frac{2(N-2K) \left[(N+1) \sum_{i=K}^N 1/i - (N-K+1) \right]}{N(N-K+1)} \quad (4.35)$$

Looking at Figure (4.6), the scaling up of the network input rate decreases the expected delay yielding an approximation that is close to the simulation results. We have derived two close approximations $\hat{T}_{N-2K}^{(3)}$, and $\hat{T}_{N-2K}^{(4)}$ such that both approximations are within 3% of the simulation results. It is interesting to note that $\hat{T}_{N-2K}^{(3)}$ is consistently greater than the simulation results for all N and K . On the other hand, $\hat{T}_{N-2K}^{(4)}$, for a given number of nodes N , starts off (at $K = 1$) greater than the corresponding simulation results and then dips below the simulation results as K increases. The reason for this behavior is that for small K , the second component of $\hat{T}_{N-2K}^{(4)}$ has a weight factor close to 1. This second component is identical to $\hat{T}_{N-2K}^{(3)}$ and is greater than the simulation results. As K increases the first component ($4K/N$), which is negligible, gets more weight. Since the first component is an underestimation (though not by much) of the actual delay at the "balanced" nodes, the overall approximation will be

N	K	$\hat{T}_{N-2K}^{(1)}$	$\hat{T}_{N-2K}^{(2)}$	$\hat{T}_{N-2K}^{(3)}$	$\hat{T}_{N-2K}^{(4)}$	Simulation
10	0	25.631	25.631	25.631	25.631	22.45
	1	3.629	5.332	4.444	3.955	3.98
	2	2.727	4.073	2.715	2.429	2.69
50	0	25.631	25.631	25.631	25.631	23.52
	1	6.143	7.466	7.178	6.971	6.96
	2	4.942	5.724	5.284	5.021	5.12
	3	4.285	4.944	4.373	4.089	4.25
100	0	25.631	25.631	25.631	25.631	23.26
	1	7.396	8.648	8.479	8.349	8.31
	2	6.126	6.811	6.544	6.362	6.42
	3	5.413	5.951	5.601	5.384	5.49
150	0	25.631	25.631	25.631	25.631	-
	1	8.155	9.380	9.257	9.160	9.13
	2	6.856	7.503	7.306	7.164	7.20
	3	6.120	6.609	6.348	6.174	6.25
500	0	25.631	25.631	25.631	25.631	24.39
	1	10.478	11.659	11.613	11.574	11.51
	2	9.127	9.709	9.632	9.571	9.56
	3	8.348	8.754	8.649	8.570	8.59
	4	7.801	8.128	7.999	7.903	7.94

Comparison of Load Sharing Processing Time With Simulation Results
 $\epsilon=10^{-6}$; 40,000 samples per simulation run
Figure (4.6)

smaller than the simulation results.

It is difficult to generalize the results obtained so far with the same techniques. As an example, we can try to approximate the expected job delay when the input rates are taken from a general distribution. The starting point is equation (4.29). The network input rate will be approximated by $N\bar{\gamma}$. The normalizing constant in the new distribution $f_{new}(x)$ is equal to $(N+1)/(N-K+1)$ for any general distribution. The new (after load sharing) distribution for the remaining nodes is:

$$f_{new}(x) = \frac{f(x) \sum_{i=K}^N \binom{N}{i} (1-F(x))^i F(x)^{N-i}}{\frac{N-K+1}{N+1}} \quad (4.36)$$

The calculation of M_{N-2K} reduces to

$$M_{N-2K} = \sum_{j=1}^{N-2K-1} \frac{\gamma_j}{1-\gamma_j} + \int_0^{1-\epsilon} \left[\frac{x}{1-x} \right] \frac{\sum_{i=K}^N \binom{N}{i} (F(x))^{N-i} (1-F(x))^i f(x) dx}{\frac{N-K+1}{N+1}}$$

Expanding the M/M/1 delay expression into a power series and exchanging the order of summation and integration, yields an extremely difficult equation to solve except for the simplest distributions.

As was done for the no load sharing network, we can replace the M/M/1 delay expression with the M/G/1 expression. The approximate network delay is:

$$\hat{T}_{N-2K}^{M/G/1} = \frac{1}{\gamma_{N-2K}} \left[\int_0^1 \cdots \int_0^{1-N-2K} \prod_{i=1}^{N-2K} f_{new}(x_i) \sum_{j=1}^{N-2K} \left[1 + \frac{x_j(1+C_b^2)}{2(1-x_j)} \right] \right]$$

In Appendix 1 we show that the solution to the above equation is:

$$\hat{T}_{N-2K}^{M/G/1} = 2 \frac{(N+1) \sum_{j=K}^N 1/j}{N-K+1} - 2 + \frac{C_b^2 - 1}{(N-K+1)(N+2)} \left[(N-K+1)(2N+3) + \frac{N(N-1)}{2} - \frac{K(K-1)}{2} + (N+2)(N+1) \sum_{j=K}^N 1/j \right]$$

In this section we derived several approximations to the expected network delay in a load sharing environment. The last two approximations yielded very tight results compared to the simulation results. In the next section, we use these approximation expressions to study the performance of the network as a function of N, K , and

the communication delay of a "shared" job.

4.5) Communication Costs and Optimal Number of Pairs

In the last three sections, we developed a number of tight approximations to the average processing time in networks with and without load sharing. Some sample values of the approximations are plotted in figure (4.7) as functions of K , the number of pairs in the network. The plots show a dramatic decrease in the average delay even for small K . The initial steep decrease justifies the intuition, mentioned in the introduction, that motivated this work. That is, that the average job delay is dominated by a few nodes that have very high utilization. By locating these few "trouble makers" and reducing their input traffic, the average processing delay can be drastically improved without too much effort. This result corroborates one of the findings in [Eage84] that even a small amount of load sharing can drastically improve performance. Successive increases in K continue to decrease the average processing time but in decreasing amounts. Eventually, the processing delay will decrease to a value of 2, which is the M/M/1 delay at each node with an input rate of $\lambda_i = \frac{1}{2}$ for a symmetric density, $f(x)$, of the input rate x .

In this section, we complete our study of the network performance by finding bounds (from above and below) to the decreases in the delay as the amount of load sharing increases. We also introduce a simple model for communication cost and show a tradeoff between decreasing average processing time and increasing communication time. Finally, we shall find the number of pairs in the network, K^* , that optimizes the sum of the processing and the communication time.

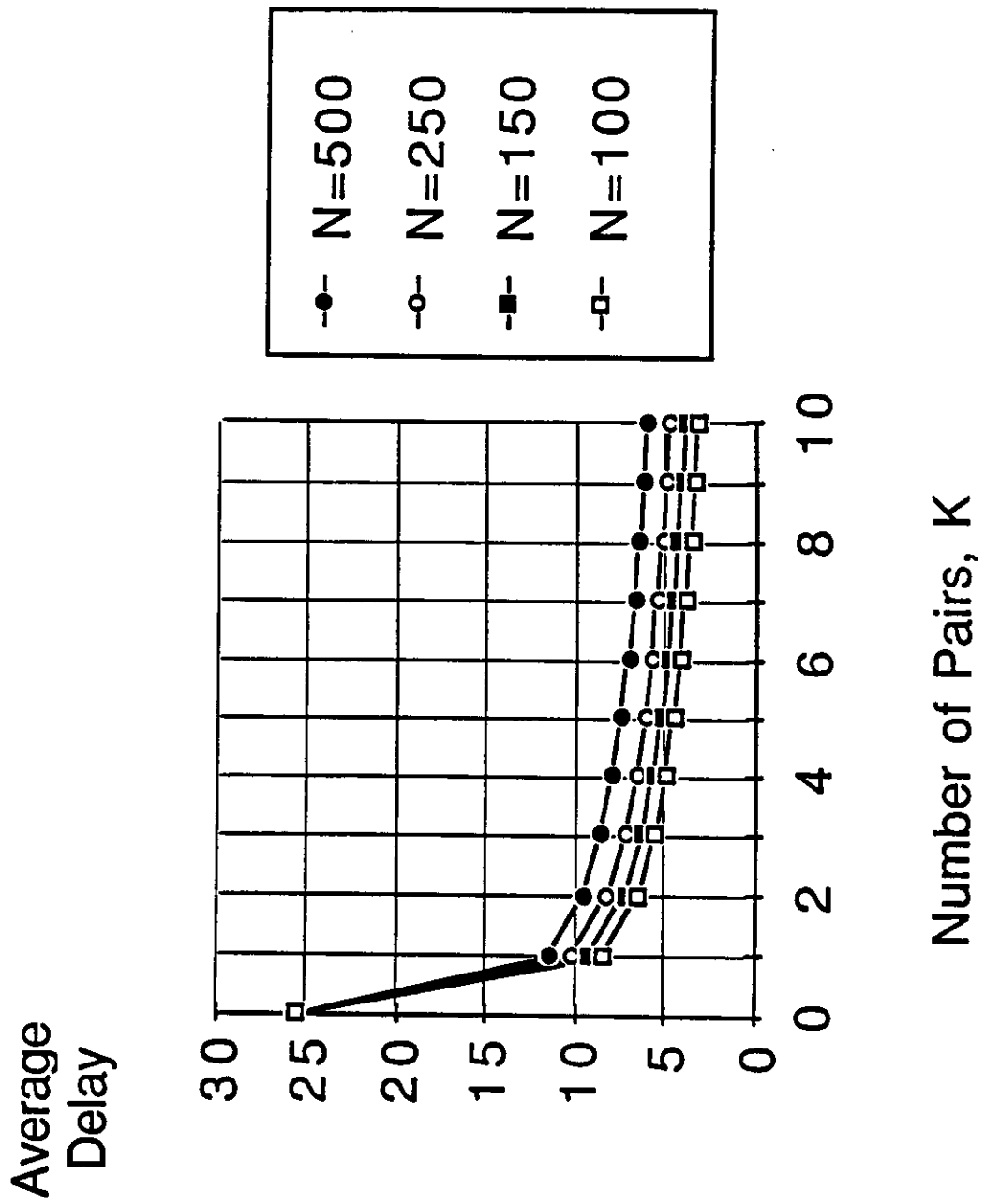


Figure (4.7)
 Approximation of Average Processing Delay and Simulation Results

The model presented here is an elaboration of the bus model that was briefly discussed in section (4.2). The communication media is a one hop multi-access broadcast channel. Assume that the load balancing traffic is a major part of the overall communications traffic being sent over the channel. When K increases, the extra load sharing traffic increases the response time of the system. We model the communications server as an $M/M/1$ node. Thus the average delay of the channel, D_K , given a flow of f_K jobs per second, is $D_K = \frac{1}{C-f_K}$ where C is the capacity of the communications channel in jobs per second. The fraction of jobs that suffer the extra communication delay is f_K/γ , so that the average system time, $G(N,K)$, for a job in the network is:

$$G(N,K) = \hat{T}_{N-2K} + \frac{f_K}{\gamma} D_K \quad (4.37)$$

when there are $2K$ nodes paired up. The object of this section is to find K^* , the optimum number of pairs that minimizes $G(N,K)$.

First, we look at the behavior of \hat{T}_{N-2K} as a function of K , and obtain expressions that bound the decrease in \hat{T}_{N-2K} (for increasing K) from above and below.

Let $\nabla_K T = \hat{T}_{N-2(K+1)} - \hat{T}_{N-2K}$ be the difference in processing time between two consecutive values of K . We use the expression for $\hat{T}_{N-2K}^{(3)}$ to start with; the reason is that $\hat{T}_{N-2K}^{(3)}$ is both very tight with respect to the simulation results, and is consistently greater than the simulation results. The approximation $\hat{T}_{N-2K}^{(3)}$ acts as an "upper bound" to the actual processing delay (as reflected in the simulation results) although we have no proof for such an assertion.

$$\nabla_K T = 2 \left[\frac{(N+1)}{N-K} \sum_{j=K+1}^N 1/j - 1 \right] - 2 \left[\frac{(N+1)}{N-K+1} \sum_{j=K}^N 1/j - 1 \right]$$

$$\begin{aligned}
&= \frac{2(N+1)}{(N-K)(N-K+1)} \left[(N-K+1) \sum_{j=K+1}^N 1/j - (N-K) \sum_{j=K}^N 1/j \right] \\
&= \frac{2(N+1)}{(N-K)(N-K+1)} \left[\sum_{j=K+1}^N 1/j - (N-K) \frac{1}{K} \right] \tag{4.38}
\end{aligned}$$

The difference, $\nabla_K T$, can be bounded from above by replacing $1/j$ in the summation with the first (largest) term, $1/(K+1)$. We will use $O(\nabla_K T)$ to denote the upper bound.

$$\begin{aligned}
\nabla_K T \leq O(\nabla_K T) &= \frac{2(N+1)}{(N-K)(N-K+1)} \left[\frac{(N-K)}{K+1} - \frac{(N-K)}{K} \right] \\
&= \frac{-2N}{(N-K)K(K+1)} \quad K \ll N \tag{4.39}
\end{aligned}$$

where N has replaced $N+1$ under the assumption that $N \gg 1$. Similarly, the $1/j$ term can be replaced by the last (smallest) term, $1/N$, to obtain the lower bound denoted by $\Omega(\nabla_K T)$.

$$\nabla_K T \geq \Omega(\nabla_K T) = \frac{2(N+1)}{(N-K)(N-K+1)} \left[\frac{(N-K)}{N} - \frac{(N-K)}{K} \right] \approx \frac{-2}{K} \quad K \ll N \tag{4.40}$$

where $N \gg 1$ has been assumed again. The above bounds formally describe the descent property of the processing delay. In figure (4.8) the two bounds for $\nabla_K T$ are plotted along with the exact results. From the figure, it is evident that the lower bound, in particular, is very tight. The close fit of $\Omega(\nabla_K T)$ is seen by noting that $1/j$ quickly approaches $1/N$ (in order of magnitude) as j increases from $K+1$ to N . We can obtain the same expression as in equation (4.40) by considering K to be a continuous variable.

When $N \gg K$ then

$$\frac{\partial \hat{T}_{N-2K}}{\partial K} \approx \frac{2 \partial \ln(N/K)}{\partial K} = \frac{-2}{K}$$

In order to analyze the communication delay as a function of K , we first have to find an expression for the traffic, f_K . Let $F_{K:N}(x)$ be the CDF (Cumulative Distribution Function) for the K^{th} maximum out of N random variables. The average communication flow is approximately half of the expected sum of the first K maxima (ignoring the small load contributed by the K minima). When the N random variables are identically and independently drawn from a uniform distribution, we then have:

$$f_K = \frac{1}{2} \sum_{i=1}^K \int_0^1 x dF_{i:N}(x)$$

Using equation (4.24), we have:

$$\begin{aligned} f_K &= \frac{1}{2} \sum_{i=1}^K \frac{N-(i-1)}{N+1} = \frac{NK - \sum_{i=0}^{K-1} i}{2(N+1)} \\ &= \frac{K [2N-K+1]}{4(N+1)} \end{aligned} \tag{4.41}$$

Assuming $N \gg 1$, we have $f_K \approx \frac{K[2N-K]}{4N}$. The above expression appears unitless; however, the upper limit of the integral is 1 *job per second*, so that f_K has units of jobs per second. The weighted communication cost of load sharing is therefore:

$$\frac{f_K}{\gamma} D_K = \frac{1}{\gamma} \frac{K[2N-K]}{4N} \frac{1}{C - \frac{K[2N-K]}{4N}}$$

Since both f_K and D_K are positive and increasing in K , then $f_K D_K$ is also positive and increasing in K .

We now assume that $N \gg K$ to see what value of K optimizes the total average job delay. Our point is to study large networks (large N) and see what effect a little

load sharing (small K) has on the average delay. We can therefore approximate the weighted communication cost as:

$$\frac{f_K}{\gamma} D_K = \frac{1}{\gamma} \frac{K/2}{C-K/2}$$

Note that K must be less than $2C$ in order for the channel to be stable. If K is considered to be a continuous variable then we can express the marginal increase in weighted communication cost as:

$$\frac{1}{\gamma} \frac{\partial f_K D_K}{\partial K} = \frac{C}{2\gamma(C-K/2)^2} \quad (4.42)$$

The increase in communication delay for an increment in K is positive as expected. Returning to equation (4.37), both terms on the right hand side have been analyzed with respect to K . Thus if the system starts out with $K=1$ (we assume that the initial drop in processing delay in going from $K=0$ to $K=1$ will dominate any communications cost) and keep increasing K by 1, we would expect to reach a point at which the increase in communication delay would outweigh the decrease in processing delay. Network managers would be interested in the number of pairs, K^* , that yields a minimum system delay. The point we are looking for is the minimum K such that $G(N, K+1) > G(N, K)$. For that K , the marginal cost of communication delay ($\nabla D_K f_K$) is greater than the marginal savings in the processing delay ($\nabla_K T$). Also, since $\nabla D_K f_K$ is increasing in K and $\nabla_K T$ is decreasing in K , the total cost $G(N, K')$ will only increase for $K' > K^*$ (for large $N \gg K$). Thus, $G(N, K^*)$ is a global minimum. Under the assumption that $N \gg K$ the first derivative of \hat{T}_{N-2K} is $\frac{-2}{K}$, thus the second derivative of \hat{T}_{N-2K} with respect to K is positive (it is equal to $\frac{2}{K^2}$), as is the second derivative of $f_K D_K$ with respect to K . Therefore, the second derivative of $G(N, K)$ is positive as well, and $G(N, K)$ is convex in K . The minimum of $G(N, K)$ is not at $K=0$

since we assume C is large enough to handle the initial load sharing traffic at little cost. Starting with equation (4.37) and differentiating $G(N,K)$ with respect to K (assuming K is continuous) yields:

$$\frac{\partial G(N,K)}{\partial K} = \frac{1}{\bar{\gamma}} \frac{\partial f_K D_K}{\partial K} + \frac{\partial \hat{T}_{N-2K}}{\partial K}$$

The value of K that optimizes the system cost satisfies the following relation:

$$\frac{1}{\bar{\gamma}} \frac{\partial f_K D_K}{\partial K} + \frac{\partial \hat{T}_{N-2K}}{\partial K} = 0$$

The optimum number of pairs, K^* can now be found by equating the expressions for the marginal processing and communication delays. We now substitute $N/2$ for $\bar{\gamma}$.

$$\frac{C}{N(C - K^*/2)^2} = \frac{2}{K^*}$$

which yields the following quadratic equation:

$$(K^*)^2 - 4K^*C(1 + 1/2N) + 4C^2 = 0$$

The solution of which is the following root:

$$K^* = 2C \left[1 + \frac{1}{2N} - \sqrt{(1 + 1/2N)^2 - 1} \right]$$

which simplifies to the following expression for K^* when we assume $N \gg 1$:

$$K^* = 2C \left[1 + \frac{1}{2N} - \frac{1}{\sqrt{N}} \right] < 2C \quad (4.43)$$

When N is large then the optimal number of pairs, K^* , tends towards $2C$ which is the upper limit on K noted above. This result means that we should push as much flow as the channel can carry. The result is surprising since the delay on the channel will be

extremely high when it is fully loaded, which is the opposite of what we set out to accomplish. However, we can explain the result as follows. By considering the channel delay, we added another "server" to the N processors in our model. When N is large there are many processors that have loads near 1 before load sharing, and the average job delay is quite high. After load sharing, the average processing time at K nodes has been reduced, while only the channel delay has been significantly increased. The overall effect is to reduce the average delay of a job since relatively few nodes are transmitting jobs over the network.

4.6) Random Pairing

In the previous sections we designed and analyzed an algorithm for load sharing. The algorithm paired up the heaviest loaded node with the most lightly loaded node. The two nodes then shared their combined load equally, thus relieving the large processing delay at the heavily loaded node. The performance of the average job delay involved a tradeoff between communication and processing costs. The question we now ask is: what would be the average processing delay in a network where the N nodes paired up at *random* (i.e. without first identifying lightly loaded and heavily loaded nodes) and shared their pooled load? In this section we neglect the communication cost and derive the average processing delay of the random scheme for sharing the load. The results of this section can be compared with those derived earlier; however, the reader should remember that the communication cost is left out.

We assume the same model as before. A broadcast network of N nodes each having one processor. The arrival process to each node is Poisson with rate γ_i and each job has a processing time that is exponentially distributed with rate $\mu=1$. The random variable γ_i is uniformly distributed over $[0,1]$. As with load sharing in section (4.4) once again we do not need to use ϵ , since there will be no probability that both

nodes of a pair will each have a load of 1.

A simple scheme is for each node to randomly pair up with another node and share the combined load equally. We refer to this scheme as *clumping*. More specifically, a clump is a set of nodes that pool their load together and share it equally. The size of the clump is the cardinality of the set. We analyze the network delay when the size of the clump is two. The total amount of load in the clump is the sum of the individual input rates. Each of the rates is an independent random variable so that the distribution of the sum is the convolution of the distributions for the individual r.v.'s. Let Y be the random variable for the load a node will see after load sharing in a clump.

$$Y = \frac{\gamma_1 + \gamma_2}{2} \quad (4.44)$$

The distribution of Y is the convolution of two uniform distributions and is expressed as:

$$f_Y(y) = \begin{cases} 4y & 0 \leq y \leq \frac{1}{2} \\ 4-4y & \frac{1}{2} \leq y \leq 1 \end{cases} \quad (4.45)$$

which is a triangular function on $[0,1]$. Since $f_Y(1)=0$ we need not to add guards against infinite delays. The approximate processing delay expression is calculated by considering just one node from each of the $\frac{N}{2}$ pairs. Each node has the same shared input rate as the other so that considering only half the network will yield the same result. The approximate processing delay expression is formulated and solved in the same manner as was done in section (4.4).

$$\hat{T}_C = \frac{1}{\gamma_0} \int_0^1 \cdots \int_0^1 \prod_{j=1}^N \left[f_Y(y_j) \sum_{i=1}^N \frac{y_i}{1-y_i} \right] dy_j$$

$$= 8 \ln 2 - 2 = 3.545$$

Note that the average processing time of a job is already close to the minimum of 2 (using the uniform distribution over $[0, 1]$ for the input rate).

The result of random pairing in the network is a dramatically reduced system time. By continuing the clumping process (adding more nodes to a clump) we can bring the system delay down to the optimum of two (when all nodes share the network load equally). As more nodes are added to the clump, the tail of the distribution for the clump load has less mass. In the limit as $N, K \rightarrow \infty$ the distribution is nearly Gaussian. If the clump size is K , we would like to bound the tail of the distribution to determine the probability of having a high delay in spite of the load sharing. Assume a network analyst has a specific delay, $T^* \geq 1$, that should not be exceeded. What is the probability that, with a clump size of K , the delay of the nodes in a clump exceeds T^* . All the nodes in a clump have the same input rate (after load sharing) and the same delay. Thus the question can be rephrased as follows: What is the probability that the mean input rate to a node after clumping ($S_K = \sum \frac{\gamma_i}{K}$) exceeds an amount γ^* which is a function of T^* (i.e. $\gamma^* = \frac{T^* - 1}{T^*}$). Since all the γ_i 's are independent random variables we can use the Chernoff bound to find an upper limit to the tail of the distribution of S_K/K . Let X_i represent the original input rate (before load sharing) to the i^{th} node, and $f(x)$ be the uniform distribution over $[0, 1]$. We are interested in

$$P\left[\frac{S_K}{K} \geq \gamma^*\right]$$

Following the procedure detailed in [Klei75] we find $M_X(v)$ the Laplace transform of the uniform distribution.

$$M_X(v) = \int_0^1 e^{-vx} dx = \frac{1}{v} \left[e^{-v} - 1 \right] \quad (4.46)$$

The parameter v is used to tighten the bound and is determined by the given parameter γ^* . We will need the following functions.

$$\gamma_X(v) = \ln(M_X(v)) = \ln \left[\frac{1}{v} \left[e^{-v} - 1 \right] \right] \quad (4.47)$$

and

$$\gamma_X^{(1)}(v) = \frac{d\gamma_X(v)}{dv} = \frac{e^{-v}[v-1]+1}{v \left[e^{-v} - 1 \right]} \quad (4.48)$$

The Chernoff bound is given by:

$$P \left[S_K \geq K\gamma^* = K\gamma_X^{(1)}(v) \right] \leq e^{K \left[\gamma_X(v) - v\gamma_X^{(1)}(v) \right]} \quad (4.49)$$

where v is determined by solving

$$\gamma^* = \gamma_X^{(1)}(v) = \frac{e^{-v}[v-1]+1}{v \left[e^{-v} - 1 \right]} \quad (4.50)$$

Figure (4.9) lists the bounds for $P[S_K/K \geq \gamma^*]$ for different values of the clump size, K . For example, in order to have a clump delay of 2.5 or less, the shared input rate must be equal to or less than .6. Looking at the first row in the table, we see that even a clump size of twenty still might yield an input rate rate greater than .6. However for higher delay requirements (e.g. $T^*=5$) even small clump sizes ($K=5$) will yield lightly loaded clumps with probability close to 1.

γ^*	T^*	Chernoff Bound on $P[\gamma > \gamma^* K]$				
		$K=2$	$K=3$	$K=5$	$K=10$	$K=20$
.6	2.5	.8856	.8334	.7381	.5447	.2967
.7	3.333	.6031	.4684	.2825	.0798	6.3×10^{-3}
.8	5	.2912	.1571	.0458	2.09×10^{-3}	4.38×10^{-6}
.9	10	.0738	.02	1.48×10^{-3}	2.19×10^{-6}	4.8×10^{-12}
.95	20	.0185	2.51×10^{-3}	4.63×10^{-6}	2.15×10^{-9}	4.63×10^{-18}

Chernoff Bounds on $P[S_K/K \geq \gamma^*]$
Figure (4.9)

4.7) Conclusions

In this chapter we developed a model of a distributed processing environment that uses a broadcast medium for transmitting the shared load among the various nodes. Previous algorithms (those presented in chapters 2 and 3 as well as in the literature) are not suited for broadcast networks, since they try to balance the load by sending too much information. Our approach, in this chapter, is to unburden the most heavily loaded nodes by sharing their processing traffic with lightly loaded nodes. To accomplish this we developed an algorithm that finds the most heavily and most lightly loaded nodes and pairs them up. Our algorithm is efficient in time and message complexity ($O(\log L_{MAX})$ bits and time per iteration) and does not require that all nodes participate. The algorithm proceeds in rounds until the processing delay at every node is within a prescribed amount or until the extra load sharing traffic starts to increase the overall job response time.

We analyzed the distributed processing environment model with and without load sharing. Based on the assumption that input rates to the users are uniformly distributed from 0 to $1-\epsilon$ we found that the processing delay \hat{T}_N is closely approximated by:

$$\hat{T}_N = \frac{2}{(1-\epsilon)^2} \left[\ln\left(\frac{1}{\epsilon}\right) - (1-\epsilon) \right]$$

where ϵ is close to, but less than 1. Using our model we found a close approximation to the processing delay in an environment with load sharing. Let K be the number of matched pairs. Then the average processing delay with K pairs is closely approximated by:

$$\hat{T}_{N-2K}^{(3)} = \frac{2 \left[(N+1) \sum_{i=K}^N 1/i - (N-K+1) \right]}{N-K+1}$$

The approximation results, along with simulation results, show that significant reductions in the average processing time of a job can be realized with relatively few pairs. Therefore, we have a method that does not utilize the network too much, yet significantly reduces the response time of a job. This confirms our thesis that in a broadcast network, only a little sharing needs to be done. Note that the results described in this paragraph are independent of the communication medium (since we are ignoring the delay due to transmitting the jobs).

We also developed a model of the communication traffic over the broadcast medium. Using this model we developed a tradeoff between savings in processing time and communication costs. We found that the number of pairs which minimizes the sum of the processing and communication delay is: $K^* \approx 2C$ where C is the capacity of the broadcast medium in jobs per second.

Another load balancing strategy was presented, in which random groups of K nodes "clump" together by collecting all their inputs and dividing the sum equally among the group members. We bounded the probability of having a high delay in the clump as a function of the clump size, K . We found that group sizes of four or five

(nodes) is enough to guarantee that moderately high delays do not occur.



CHAPTER 5

ANALYSIS OF TWO QUEUES WITH THRESHOLD

In Chapter 3 we looked at a load balancing problem in a distributed processing environment using a point to point data network for communications. The load balancing was accomplished by using a threshold policy at each node in the network to decide when to transmit jobs for remote processing. The analysis in Chapter 3 assumes that each node in the network can be modeled as a stochastically independent system. In this chapter we analyze the threshold queue in a network environment without making any independence assumption. Our motivation for this chapter is to provide a numerical technique to obtain the average number in system as a function of the parameters.

5.1) Model

The network we are modeling in this chapter is a two node load balancing system. One node may be considered to be a "network server" by accepting jobs transferred from the other system node as well as jobs from external sources.¹ The other node accepts jobs only from external sources and either processes those jobs locally or transmits them to the network server for processing. This is the dynamic load balancing policy. The model is a small (yet extremely difficult to analyze) example of the central server model studied by [Tant85]. In that model there are N satellite processors that are connected to a central server in a star topology. The satellite proces-

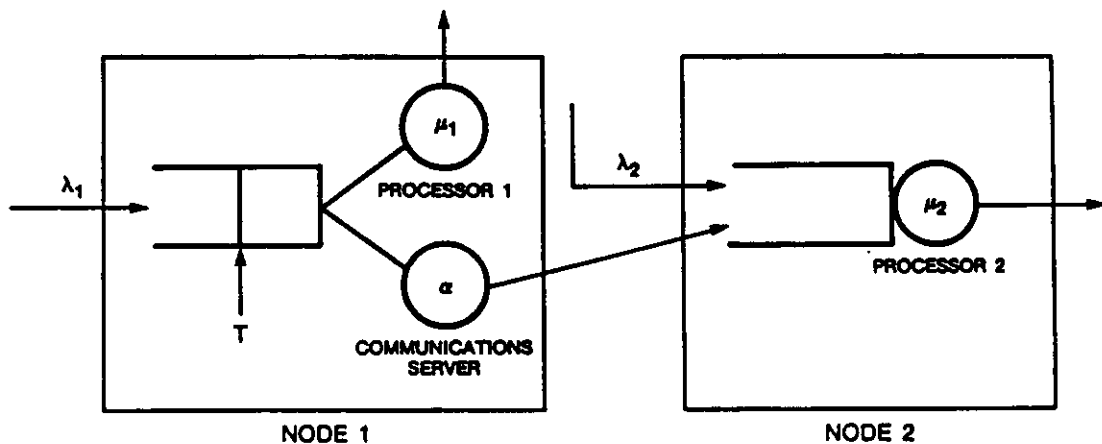
¹ A source is *external* if it is not a node in the communication network.

sors receive external input and process a fraction of those jobs locally. The rest of the jobs are transmitted to the central server for processing. The central server may also accept jobs from external sources. The load balancing policy used in this chapter is the threshold method. A satellite node will start transmitting jobs to the central server whenever the number of jobs that are waiting to be processed (at the satellite) meets or exceeds a threshold T . See figure (5.1) for a graphic description of the model.

Although the model focuses on one satellite node and the central server, the other $N-1$ satellites are included as well, as we now explain. In Chapter 3 we showed (using simulation results) that the superposition of the output processes of a large number of threshold queues is nearly Poisson. In this chapter we focus on one satellite node, and represent the output from the other $N-1$ satellites as a Poisson source. The Poisson source, with rate λ_2 , of external jobs to the second node shown in figure (5.1) represents the superposition of the transmitter output processes from the other $N-1$ satellite nodes.

Formally, there are two queueing systems, nodes 1 and 2. Node 1 is the satellite while node 2 is the network server. Jobs arrive from independent external sources to nodes 1 and 2 in a Poisson manner with rates λ_1 and λ_2 respectively. The processor at node 1 will take the first job waiting at the head of the queue (if any) whenever the processor is idle. If there is no job waiting then the processor will remain idle until a job arrives to the queue. Jobs that are processed at node 1 have a service time that is negatively exponentially distributed with mean length $\bar{x}_1 = \frac{1}{\mu_1}$. If there are less than T jobs in the queue at node 1 then the communications link (to the central server) remains idle. When the number of jobs waiting is T or more then the communications link is activated and starts transmitting the job at the head of the queue. The transmission time for a job is negatively exponentially distributed with mean time $\bar{r} = \frac{1}{\alpha}$. If,

while the communication link is busy, the number of jobs waiting drops below T then the communication link immediately ceases transmission of the job it was working on. The pre-empted job returns to the head or the back of the queue. In the analysis of this problem we are concerned with the average number in the system so the Markov description will be identical for any pre-emption policy (i.e. where to place the pre-empted job). Transmitted jobs and external arrivals to node 2 queue up for processor 2. Jobs in queue 2 are processed in a FCFS manner and have a service time that is negatively exponentially distributed with mean $\bar{x}_2 = 1/\mu_2$. Jobs leave the system when they have been processed at either of the two nodes.



Two Node Threshold Queueing System
Figure (5.1)

The model is intended to analyze the effect of the threshold on the system behavior. The problem is difficult enough, thus we have made a conscious decision to leave out some aspects of a more realistic load balancing model. In particular, a job that is transmitted from node 1 to node 2 for processing does not have to return to

node 1 before exiting the system. If the central server and satellites are geographically dispersed then omitting the return leg makes the model less convincing. However, if all the machines are colocated then one may assume that users are directly connected to both machines, and that a job transmitted from node 1 to node 2 includes the address for directing the output. A second problem arises with data files that might be located at a satellite node that are used in processing jobs at that node. Transmitting a job from one node to another might entail shipping a large data file as well. There are two ways to handle this modeling aspect. If the real system has shared memory then only jobs are transmitted and data files are accessible to either machine. If there is no shared memory, then we assume that the data files are to be shipped as well as the jobs and the average transmission time of a job will take into account the mean job length and the mean data file length in bits .

5.2) Problem Formulation and Solution

5.2.1) System Equations

The system can be modeled by a two dimensional state space. Let $n_1(t)$ be the total number of jobs in node 1 (including any job in the transmission link or the processor), $n_2(t)$ be the number of jobs in node 2, at time t . Then $(n_1(t), n_2(t))$ $n_1 \geq 0$, $n_2 \geq 0$, $t \geq 0$ is the complete set of states that the system can be in. The threshold policy depends only on the total number of jobs in node 1. If $n_1(t) \geq T+1$ then there is a job in the transmission link, otherwise it is idle. The preemptive discipline precludes the state where there are less than $T+1$ in system yet the transmission link is busy. The distributions for the processing and transmission time are negatively exponentially distributed, thus the system is memoryless. Let $P_{i,j}(t)$ be the probability that the system is in state (i, j) at time t .

We now state the system conditions for ergodicity (i.e. stability requirements), which will be derived later in this section. At this point we provide the intuition and state the requirements so as to move on with the steady state analysis.

The intuition for the following three conditions is that the input rate must be strictly less than the service rate of the system considered. Considering the first queue alone; we have

$$\lambda_1 < \mu_1 + \alpha$$

We now give a stricter condition (to be proved in section (5.2.2)):

$$\lambda_1 < \mu_1 + \alpha f_T \tag{S1}$$

where f_T represents the fraction of time the communication server is busy transmitting jobs. It is given by

$$f_T = \sum_{i=T+1}^{\infty} P_i < 1$$

where P_i is the steady state probability of finding i jobs in node 1. Essentially, (S1) states that the input rate must be strictly less than the server rates, where the second server (transmitter) is only busy a fraction f_T of the time (when the threshold is T). Thus the effective rate for the transmitter is αf_T . We now derive stability condition (S1). If $\lambda_1 < \mu_1 + \alpha f_T$ then $\lambda_1 < \mu_1 + \alpha$ since $f_T < 1$. We now show that inequality $\lambda_1 < \mu_1 + \alpha$ implies (S1). When $\lambda_1 / \mu_1 \leq 1$ then the assertion is obviously true. We now consider the range $\lambda_1 > \mu_1$. We use equations that are derived later to prove our assertion. From equation (5.20) we have

$$\lambda_1 < \mu_1 + \alpha f_T = \mu_1 + \alpha \sum_{i=T+1}^{\infty} P_i$$

$$= \mu_1 + \alpha P(0,1)(\lambda_1/\mu_1)^T (\lambda_1/(\mu_1+\alpha)) \sum_{i=T+1}^{\infty} \left[\frac{\lambda_1}{\mu_1+\alpha} \right]^{i-(T+1)}$$

Summing the above we get

$$= \mu_1 + \frac{\alpha(\lambda_1/\mu_1)^T [\lambda_1/(\mu_1+\alpha-\lambda_1)]}{\frac{1-(\lambda_1/\mu_1)^T}{1-(\lambda_1/\mu_1)} + \frac{(\lambda_1/\mu_1)^T}{1-[\lambda_1/(\mu_1+\alpha)]}}$$

After some algebraic manipulation to simplify the denominator we obtain

$$\lambda_1 < \mu_1 + \frac{\lambda_1 \alpha \left[\frac{\lambda_1}{\mu_1} - 1 \right]}{\lambda_1 \alpha / \mu_1 - (\mu_1 + \alpha - \lambda_1)(\mu_1 / \lambda_1)^T}$$

or

$$\mu_1 > \lambda_1 \frac{\alpha - (\mu_1 + \alpha - \lambda_1)(\mu_1 / \lambda_1)^T}{\lambda_1 \alpha / \mu_1 - (\mu_1 + \alpha - \lambda_1)(\mu_1 / \lambda_1)^T}$$

The denominator may be written as

$$(\lambda_1 - \mu_1)(\mu_1 / \lambda_1)^T + \alpha(\lambda_1 / \mu_1 - (\mu_1 / \lambda_1)^T) > 0$$

Since $\lambda_1 > \mu_1$, we see that the denominator is positive. We have, after multiplying the denominator through and canceling like terms,

$$-\mu_1(\mu_1 + \alpha - \lambda_1)(\mu_1 / \lambda_1)^T > -\lambda_1(\mu_1 + \alpha - \lambda_1)(\mu_1 / \lambda_1)^T$$

which is true since $\lambda_1 > \mu_1$ and $\mu_1 + \alpha - \lambda_1 > 0$. QED.

The stability condition for the second queue (derived in section (5.2.5)) is

$$\lambda_2 + \alpha f_T < \mu_2 \tag{S2}$$

which implies that $\lambda_2 < \mu_2$. A third condition which immediately follows from (S1) and (S2) considers the two queues as an entire system.

$$\lambda_1 + \lambda_2 < \mu_1 + \mu_2 \quad (S3)$$

We assume that (under the above conditions) the system reaches steady state and that the long run probability of finding the system in a state (i,j) ($i \geq 0, j \geq 0$) has a unique limit

$$\lim_{t \rightarrow \infty} P_{i,j}(t) = P_{i,j}$$

Where $P_{i,j}$ represent the steady state probability that the system is in state (i,j) .

The two dimensional state diagram is illustrated in figure (5.2). The steady state transition equations are now presented.

For $i, j = 0$

$$(\lambda_1 + \lambda_2) P_{0,0} = \mu_1 P_{1,0} + \mu_2 P_{0,1} \quad (5.1)$$

For $j=0$ and $1 \leq i \leq T$

$$(\lambda_1 + \lambda_2 + \mu_1) P_{i,0} = \lambda_1 P_{i-1,0} + \mu_1 P_{i+1,0} + \mu_2 P_{i,1} \quad (5.2)$$

For $j=0$ and $i \geq T+1$

$$(\lambda_1 + \lambda_2 + \mu_1 + \alpha) P_{i,0} = \lambda_1 P_{i-1,0} + \mu_1 P_{i+1,0} + \mu_2 P_{i,1} \quad (5.3)$$

For $j > 0$ and $i=0$

$$(\lambda_1 + \lambda_2 + \mu_2) P_{0,j} = \lambda_2 P_{0,j-1} + \mu_1 P_{1,j} + \mu_2 P_{0,j+1} \quad (5.4)$$

For $j > 0$ and $0 < i < T$

$$(\lambda_1 + \lambda_2 + \mu_1 + \mu_2) P_{i,j} = \lambda_1 P_{i-1,j} + \lambda_2 P_{i,j-1} + \mu_1 P_{i+1,j} + \mu_2 P_{i,j+1} \quad (5.5)$$

For $j > 0$ and $i=T$

$$(\lambda_1 + \lambda_2 + \mu_1 + \mu_2) P_{T,j} = \quad (5.6)$$

$$\lambda_1 P_{T-1,j} + \lambda_2 P_{T,j-1} + \mu_1 P_{T+1,j} + \mu_2 P_{T,j+1} + \alpha P_{T+1,j-1}$$

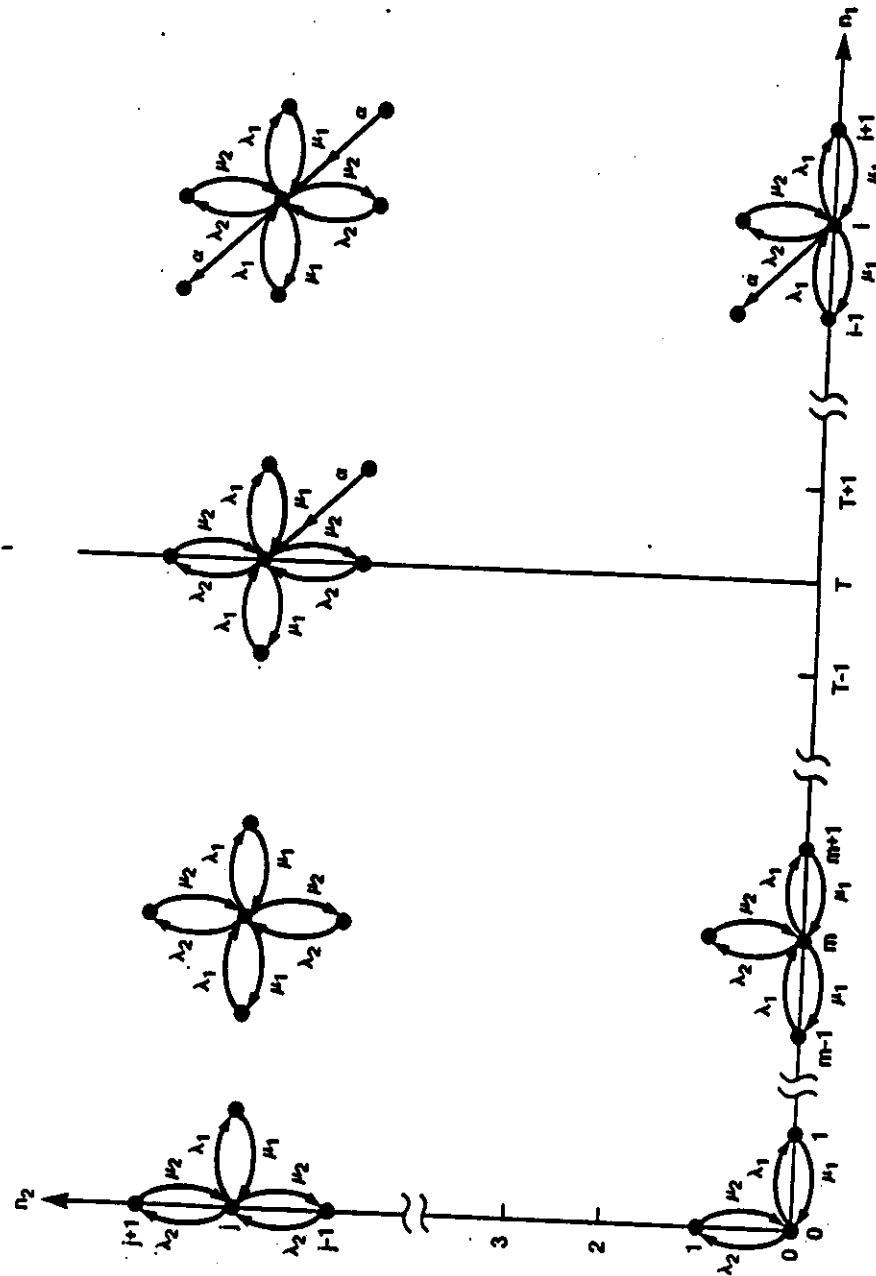


Figure (5.2)
State Diagram for the Threshold Queueing System

For $j > 0$ and $i > T$

$$(\lambda_1 + \lambda_2 + \mu_1 + \mu_2 + \alpha) P_{i,j} = \quad (5.7)$$

$$\lambda_1 P_{i-1,j} + \lambda_2 P_{i,j-1} + \mu_1 P_{i+1,j} + \mu_2 P_{i,j+1} + \alpha P_{i+1,j-1}$$

We define the following transforms:

$$R_j(z) = \sum_{i=0}^{\infty} P_{i,j} z^i$$

$$P_i(w) = \sum_{j=0}^{\infty} P_{i,j} w^j$$

$$P(z, w) = \sum_{j=0}^{\infty} P_j(z) w^j = \sum_{i=0}^{\infty} P_i(w) z^i$$

Multiply equations (5.1) through (5.7) by w^j and sum from $j=0$ to infinity to obtain the following.

For $i=0$

$$(\lambda_1 + \lambda_2 + \mu_2) P(0, w) - \mu_2 P_{0,0} = \lambda_2 w P(0, w) + \mu_1 P_1(w) + \frac{\mu_2}{w} [P(0, w) - P_{0,0}] \quad (5.8)$$

For $0 < i < T$

$$(\lambda_1 + \lambda_2 + \mu_1 + \mu_2) P_i(w) - \mu_2 P_{i,0} = \quad (5.9)$$

$$\lambda_1 P_{i-1}(w) + \lambda_2 w P_i(w) + \mu_1 P_{i+1}(w) + \frac{\mu_2}{w} [P_i(w) - P_{i,0}]$$

For $i=T$

$$(\lambda_1 + \lambda_2 + \mu_1 + \mu_2) P_T(w) - \mu_2 P_{T,0} = \quad (5.10)$$

$$\lambda_1 P_{T-1}(w) + \lambda_2 w P_T(w) + \mu_1 P_{T+1}(w) + \frac{\mu_2}{w} [P_T(w) - P_{T,0}] + \alpha w P_{T+1}(w)$$

For $i > T$

$$(\lambda_1 + \lambda_2 + \mu_1 + \mu_2 + \alpha) P_i(w) - \mu_2 P_{i,0} = \quad (5.11)$$

$$\lambda_1 P_{i-1}(w) + \lambda_2 w P_i(w) + \mu_1 P_{i+1}(w) + \frac{\mu_2}{w} [P_i(w) - P_{i,0}] + \alpha w P_{i+1}(w)$$

We now multiply equations (5.8) through (5.11) by z^i and sum from $i=0$ to infinity to obtain the generating function for the two dimensional Markov process (n_1, n_2) .

$$K(z, w) P(z, w) = A(z, w) P(0, w) + B(z, w) P(z, 0) + C(z, w) \sum_{i=0}^T P_i(w) z^i \quad (5.12)$$

where

$$K(z, w) = \lambda_1(1-z) + \mu_1(1-1/z) + \lambda_2(1-w) + \mu_2(1-1/w) + \alpha(1-w/z) \quad (5.13)$$

$$A(z, w) = \mu_1(1-1/z) \quad (5.14)$$

$$B(z, w) = \mu_2(1-1/w) \quad (5.15)$$

$$C(z, w) = \alpha(1-w/z) \quad (5.16)$$

5.2.2) Performance Measures

Equation (5.12) is too difficult to invert for $P(z, w)$ since there are $T+2$ unknown functions on the right hand side. However, we can extract some more information about the performance of the system (rather than the steady state probabilities) from the above equations. We know from the sum of total probability that $P(z, w)|_{z,w=1} = 1$. From equation (5.12) we have:

$$P(z, w)|_{z,w=1} = 1 = \frac{\left[A(z, w) P(0, w) + B(z, w) P(z, 0) + C(z, w) \sum_{i=0}^T P_i(w) z^i \right]_{w,z=1}}{K(z, w)|_{z,w=1}}$$

The equation at $w, z=1$ evaluates to an indeterminate (0/0) form. Using L'Hospital's rule (with respect to either z or w) we can derive a relationship between $P(1,0)$ and $P(0,1)$. Instead, we will use a simple argument to derive these same results.

By putting a black box around the second node only we observe that the input rate must equal the output rate. The input rate consists of the external arrival rate (λ_2) plus the transfer rate of jobs from node 1. The transmission link is busy only when $n_1 > T$ and then transmits with rate α . The departure rate of jobs from node 2 is μ_2 when the node is nonempty otherwise it is 0. Equating the input rate with the departure rate yields:

$$\mu_2(1-P(1,0)) = \lambda_2 + \alpha \sum_{i=T+1}^{\infty} P_i(1)$$

or

$$P(1,0) = \left(1 - \frac{\lambda_2}{\mu_2}\right) - \frac{\alpha}{\mu_2} \left[1 - \sum_{i=0}^T P_i(1)\right] \quad (5.17)$$

where $P(1,0) = \sum_{i=0}^{\infty} P_{i,0}$ is the probability of no jobs in node 2 and $P_i(1) = \sum_{j=0}^{\infty} P_{i,j}$ is the probability of i jobs in node 1.

For node 1, the input rate consists only of the external arrival rate, λ_1 . The departure rate consists of the processor rate, μ_1 , when the node is nonempty, plus the transmission rate, α , when there are $n_1 > T$ jobs in the node. We then have

$$\mu_1[1-P(0,1)] + \alpha \left[1 - \sum_{i=0}^T P_i(1)\right] = \lambda_1$$

$$P(0,1) = \left(1 - \frac{\lambda_1}{\mu_1}\right) + \frac{\alpha}{\mu_1} \left[1 - \sum_{i=0}^T P_i(1)\right] \quad (5.18)$$

Combining equations (5.17) and (5.18) yields the following equation:

$$\lambda_1 + \lambda_2 = \mu_1(1-P(0,1)) + \mu_2(1-P(1,0))$$

Alternatively, we could have derived the above equation from considering the flow through both nodes. From this observation we get:

$$\frac{\mu_1}{\mu_1 + \mu_2} P(0,1) + \frac{\mu_2}{\mu_1 + \mu_2} P(1,0) = 1 - \frac{\lambda_1 + \lambda_2}{\mu_1 + \mu_2} \quad (5.19)$$

Exact expressions can be obtained for $P(0,1)$ and $P(1,0)$ by considering node 1 alone. Since node 1's behavior is not dependent on node 2 at all, we can determine the probability distribution $P_i(1)$ for all i . The distribution of number in node 1 (unconditioned on the number in node 2) can be easily derived as was done in Chapter 3 (from a birth death chain) and we will repeat the results only. Note that the stability condition $\lambda_1 < \mu_1 + \alpha$ is easily derived from the birth death chain (see [Klei75]).

$$P_i(1) = \begin{cases} P(0,1) \left(\frac{\lambda_1}{\mu_1}\right)^i & i < T \\ P(0,1) \left(\frac{\lambda_1}{\mu_1}\right)^T \left[\frac{\lambda_1}{\mu_1 + \alpha}\right]^{i-T} & i \geq T \end{cases}$$

where

$$P(0,1) = \begin{cases} \frac{1 - \frac{\lambda_1}{\mu_1 + \alpha}}{\frac{\alpha}{\mu_1 + \alpha} \left[\frac{1 - (\lambda_1/\mu_1)^{T+1}}{1 - \lambda_1/\mu_1} \right] + \frac{\mu_1}{\mu_1 + \alpha}} & \lambda_1 \neq \mu_1 \\ \left[T + \frac{\mu_1 + \alpha}{\alpha} \right]^{-1} & \lambda_1 = \mu_1 \end{cases} \quad (5.20)$$

From equation (5.19), the probability of no jobs in node 2 is:

$$P(1,0) = \begin{cases} \frac{\mu_1 + \mu_2 - (\lambda_1 + \lambda_2)}{\mu_2} - \frac{\mu_1}{\mu_2} \frac{\mu_1 + \alpha - \lambda_1}{\alpha \left[\frac{1 - (\lambda_1/\mu_1)^{T+1}}{1 - \lambda_1/\mu_1} \right]} + \mu_1 & \lambda_1 \neq \mu_1 \\ \frac{\mu_1 + \mu_2 - (\lambda_1 + \lambda_2)}{\mu_2} - \frac{\mu_1/\mu_2}{\left[T + \frac{\mu_1 + \alpha}{\alpha} \right]} & \lambda_1 = \mu_1 \end{cases} \quad (5.21)$$

Unfortunately, given the set of transforms (equations (5.12) through (5.16)) we are not able to extract other probabilities such as $P_{0,0}$, or $R_j(1)$, the probability there are j jobs in node 2 (unconditioned on node 1), for any j . The same problem holds for the average number in system. That is, while we are able to derive a closed form expression for the average number in node 1, \bar{n}_1 we are unable to do so for node 2. A simple way to obtain \bar{n}_1 is to use equations (5.12) through (5.16) with $w=1$.

$$P(z, 1) = \frac{\alpha \sum_{i=0}^T P_i(1) z^i + \mu_1 P(0, 1)}{(\mu_1 + \alpha) - z\lambda_1} \quad (5.22)$$

Taking the derivative of the above equation (w.r.t. z), setting $z=1$ and using equation (5.18) yields

$$\bar{n}_1 = \frac{\frac{\lambda_1}{\mu_1 + \alpha}}{1 - \frac{\lambda_1}{\mu_1 + \alpha}} + \frac{\frac{\alpha}{\lambda_1 + \alpha} \sum_{i=0}^T i P_i(1)}{1 - \frac{\lambda_1}{\mu_1 + \alpha}} \quad (5.23)$$

which is the same result as equation (3.8). To derive \bar{n}_2 we set $z=1$ in equations (5.12) through (5.16) and obtain:

$$P(1, w) = \frac{\mu_2 P(1, 0) - \alpha w \sum_{i=0}^T P_i(w)}{\mu_2 - w(\lambda_2 + \alpha)} \quad (5.24)$$

Taking the derivative of (5.24) (w.r.t. w), setting $w=1$ and using equation (5.17) we obtain

$$\bar{n}_2 = \frac{\mu_2(1-P(1,0)) - \alpha \sum_{i=0}^T \frac{\partial P_i(w)}{\partial w} \Big|_{w=1}}{\mu_2 - (\lambda_2 + \alpha)} \quad (5.25)$$

When $\mu_2 - (\lambda_2 + \alpha) \neq 0$ then the last equation has $T+1$ unknown functions. The term $\frac{\partial P_i(w)}{\partial w} \Big|_{w=1}$ denotes the average number in node 2 when there are i jobs in node 1.

The $T+1$ unknown terms can be reduced to one unknown $\frac{\partial P_0(w)}{\partial w} \Big|_{w=1}$, and a set of unknown constants $P_{i,0}$ ($0 \leq i < T$) by repeatedly using equations (5.8) and (5.9). When $\mu_2 = \lambda_2 + \alpha$ then equation (5.25) is indeterminate. However, we will use the following simpler approach to derive an expression for the total number in system.

Consider the following transform $P^*(z)$.

$$P^*(z) = \sum_{n=0}^{\infty} \sum_{i=0}^n P_{i,n-i} z^n \quad (5.26)$$

The inner sum collects all the states that represent n jobs in the total system, into one "diagonal" group (see figure (5.3)). Let P_n be the steady state probability of n jobs in the total system (whose z transform is $P^*(z)$). Probabilities $P_{n,0}$, and $P_{0,n}$ remain as before. The state equations for this grouping can be written as follows:

$$(\lambda_1 + \lambda_2) P_0 = \mu_1 P_{1,0} + \mu_2 P_{0,1} = (\mu_1 + \mu_2) P_1 - \mu_2 P_{1,0} + \mu_1 P_{0,1} \quad (5.27)$$

For $i \geq 1$

$$(\lambda_1 + \lambda_2 + \mu_1 + \mu_2) P_i - (\mu_2 P_{1,0} + \mu_1 P_{0,1}) = \quad (5.28)$$

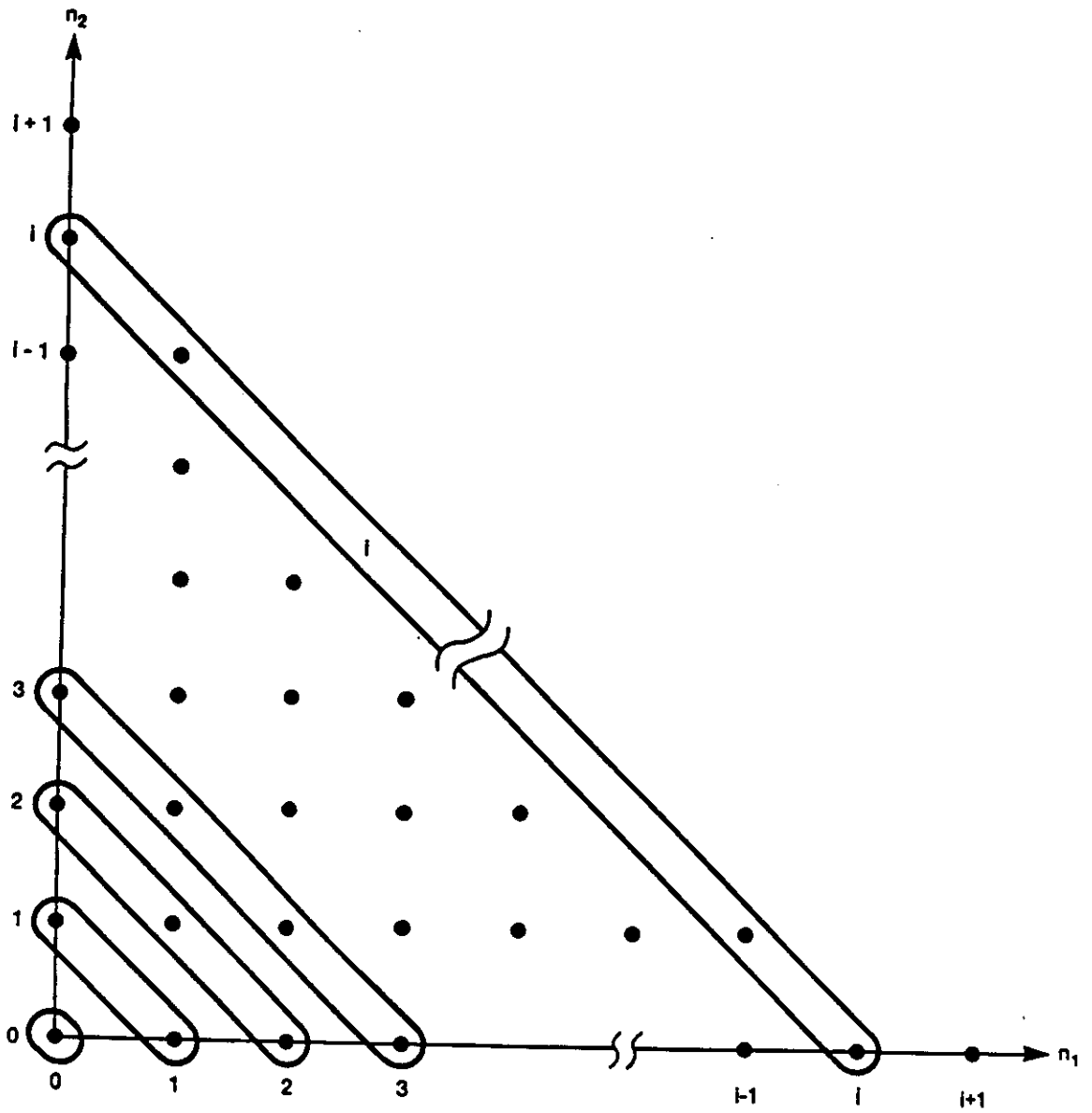


Figure (5.3)
Alternate Representation of the State Space

$$(\lambda_1 + \lambda_2) P_{i-1} + (\mu_1 + \mu_2) P_{i+1} - (\mu_2 P_{i+1,0} + \mu_1 P_{0,i+1})$$

Multiplying equation (5.28) by z^i and summing from $i=1$ to infinity yields:

$$\begin{aligned} & (\lambda_1 + \lambda_2 + \mu_1 + \mu_2) [P^*(z) - P_0] - \mu_1 [P(0,z) - P_0] - \mu_2 [P(z,0) - P_0] \\ &= (\lambda_1 + \lambda_2) z P^*(z) + \frac{\mu_1 + \mu_2}{z} [P^*(z) - P_0 - z P_1] \\ & \quad - \frac{\mu_1}{z} [P(0,z) - P_0 - z P_{0,1}] - \frac{\mu_2}{z} [P(z,0) - P_0 - z P_{1,0}] \end{aligned}$$

The above equation can be simplified using equation (5.27) to obtain

$$P^*(z) = \frac{\mu_1 P(0,z) + \mu_2 P(z,0)}{\mu_1 + \mu_2 - z(\lambda_1 + \lambda_2)} \quad (5.29)$$

The functions $P(0,z)$ and $P(z,0)$ are the transforms of the sets of boundary states $\{P_{0,j}\}$ and $\{P_{i,0}\}$ respectively. By setting $z=1$ and equating $P^*(1)$ with one we obtain equation (5.19) once more. Taking the derivative of $P^*(z)$ with respect to z at $z=1$ gives an expression for the average number in the system, \bar{n} .

$$\bar{n} = \frac{\lambda_1 + \lambda_2}{\mu_1 + \mu_2 - (\lambda_1 + \lambda_2)} + \frac{\mu_1 P'(0,1) + \mu_2 P'(1,0)}{\mu_1 + \mu_2 - (\lambda_1 + \lambda_2)} \quad (5.30)$$

The denominator is strictly positive when stability condition (S3) is true. We will use this equation to evaluate \bar{n} in section (5.3) after we obtain analytic solutions (by solving a boundary value problem) for $P(0,z)$ and $P(z,0)$.

Our goal is to obtain numerical solutions to $P'(0,1)$ and $P'(1,0)$. To do so, we first define a two dimensional boundary valued problem involving $P(z,0)$ and $P(0,z)$ and solve it numerically. From the numerical solution we can obtain the numerical derivatives we are seeking to evaluate \bar{n}_2 .

5.2.3) Boundary Value Problem Formulation

A Boundary Value problem in two variables can be formulated as follows. Find two functions $\Phi(z)$, and $\Psi(w)$ analytic in their respective disks ($|z| < 1, |w| < 1$), such that the following equation holds for $K(z,w)=0$ (the zero set of points)

$$A(z,w)\Phi(z) + B(z,w)\Psi(w) = C(z,w) \quad (5.31)$$

where $A(z,w)$, $B(z,w)$ and $C(z,w)$ are known functions. For our problem of section (5.2.2) we will shortly be able identify that $K(z,w)$ is the kernel function of the state space transform $P(z,w)$. The function $\Phi(z)$ is the transform of the boundary probabilities $P_{i,0}$, and $\Psi(w)$ is the transform of the boundary probabilities $P_{0,j}$.

To reduce the threshold queue problem to a boundary value problem, we organize the transforms a little differently than was done in the previous section. We start off with equations (5.8) through (5.11). Let $G(z,w) = \sum_{i=T+1}^{\infty} P_i(w)z^i$. Multiplying equation (5.11) by z^i and summing both sides from $T+1$ to ∞ yields:

$$\left[\lambda_2(1-w) + \mu_2(1-1/w) + \lambda_1 + \mu_1 + \alpha \right] G(z,w) - \mu_2(1-1/w)R'(z) = \quad (5.32)$$

$$\lambda_1 z \left[G(z,w) + P_T(w)z^T \right] + \left[\frac{\mu_1 + \alpha w}{z} \right] \left[G(z,w) - P_{T+1}(w)z^{T+1} \right]$$

where $R'(z) = \sum_{i=T+1}^{\infty} P_{i,0}z^i$. Collecting the $G(z,w)$ terms on the left hand side simplifies the above equation to:

$$K(z,w) G(z,w) = \mu_2(1-1/w)R'(z) + \lambda_1 z^{T+1} P_T(w) - \left[\frac{\mu_1 + \alpha w}{z} \right] P_{T+1}(w)z^{T+1} \quad (5.33)$$

Substituting in equation (5.10) above to eliminate $P_{T+1}(w)$, we obtain:

$$K(z,w)G(z,w) = \mu_2(1-1/w)z^T R(z) + \lambda_1 z^T P_{T-1}(w) -$$

$$\left[\lambda_1(1-z) + \mu_1 + \lambda_2(1-w) + \mu_2(1-1/w) \right] P_T(w) z^T$$

where $R(z) = \sum_{i=T}^{\infty} P_{i,0} z^{i-T} = z^{-T} (R'(z) + P_{T,0} z^T)$ We now factor out the z^T term and

have:

$$K(z,w)G(z,w) = \tag{5.34}$$

$$z^T \left[\mu_2(1-1/w)R(z) + \lambda_1 P_{T-1}(w) - \left[\lambda_1(1-z) + \mu_1 + \lambda_2(1-w) + \mu_2(1-1/w) \right] P_T(w) \right]$$

The above equation has three unknowns ($P_{T-1}(w)$, $P_T(w)$, and $R(z)$) on the right hand side. We can reduce the number of unknowns to two by using equation (5.9) repeatedly to find the relationship between $P_{T-1}(w)$, $P_T(w)$, and $P(0,w)$. We now use equation (5.9) and an identity for $P_i(w)$ to write the following matrix equations for $0 < i < T$

$$\begin{bmatrix} P_{i+1}(w) \\ P_i(w) \end{bmatrix} = \begin{bmatrix} M_{11}(w) & -\frac{\lambda_1}{\mu_1} \\ 1 & 0 \end{bmatrix} \begin{bmatrix} P_i(w) \\ P_{i-1}(w) \end{bmatrix} - \begin{bmatrix} \frac{\mu_2}{\mu_1} P_{i,0} \\ 0 \end{bmatrix} \tag{5.35}$$

where $M_{11}(w) = \frac{\lambda_1 + \mu_1 + \lambda_2(1-w) + \mu_2(1-1/w)}{\mu_1}$. Equation (5.8) also lends itself to a

vector formulation

$$\begin{bmatrix} P_1(w) \\ P(0,w) \end{bmatrix} = \begin{bmatrix} L(w) \\ 1 \end{bmatrix} P(0,w) - \begin{bmatrix} \frac{\mu_2}{\mu_1} P_{0,0} \\ 0 \end{bmatrix} \tag{5.36}$$

where $L(w) = \frac{\lambda_1 + \lambda_2(1-w) + \mu_2(1-1/w)}{\mu_1}$. We want to reduce the following expression

in equation (5.34) to one involving $P(0,w)$ and the constants $P_{i,0}$ ($0 \leq i < T-1$).

$$\lambda_1 P_{T-1}(w) - \left[\lambda_1(1-z) + \mu_1 + \lambda_2(1-w) + \mu_2(1-1/w) \right] P_T(w) =$$

$$B^T(z, w) P(0, w) + \sum_{i=0}^{T-1} C_i(z, w) P_{i,0}$$

For simplicity, we write the above equation in vector form

$$\begin{bmatrix} -(\lambda_1(1-z) + \mu_1 + \lambda_2(1-w) + \mu_2(1-1/w)) & \lambda_1 \end{bmatrix} \begin{bmatrix} P_T(w) \\ P_{T-1}(w) \end{bmatrix} =$$

$$B^T(z, w) P(0, w) + \sum_{i=0}^{T-1} C_i(z, w) P_{i,0}$$

Now we use equation (5.35) recursively to obtain the following:

$$\vec{V}_1 \begin{bmatrix} P_T(w) \\ P_{T-1}(w) \end{bmatrix} = \vec{V}_1 M^{T-1} \begin{bmatrix} P_1(w) \\ P(0, w) \end{bmatrix} - \sum_{i=1}^{T-1} \vec{V}_1 M^{T-1-i} \begin{bmatrix} \mu_2(1-1/w) \\ 0 \end{bmatrix} P_{i,0}$$

where

$$M = \begin{bmatrix} M_{11}(w) & \frac{\lambda_1}{\mu_1} \\ 1 & 0 \end{bmatrix} \quad \vec{V}_1 = \begin{bmatrix} -(\lambda_1(1-z) + \mu_1 + \lambda_2(1-w) + \mu_2(1-1/w)) & \lambda_1 \end{bmatrix}$$

Finally, using equation (5.36) we end up with

$$\vec{V}_1 \begin{bmatrix} P_T(w) \\ P_{T-1}(w) \end{bmatrix} = \vec{V}_1 M^{T-1} \vec{V}_2^\dagger P(0, w) - \sum_{i=0}^{T-1} \vec{V}_1 M^{T-1-i} \vec{V}_3^\dagger P_{i,0} \quad (5.37)$$

where

$$\vec{V}_2 = \begin{bmatrix} L(w) & 1 \end{bmatrix} \quad \vec{V}_3 = \begin{bmatrix} \frac{\mu_2}{\mu_1}(1-1/w) & 0 \end{bmatrix}$$

We use the notation \vec{V}^\dagger to denote the transpose of vector \vec{V} . Substituting equation (5.37) into (5.34) yields

$$K(z,w) G(z,w) = \tag{5.38}$$

$$z^T \left[\mu_2(1-1/w)R(z) + \vec{V}_1 M^{T-1} \vec{V}_2^\dagger P(0,w) - \sum_{i=0}^{T-1} \vec{V}_1 M^{T-1-i} \vec{V}_3^\dagger P_{i,0} \right]$$

At this point we are almost done. The problem will be to (numerically) determine the boundary functions $R(z)$ and $P(0,w)$. The last step in the reduction to a boundary value problem, concerns the latter function, $P(0,w)$. In the solution of the boundary value problem we will find it necessary to apply the logarithm function to $B^T(z,w)$, the coefficient of $P(0,w)$. In order that the logarithm function be analytic over the domain of interest, we need to guarantee that $B^T(z,w)$ has no poles in the domain. Our domain of interest includes the point $w=0$ which is a pole of order $T+1$ in $B^T(z,w)$. One pole will be canceled when the equation is divided by the term in front of $R(z)$. The other T poles will be canceled by separating from $P(0,w)$ the first T terms ($P_{0,j}w^j$ ($0 \leq j < T$)) and pulling out a factor w^T from the remaining terms. The boundary function will then be $w^T P^T(0,w) = (P(0,w) - \sum_{i=0}^{T-1} P_{0,i}w^i)$ where

$$P^T(0,w) = \sum_{j=T}^{\infty} P_{0,j}w^{j-T}. \text{ The system equation assumes its final form:}$$

$$K(z,w) G(z,w) = \tag{5.39}$$

$$z^T \left[\mu_2(1-1/w)R(z) + w^T \vec{V}_1 M^{T-1} \vec{V}_2^\dagger P^T(0,w) - \sum_{i=0}^{T-1} \vec{V}_1 M^{T-1-i} \vec{V}_3^\dagger P_{i,0} + \sum_{j=0}^{T-1} \vec{V}_1 M^{T-1} \vec{V}_2^\dagger P_{0,j}w^j \right]$$

We are looking for analytic functions $G(z,w)$, $R(z)$, and $P^T(0,w)$. When $K(z,w)=0$ the left hand side of (5.39) vanishes. The right hand side must also vanish if $G(z,w)$ is analytic. We thus obtain the following boundary value problem defined on the set of

points (z, w) where $K(z, w)=0$:

$$\mu_2(1-1/w)R(z) + B^T(z, w)P^T(0, w) + \sum_{i=0}^{T-1} C_i(z, w)P_{i,0} + \sum_{j=0}^{T-1} C_j(z, w)P_{0,j} = 0 \quad (5.40)$$

where

$$B^T(z, w) = w^T \vec{V}_1 M^{T-1} \vec{V}_2^\dagger \quad (5.41)$$

$$C_i(z, w) = -\vec{V}_1 M^{T-1-i} \vec{V}_3^\dagger \quad (5.42)$$

$$C_j(z, w) = \vec{V}_1 M^{T-1} \vec{V}_2^\dagger w^j \quad (5.43)$$

Note that equation (5.40) has two unknown boundary functions $R(z)$, and $P^T(0, w)$, each a function of one variable (z and w , respectively). The other functions are known except for the $2T-1$ constants. This is the form of the general two dimensional boundary value problem that we specified in the beginning of the section. The only differences are the $2T-1$ constants. In the next section we will provide a method for solving our boundary value problem in two parts. The first part consists of solving $2T-1$ separate boundary value problems as formulated in the beginning of this section. The second part uses the $2T-1$ solutions along with the $2T-1$ constants in a linear program that resolves the values of the constants.

5.2.4) Method of Solution

The solution of a two variable Boundary Value problem consists of constructing the analytic functions $\Phi(z)$, and $\Psi(w)$ so that equality (5.31) holds when the pair (z, w) satisfies $K(z, w)=0$. For this purpose we use the Splitting Method and Theorem developed by [Tay188]. The Splitting Theorem is stated without proof. Given

1. $K(z, w)$ analytic in $|z| < 1+\epsilon$, $|w| < 1+\epsilon$.
2. For all $|z|=1$, $z \neq 1$, there exists unique $|w| < 1$ such that $K(z, w)=0$.

3. For all $|w|=1, w \neq 1$, there exists unique $|z| < 1$ such that $K(z, w) = 0$.
4. $V_\epsilon = \{(z, w): |z| < 1+\epsilon, |w| < 1+\epsilon, K(z, w) = 0\}$
5. $C(z, w)$ a function analytic on V_ϵ .

Then there exists $0 \leq \epsilon' \leq \epsilon$ and $\Phi(z), \Psi(w)$ analytic for $|z| < 1+\epsilon', |w| < 1+\epsilon'$ such that

$$\Phi(z) + \Psi(w) = C(z, w) \quad \text{for } (z, w) \in V_{\epsilon'}$$

Further, $\Phi(z)$, and $\Psi(w)$ are *unique* up to an additive constant.

We now show that the system equations given in (5.40) through (5.43) (when $K(z, w) = 0$) satisfies the first three criteria of the splitting theorem.

For the threshold problem, $K(z, w)$ (given in equation (5.13)) is a polynomial of degree two in the z and w variables. Thus, $K(z, w)$ is analytic in the polydisc (z, w) $|z| \leq 1+\epsilon, |w| \leq 1+\epsilon$ for any $\epsilon \geq 0$.

The following two definitions will be used throughout the next two sections.

DEFINITION 1: Let $w(z)$ denote the set of roots $\{w: K(w, z) = 0, |w| < 1\}$ for a given z .

DEFINITION 2: Let $z(w)$ denote the set of roots $\{z: K(w, z) = 0, |z| < 1\}$ for a given w .

The following two theorems prove that the cardinality of $w(z)$ ($z(w)$) is one when $|z| = 1+\epsilon$ ($|w| = 1+\epsilon$), satisfying criteria two and three of the splitting theorem.

THEOREM 5.1: For all $z, |z| = 1+\epsilon$, there exists a unique root $w(z)$, $|w(z)| < 1$, such that $K(z, w(z)) = 0$.

PROOF: We use Rouché's theorem to show that there is one root w_z in the w unit disk for a given $|z| < 1+\epsilon$.

$$\begin{aligned} K(z, w) &= \lambda_1(1-z) + \mu_1(1-1/z) + \lambda_2(1-w) + \mu_2(1-1/w) + \alpha(1-w/z) = 0 \\ &= (\lambda_2 + \alpha/z)w^2 - (\lambda_2 + \mu_2 + \alpha + \lambda_1(1-z) + \mu_1(1-1/z))w + \mu_2 = 0 \end{aligned}$$

Define the following two functions:

$$f(z, w) = w(\lambda_2 + \mu_2 + \alpha + \lambda_1(1-z) + \mu_1(1-1/z))$$

$$g(z, w) = (\lambda_2 + \alpha/z)w^2 + \mu_2$$

and compare the absolute values of f and g on the contour $|w| = 1$. Then

$$\begin{aligned} |g(z, w)| &= |(\lambda_2 + \alpha/z)w^2 + \mu_2| \\ &\leq \mu_2 + |w^2| |\lambda_2 + \alpha/z| \\ &\leq \mu_2 + \lambda_2 + \frac{\alpha}{|z|} \end{aligned}$$

where we used the triangle inequality to obtain the upper bound.

$$\begin{aligned} |f(z, w)| &= |w(\lambda_2 + \mu_2 + \alpha + \lambda_1(1-z) + \mu_1(1-1/z))| \\ &= |(\lambda_2 + \mu_2 + \alpha + \lambda_1(1-z) + \mu_1(1-1/z))| \quad \text{on } |w|=1 \\ &\geq |\lambda_2 + \mu_2 + \lambda_1 + \mu_1 + \alpha - |\lambda_1 z + \frac{\mu_1}{z}|| \\ &\geq \lambda_2 + \mu_2 + \lambda_1 + \mu_1 + \alpha - |\lambda_1 z + \frac{\mu_1}{z}| \\ &\geq \lambda_2 + \mu_2 + \lambda_1 + \mu_1 + \alpha - \lambda_1 |z| - \frac{\mu_1}{|z|} \end{aligned}$$

The last set of inequalities were derived using $|x-y| \geq ||x| - |y||$. We need to show

that $|f(z, w)| > |g(z, w)|$ on $|w|=1$, which is:

$$\lambda_2 + \mu_2 + \lambda_1 + \mu_1 + \alpha - \lambda_1 |z| - \frac{\mu_1}{|z|} > \lambda_2 + \mu_2 + \frac{\alpha}{|z|}$$

which implies

$$\mu_1 \left(1 - \frac{1}{|z|}\right) + \alpha \left(1 - \frac{1}{|z|}\right) > \lambda_1 (|z| - 1)$$

$$\mu_1 + \alpha > \lambda_1 |z| = \lambda_1 (1 + \epsilon)$$

which holds if we choose $0 < \epsilon < \frac{\mu_1 + \alpha - \lambda_1}{\lambda_1}$. By Rouché's theorem the number of zeroes of $f + g$ (inside $|w|=1$) is equal to the number of zeroes of f (inside $|w|=1$) which is one. QED.

The above proof works for $0 < \epsilon < \frac{\mu_1 + \alpha - \lambda_1}{\lambda_1}$. The same result holds (i.e. we can apply Rouché's theorem to prove there is one w root in the unit disk) for $\epsilon=0$ since $|f(z, w)| > \lambda_2 + \mu_2 + \alpha \geq |g(z, w)|$ for all $|z|=1$ except at $z=1$. At $z=1$ the two roots of w are $w_1 = 1$ and $w_2 = \frac{\mu_2}{\alpha + \lambda_2}$. If $\frac{\mu_2}{\alpha + \lambda_2} < 1$ then there are two roots in (and on) the unit circle.

THEOREM 5.2: For all $|w|=1$, $w \neq 1$, there exists a unique root $z(w)$, $|z(w)| < 1$, such that $K(z(w), w) = 0$.

PROOF: We write $K(z, w) = 0$ as a quadratic in z .

$$\lambda_1 z^2 - z(\lambda_1 + \mu_1 + \alpha + \lambda_2(1-w) + \mu_2(1-1/w)) + \mu_1 + \alpha w = 0$$

Define the following functions:

$$f(z, w) = z(\lambda_1 + \mu_1 + \alpha + \lambda_2(1-w) + \mu_2(1-1/w))$$

$$g(z, w) = \lambda_1 z^2 + \mu_1 + \alpha w$$

On the contour $|z|=1$, we have for fixed w , $|w|=1, w \neq 1$

$$|g(z, w)| = |\lambda_1 z^2 + \mu_1 + \alpha w|$$

$$\leq |\lambda_1 z^2| + |\mu_1 + \alpha w|$$

$$\leq \lambda_1 + |\mu_1 + \alpha w|$$

$$< \lambda_1 + \mu_1 + \alpha |w| \quad \text{on } |w|=1, w \neq 1$$

$$< \lambda_1 + \mu_1 + \alpha$$

$$|f(z, w)| = |z(\lambda_1 + \mu_1 + \alpha + \lambda_2(1-w) + \mu_2(1-1/w))|$$

$$\geq |\lambda_1 + \mu_1 + \alpha + \lambda_2 + \mu_2 - |\lambda_2 w + \frac{\mu_2}{w}||$$

$$\geq \lambda_1 + \mu_1 + \alpha + \lambda_2 + \mu_2 - |\lambda_2 w + \frac{\mu_2}{w}|$$

$$\geq \lambda_1 + \mu_1 + \alpha + \lambda_2 + \mu_2 - \lambda_2 |w| - \frac{\mu_2}{|w|}$$

$$\geq \lambda_1 + \mu_1 + \alpha > |g(z, w)|$$

By Rouché's theorem, the number of zeroes of $f+g$ (inside $|z|=1$) is equal to the number of zeroes of f (inside $|z|=1$) which is one. For $w=1$, solving $K(z, 1)=0$ yields two roots $z_1=1$ and $z_2=\frac{\mu_1+\alpha}{\lambda_1}$. The root z_2 must lie outside the unit disk ($|z_2|>1$)

since the stability condition for the first queue dictates that $\lambda_1 < \mu_1 + \alpha$. QED.

We will show the solution technique using a "splitter" subroutine [Tayl88] for the general threshold problem ($T > 0$). However, we prove (in section 5.2.5) the correctness of condition four of the splitting theorem only for the restrictive case $T=1$ (plus a restriction on the values of the parameters). Numerical evidence will be presented for the other range of parameters and also for $T > 1$; this evidence will allow us to conjecture that the functions we need to split are analytic on the zero set V_ϵ for all $T > 0$. We have the following equation for general T , from the previous section.

$$K(z,w)G(z,w) =$$

$$z^T \left[\mu_2(1-1/w)P^T(z,0) + B^T(z,w)P^T(0,w) + \sum_{i=0}^{T-1} C_i(z,w)P_{i,0} + \sum_{j=0}^{T-1} C_j(z,w)P_{0,j} \right]$$

where $B^T(z,w)$, $C_i(z,w)$, and $C_j(z,w)$ are defined in equations (5.41) through (5.43). When $K(z,w)=0$ ($|z| \leq 1, |w| \leq 1$) the right hand side must vanish as well, otherwise the probability generating function, $G(z,w)$ will not be analytic. We then have the following equation holding on the zero set $\{ (z,w): K(z,w)=0, |w| \leq 1, |z| \leq 1 \}$:

$$P^T(z,0) + \frac{B^T(z,w)}{\mu_2(1-1/w)} P^T(0,w) = \sum_{i=0}^{T-1} C_i^*(z,w)P_{i,0} + \sum_{j=0}^{T-1} C_j^*(z,w)P_{0,j} \quad (5.44)$$

where $C_i^*(z,w)$ ($C_j^*(z,w)$) is $C_i(z,w)$ ($C_j(z,w)$) divided by $\mu_2(1-1/w)$. There are two unknown functions $P^T(z,0)$, and $P^T(0,w)$, and $2T-1$ unknown constants $P_{0,0}$; $P_{i,0}$ ($0 < i < T$); $P_{0,j}$ ($0 < j < T$).

STEP 1: Take the logarithm of the function $\frac{B^T(z,w)}{\mu_2(1-1/w)}$ on the zero set $K(z,w)=0$ and use the splitting subroutine on the result to obtain two functions $\psi(w), \phi(z)$ that are analytic in their respective unit disks and are the unique solution to

$$\log \left[\frac{B^T(z, w)}{\mu_2(1-1/w)} \right] = \psi(w) - \phi(z)$$

Assuming that we can apply the logarithm and split the result into $\psi(w)$, and $\phi(z)$ we then have

$$\frac{B^T(z, w)}{\mu_2(1-1/w)} = \frac{e^{\psi(w)}}{e^{\phi(z)}}$$

It turns out that there is an extra complication in applying the logarithm function. In order for the log function to be single valued we need to remove T zeroes from $\frac{B^T(z, w)}{\mu_2(1-1/w)}$. These zeroes are located in the unit disk in the complex w plane. Let us

rewrite equation (5.44) by multiplying and dividing the T zeroes through.

$$P^T(z, 0) + \prod_{m=1}^T (w-w_m) B^*(z, w) P^T(0, w) = \sum_{i=0}^{T-1} C_i^*(z, w) P_{i,0} + \sum_{j=0}^{T-1} C_j^*(z, w) P_{0,j} \quad (5.45)$$

where

$$B^*(z, w) = \frac{B^T(z, w)}{\mu_2(1-1/w) \prod_{m=1}^T (w-w_m)}$$

We now apply the logarithm function to $B^*(z, w)$ and split the result into $\psi(w)$, and $\phi(z)$. Replacing $B^*(z, w)$ in equation (5.45) with $\psi(w)$ and $\phi(z)$ yields

$$e^{\phi(z)} P^T(z, 0) + \prod_{m=1}^T (w-w_m) e^{\psi(w)} P^T(0, w) = \sum_{i=0}^{T-1} e^{\phi(z)} C_i^*(z, w) P_{i,0} + \sum_{j=0}^{T-1} e^{\phi(z)} C_j^*(z, w) P_{0,j} \quad (5.46)$$

STEP 2: We split the right hand side of equation (5.46) using the "splitter" subroutine to obtain:

$$\sum_{i=0}^{T-1} e^{\phi(z)} C_i^*(z, w) P_{i,0} + \sum_{j=0}^{T-1} e^{\pi(z)} C_j^*(z, w) P_{0,j} = \quad (5.47)$$

$$\sum_{i=1}^{T-1} \left[\Phi_{i,0}(z) + \Psi_{i,0}(w) \right] P_{i,0} + \sum_{j=1}^{T-1} \left[\Phi_{0,j}(z) + \Psi_{0,j}(w) \right] P_{0,j} + \left[\Phi_{0,0}(z) + \Psi_{0,0}(w) \right] P_{0,0} + K - K$$

which yields the unique functions $(\Phi_{i,0}(z), \Phi_{0,j}(z), \Psi_{i,0}(w), \Psi_{0,j}(w))$, up to an additive constant K , that solve the above equation. We include the $K - K$ term in the above equation to show that there is an extra constant that must be solved for when we separate the w terms from the z terms. Rewriting equation (5.46) with the functions obtained in equation (5.47) yields:

$$e^{\Phi(z)} P^T(z, 0) + \prod_{m=1}^T (w - w_m) e^{\Psi(w)} P^T(0, w) = \quad (5.48)$$

$$\sum_{i=1}^{T-1} \left[\Phi_{i,0}(z) + \Psi_{i,0}(w) \right] P_{i,0} + \sum_{j=1}^{T-1} \left[\Phi_{0,j}(z) + \Psi_{0,j}(w) \right] P_{0,j} + \left[\Phi_{0,0}(z) + \Psi_{0,0}(w) \right] P_{0,0} + K - K$$

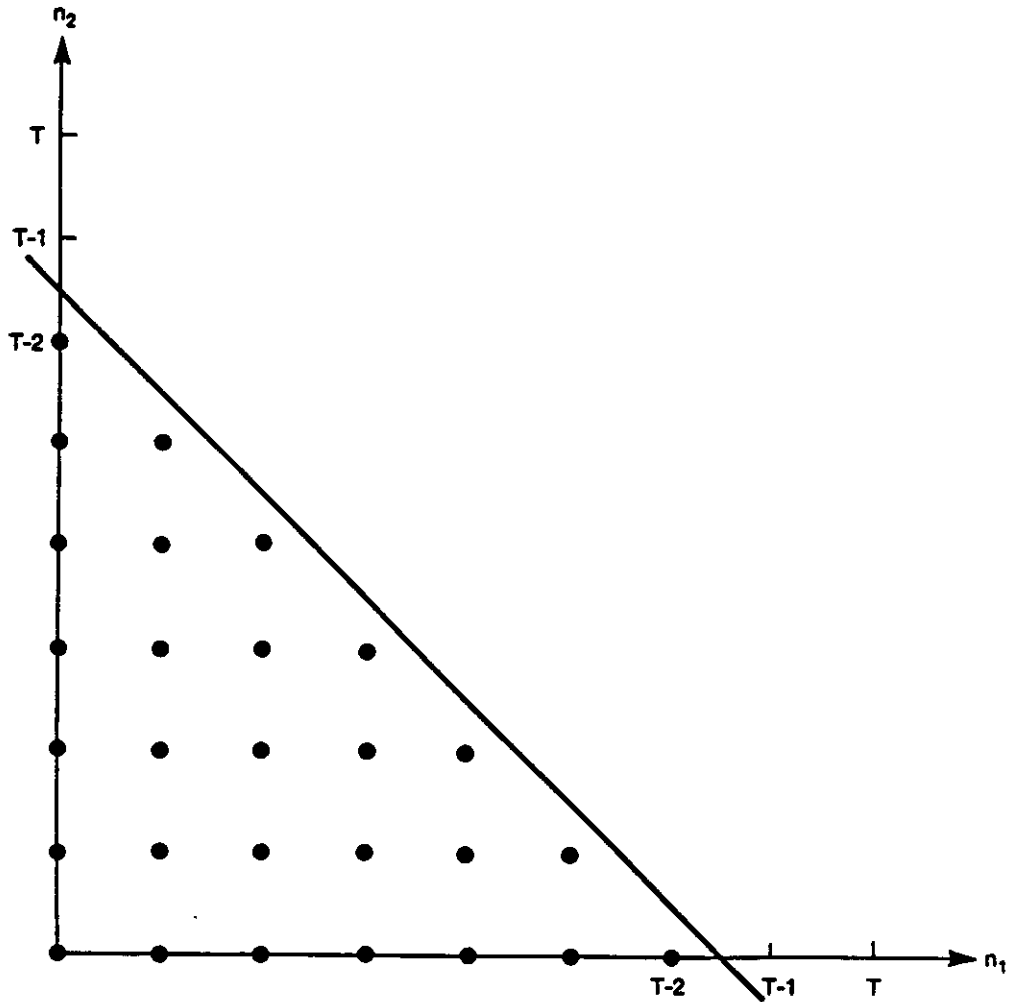
By the splitting theorem the right hand side is the unique way to split $C(z, w)$ into two analytic functions $\Phi(z)$ and $\Psi(w)$ such that $\Phi(z) + \Psi(w) = C(z, w)$ on the zero set $K(z, w) = 0$. However, equation (5.48) also provides a way to split $C(z, w)$ into two analytic functions $\Phi'(z) = e^{\Phi(z)} R(z)$ and $\Psi'(w) = \prod_{m=1}^T (w - w_m) e^{\Psi(w)} P^T(0, w)$ such that $\Phi'(z) + \Psi'(w) = C(z, w)$ on the zero set. By the uniqueness principle of the splitting theorem we must have $\Phi'(z) = \Phi(z)$ and $\Psi'(w) = \Psi(w)$, i.e.

$$e^{\Phi(z)} P^T(z, 0) = \sum_{i=0}^{T-1} \Phi_{i,0}(z) P_{i,0} + \sum_{j=0}^{T-1} \Phi_{0,j}(z) P_{0,j} + K \quad (5.49)$$

$$\prod_{m=1}^T (w - w_m) e^{\Psi(w)} P^T(0, w) = \sum_{i=0}^{T-1} \Psi_{i,0}(w) P_{i,0} + \sum_{j=0}^{T-1} \Psi_{0,j}(w) P_{0,j} - K \quad (5.50)$$

Solving for $P^T(0, w)$ we note that there are now $2T$ unknown constants

$K; P_{0,0}; P_{i,0} \ 0 < i < T; P_{0,j} \ 0 < j < T$, and T equations (one for each zero w_m). We can get an equal number of equations and unknown constants by considering the triangle zone of states shown in figure (5.4) along with their flow equations. From these states, and the T equations above a linear set of equations can be set up to solve for all the unknowns by introducing the flow equations for the states $P_{i,j} \ 0 \leq i+j < T-1$.



Triangle Zone of States $0 \leq i+j < T-1$
Figure (5.4)

The number of unknowns is now $1 + \frac{T(T+1)}{2}$ (K and states $P_{i,j}$ $0 \leq i+j < T$) while the number of independent equations is also $\frac{T(T+1)}{2}$ (which is the sum of the T equations from the zeroes w_m in equation (5.50), plus the $\frac{T(T-1)}{2}$ flow equations we introduced from the triangle zone). The last equation (independent of the other equations) is that the total probability is equal to one. Note also that

$$P^T(0,1) + \sum_{j=0}^{T-1} P_{0,j} = P(0,1)$$

The probability $P(0,1)$ ($\Pr\{\text{no jobs in the first queue}\}$) is easily calculated from equation (5.18). Thus the last equation to the set of independent equations is equation (5.50) when $w=1$. Solving the linear set of equations yields numerical values for the constants $P_{i,j}$ $0 \leq i+j < T$ and the boundary functions $P(z,0)$, and $P(0,w)$ (for $|z| \leq 1$ and $|w| \leq 1$). This information is enough to derive any state probability, $P_{i,j}$, by using equations (5.8) through (5.10) to determine $P_i(w)$ and its j^{th} derivative at $w=0$. We also have enough numerical information to derive the average number in system from equation (5.30).

The above procedure to obtain the average number is a general one that is used for many problems of this type. However, for each problem it must be shown that the solution technique yields a unique solution. If there is more than one solution then our queueing problem has more than one solution and must be nonergodic.

5.2.5) Proof of Uniqueness for $T=1$

This section contains a proof for the uniqueness of a solution to our Boundary Value problem along with a proof of existence of such a solution. We limit our attention to the case where the threshold, $T = 1$. We discuss the general case $T \geq 1$ in the

next section. The following material includes proofs that the functions we split satisfy the criteria of the Splitting Theorem listed in section (5.2.4).

For the case $T=1$ we have the following Boundary Value problem from equations (5.40) through (5.43).

$$P^1(z, 0) + \frac{B^1(z, w)}{\mu_2(1-1/w)} P^1(0, w) = P_{0,0} C(z, w) \quad (5.51)$$

Where $C(z, w)$ is the sum of the coefficients for $P_{0,0}$ in equations (5.42) and (5.43). We provide a constructive proof of uniqueness and existence by stepping through the technique presented in the previous section. In order to take an analytic logarithm (step 1) of $\frac{B^1(z, w)}{\mu_2(1-1/w)}$ we must show (among other things) that the function is analytic over the domain of interest. For the remainder of this section we use the following definition:

$$B(z, w) = \frac{B^1(z, w)}{\mu_2(1-1/w)} = \frac{w^2[(\lambda_1 + m(w))(\mu_1 + \lambda_1(1-z) + m(w)) - \lambda_1 \mu_1]}{\mu_1 \mu_2(1-w)}$$

with

$$m(w) = \lambda_2(1-w) + \mu_2(1-1/w)$$

The function on the right hand side is a polynomial (in z and w) divided by a polynomial. $B(z, w)$ is analytic over the zero set, V_ϵ , if the polynomial in the denominator does not vanish anywhere in V_ϵ .

The factors $(\lambda_1 + m(w))$ and $(\mu_1 + \lambda_1(1-z) + m(w))$ each have pole at $w=0$. However, the w^2 term in front cancels both poles. At $w=1$ and $z=1$ (on the zero set) both the numerator and the denominator vanish. We now show that this point is not a pole. Rewriting $B(z, w)$ we have

$$B(z, w) = \frac{w^2}{\mu_1 \mu_2} \left[\lambda_1^2 \frac{(1-z)}{(1-w)} + \frac{m(w)}{(1-w)} (\mu_1 + \lambda_1 + \lambda_1(1-z) + m(w)) \right]$$

From the kernel $K(z, w) = 0$ we have

$$\lambda_1(1-z) + \mu_1(1-1/z) + \lambda_2(1-w) + \mu_2(1-1/w) + \alpha(1-w/z) = 0$$

By adding and subtracting α/z , and grouping terms we obtain:

$$\frac{(1-z)}{(1-w)} = \frac{(\mu_2/w - \lambda_2 - \alpha/z)}{\lambda_1 - \frac{(\mu_1 + \alpha)}{z}} \quad (5.52)$$

at $w=1, z=1$ we have

$$\left. \frac{(1-z)}{(1-w)} \right|_{w,z=1} = \frac{\mu_2 - (\lambda_2 + \alpha)}{\lambda_1 - (\mu_1 + \alpha)}$$

and

$$B(1, 1) = \frac{1}{\mu_1 \mu_2} \left[\lambda_1^2 \frac{\mu_2 - (\lambda_2 + \alpha)}{\lambda_1 - (\mu_1 + \alpha)} + (\lambda_2 - \mu_2)(\mu_1 + \lambda_1) \right] \quad (5.53)$$

Since $\lambda_1 < \mu_1 + \alpha$ for stability (implied from condition (S1)) we see that $B(1, 1) \neq \infty$. The function $B(z, w)$ is thus analytic on the zero set V_ϵ and yet may have zeroes on that domain. This is a problem since the logarithm of $B(z, w)$ is not analytic at a point (z, w) (in V_ϵ) where $B(z, w) = 0$. The number of zeroes of $B(z, w)$, if any, over the zero set V_ϵ can be determined by computing a quantity known as the winding number (or index) of the function around the boundary of the zero set ($|w|=1$, and $|z|=1$). The zeroes, if any, can be removed from $B(z, w)$ and then the logarithm function may be applied.

The winding number, or index, of a function $f(x)$ on contour Ω is the number of times $f(x)$ wraps around the origin as x traverses Ω . Let $W_\Omega^\#(f(x))$ denote the winding number of a function $f(x)$ as x traverses Ω . From [Chur74], if Ω is a simple

closed curve and f is analytic within and on Ω (except for poles interior to Ω) and does not have any zeroes on Ω then

$$W_{\Omega}^{\#}(f(x)) = \frac{1}{2\pi i} \int_{\Omega} \frac{f'(x)}{f(x)} dx = N_z - N_p$$

where N_z (N_p) is the number of zeroes (poles) of f interior to Ω . In this section we provide proofs for the winding numbers listed in the table in figure (5.5).

$f(z, w)$	$W_{ w =1}^{\#}(f(z, w))$	$W_{ z =1}^{\#}(f(z, w))$	
		$\lambda_1 + \mu_1 < \alpha(1 - \frac{\mu_2}{\lambda_2 + \alpha})$	$\lambda_1 + \mu_1 > \alpha(1 - \frac{\mu_2}{\lambda_2 + \alpha})$
$\frac{B^1(z, w)}{1-w}$	0	0	-1
$\frac{wB^1(z, w)}{1-w}$	1	0	-1
$\frac{wB^1(z, w)}{(1-w)(w-p)}$	0	0	0

Table of Winding Numbers for $B(z, w)$
Figure (5.5)

We now prove that the winding number $W_{|w|=1}^{\#}B(z(w), w) = 1$.

THEOREM (5.3):

$$W_{|w|=1}^{\#}(B(z(w), w)) =$$

$$W_{|w|=1}^{\#} \left[\frac{w^2}{\mu_1 \mu_2} \left[\lambda_1^2 \frac{(1-z)}{(1-w)} + \frac{m(w)}{(1-w)} (\mu_1 + \lambda_1 + \lambda_1(1-z) + m(w)) \right] \right] = 1$$

Where $z=z(w)$ is defined by $K(z, w)=0$.

PROOF: The winding number of a product is equal to the sum of the individual winding numbers (i.e. $W_{\Omega}^{\#}(fg) = W_{\Omega}^{\#}(f) + W_{\Omega}^{\#}(g)$). Since $W_{|w|=1}^{\#}(w) = 1$ we need to show that

$$W_{|w|=1}^{\#} \left[\frac{w}{\mu_1 \mu_2} \left[\lambda_1^2 \frac{(1-z)}{(1-w)} + \frac{m(w)}{(1-w)} (\mu_1 + \lambda_1 + \lambda_1(1-z) + m(w)) \right] \right]$$

$$= W_{|w|=1}^{\#} \left[\frac{B^1(z, w)}{1-w} \right] = 0$$

Assume that the above winding number is not zero, then the function $\frac{B^1(z, w)}{(1-w)}$ must touch the positive real axis for some value w' , $|w'|=1$, and $|z'| \leq 1$.

CASE 1: $|w'|=1$, and $w' \neq 1$. Assume $\frac{\mu_1 \mu_2 B^1(z', w')}{1-w'} = C$ where C is real and $C \geq 0$.

Then

$$(\lambda_1 + m(w'))(\mu_1 + \lambda_1(1-z') + m(w')) - \lambda_1 \mu_1 = -C(1-1/w')$$

We now isolate the $(1-z')$ term from the rest, leaving:

$$\lambda_1(1-z') = \frac{\lambda_1 \mu_1 - C(1-1/w')}{\lambda_1 + m(w')} - m(w') - \mu_1$$

Looking at the real parts of both sides yields:

$$\lambda_1(1-\text{Re}(z')) = \text{Re} \left[\frac{\lambda_1 \mu_1 - C(1-1/w')}{\lambda_1 + m(w')} \right] - \text{Re}(m(w')) - \mu_1$$

Looking at the first term on the right hand side more closely, we can find an upper bound for it (by multiplying and dividing by the complex conjugate of $\lambda_1 + m(w')$).

$$\text{Re} \left[\frac{\lambda_1 \mu_1 - C(1-1/w')}{\lambda_1 + m(w')} \right]$$

$$= \frac{\text{Re}(\lambda_1 \mu_1 - C(1-1/w')) \text{Re}(\lambda_1 + m(w')) + \text{Im}(\lambda_1 \mu_1 - C(1-1/w')) \text{Im}(\lambda_1 + m(w'))}{|\lambda_1 + m(w')|^2}$$

$$= \frac{(\lambda_1 \mu_1 - C(1 - \operatorname{Re}(w')) \operatorname{Re}(\lambda_1 + m(w')) - C \operatorname{Im}^2(w')(\mu_2 - \lambda_2))}{|\lambda_1 + m(w')|^2}$$

The right hand side of the equation is maximized when $C=0$ since $\mu_2 > \lambda_2$ for stability (condition (S2)). Furthermore, we have that

$$\begin{aligned} |\lambda_1 + m(w')|^2 &= \operatorname{Re}^2(\lambda_1 + m(w')) + \operatorname{Im}^2(\lambda_1 + m(w')) \\ &> \lambda_1 \operatorname{Re}(\lambda_1 + m(w')) \end{aligned}$$

since $m(w') = \lambda_2(1 - w') + \mu_2(1 - 1/w') > 0$ when $|w'|=1$. We then have the following upper bound:

$$\operatorname{Re} \left[\frac{\lambda_1 \mu_1 - C(1 - 1/w')}{\lambda_1 + m(w')} \right] < \frac{\lambda_1 \mu_1 \operatorname{Re}(\lambda_1 + m(w'))}{\lambda_1 \operatorname{Re}(\lambda_1 + m(w'))} = \mu_1$$

Thus $\lambda_1(1 - z')$ is also bounded from above:

$$\lambda_1(1 - \operatorname{Re}(z')) < \mu_1 - \mu_1 - (\mu_2 + \lambda_2)(1 - \operatorname{Re}(w')) < 0$$

since $1 - \operatorname{Re}(w') > 0$ for $|w'|=1, w' \neq 1$. However, this inequality contradicts Theorem 5.2 which states that for any $w, |w|=1$, there exists a root $z(w)$ inside the unit disk, hence $1 - \operatorname{Re}(z(w)) \geq 0$. Therefore, the assumption that $W_{|w|=1}^{\#} \left[\frac{B^1(z(w), w)}{1 - w} \right] \neq 0$ is incorrect for case 1.

CASE 2: $w'=1$ and $z(w')=1$ (CASE 1 included all $|w'|=1$ except the point $w'=1$).

From equation (5.53) we know that

$$\begin{aligned} \frac{w B^1(z, w)}{(1 - w)} \Big|_{w, z=1} &= \frac{1}{\mu_1 \mu_2} \left[\lambda_1^2 \frac{\mu_2 - (\lambda_2 + \alpha)}{\lambda_1 - (\mu_1 + \alpha)} + (\lambda_2 - \mu_2)(\mu_1 + \lambda_1) \right] \\ &= \frac{1}{\mu_1 \mu_2} \left[(\lambda_2 - \mu_2) \left[\mu_1 + \lambda_1 + \frac{\lambda_1^2}{(\mu_1 + \alpha) - \lambda_1} \right] + \frac{\lambda_1^2 \alpha}{(\mu_1 + \alpha) - \lambda_1} \right] \end{aligned}$$

The right hand side is negative if

$$\frac{\lambda_1^2 \alpha}{(\mu_1 + \alpha) - \lambda_1} < (\mu_2 - \lambda_2) \left[\mu_1 + \lambda_1 + \frac{\lambda_1^2}{(\mu_1 + \alpha) - \lambda_1} \right]$$

or

$$\mu_2 - \lambda_2 > \alpha \left[\frac{\frac{\lambda_1^2}{(\mu_1 + \alpha) - \lambda_1}}{\mu_1 + \lambda_1 + \frac{\lambda_1^2}{(\mu_1 + \alpha) - \lambda_1}} \right] \quad (5.54)$$

This last inequality is the stability condition (S2) for the second queue. Together with case 1 we have that $\frac{B^1(z, w)}{(1-w)}$ never touches the positive (≥ 0) real axis for any $w', |w'|=1$. Therefore, $W^*_{|w|=1} \left[\frac{B^1(z, w)}{(1-w)} \right] = 0$, and $W^*_{|w|=1} (B(z, w)) = 1$. QED.

The stability criterion for the second queue is that the external input rate, λ_2 , plus the internal transfer rate (call it f) must be less than μ_2 . The internal input rate f can be calculated as follows.

$$f = \alpha \sum_{i=2}^{\infty} Pr\{i \text{ in } Q1\} = \alpha P_0 \left(\frac{\lambda_1}{\mu_1} \right) \sum_{i=2}^{\infty} \left[\frac{\lambda_1}{\mu_1 + \alpha} \right]^{i-1} \quad \text{from equation (5.20)}$$

$$= \frac{\alpha P_0 \left(\frac{\lambda_1}{\mu_1} \right) \left[\frac{\lambda_1}{\mu_1 + \alpha} \right]}{1 - \frac{\lambda_1}{\mu_1 + \alpha}} = \frac{\alpha \left(\frac{\lambda_1}{\mu_1} \right) \left[\frac{\lambda_1}{\mu_1 + \alpha} \right]}{1 - \frac{\lambda_1}{\mu_1 + \alpha} + \frac{\lambda_1}{\mu_1}}$$

$$= \frac{\alpha \lambda_1^2}{(\mu_1 + \lambda_1)(\mu_1 + \alpha) - \lambda_1 \mu_1} = \frac{\frac{\alpha \lambda_1^2}{(\mu_1 + \alpha) - \lambda_1}}{\frac{\mu_1 ((\mu_1 + \alpha) - \lambda_1) + \lambda_1 ((\mu_1 + \alpha) - \lambda_1) + \lambda_1^2}{(\mu_1 + \alpha) - \lambda_1}}$$

$$= \alpha \left[\frac{\frac{\alpha \lambda_1^2}{(\mu_1 + \alpha) - \lambda_1}}{\mu_1 + \lambda_1 + \frac{\lambda_1^2}{(\mu_1 + \alpha) - \lambda_1}} \right]$$

Then the stability condition requires that

$$\mu_2 - \lambda_2 > \alpha \left[\frac{\frac{\lambda_1^2}{(\mu_1 + \alpha) - \lambda_1}}{\mu_1 + \lambda_1 + \frac{\lambda_1^2}{(\mu_1 + \alpha) - \lambda_1}} \right]$$

the same inequality found in (5.54).

The second boundary of the zero set V_ε is the set of points $\{(w, z) : |z|=1, |w|<1, K(z, w)=0\}$. We show that the winding number of $B(z, w)$ on the contour $|z|=1$ is either 0 or -1 depending on the parameters. In order to prove the assertion about the winding number, we first need to prove the following lemmas.

LEMMA 5.1: If $\frac{\mu_2}{\lambda_2 + \alpha} < 1$ then there exists a unique root $w(z)$ defined by $K(z, w)=0$

such that $|w(z)| < \frac{\mu_2}{\lambda_2 + \alpha}$ when $|z|=1, z \neq 1$. At $z=1, w_1 = \frac{\mu_2}{\lambda_2 + \alpha}$ and $w_2=1$.

PROOF: From the kernel equation $K(z, w)=0$, let

$$f(z, w) = w(\lambda_2 + \mu_2 + \alpha + \lambda_1(1-z) + \mu_1(1-1/z))$$

$$g(z, w) = (\lambda_2 + \alpha/z)w^2 + \mu_2$$

We are interested in the contour $|w| = \frac{\mu_2}{\lambda_2 + \alpha}$, and a fixed z such that $|z|=1, z \neq 1$.

$$|f(z, w)| = \frac{\mu_2}{\lambda_2 + \alpha} |\lambda_1 + \lambda_2 + \mu_1 + \mu_2 + \alpha - \lambda_1 z - \mu_1/z|$$

$$\geq \frac{\mu_2}{\lambda_2 + \alpha} (\lambda_1 + \lambda_2 + \mu_1 + \mu_2 + \alpha - (\lambda_1 + \mu_1)) \quad \text{by the triangle inequality}$$

$$\geq \mu_2 \left(1 + \frac{\mu_2}{\lambda_2 + \alpha}\right)$$

$$|g(z, w)| = |w^2(\lambda_2 + \alpha/z) + \mu_2|$$

$$\leq \mu_2 + |w^2(\lambda_2 + \alpha/z)|$$

$$\leq \mu_2 + \left(\frac{\mu_2}{\lambda_2 + \alpha}\right)^2 |\lambda_2 + \alpha/z|$$

$$< \mu_2 + \left(\frac{\mu_2}{\lambda_2 + \alpha}\right)^2 (\lambda_2 + \alpha) \quad \text{on } |z|=1, z \neq 1$$

$$< \mu_2 \left(1 + \frac{\mu_2}{\lambda_2 + \alpha}\right)$$

therefore $|f(z, w)| > |g(z, w)|$ on the contour $|w| = \frac{\mu_2}{\lambda_2 + \alpha}$. By Rouché's theorem

$f+g$ has the same number of zeroes as f inside $|w| = \frac{\mu_2}{\lambda_2 + \alpha}$ which is one. At $z=1$ we

obtain two roots from $K(1, w)=0$, $w_1 = \frac{\mu_2}{\lambda_2 + \alpha}$ and $w_2=1$. QED

The next lemma shows that for $|z|=1$ the real part of the w root (inside the unit disk) is strictly positive.

LEMMA (5.2): $\text{Re}(w(z)) > 0$ for $|z|=1$ and $K(z, w)=0$

PROOF: The kernel equation is:

$$\lambda_1(1-z) + \mu_1(1-1/z) + \lambda_2(1-w) + \mu_2(1-1/w) = 0$$

Taking the real part of both sides (for $|z|=1$) yields:

$$(\mu_1 + \lambda_1)(1 - \operatorname{Re}(z)) + \lambda_2(1 - \operatorname{Re}(w)) + \mu_2 \left[1 - \frac{\operatorname{Re}(w)}{|w|^2} \right] + \alpha(1 - \operatorname{Re}(w)\operatorname{Re}(z) - \operatorname{Im}(w)\operatorname{Im}(z)) = 0$$

Solving for $\operatorname{Re}(w)$ leaves

$$\operatorname{Re}(w) = \frac{(\mu_1 + \lambda_1)(1 - \operatorname{Re}(z)) + \lambda_2 + \mu_2 + \alpha(1 - \operatorname{Im}(z)\operatorname{Im}(w))}{\lambda_2 + \frac{\mu_2}{|w|^2} + \alpha \operatorname{Re}(z)}$$

From Lemma (5.1), if $\frac{\mu_2}{\lambda_2 + \alpha} \leq 1$ then

$$\lambda_2 + \frac{\mu_2}{|w|^2} + \alpha \operatorname{Re}(z) > \lambda_2 + \frac{\lambda_2 + \alpha}{|w|} + \alpha \operatorname{Re}(z) \geq \lambda_2 + \frac{\lambda_2 + \alpha}{|w|} - \alpha > 0$$

On the other hand, if $\frac{\mu_2}{\lambda_2 + \alpha} \geq 1$ then $|w| < 1$ when $|z|=1, z \neq 1$ and

$$\lambda_2 + \frac{\mu_2}{|w|^2} + \alpha \operatorname{Re}(z) > \lambda_2 + \mu_2 - \alpha \geq 0$$

Thus the denominator is positive for $|z|=1, z \neq 1$. From Theorem (5.2) we know that the product $\operatorname{Im}(z)\operatorname{Im}(w) \leq 1$ thus the numerator is strictly positive and $\operatorname{Re}(w) > 0$ for $|z|=1, z \neq 1$. At $z=1$ we have two roots $w_1 = \frac{\mu_2}{\lambda_2 + \alpha}$ and $w_2=1$, both having strictly positive real parts. QED.

The next lemma involves the real part of $m(w) = \lambda_2(1-w) + \mu_2(1-1/w)$ when $|z|=1$.

LEMMA (5.3): if $\frac{\mu_2}{\lambda_2 + \alpha} < 1$ then for $|z|=1, z \neq 1$,

$$\operatorname{Re}(m(w)) = \operatorname{Re}(\lambda_2(1-w) + \mu_2(1-1/w)) < -\alpha \left(1 - \frac{\mu_2}{\lambda_2 + \alpha} \right)$$

PROOF: From the kernel equation we have

$$-(\lambda_2(1-w) + \mu_2(1-1/w)) = \lambda_1(1-z) + \mu_1(1-1/z) + \alpha(1-w/z)$$

The real part of the above equation is

$$-\operatorname{Re}(m(w)) = (\lambda_1 + \mu_1)(1 - \operatorname{Re}(z)) + \alpha(1 - \operatorname{Re}(w/z))$$

We know that $(\lambda_1 + \mu_1)(1 - \operatorname{Re}(z)) \geq 0$ for $|z|=1$. From Lemma (5.1) we know that $|w| < 1$ when $|z|=1$, $z \neq 1$, however, dividing w by z (z on the unit circle) merely changes the angle, not the magnitude of w . Thus $\operatorname{Re}(w/z) < \frac{\mu_2}{\lambda_2 + \alpha}$ and

$$-\operatorname{Re}(m(w)) > \alpha\left(1 - \frac{\mu_2}{\lambda_2 + \alpha}\right)$$

and finally, our result

$$\operatorname{Re}(m(w)) = \operatorname{Re}(\lambda_2(1-w) + \mu_2(1-1/w)) < -\alpha\left(1 - \frac{\mu_2}{\lambda_2 + \alpha}\right)$$

QED

THEOREM (5.4):

$$W_{|z|=1}^\#(B(z, w)) = \begin{cases} 0 & \lambda_1 + \mu_1 < \alpha\left(1 - \frac{\mu_2}{\lambda_2 + \alpha}\right) \\ -1 & \lambda_1 + \mu_1 \geq \alpha\left(1 - \frac{\mu_2}{\lambda_2 + \alpha}\right) \end{cases}$$

PROOF: We first prove that $W_{|z|=1}^\#(B(z, w)) = 0$ when $\lambda_1 + \mu_1 < \alpha\left(1 - \frac{\mu_2}{\lambda_2 + \alpha}\right)$. Note

that $\frac{\mu_2}{\lambda_2 + \alpha} < 1$ is implied when the above inequality holds since λ_1 and μ_1 are both positive. We need two corollaries of Lemma (5.3).

COROLLARY (5.1): For $|z|=1$, $z \neq 1$, when $\lambda_1 + \mu_1 < \alpha\left(1 - \frac{\mu_2}{\lambda_2 + \alpha}\right)$ then

$$\operatorname{Re}(\lambda_1 + m(w)) < \lambda_1 - \alpha \left(1 - \frac{\mu_2}{\lambda_2 + \alpha}\right) < -\mu_1$$

COROLLARY (5.2): For $|z|=1$, $z \neq 1$, when $\lambda_1 + \mu_1 < \alpha \left(1 - \frac{\mu_2}{\lambda_2 + \alpha}\right)$ then

$$-\operatorname{Re}(\mu_1 + \lambda_1(1-z) + m(w)) = -\operatorname{Re} \left[\frac{\mu_1}{z} - \alpha(1-w/z) \right] \quad \text{from } K(z, w) = 0$$

$$> -\mu_1 + \alpha \left(1 - \frac{\mu_2}{\lambda_2 + \alpha}\right) > \lambda_1$$

Therefore, $\operatorname{Re}(\mu_1 + \lambda_1(1-z) + m(w)) < -\lambda_1$. Now for the proof of Theorem (5.4).

When $\frac{\mu_2}{\lambda_2 + \alpha} < 1$ then the function $1-w(z)$ never winds around the point zero

for $|z|=1$ since $\operatorname{Re}(1-w(z)) = 1 - \operatorname{Re}(w(z)) \geq 1 - \frac{\mu_2}{\lambda_2 + \alpha} > 0$. The same result holds for the function $w(z)$. Therefore,

$$W_{|z|=1}^\#(B(z, w)) = W_{|z|=1}^\# \left[\frac{w^2 [(\lambda_1 + m(w))(\mu_1 + \lambda_1(1-z) + m(w)) - \lambda_1 \mu_1]}{\mu_1 \mu_2 (1-w)} \right]$$

$$= W_{|z|=1}^\# \left[(\lambda_1 + m(w))(\mu_1 + \lambda_1(1-z) + m(w)) - \lambda_1 \mu_1 \right]$$

$$+ 2W_{|z|=1}^\#(w) - W_{|z|=1}^\#(1-w)$$

$$= W_{|z|=1}^\# \left[(\lambda_1 + m(w))(\mu_1 + \lambda_1(1-z) + m(w)) - \lambda_1 \mu_1 \right]$$

If the winding number of the parenthesized function on the right hand side is not zero the function must "touch" the negative real axis for some value of z' , $|z'|=1$ and $w' = w(z')$. Assume this is the case, then for $|z'|=1$, $z' \neq 1$, there is a real number $C > 0$ such that

$$(\lambda_1 + m(w'))(\mu_1 + \lambda_1(1-z') + m(w')) - \lambda_1\mu_1 = -C$$

then rearranging terms leaves

$$\mu_1 + \lambda_1(1-z') + m(w') = \frac{\lambda_1\mu_1 - C}{\lambda_1 + m(w')}$$

Taking the real part of the above equation yields:

$$\begin{aligned} \operatorname{Re}(\mu_1 + \lambda_1(1-z') + m(w')) &= (\lambda_1\mu_1 - C) \operatorname{Re} \left[\frac{1}{\lambda_1 + m(w')} \right] \\ &= \frac{(\lambda_1\mu_1 - C) \operatorname{Re}(\lambda_1 + m(w'))}{|\lambda_1 + m(w')|^2} \\ &\geq \frac{(\lambda_1\mu_1) \operatorname{Re}(\lambda_1 + m(w'))}{|\lambda_1 + m(w')|^2} \end{aligned}$$

The last inequality is a result of corollary (5.1). Also from Corollary (5.1) and $|\lambda_1 + m(w')|^2 = \operatorname{Re}^2(\lambda_1 + m(w')) + \operatorname{Im}^2(\lambda_1 + m(w'))$ we have

$$\operatorname{Re}(\mu_1 + \lambda_1(1-z') + m(w')) \geq \frac{(\lambda_1\mu_1) \operatorname{Re}(\lambda_1 + m(w'))}{\operatorname{Re}^2(\lambda_1 + m(w'))} = \frac{\lambda_1\mu_1}{\operatorname{Re}(\lambda_1 + m(w'))} > -\lambda_1$$

However, this last inequality contradicts Corollary (5.2). Therefore, the assumption is incorrect and $(\lambda_1 + m(w'))(\mu_1 + \lambda_1(1-z') + m(w')) - \lambda_1\mu_1$ does not touch the negative real axis for $|z'|=1, z' \neq 1$. At the point $z'=1$ we have

$$\begin{aligned} B^1(1, w(1)) &= \frac{1}{\mu_1\mu_2} \left[(\lambda_1 - \alpha(1 - \frac{\mu_2}{\lambda_2 + \alpha}))(\mu_1 - \alpha(1 - \frac{\mu_2}{\lambda_2 + \alpha})) - \lambda_1\mu_1 \right] \\ &= \frac{1}{\mu_1\mu_2} \left[\alpha(1 - \frac{\mu_2}{\lambda_2 + \alpha}) \left[\alpha(1 - \frac{\mu_2}{\lambda_2 + \alpha}) - (\lambda_1 + \mu_1) \right] \right] > 0 \end{aligned}$$

Therefore

$$((\lambda_1 + m(w'))(\mu_1 + \lambda_1(1-z') + m(w')) - \lambda_1 \mu_1) \neq -C$$

for any $C \geq 0$ and $z', |z'|=1$. Thus $W_{|z|=1}^\#(B(z, w))=0$.

The second part of the theorem assumes that $\lambda_1 + \mu_1 \geq \alpha(1 - \frac{\mu_2}{\lambda_2 + \alpha})$, let us call this inequality condition C_1 . We show that $W_{|z|=1}^\#(B(z, w)) = -1$ by going through a two step procedure. First, we prove that $B(z, w) \neq 0$ for any $|z|=1, w=w(z)$ when C_1 holds. Second, we prove that $W_{|z|=1}^\#(B(z, w)) = -1$ for a certain subset of the parameters $(\lambda_1, \mu_1, \lambda_2, \mu_2, \alpha)$ that satisfy C_1 . If $W_{|z|=1}^\#(B(z, w)) = -1$ for a subset of parameter values satisfying C_1 , then $W_{|z|=1}^\#(B(z, w)) = -1$ for all parameter values satisfying C_1 . The reason is that the function $B(z, w)$, which is continuous in the parameter space $(\lambda_1, \mu_1, \lambda_2, \mu_2, \alpha)$, never goes through the point zero. Hence, the winding number, which is a discrete function of $B(z, w)$, cannot change throughout the parameter subspace which satisfies condition C_1 .

In order to continue with the proof, it is necessary to investigate the image of $|z|=1$ in the complex w plane. The next lemma describes the behavior of $\text{Im}(w(z))$ for $|z|=1$. The different types of curves for $w(z)$ ($|z|=1$) are displayed in figure (5.6).

LEMMA (5.4): If $\text{Re}(w(z)) > \frac{\lambda_1 - \mu_1}{\alpha}$ for all $|z|=1$, then $\text{Im}(z)\text{Im}(w(z)) < 0$ for all $|z|=1$ except at $z=1, -1$ where $\text{Im}(z)=\text{Im}(w(z))=0$. If $Z = \{z: |z|=1, \text{Re}(w(z)) < \frac{\lambda_1 - \mu_1}{\alpha}\}$ is not empty then

$$\text{Im}(z)\text{Im}(w(z)) = \begin{cases} >0 & z \in Z \\ =0 & z=1, -1, z_1, \bar{z}_1 \\ <0 & \text{all other } z, |z|=1 \end{cases}$$

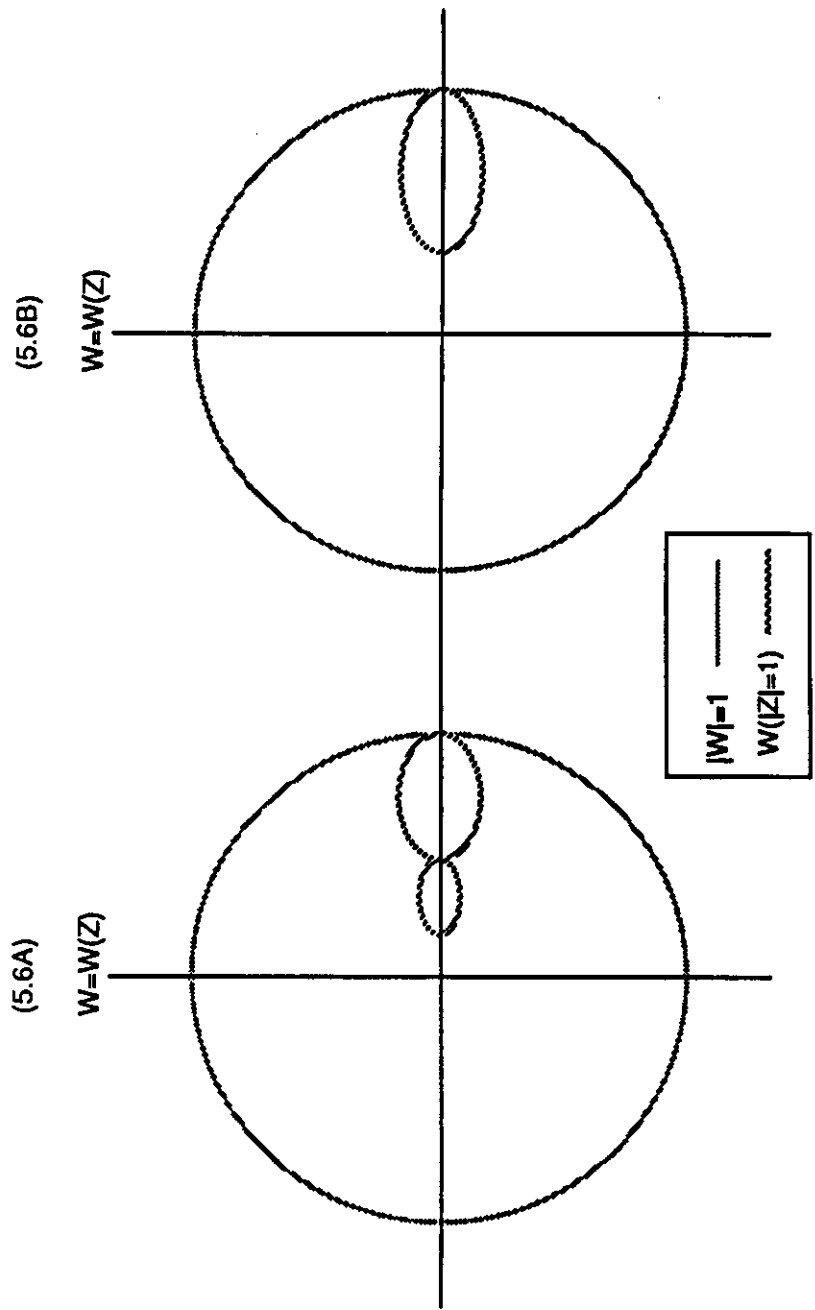


Figure (5.6)
Image of $w(z)$ for $|z| = 1$

Where $|z_1| = |\bar{z}_1| = 1$. PROOF: From $\text{Im}(K(z, w)) = 0$ on $|z| = 1$ we have

$$(\mu_1 - \lambda_1)\text{Im}(z) + \left[\frac{\mu_2}{|w|^2} - \lambda_2 \right] \text{Im}(w) - \alpha(\text{Re}(z)\text{Im}(w) - \text{Re}(w)\text{Im}(z)) = 0$$

Solving for $\text{Im}(w)$ yields:

$$\text{Im}(w) = -\text{Im}(z) \frac{(\mu_1 - \lambda_1 + \alpha \text{Re}(w))}{\left(\frac{\mu_2}{|w|^2} - \lambda_2 - \alpha \text{Re}(z) \right)}$$

The denominator is always positive if $\frac{\mu_2}{\lambda_2 + \alpha} > |w|^2$ which obviously holds if $\frac{\mu_2}{\lambda_2 + \alpha} > 1$. If $\frac{\mu_2}{\lambda_2 + \alpha} < 1$ then Lemma (5.1) guarantees the inequality to be true. If $\text{Re}(w) > \frac{\lambda_1 - \mu_1}{\alpha}$ for all $|z| = 1$ then the numerator is also positive, the set Z is empty and the first part of the lemma is established. If $\text{Re}(w(z)) < \frac{\lambda_1 - \mu_1}{\alpha}$ for some $z, |z| = 1$, then the numerator is negative for those points $z \in Z$ and the second part of the lemma is established. Since $\text{Re}(w(z))$ is a continuous function of z then we must have $\text{Re}(w(z)) = \frac{\lambda_1 - \mu_1}{\alpha}$ (and $\text{Im}(w(z)) = 0$) for some values of z . Since z is quadratic in w there can be at most two such points, call them z_1 , and z_2 . The form of $K(z, \frac{\lambda_1 - \mu_1}{\alpha}) = 0$ is a quadratic equation in z with real coefficients. Since $|z_1| = |z_2| = 1$ and $z_1, z_2 \neq 1$ then z_1 and z_2 must be a conjugate pair (i.e. $z_2 = \bar{z}_1$) and the proof is complete.

We use this lemma to help prove that $B(z, w) \neq 0$ while condition (or inequality) C_1 holds.

LEMMA (5.5): $B(z, w) \neq 0$ for all $z, |z| = 1, w = w(z)$ when condition C_1 holds (i.e.

$$\lambda_1 + \mu_1 > \alpha \left(1 - \frac{\mu_2}{\lambda_2 + \alpha} \right)$$

PROOF: First, we check the point $z=1$

$$B(1, w(1)) = \begin{cases} \frac{\alpha \frac{\mu_2^2}{\lambda_2 + \alpha}}{(1 - \frac{\mu_2}{\lambda_2 + \alpha}) \mu_1 \mu_2} [\alpha(1 - \frac{\mu_2}{\lambda_2 + \alpha}) - (\lambda_1 + \mu_1)] < 0 & \frac{\mu_2}{\lambda_2 + \alpha} < 1 \\ \frac{1}{\mu_1 \mu_2} [(\lambda_2 - \mu_2)(\mu_1 + \lambda_1 + \frac{\lambda_1^2}{\mu_1 + \alpha - \lambda_1}) + \frac{\alpha \lambda_1^2}{\mu_1 + \alpha - \lambda_1}] < 0 & \frac{\mu_2}{\lambda_2 + \alpha} \geq 1 \end{cases}$$

and at the point $z \neq 1$ we have (let w_{-1} denote $w(z=-1)$)

$$B(-1, w_{-1}) = \frac{w_{-1}^2}{\mu_1 \mu_2 (1 - w_{-1})} [(\lambda_1 + m(w_{-1}))(\mu_1 + 2\lambda_1 + m(w_{-1})) - \lambda_1 \mu_1]$$

Note that $w_{-1} > 0$ from Lemma (5.2). From the kernel equation (evaluated at $z = -1$) we have

$$\lambda_1 + m(w_{-1}) = -\lambda_1 - 2\mu_1 - \alpha(1 + w_{-1})$$

$$\mu_1 + 2\lambda_1 + m(w_{-1}) = -\mu_1 - \alpha(1 + w_{-1})$$

Substituting in the right hand sides of the above two equations into the equation for $B(-1, w_{-1})$ yields:

$$B(-1, w_{-1}) = \frac{w_{-1}^2}{\mu_1 \mu_2 (1 - w_{-1})} [(\lambda_1 + 2\mu_1 + \alpha(1 + w_{-1}))(\mu_1 + \alpha(1 + w_{-1})) - \lambda_1 \mu_1]$$

The first parenthesized term (on the right hand side) is strictly greater than λ_1 , while the second is strictly greater than μ_1 . Thus, the bracketed term is strictly positive and so is $B(z, w(z))$ at $z = -1$. This excludes the points where $\text{Im}(z) = 0$ from further consideration. The rest of the proof is divided into two cases.

CASE 1: $\text{Re}(w(z)) > \frac{\lambda_1 - \mu_1}{\alpha}$ for all $z, |z|=1$.

By Lemma (5.4) $\text{Im}(z)\text{Im}(w) < 0$ (excluding the points $z=1, -1$). Assume $B(z', w') = 0$ for some $z', w' = w(z')$. Then

$$\lambda_1(1-z) + \mu_1 + m(w) = \frac{\lambda_1 \mu_1}{\lambda_1 + m(w)}$$

By looking at the imaginary part of the above equation we obtain:

$$-\lambda_1 \operatorname{Im}(z) + \left[\frac{\mu_2}{|w'|^2} - \lambda_2 \right] \operatorname{Im}(w) = \frac{-\lambda_1 \mu_1}{|\lambda_1 + m(w)|^2} \left[\frac{\mu_2}{|w'|^2} - \lambda_2 \right] \operatorname{Im}(w)$$

From Lemma (5.4) we know that the left hand side is negative when $\operatorname{Im}(z) > 0$, yet the right hand side is positive when $\operatorname{Im}(z) > 0$, thus leading to a contradiction. The same contradiction results when considering $\operatorname{Im}(z) < 0$, therefore $B(z, w(z)) \neq 0$ for $|z| = 1$

when $\operatorname{Re}(w(z)) > \frac{\lambda_1 - \mu_1}{\alpha}$.

CASE 2: $\operatorname{Re}(w(z)) \leq \frac{\lambda_1 - \mu_1}{\alpha}$ for some values of z , $|z| = 1$.

The same contradiction (from CASE 1) holds for all z , $|z| = 1$ such that

$\operatorname{Re}(w(z)) < \frac{\lambda_1 - \mu_1}{\alpha}$. Unfortunately, we have not been able to find a proof for all other

z , $|z| = 1$ $\operatorname{Re}(w(z)) \leq \frac{\lambda_1 - \mu_1}{\alpha}$. By considering all the numerical evidence we ob-

served for different parameter values satisfying $\lambda_1 + \mu_1 > \alpha(1 - \frac{\mu_2}{\lambda_2 + \alpha})$, we conjecture

that $B(z, w) \neq 0$ in this case as well. For the remainder of this proof (for Theorem (5.4)),

we assume the conjecture to be true.

The last step involves a proof that $W_{|z|=1}^\#(B(z, w(z))) = -1$ for a set of parameters that satisfy condition C_1 . We utilize the fact that $W_\Omega^\#(B/A) = W_\Omega^\#(B) - W_\Omega^\#(A)$.

In this proof, we divide $B(z, w)$ by a function that has winding number -1 (Lemma (5.6)) and show that the resulting function has a winding number of 0 (on $|z| = 1$).

Thus, $B(z, w)$ must have winding number -1 on $|z| = 1$.

LEMMA (5.6): $W_{|z|=1}^\#(\mu_1 + \lambda_1(1-z) + m(w)) = -1$ when $\mu_1 > \alpha$.

PROOF: $\mu_1 + \lambda_1(1-z) + m(w) = \frac{\mu_1}{z} - \alpha(1-w/z)$ from the kernel equation. We prove that the winding number of the right hand side is -1 on $|z|=1$. Multiplying the right hand side by z (which obviously has winding number 1) we obtain $\mu_1 - \alpha(z-w)$. However,

$$\begin{aligned} \operatorname{Re}(\mu_1 - \alpha(z-w)) &= \mu_1 - \alpha(\operatorname{Re}(z) - \operatorname{Re}(w)) \\ &> \mu_1 - \alpha \operatorname{Re}(z) \quad \text{since } \operatorname{Re}(w) > 0 \\ &> \mu_1 - \alpha > 0 \end{aligned}$$

which proves that $W_{|z|=1}^\#(\mu_1 - \alpha(z-w)) = 0$. Therefore,

$$\begin{aligned} W_{|z|=1}^\#(\mu_1 + \lambda_1(1-z) + m(w)) &= W_{|z|=1}^\# \left(\frac{\mu_1}{z} - \alpha(1-w/z) \right) = \\ &W_{|z|=1}^\#(\mu_1 - \alpha(z-w)) - W_{|z|=1}^\#(z) = -1 \end{aligned}$$

thus completing the proof of Lemma (5.6).

LEMMA (5.7): $W_{|z|=1}^\# \left[\frac{B(z,w)}{\lambda_1(1-z) + \mu_1 + m(w)} \right] = 0$ when $\mu_1 > \alpha$ and $\mu_1 > \lambda_1$.

PROOF:

$$\frac{B(z,w)}{\lambda_1(1-z) + \mu_1 + m(w)} = \frac{w^2}{\mu_1 \mu_2 (1-w)} \left[\lambda_1 + m(w) - \frac{\lambda_1 \mu_1}{\lambda_1(1-z) + \mu_1 + m(w)} \right]$$

Assume $W_{|z|=1}^\# \left[\frac{B(z,w)}{\lambda_1(1-z) + \mu_1 + m(w)} \right] \neq 0$ then

$$W_{|z|=1}^\# \left[\lambda_1 + m(w) - \frac{\lambda_1 \mu_1}{\lambda_1(1-z) + \mu_1 + m(w)} \right] \neq 0 \quad \text{since } W_{|z|=1}^\#(w) = W_{|z|=1}^\#(1-w) = 0.$$

Then there must exist a point z' , $w' = w(z')$ and a real number $C > 0$ such that the function in the parentheses touches the real axis at C when $z = z'$.

$$\lambda_1 + m(w') - \frac{\lambda_1 \mu_1}{\lambda_1(1-z') + \mu_1 + m(w')} = C$$

The imaginary part of the function must be equal to zero at $z=z'$.

$$\text{Im} \left[\lambda_1 + m(w') - \frac{\lambda_1 \mu_1}{\lambda_1(1-z') + \mu_1 + m(w')} \right] = 0$$

and that

$$\text{Im}(\lambda_1 + m(w')) = \frac{-\lambda_1 \mu_1}{|\lambda_1(1-z') + \mu_1 + m(w')|^2} \text{Im}(\lambda_1(1-z') + \mu_1 + m(w'))$$

but from Lemma (5.4) we know that $\text{Im}(\lambda_1 + m(w')) = \text{Im}(w') < 0$ when $\text{Im}(z') > 0$ yet the right hand side is positive since $\text{Im}(\lambda_1(1-z') + \mu_1 + m(w')) = -\lambda_1 \text{Im}(z') + \text{Im}(m(w')) < 0$ for $\text{Im}(z') > 0$. The same contradiction holds for $\text{Im}(z') < 0$. Thus,

$$W_{|z|=1}^\# \left[\frac{B(z, w)}{\lambda_1(1-z) + \mu_1 + m(w)} \right] = 0 \quad \text{when } \mu_1 > \alpha \text{ and } \mu_1 > \lambda_1$$

COROLLARY (5.3): $W_{|z|=1}^\#(B(z, w)) = -1$ when $\mu_1 > \alpha$ and $\mu_1 > \lambda_1$.

The corollary results from Lemmas (5.6), (5.7) and that

$$W_{|z|=1}^\#(B(z, w)) = W_{|z|=1}^\# \left[\frac{B(z, w)}{\lambda_1(1-z) + \mu_1 + m(w)} \right] + W_{|z|=1}^\#(\lambda_1(1-z) + \mu_1 + m(w))$$

Corollary (5.3) and Lemma (5.5) together imply that $W_{|z|=1}^\#(B(z, w)) = -1$ when $\lambda_1 + \mu_1 > \alpha(1 - \frac{\mu_2}{\lambda_2 + \alpha})$, completing the proof for Theorem (5.4), subject to the conjecture made in CASE 2 of Lemma (5.5). (When $\lambda_1 + \mu_1 = \alpha(1 - \frac{\mu_2}{\lambda_2 + \alpha})$ then $B(z, w)|_{z=1} = 0$. We handle this situation later on in this section.)

The winding number results obtained so far are displayed in the first two rows of the table in figure (5.5).

For the case $\lambda_1 + \mu_1 < \alpha(1 - \frac{\mu_2}{\lambda_2 + \alpha})$ we have $W_{|w|=1}^\#(B(z, w)) = 1$ and $W_{|z|=1}^\#(B(z, w)) = 0$. Recall that the argument principle states that for a function, $f(x)$, analytic in a domain that has boundary Ω .

$$W_\Omega^\#(f(x)) = N_{\text{zeroes}} - N_{\text{poles}}$$

where N_{zeroes} (N_{poles}) is the number of zeroes (poles) of $f(x)$ in the domain. We already showed that $B(z, w)$ has no poles on the zero set V_ϵ . The boundary of the zero set is given by $\Omega_1 = \{(z, w) : K(z, w) = 0, |w| = 1\}$ and $\Omega_2 = \{(z, w) : K(z, w) = 0, |z| = 1\}$ (as seen in figure (5.6)). Thus, the number of zeroes of $B(z, w)$ on the zero set is one. In order to take an analytic logarithm of $B(z, w)$ (step 1 in section 2D), we have to remove the zero (let it be denoted by p). In fact, the zero, p , must lie on the real axis in the w complex plane between $w = \frac{\mu_2}{\lambda_2 + \alpha}$ and $w = 1$. At the end of the proof of Theorem (5.3) we showed that $B(1, w=1) < 0$ for a stable system. In the proof for Theorem (5.4) we saw that $B(1, \frac{\mu_2}{\lambda_2 + \alpha}) > 0$ for $\lambda_1 + \mu_1 < \alpha(1 - \frac{\mu_2}{\lambda_2 + \alpha})$. Since $B(z, w)$ changes sign as w moves from $\frac{\mu_2}{\lambda_2 + \alpha}$ to 1, then $B(z, w) = 0$ at some point $w = p$ in $(\frac{\mu_2}{\lambda_2 + \alpha}, 1)$. We then take the logarithm of $\frac{B(z, w)}{(w-p)}$. Since p is located on the zero set V_ϵ between the two boundaries (Ω_1 and Ω_2 ; see figure (5.7)) we have $W_{|w|=1}^\#(w-p) = 1$ and $W_{|z|=1}^\#(w-p) = 0$ so that

$$W_{|w|=1}^\# \left[\frac{B(z, w)}{w-p} \right] = W_{|z|=1}^\# \left[\frac{B(z, w)}{w-p} \right] = 0$$

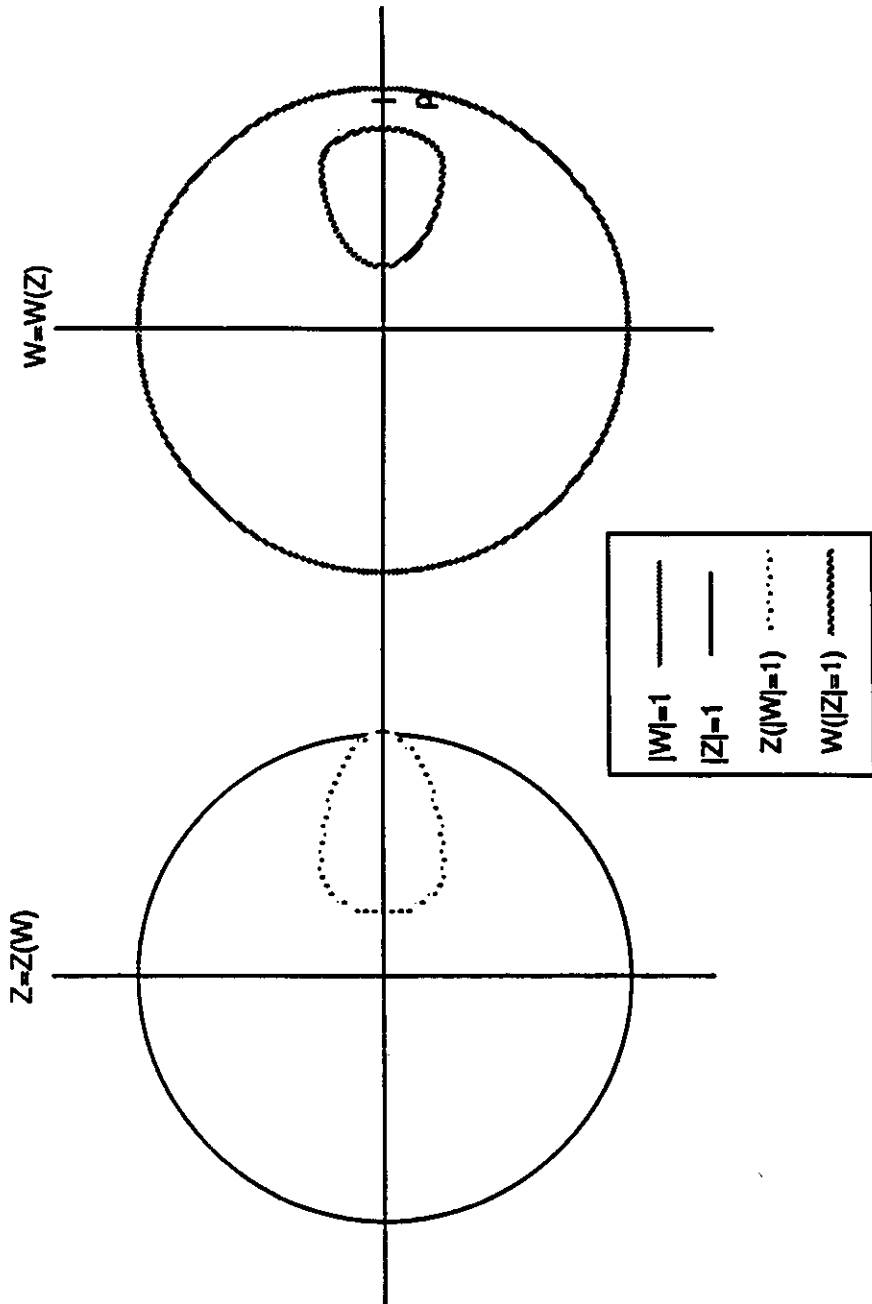


Figure (5.7)
 Zero of $B(z,w)$ for $\lambda_1 + \mu_1 < \alpha(1 - \mu_2/(\lambda_2 + \alpha))$

For the other case ($\lambda_1 + \mu_1 > \alpha(1 - \frac{\mu_2}{\lambda_2 + \alpha})$), the winding numbers (from the boundaries Ω_1 , and Ω_2) sum to zero, so that $B(z, w)$ has no zeroes on the zero set. A problem remains in that the logarithm is not single valued.

Recall that the winding number (or index) of an function $f(x)$ is defined as

$$W_{\Omega}^{\#}(f(x)) = \frac{1}{2\pi i} \int_{\Omega} \frac{f'(x)}{f(x)} dx$$

where Ω is the boundary of the domain of interest. If the domain is a simply connected annulus then $\Omega = \Omega_1 + \Omega_2$ where Ω_1 is the outer boundary and Ω_2 is the inner boundary. See figure (5.8) for an example where the domain of interest is $D = \{x: \frac{1}{2} \leq |x| \leq 1\}$. Assume $f(x)$ is analytic with no poles over D (for example $f(x) = x$). Then the logarithm of $f(x)$ is (from) [Chur74]

$$\log(f(x)) = \log|f(x)| + i\theta_{f(x)}$$

where $\theta_{f(x)}$ is the argument of $f(x)$. If the winding number of $f(x)$ is 1 around both boundaries then the winding number (of $f(x)$) is 1 for any closed curve homotopic¹ to Ω_1 or Ω_2 [Conw78]. Let ω be a closed curve in D homotopic to Ω_1 or Ω_2 . Then

$$W_{\omega}^{\#}(f(x)) = \frac{1}{2\pi i} \int_{\omega} \frac{f'(x)}{f(x)} dx = 1$$

But the integral is the logarithm of $f(x)$ over the closed curve ω , so that

$$\frac{1}{2\pi i} \int_{\omega} \frac{f'(x)}{f(x)} dx = \frac{1}{2\pi i} \log f(x) \Big|_{\omega} = 1$$

The rightmost equality in the above equation tells us that the result of traversing a loop leaves us with a non-zero answer

¹ Two simple closed curves, Γ_0 and Γ_1 are homotopic if there is a continuous parameter $0 \leq t \leq 1$ and a continuous function, $\Gamma(t)$ such that $\Gamma(0) = \Gamma_0$ and $\Gamma(1) = \Gamma_1$, and $\Gamma(t)$ is simple and closed for any $t \in [0, 1]$.

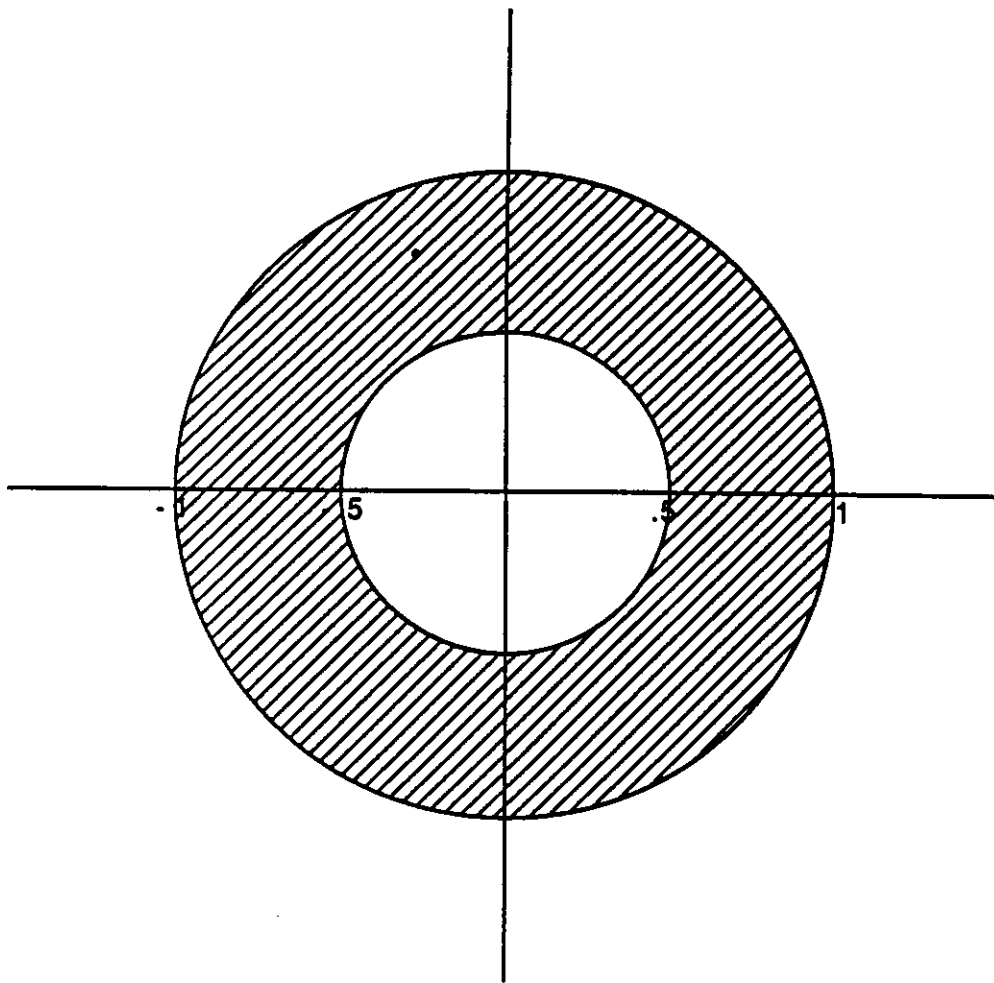


Figure (5.8)
Example of Simply Connected Annulus

$$\log f(x) \Big|_{\omega} = 2\pi i$$

and that

$$\log f(x) = \begin{cases} \log |f(x)| + i\theta_{f(x)} \\ \log |f(x)| + i(\theta_{f(x)} + 2\pi) \end{cases}$$

i.e. $\log f(x)$ is double valued. Thus, there is no analytic logarithm of $f(x)$ over domain D as stated. However, dividing $f(x)$ by $x-a$ where $|a| < 1/2$ yields a function, $\frac{f(x)}{x-a}$ that does have an analytic logarithm. To see this, note that as x traverses ω in the positive direction (counter clockwise), $\log f(x)$ increases its argument. However, $\frac{1}{(x-a)}$ traverses ω in the opposite direction and $\log \frac{1}{(x-a)}$ decreases the argument by an equal amount. When x returns to the starting point the argument portion of the logarithm (for the entire function $\frac{f(x)}{x-a}$) has been canceled out. Also note the winding numbers $W_{\Omega_1}^{\#}(x-a) = W_{\Omega_2}^{\#}(x-a) = 1$ so that $W_{\Omega_1}^{\#} \left[\frac{f(x)}{(x-a)} \right] = W_{\Omega_2}^{\#} \left[\frac{f(x)}{(x-a)} \right] = 0$.

In our problem, we divide $B(z,w)$ by $(w-p)$, where p is a point in the w unit disk that is not included in the zero set V_g (i.e. there is no z , $|z| \leq 1$ such that $K(z(p), p) = 0$). In Theorem (II.1) (in Appendix 2) we prove that such a p always exists in a stable system. Since $|p| < 1$ it is easily seen that $W_{|w|=1}^{\#}(w-p) = 1$, and so $W_{|w|=1}^{\#} \left[\frac{B(z,w)}{(w-p)} \right] = 0$. From Theorem (II.1) it is obvious that the point p lies immediately to the left (on the real axis) of $w(z=1)$ (see figure (5.9)). Thus the w image of $|z| = 1$ wraps around the point p one time in the opposite manner of the z variable, and $W_{|z|=1}^{\#}(w(z)-p) = -1$. We now have $W_{|z|=1}^{\#} \left[\frac{B(z,w)}{w(z)-p} \right] = 0$ for the case

$$\lambda_1 + \mu_1 > \alpha \left(1 - \frac{\mu_2}{\lambda_2 + \alpha} \right).$$

When $\lambda_1 + \mu_1 = \alpha(1 - \frac{\mu_2}{\lambda_2 + \alpha})$ we have $B(z, w)|_{z=1} = 0$. We deal with this problem by dividing out the zero at $w = \frac{\mu_2}{\lambda_2 + \alpha}$, and take the logarithm of $\frac{B(z, w)}{w-p}$.

Thus, we now have

$$W_{|w|=1}^{\#} \left[\frac{B(z, w)}{w-p} \right] = W_{|z|=1}^{\#} \left[\frac{B(z, w)}{w-p} \right] = 0$$

for all cases and is displayed in the last row of the table in figure (5.5). From [Maly72] we know that any closed curve ω on the zero set, V_ϵ , is homotopic to the boundaries $|w| = 1$, and $|z| = 1$ so that

$$W_{\omega}^{\#} \left[\frac{B(z, w)}{w-p} \right] = 0$$

We have now proved that $\log \left[\frac{B(z, w)}{w-p} \right]$ satisfies condition 5 of the Splitting Theorem in that the logarithm is analytic on V_ϵ . The system equation then becomes

$$P^1(z, 0) + (w-p) \frac{B(z, w)}{(w-p)} P^1(0, w) = P_{0,0} C(z, w) \quad (5.55)$$

We apply the logarithm function to $\frac{B(z, w)}{(w-p)}$ and split the result using the "splitter" subroutine. By the Splitting Theorem $\phi(z)$ and $\psi(w)$ are the unique analytic functions that satisfy

$$\ln \frac{B(z, w)}{(w-p)} = \psi(w) - \phi(z)$$

so that

$$\frac{B(z, w)}{(w-p)} = \frac{e^{\psi(w)}}{e^{\phi(z)}} \quad (5.56)$$

Substituting (5.56) into (5.55) we obtain the system equation:

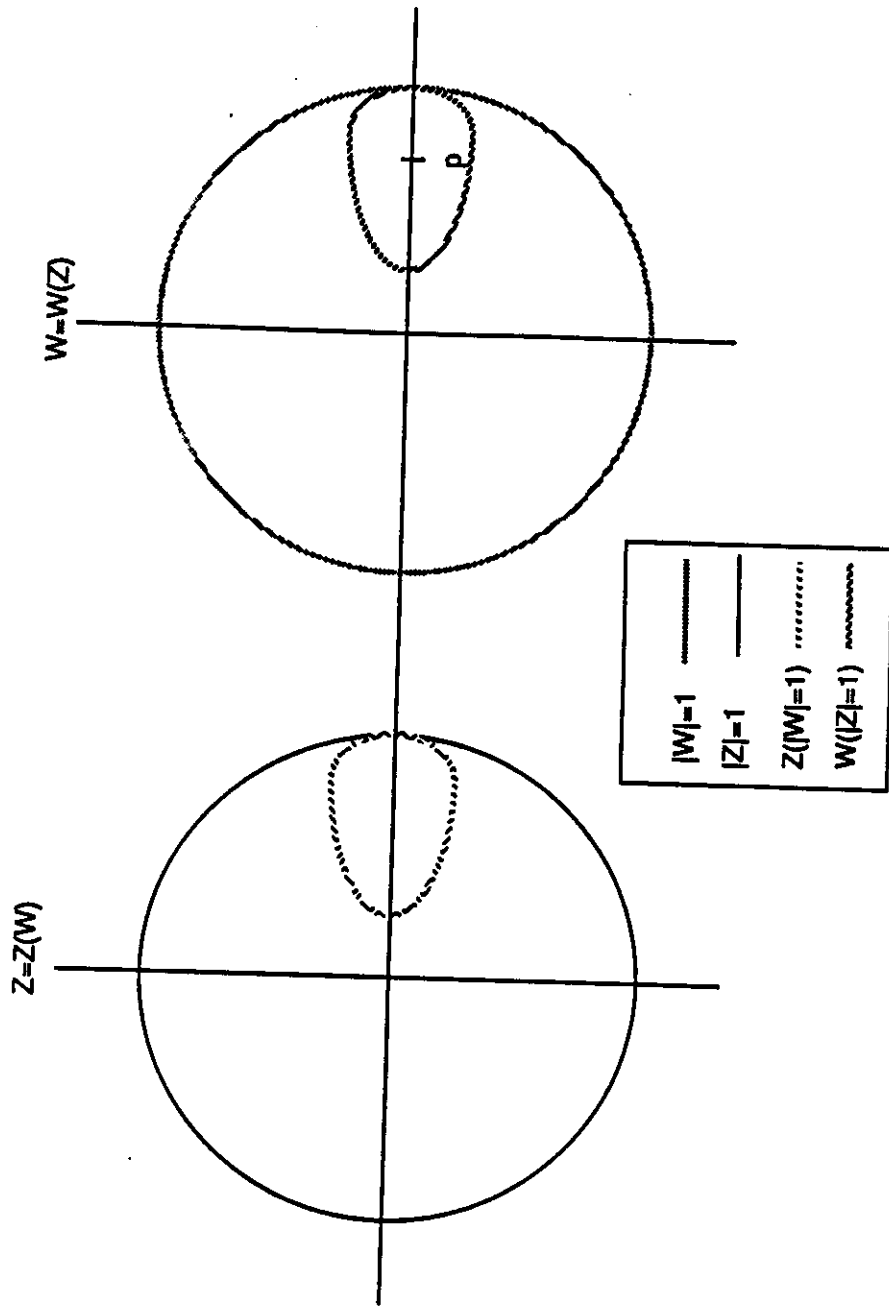


Figure (5.9)
Zero of $B(z,w)$ for $\lambda_1 + \mu_1 > \alpha(1 - \mu_2/(\lambda_2 + \alpha))$

$$e^{\phi(z)}P^1(z, 0) + (w-p)e^{\psi(w)}P^1(0, w) = e^{\phi(z)}P_{0,0}C(z, w) \quad (5.57)$$

The next step is to split the right hand side into two functions. One function is analytic throughout the z unit disk and is independent of w , while the other is analytic throughout the w disk and is independent of z . In order to split the right hand side we must show that $e^{\phi(z)}C(z, w)$ is analytic on the zero set V_ε . The first term, $e^{\phi(z)}$ is analytic since the exponential of an analytic function is analytic. From equations (5.40),(5.42) and (5.43) we have the following expression for $C(z, w)$:

$$\begin{aligned} C(z, w) &= \frac{\vec{V}_1 \vec{V}_3^\dagger - \vec{V}_1 \vec{V}_2^\dagger}{\mu_2(1-1/w)} = \frac{\vec{V}_1 [\vec{V}_2^\dagger - \vec{V}_3^\dagger]}{\mu_2(1-1/w)} \\ &= [-(\lambda_1(1-z) + \mu_1 + m(w)) \quad \lambda_1] \begin{bmatrix} -(\lambda_1 + \lambda_2(1-w)) \\ \mu_1 \\ -1 \end{bmatrix} \frac{1}{\mu_2(1-1/w)} \\ &= \frac{w \left[(\lambda_1(1-z) + \mu_1 + m(w))(\lambda_1 + \lambda_2(1-w)) - \lambda_1 \mu_1 \right]}{\mu_1 \mu_2 (w-1)} \end{aligned} \quad (5.58)$$

The w term in front cancels out the (only) pole at $w=0$, located in the $m(w)$ term. The $(w-1)$ term can be divided out of the numerator as was done for $B(z, w)$. Since $z(w)$, and $w(z)$ are analytic mappings on the zero set V_ε we conclude that $C(z, w)$ is analytic on V_ε . Applying the splitting subroutine to the right hand side of equation (5.57) yields:

$$e^{\phi(z)}C(z, w) = (\Phi(z) + K) + (\Psi(w) - K) \quad (5.59)$$

Where K is an additive constant to be solved for. Our system equation, which holds on V_ε , is now

$$e^{\phi(z)}P^1(z, 0) + (w-p)e^{\psi(w)}P^1(0, w) = P_{0,0}((\Phi(z) + K) + (\Psi(w) - K)) \quad (5.60)$$

Since, by the Splitting Theorem, equation (5.59) is the unique way to split

$e^{\Phi(z)}C(z, w)$, then we must have that

$$e^{\Phi(z)}P^1(z, 0) = P_{0,0}(\Phi(z) + K) \quad (5.61)$$

$$(w-p)e^{\Psi(w)}P^1(0, w) = P_{0,0}(\Psi(w) - K) \quad (5.62)$$

Setting $w=p$ in the above equation forces the constant to be $\Psi(p)$ so that

$$P^1(0, w) = \frac{P_{0,0}(\Psi(w) - \Psi(p))}{e^{\Psi(w)}(w-p)} \quad (5.63)$$

Using the equation $\sum_{i=0}^{\infty} \sum_{j=0}^{\infty} P_{i,j} = 1$ we obtain (from equation (5.20) in particular)

$$P(0, 1) = P_{0,0} \left[1 + \frac{\Psi(1) - \Psi(p)}{e^{\Psi(1)}(1-p)} \right] = \frac{\frac{\lambda_1}{\mu_1} \left(1 - \frac{\lambda_1}{\mu_1 + \alpha} \right)}{\left[\frac{\lambda_1}{\mu_1} + 1 - \frac{\lambda_1}{\mu_1 + \alpha} \right]}$$

Thus, solving for $P_{0,0}$ yields:

$$P_{0,0} = \frac{\frac{\lambda_1}{\mu_1} \left(1 - \frac{\lambda_1}{\mu_1 + \alpha} \right)}{\left[\frac{\lambda_1}{\mu_1} + 1 - \frac{\lambda_1}{\mu_1 + \alpha} \right]} \left[\frac{1}{1 + \frac{\Psi(1) - \Psi(p)}{e^{\Psi(1)}(1-p)}} \right] \quad (5.64)$$

Thus we have the proof of existence (and form) of a solution. In order to prove that there exists only one solution to the system equation we must show that the solution space to the homogeneous problem, namely,

$$\mu_2(1-1/w)P^1(z, 0) + B(z, w)P^1(0, w) = 0$$

includes only the trivial solution ($P^1(z, 0)=0$ and $P^1(0, w)=0$). We duplicate the first step exactly as for the proof of existence and obtain

$$e^{\Phi(z)}P^1(z, 0) + (w-p)e^{\Psi(w)}P^1(0, w) = 0$$

We now split the function 0 into two unique functions up to an additive constant.

Since $\Phi(z) = 0$, and $\Psi(w) = 0$ do indeed split 0, they must therefore be the unique solution to the previous equation. Then we have

$$e^{\Phi(z)} P^1(z, 0) = 0 + K$$

$$(w-p)e^{\Psi(w)} P^1(0, w) = 0 - K$$

Setting $w=p$ forces $K=0$ and we are left with

$$P^1(0, w) = 0 \text{ and } P^1(z, 0) = 0$$

In order to see that inequality (S2) is indeed a condition for ergodicity, we analyze the situation when the condition is violated. The winding number of $\frac{B^1(z, w)}{1-w}$ on $|w| = 1$ will be 1 and not 0, and $W_{|w|=1}^{\#}(B(z, w)) = 2$. In order to take an analytic logarithm of $B(z, w)$ it will be necessary to divide $B(z, w)$ by two factors, $(w-p_1)$ and $(w-p_2)$ (not one as in the stable case). At least one of the p_i 's will be on the zero set, V_{ϵ} . After applying the splitter the second time (Step 2 in section (5.2.4)) we have

$$e^{\Phi(z)} P^1(z, 0) + (w-p_1)(w-p_2)e^{\Psi(w)} P^1(0, w) = P_{0,0}((\Phi(z) + K) + (\Psi(w) - K))$$

and by the uniqueness of the Splitting Theorem we must have

$$(w-p_1)(w-p_2)e^{\Psi(w)} P^1(0, w) = P_{0,0}(\Psi(w) - K)$$

The above equation has two conditions to solve for two unknowns $P_{0,0}$, and K . However, we still have the equation for $P(0, 1)$ which overdetermines our system and yields no solution. Therefore, (S2) is a condition for existence of a solution, i.e. an ergodicity condition.

This concludes our proof of uniqueness and existence for the case $T=1$ (except for the set of parameters in CASE 2 of Lemma (5.5)). In the next section we discuss

the general case $T \geq 1$.

5.2.6) The General Problem, T greater than 1

The conditions (S1) and (S2) listed in section (5.2.1) are the conditions for ergodicity for the general case $T \geq 1$. Condition (S1) was proved necessary for ergodicity (for any T). However, we have been unable to find a proof that (S2) is necessary for the stability of the system although, we have strong numerical evidence that supports our conjecture that it is. The evidence concerns the winding number of $B^T(z, w)$ on the contour $|w| = 1$. For the remainder of this section let $B(z, w) = \frac{B^T(z, w)}{\mu_2(1-1/w)}$. From equation (5.41) we have that

$$B(z, w) = \frac{B^T(z, w)}{\mu_2(1-1/w)} = \frac{w^{T+1} \vec{V}_1 M^{T-1} \vec{V}_2^\dagger}{\mu_2(w-1)} \quad (5.65)$$

where

$$M = \frac{1}{\mu_1} \begin{bmatrix} \lambda_1 + \mu_1 + m(w) & -\lambda_1 \\ \mu_1 & 0 \end{bmatrix} \quad \vec{V}_1 = \begin{bmatrix} -(\lambda_1(1-z) + \mu_1 + m(w)) & \lambda_1 \end{bmatrix}$$

$$\vec{V}_2 = \begin{bmatrix} \lambda_1 + m(w) \\ \mu_1 \end{bmatrix} \quad \vec{V}_3 = \begin{bmatrix} \mu_2(1-1/w) & 0 \\ \mu_1 \end{bmatrix} \quad (5.66)$$

The w^{T+1} term cancels the $T+1$ poles (at $w = 0$) that occur in $\vec{V}_1 M^{T-1} \vec{V}_2^\dagger$. At $w=1$, $z=1$ the denominator of equation (5.65) is 0 whereas the numerator is

$$\frac{1}{\mu_1^{T-1}} \begin{bmatrix} -\mu_1 & \lambda_1 \end{bmatrix} \begin{bmatrix} \lambda_1 + \mu_1 & -\lambda_1 \\ \mu_1 & 0 \end{bmatrix}^{T-1} \begin{bmatrix} \lambda_1 / \mu_1 \\ 1 \end{bmatrix}$$

We can show that the above expression is equal to 0 for any $T \geq 1$, by transforming matrix M into its eigenvalue matrix, D , and associated eigenvectors, i.e.

$$M = ADA^{-1}$$

where A is a 2×2 matrix with each column being an eigenvector of M . The two eigenvalues of M , denoted by e_1 and e_2 , obey the following equations

$$e_1 e_2 = \det M = \lambda_1 \mu_1$$

$$e_1 + e_2 = \text{trace } M = \sum_{i=1}^2 m_{ii} = \lambda_1 + \mu_1$$

where m_{ii} is the i^{th} diagonal element of M . It is obvious that $e_1 = \lambda_1$ and $e_2 = \mu_1$ satisfy the above equations. Also note that $\det M \neq 0$, so that M is not singular at $w=1$, $z=1$. The matrix A is found by solving the following two vector equations.

$$M \begin{bmatrix} a_{11} \\ a_{21} \end{bmatrix} = e_1 \begin{bmatrix} a_{11} \\ a_{21} \end{bmatrix}$$

$$M \begin{bmatrix} a_{12} \\ a_{22} \end{bmatrix} = e_2 \begin{bmatrix} a_{12} \\ a_{22} \end{bmatrix}$$

Each vector equation leads to two equations in two unknowns. The following values for a_{ij} , $i, j = 1, 2$ satisfy the vector equations up to a multiplicative constant.

$$A = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} = \begin{bmatrix} \lambda_1 & 1 \\ \mu_1 & 1 \end{bmatrix}$$

and

$$A^{-1} = \frac{1}{\lambda_1 - \mu_1} \begin{bmatrix} 1 & -1 \\ -\mu_1 & \lambda_1 \end{bmatrix}$$

Now

$$M^{T-1} = AD^{T-1}A^{-1}$$

$$\begin{aligned}
&= \frac{1}{\lambda_1 - \mu_1} \begin{bmatrix} \lambda_1 & 1 \\ \mu_1 & 1 \end{bmatrix} \begin{bmatrix} \lambda_1^{T-1} & 0 \\ 0 & \mu_1^{T-1} \end{bmatrix} \begin{bmatrix} 1 & -1 \\ -\mu_1 & \lambda_1 \end{bmatrix} \\
&= \frac{1}{\lambda_1 - \mu_1} \begin{bmatrix} \lambda_1^T - \mu_1^T & \lambda_1 \mu_1^{T-1} - \lambda_1^T \\ \mu_1 \lambda_1^{T-1} - \mu_1^T & \lambda_1 \mu_1^{T-1} - \mu_1 \lambda_1^{T-1} \end{bmatrix}
\end{aligned}$$

After plugging in the above expression into $\vec{V}_1 M^{T-1} \vec{V}_2^\dagger$ it is easily verified that the numerator of equation (5.65) is 0 at $w=1, z=1$. The zero in the numerator cancels the zero in the denominator and $B(z, w)$ has no poles on the zero set, V_ε . In the table in figure (5.10) we present numerical data on the winding numbers $W_{|w|=1}^\#(B(z, w))$ and $W_{|z|=1}^\#(B(z, w))$ for different parameter values of stable systems. From the table we see that $W_{|w|=1}^\#(B(z, w)) = T$ and $-T \leq W_{|z|=1}^\#(B(z, w)) \leq 0$. The number of zeroes of $B(z, w)$ on the zero set is $n = W_{|w|=1}^\#(B(z, w)) + W_{|z|=1}^\#(B(z, w))$ which can range from 0 to T . In order to take an analytic logarithm of $B(z, w)$ on the zero set we will have to remove those n zeroes plus $T-n$ other points in the "hole" (points (z, w) on the polydisc $|z| \leq 1, |w| \leq 1, K(z, w) \neq 0$). After removing these T factors, both winding

numbers, $W_{|w|=1}^\# \left(\frac{B(z, w)}{\prod_{m=1}^T (w - w_m)} \right)$ and $W_{|z|=1}^\# \left(\frac{B(z, w)}{\prod_{m=1}^T (w - w_m)} \right)$ will be 0. As in the case

$T=1$, these T factors that were removed give us T equations to help solve for the unknown constants in equations (5.40) through (5.43). If the winding number $W_{|w|=1}^\#(B(z, w)) = T+1$, then we will have $T+1$ factors to remove, and $T+1$ equations to help solve for the unknowns. However, this extra equation will overdetermine the system of equations thus yielding no solution. The question as to whether or not there exists a solution depends on the value of the winding number $W_{|w|=1}^\#(B(z, w))$. We conjecture that in a stable system the winding number is T , whereas in an unstable system the winding number is $T+1$.

λ_1	λ_2	μ_1	T	$W_{ w =1}^{\#}(B^T(z,w))$	$W_{ z =1}^{\#}(B^T(z,w))$
.01	1	.2	2	2	0
.15	1	.2	2	2	-1
1.2	1	.8	2	2	-2
.01	1	.2	3	3	0
.1	1	.2	3	3	-1
.15	1	.2	3	3	-2
1.2	1	.8	3	3	-3
.05	1	.1	4	4	0
.1	1	.1	4	4	-1
.2	1	.1	4	4	-2
.4	1	.1	4	4	-3
1.2	1	.8	4	4	-4
.05	1.5	.2	5	5	0
.1	1.5	.2	5	5	-1
.2	1.5	.2	5	5	-2
.4	1.5	.2	5	5	-3
.67	1.5	.2	5	5	-4
1.2	1	.8	5	5	-5

Table of Values for $W_{|w|=1}^{\#}(B^T(z,w))$ and $W_{|z|=1}^{\#}(B^T(z,w))$
 $\mu_2 = 2, \alpha = 1.5$ (All cases are for stable systems)
Figure (5.10)

In the next section we discuss some of the numerical results we obtained for the performance measures of the queueing system.

5.3) Discussion of Results

In this section we discuss the numerical results obtained from the Boundary Value approach to the threshold queueing problem. We present numerical results obtained using the *splitter*, and compare those results to numbers obtained using two other approaches.

We limit the presentation of results to the case where the threshold $T = 1$. The numerical algorithms used in the *splitter* did not converge to particular values for

$T > 1$. We do not believe this is a problem with the Boundary Value approach, but rather an implementation problem with the algorithms used to compute the Poisson integral.

In the table in figure (5.11) we present the results, in terms of the average number in the system, for three different methods. The first is the Boundary Value approach. The second is the State Truncation approach. The two dimensional state space was truncated at $n_1 = M$ and at $n_2 = N$. For light loads ($\rho_1 = \frac{\lambda_1}{\mu_1 + \alpha} < .7$, $\rho_2 = \frac{\lambda_2}{\mu_2} < .5$) we used $N = M = 50$. For higher loads we increased both N , and M from 50 to 100. The steady state equation for each state (i, j) $0 \leq i < M$ $0 \leq j < N$, is the same as in section (5.2.1). On the boundary, the equations are modified as follows:

$$(\lambda_1 + \mu_2)P_{0,N} = \lambda_2 P_{0,N-1} + \mu_1 P_{1,N}$$

For $0 < i < T$

$$(\lambda_1 + \mu_1 + \mu_2)P_{i,N} = \lambda_1 P_{i-1,N} + \mu_1 P_{i+1,N} + \lambda_2 P_{i,N-1}$$

For $T \leq i < M$

$$(\lambda_1 + \mu_1 + \mu_2)P_{i,N} = \lambda_1 P_{i-1,N} + \mu_1 P_{i+1,N} + \lambda_2 P_{i,N-1} + \alpha P_{i+1,N-1}$$

$$(\mu_1 + \mu_2)P_{M,N} = \lambda_1 P_{M-1,N} + \lambda_2 P_{M,N-1}$$

For $0 < j < N$

$$(\lambda_2 + \mu_1 + \mu_2 + \alpha)P_{M,j} = \lambda_1 P_{M-1,j} + \lambda_2 P_{M,j-1} + \mu_2 P_{M,j+1}$$

$$(\lambda_2 + \mu_1 + \alpha)P_{M,0} = \lambda_1 P_{M-1,0} + \mu_2 P_{M,1}$$

The state probabilities, $P_{i,j}$, were given initial values as if the system consisted of two independent $M/M/1$ queues. In each iteration the set of equations was used to calcu-

late the new value for each $P_{i,j}$. The iterations continued until the probabilities stabilized. The last method was a simulation of the two queues. The simulation was written in the C language and run on a DEC VAX¹ 11/750. Each simulation was stopped after the second queue had processed from 750,000 to 1,000,000 jobs (depending on the parameters).

λ_1	λ_2	Simulation	SST	B-V
0.25	0.25	0.3321	0.3324	0.3324
0.5		0.5798	0.5798	0.5798
0.75		0.9135	0.9138	0.9139
1.5		3.586	3.588	3.582
1.7		6.349	6.351	6.407
1.9		19.793	19.213	18.228
0.25	0.5	0.5243	0.5253	0.5254
0.5		0.7802	0.7800	0.7800
0.75		1.127	1.127	1.126
1.25		2.406	2.408	2.407
1.5		3.877	3.881	3.886
1.7		6.678	6.679	6.742
1.9	20.166	19.586	19.032	
0.75	1	1.903	1.901	1.910
1.5		5.117	5.100	5.157
1.7		8.140	8.136	8.285
1.9		21.967	21.372	19.719
0.75	1.5	4.541	4.537	4.562
1.5		12.760	12.088	12.794
1.6		15.461	14.776	17.088
1.7		21.941	20.736	18.795
1.9		61.186	45.402	49.848
0.25	1.75	7.618	7.511	7.965

Average Number in the System
 $T=1, \mu_1=1.5, \mu_2=2, \alpha=0.5$
 Figure (5.11)

When the utilization is low for both queues (less than .75) all methods are within 1% of each other. Comparing the R-H (Boundary Value or Riemann-Hilbert) with the SST (state space truncation) numbers the difference is less than .5%.

¹ VAX is a registered trademark of the Digital Equipment Corporation

As the utilization in the first queue increases beyond .75 the differences between the three approaches starts to increase. However, if the utilization in the second queue remains low then the differences are bounded by 1%. This effect is due to errors in computing the Poisson integral and in computing the derivatives of the constructed functions. When the utilization in both queues tends towards 1 then the results from the three approaches starts to diverge. In particular, the three results for the case $\lambda_1=1.6$, $\lambda_2=1.5$ ($\rho_1=.8$, $\rho_2=.75$) differ by 10%. The effect is even more dramatic for the case $\lambda_1=1.9$, $\lambda_2=1.5$ ($\rho_1=.95$, $\rho_2=.75$). This effect is expected due to computational errors with the R-H, and SST approaches. In particular, the SST approach underestimates the average number due to the truncation. At high load there is a non-negligible amount of probability mass in the states that have been cutoff. However, by increasing the state size (increasing M and N) the computational complexity increases drastically. A further problem with the SST approach is that the number of iterations needed also increases with increasing load. The problems mentioned earlier with the Poisson integrator and taking derivatives, are exacerbated at heavy loads. The last approach, simulation, provided stable values for $\lambda_1 \leq 1.7$ and $\lambda_2 \leq 1.5$ ($\rho_1=.85$, $\rho_2=.75$). That is, different runs for the same parameter values yielded results that were within 5% of each other. However, for $\lambda_1 = 1.9$ and $\lambda_2 = 1.5$ ($\rho_1=.95$, $\rho_2=.75$) the different runs differed by as much as 25%.

5.4) Conclusion

In this chapter we introduced a coupled queue problem that is difficult, if not impossible, to solve using the standard methods in queueing theory [Klei75]. We presented an alternative approach that reduced the queueing problem to a two dimensional boundary value problem. The new problem is solved using Complex Analysis to construct analytic functions that satisfy the boundary conditions. The advantage of

our approach is that it yields an exact solution (albeit numerical) to the problem. Other approaches (State Space Truncation, Simulation) that yield numbers are approximations, not exact. A second advantage of our approach is that the method yields conditions for ergodicity as part of the proof of uniqueness. In many coupled queueing problems, the stability conditions are intuitively obvious (as in our problem), however, proving those requirements to be the ergodicity conditions is far more difficult.

The last advantage is a double edged sword. The main problem with our approach is the complexity of proving the solution to be unique. We were able to prove uniqueness for $T=1$ only, not for general T . Even for the case $T=1$ the proof is incomplete for the particular case of lemma (5.5). We hoped that a method could be developed that would apply to other coupled queue problems as well. However, the proofs derived here are only for the particular problem at hand. There are no general theorems that we could extract from our work.

We encountered a problem with the particular splitting subroutine that was used. The *splitter* was not sensitive enough for high loads or for $T>1$. This is not a problem with the method in general, but with the implementation of the splitting routine. A more sensitive Poisson integrator would yield more exact results.



CHAPTER 6

FUTURE RESEARCH

In Chapters 2 and 3 we investigated algorithms for load balancing in point to point networks. In Chapter 4 we looked at load sharing in a broadcast environment. An area for future research is load balancing in a mixed media network. For example, it would be interesting to develop and analyze effective, yet efficient, load balancing algorithms for many broadcast networks connected together in a point to point network. How could the algorithms in Chapters 2,3, and 4 be adapted to work in such a network? On the flip side we can design a point to point network that utilizes a low bandwidth channel for information passing only. Is there an advantage to using the channel for a load balancing application? Would the extra channel help speed the convergence of the algorithms in Chapters 2 and 3?

In Chapter 4 we analyzed the delay in a broadcast network before and after load sharing when the arrival rates were sampled from a uniform distribution. More work is needed to determine other distributions that are useful and amenable to analysis. We found that the average delay was sharply reduced when load sharing was used. However, we limited the external input rates to the nodes to be no larger than the service rate at the nodes. This limitation was used to bound the average job delay at a node without load sharing. In a real distributed processing environment, however, there may exist processors that are overloaded which clearly dictates the need for load sharing. Another performance measure (or different delay function) is needed to capture the effect of distributing the traffic from overloaded processors.

The proof of uniqueness in Chapter 5 is for the case $T=1$. A new approach for obtaining the winding numbers is needed for the general threshold case ($T>1$). An asymmetry remains between probabilistic analysis and complex analysis. If we are able to construct a solution using complex variables theory, perhaps we should be able to do so using probabilistic methods. An alternative method (using probability) would (hopefully) yield meaningful equations. The complex analysis approach yielded equations that did not help us understand the system. A closed form approximation to the average response time would give us a better feel to the system, and allow us to fine tune the system (by changing the threshold) thus minimizing the average response time of a job.

The system analyzed in Chapter 5 modeled a star topology in which the satellite nodes were connected to a central mainframe server. The satellites used thresholds to determine when to send jobs to the central server. A more complete model would include a threshold for the central server as well. The server could accept jobs only if the number in its queue is below its threshold, otherwise, the satellite nodes are prohibited from sending jobs to the central server. Another modification would be to allow the central server to send jobs to the satellite nodes if it is congested.

APPENDIX 1

DERIVATION OF M/G/1 DELAY WITH LOAD SHARING

We start off with the equation for $\hat{T}_{N-2K}^{M/G/1}$ using the same value for $\bar{\gamma}_{N-2K}$ as in the M/M/1 network.

$$\hat{T}_{N-2K}^{M/G/1} = \frac{1}{\bar{\gamma}_{N-2K}} \int_0^1 \cdots \int_0^1 \prod_{j=1}^{N-2K} f(x_j) \left[\sum_{i=1}^{N-2K} \left[1 + \frac{x_i(1+C_b^2)}{2(1-x_i)} \right] \right] dx_j \quad (\text{A.1})$$

For notational simplicity, we let D_i represent the average delay at one node in the network.

$$D_i = 1 + \frac{x_i(1+C_b^2)}{2(1-x_i)} = \frac{2+x_i(C_b^2-1)}{2(1-x_i)}$$

Replacing this value in the network equation and isolating the terms for the random variable x_{N-2K} yields:

$$\hat{T}_{N-2K}^{M/G/1} = \frac{1}{\bar{\gamma}_{N-2K}} \int_0^1 \cdots \int_0^1 \prod_{j=1}^{N-2K} f(x_j) dx_j M \quad (\text{A.2})$$

where

$$M = \left[\int_0^1 \left[\sum_{i=1}^{N-2K-1} x_i D_i + \frac{2x_{N-2K} + x_{N-2K}^2(C_b^2-1)}{2(1-x_{N-2K})} \right] f_{N-2K}(x_{N-2K}) dx_{N-2K} \right]$$

The solution of the bracketed term is made up of three separate parts.

$$M = \sum_{i=1}^{N-2K-1} x_i D_i + \int_0^1 \frac{x}{1-x} f(x) dx + \frac{[C_b^2-1]}{2} \int_0^1 \frac{x^2}{1-x} f(x) dx$$

The middle term on the right hand side of the above equation, call it A, is

$$A = \frac{N+1}{N-K+1} \int_0^1 \sum_{j=K}^N \binom{N}{j} (1-x)^{j-1} x^{N-j+1} dx = \frac{N+1}{N-K+1} \sum_{j=K}^N \binom{N}{j} \int_0^1 (1-x)^{j-1} x^{N-j+1} dx$$

The integral in the rightmost term is the normalization constant for a Beta distribution with parameters $j-1, N-j+1$. Substituting the constant into the equation yields:

$$\begin{aligned} A &= \frac{N+1}{N-K+1} \sum_{j=K}^N \binom{N}{j} \frac{\Gamma(j)\Gamma(N-j+2)}{\Gamma(N+2)} \\ &= \frac{1}{N-K+1} \sum_{j=K}^N \frac{N-j+1}{j} = \frac{1}{N-K+1} \left[(N+1) \sum_{j=K}^N \frac{1}{j} - 1 \right] \end{aligned} \quad (A.3)$$

The third part of M is likewise solved.

$$\begin{aligned} B &= \frac{(N+1)(C_b^2-1)}{2(N-K+1)} \int_0^1 \sum_{j=K}^N \binom{N}{j} (1-x)^{j-1} x^{N-j+2} dx \\ &= \frac{(N+1)(C_b^2-1)}{2(N-K+1)} \sum_{j=K}^N \binom{N}{j} \int_0^1 (1-x)^{j-1} x^{N-j+2} dx \end{aligned}$$

As with the A term, the integral in the above term is the normalization condition for a Beta distribution with parameters $j-1, N-j+2$.

$$\begin{aligned} B &= \frac{(N+1)(C_b^2-1)}{2(N-K+1)} \sum_{j=K}^N \binom{N}{j} \frac{\Gamma(j)\Gamma(N-j+3)}{\Gamma(N+3)} \\ &= \frac{(C_b^2)}{2(N-K+1)} \sum_{j=K}^N \frac{(N-j+2)(N-j+1)}{j(N+2)} \\ &= \frac{(C_b^2)}{2(N-K+1)(N+2)} \left[(N-K+1)(2N+3) + \frac{N(N+1)}{2} - \frac{K(K-1)}{2} + (N+2)(N+1) \sum_{j=K}^N \frac{1}{j} \right] \end{aligned} \quad (A.4)$$

Replacing A, B, and $\sum x_i D_i$ in equation (A.2) yields the same functional equation but with one less random variable (x_{N-2K}). By repeating the same procedure for the other $N-2K-1$ random variables we have:

$$\hat{T}_{N-2K}^{MIG1} = \frac{2}{N-2K} \left[(N-2K)(A+B) \right]$$

Replacing A and B with equations (A.3) and (A.4) yields the final result in section 4.4.



APPENDIX 2
STUDY OF THE KERNEL EQUATION

THEOREM (II.1): For $\mu_1 + \mu_2 < \lambda_1 + \lambda_2$, there exists a point p in the w complex plane such that $K(z, p) \neq 0$ for any z , $|z| \leq 1$.

The inequality $\mu_1 + \mu_2 < \lambda_1 + \lambda_2$ is the system stability requirement (condition (S3) from section (5.2.1)). The input rate $\lambda_1 + \lambda_2$ should be strictly less than the sum of the service rates. In order to prove Theorem (II.1) we need the following lemma.

LEMMA (II.1):
$$\frac{\mu_2}{\lambda_2 + \alpha} > \frac{\lambda_1 - \mu_1}{\alpha}$$

PROOF: If $\frac{\mu_2}{\lambda_2 + \alpha} \geq 1$ then we must have $\lambda_1 < \mu_1 + \alpha$ for stability in the first queue. This

condition immediately gives us $\frac{\lambda_1 - \mu_1}{\alpha} < 1 < \frac{\mu_2}{\lambda_2 + \alpha}$. If $\frac{\mu_2}{\lambda_2 + \alpha} < 1$ then assume

$\frac{\lambda_1 - \mu_1}{\alpha} \geq \frac{\mu_2}{\lambda_2 + \alpha}$. By considering the system stability condition and the assumption we

have the following inequality

$$\mu_2 - \lambda_2 > \lambda_1 - \mu_1 > \alpha \frac{\mu_2}{\lambda_2 + \alpha}$$

but $\alpha \frac{\mu_2}{\lambda_2 + \alpha} = \mu_2 \left(1 - \frac{\lambda_2}{\lambda_2 + \alpha}\right) = \mu_2 - \lambda_2 \frac{\mu_2}{\lambda_2 + \alpha} > \mu_2 - \lambda_2$ which contradicts the above inequality. QED.

PROOF (of Theorem II.1): The kernel equation is

$$\lambda_1 z^2 - (\lambda_1 + \mu_1 + \alpha + m(w))z + (\mu_1 + \alpha w) = 0$$

where $m(w) = \lambda_2(1-w) + \mu_2(1-1/w)$. Let $w = re^{i\theta}$. Taking the partial derivative of the kernel equation with respect to r we obtain:

$$2\lambda_1 z \frac{\partial z}{\partial r} - \frac{\partial z}{\partial r} (\lambda_1 + \mu_1 + \alpha + m(w)) - z \left[\frac{\mu_2}{r^2} e^{-i\theta} - \lambda_2 e^{i\theta} \right] + \alpha e^{i\theta} = 0$$

Isolating for $\frac{\partial z}{\partial r}$ yields:

$$\frac{\partial z}{\partial r} = \frac{z \left[\frac{\mu_2}{r^2} e^{-i\theta} - \lambda_2 e^{i\theta} \right] - \alpha e^{i\theta}}{2\lambda_1 z - (\lambda_1 + \mu_1 + \alpha + m(w))} \quad (\text{II.1})$$

CASE 1: $\frac{\mu_2}{\lambda_2 + \alpha} < 1$. We want to show that $p = \frac{\mu_2}{\lambda_2 + \alpha} - \epsilon$, for small enough ϵ , has no z

roots in the unit disk. From the kernel equation, when $w = \frac{\mu_2}{\lambda_2 + \alpha}$ there are two roots

$z_1(w) = 1$ and $z_2(w) = \frac{\mu_1 + \alpha \frac{\mu_2}{\lambda_2 + \alpha}}{\lambda_1} > 1$ (from Lemma (II.1)). The derivative of each

root is obtained from equation (II.1).

$$\left. \frac{\partial z_1(w)}{\partial r} \right|_{w = \frac{\mu_2}{\lambda_2 + \alpha}} = \frac{\left[\frac{\mu_2}{\left[\frac{\mu_2}{\lambda_2 + \alpha} \right]^2} - \lambda_2 \right] - \alpha}{\lambda_1 - \left(\mu_1 + \alpha \frac{\mu_2}{\lambda_2 + \alpha} \right)}$$

The denominator is negative (from Lemma (II.1)) and the numerator is positive since

$$\frac{\mu_2}{\left[\frac{\mu_2}{\lambda_2 + \alpha} \right]^2} - (\lambda_2 + \alpha) > \frac{\mu_2}{\lambda_2 + \alpha} - (\lambda_2 + \alpha) = 0$$

Thus $\frac{\partial z_1(w)}{\partial r}$ is negative and picking $p = \frac{\mu_2}{\lambda_2 + \alpha} - \epsilon$ (small ϵ) moves $z_1(w)$ to a point

on the real axis greater than one. The derivative of $z_2(w)$ at $w = \frac{\mu_2}{\lambda_2 + \alpha}$ is

$$\left. \frac{\partial z_2(w)}{\partial r} \right|_{w = \frac{\mu_2}{\lambda_2 + \alpha}} = \frac{z_2(w) \left[\frac{\mu_2}{\lambda_2 + \alpha} - \lambda_2 \right] - \alpha}{2\lambda_1 z_2(w) - (\lambda_1 + \mu_1 + \alpha + \lambda_2 (1 - \frac{\mu_2}{\lambda_2 + \alpha}) + \mu_2 (1 - \frac{\lambda_2 + \alpha}{\mu_2}))}$$

We see after some algebraic manipulation that both the numerator and denominator are positive. As w moves from $\frac{\mu_2}{\lambda_2 + \alpha}$ to point $p = \frac{\mu_2}{\lambda_2 + \alpha} - \epsilon$, $z_2(w)$ also moves on the positive real axis to the left. However, if ϵ is chosen small enough then $z_2(w)$ will still be out of the unit circle. So for $p = \frac{\mu_2}{\lambda_2 + \alpha} - \epsilon$ both z roots will be out of the unit circle.

CASE 2: $\frac{\mu_2}{\lambda_2 + \alpha} \geq 1$. Here $p = 1 - \epsilon$. The two z roots of $w = 1$ are $z_1(1) = 1$ and $z_2(1) = \frac{\mu_1 + \alpha}{\lambda_1} > 1$. The corresponding derivatives at $w = 1$ are

$$\left. \frac{\partial z_1(w)}{\partial r} \right|_{w=1} = \frac{\mu_2 - (\lambda_2 + \alpha)}{\lambda_1 - (\mu_1 + \alpha)} \leq 0$$

A small decrease in r (by ϵ) will result in an increase for $z_1(w)$ to a point outside the unit circle. For $z_2(w)$ we have

$$\left. \frac{\partial z_2(w)}{\partial r} \right|_{w=1} = \frac{\frac{\mu_1 + \alpha}{\lambda_1} (\mu_2 - \lambda_2) - \alpha}{2(\mu_1 + \alpha) - (\lambda_1 + \mu_1 + \alpha)} > 0$$

Decreasing r by ϵ will bring $z_2(w)$ closer to the unit disk. However, for small enough ϵ , $z_2(w)$ will remain outside. QED



References

- [Abra72] M. Abramowitz and I. A. Stegun, *Handbook of Mathematical Functions*, Dover Publications, New York (1972).
- [Agra84] A. Agrawala, E. Coffman, M. Garey, and S. Tripathi, "A Stochastic Optimization Algorithm Minimizing Expected Flow Times on Uniform Processes," *IEEE Trans. Comput.* C-33(4), pp.351-356 (1984).
- [Bert78] D. P. Bertsekas, "Algorithms for Optimal Routing of Flow in Networks," , Coordinated Science Laboratory Working Paper, University of Ill. at Champaign-Urbana (June 1978).
- [Bert84] D. Bertsekas, E. Gafni, and R. Gallager, "Second Derivative Algorithms for Minimum Delay Routing in Networks," *IEEE Trans. Commun.* COM-32(8), pp.911-919 (Aug. 1984).
- [Brya81] R. M. Bryant and R. A. Finkel, "A stable distributed scheduling algorithm," pp. 314-323 in *Proceedings 2nd International Conference in Distributed Computing Systems*, Los Alamitos, CA (1981).
- [Cant74] D. G. Cantor and M. Gerla, "Optimal Routing in a Packet-Switched Computer Network," *IEEE Trans. Comput.* C-23, pp.1062-1069 (Oct. 1974).
- [Cape79] J. I. Capetanakis, "Generalized TDMA: The Multi-Accessing Tree Protocol," *IEEE Trans. Commun.* COMM-27, pp.1476-1484 (Oct. 1979).
- [Case81] L. M. Casey, "Decentralized Scheduling," *Australian Comput. J.* 13, pp.58-63 (May 1981).
- [Chou82] T. C. K. Chou and J. A. Abraham, "Load Balancing in Distributed Systems," *IEEE Trans. Software Eng.* SE-8, pp.401-412 (July 1982).
- [Chow79] Y. Chow and W. Kohler, "Models for dynamic load balancing in a heterogeneous multiple processor system," *IEEE Trans. Comput.* 28(5), pp.354-361 (May 1979).

- [Chur74] R. V. Churchill, J. W. Brown, and R. F. Verhey, *Complex Variables and Applications*, McGraw-Hill, New York (1974).
- [Chu80] W. Chu, L. J. Holloway, M. T. Lan, and K. Efe, "Task Allocation in Distributed Data Processing," *IEEE Computer* 13, pp.57-69 (Nov. 1980).
- [Cinl67] Erhan Cinlar and Ralph L. Disney, "Stream of Overflows from a Finite Queue," *Operations Research* 15(1), pp.131-134 (Jan. - Feb. 1967).
- [Cohe83] J. W. Cohen and O. J. Boxma, *Boundary Value Problems in Queueing System Analysis*, North-Holland Math. Studies, Amsterdam (1983).
- [Conw78] J. B. Conway, *Functions of One Complex Variable*, Springer-Verlag, New York (1978).
- [Davi70] H. A. David, *Order Statistics*, Wiley, New York (1970).
- [Eage84] Derek L. Eager, Edward D. Lazowska, and John Zahorjan, "Dynamic Load Sharing in Homogenous Distributed Systems," 84-10-01, Dept. of Computer Science, Univ. of Washington (Oct. 1984).
- [Ephr80] A. Ephremides, P. Varaiya, and J. Walrand, "A simple dynamic routing problem," *IEEE Trans. Automatic Control* 25(4), pp.690-693 (Aug. 1980).
- [Farb72] D. C. Farber and K. C. Larson, "The Distributed Computer System," in *Proceedings Symp. Comput. Commun. Networks and Teletraffic* (1972).
- [Fayo79] G. Fayolle and R. Iasnogorodski, "Two Coupled Processors: The Reduction to a Riemann-Hilbert Problem," *Z. Wahrscheinlichkeitstheorie* 47, pp.325-351 (1979).
- [Frat73] L. Fratta, M. Gerla, and L. Kleinrock, "The Flow Deviation Method: An Approach to Store-and-Forward Communications Network Design," *Networks* 3, pp.97-133 (1973).
- [Gakh66] F. D. Gakhov, *Boundary Value Problems*, Pergamon Press, Oxford (1966).
- [Gall77] Robert G. Gallager, "A Minimum Delay Routing Algorithm Using Distributed Computation," *IEEE Trans. Commun.* COM-25(1), pp.73-85 (Jan. 1977).
- [Gold83] A. Goldberg, G. Popek, and S. S. Lavenberg, "A Validated Distributed System Performance Model," pp. 251-268 in *Performance 83 Proceedings*, ed. A. K. Agrawala, S. K. Tripathi (1983).

- [Haje84] B. Hajek, "Optimal Control of Two Interacting Service Stations," *IEEE Trans. Automatic Control* 29(6), pp.491-499 (June 1984).
- [Klei64] L. Kleinrock, *Communication Nets; Stochastic Message Flow and Delay*, McGraw-Hill, New York (1964).
- [Klei84] L. Kleinrock, "On the Theory of Distributed Processing," *Twenty-second Annual Allerton Conference on Communication, Control, and Computing* (October 3, 1984).
- [Klei75] L. Kleinrock, *Queueing Systems. Vol. 1, Theory.*, Wiley, New York (1975).
- [Klei76] L. Kleinrock, *Queueing Systems. Vol. 2, Computer Applications*, Wiley, New York (1976).
- [Kuro86] James F. Kurose and Suresh Singh, "A Distributed Algorithm for Optimum Static Load Balancing in Distributed Computer Systems," pp. 458-467 in *Proceedings INFOCOM* (1986).
- [Lave83] S. S. Lavenberg (Editor), *Computer Performance Modeling Handbook*, Academic Press, New York (1983).
- [Lazo84] E. D. Lazowska, J. Zahorjan, D. R. Cheriton, and W. Zwaenepoel, "File Access Performance of Diskless Workstations," 84-06-01, Dept. of Computer Science, Univ. of Washington (June 1984).
- [Lee87] K. J. Lee, *Load Balancing in Distributed Computer Systems*, Dept. of Computer and Information Science, University of Massachusetts, Amherst (1987). Ph.D. Dissertation.
- [Livn82] M. Livny and M. Melman, "Load Balancing in Homogeneous Broadcast Distributed Systems," pp. 47-55 in *Proceedings ACM Computer Network Performance Symposium* (April 1982).
- [Maju80] S. Majumdar and M. L. Green, "A distributed real time resource manager," in *Proceedings IEEE Symp. Distrib. Data Acquisition, Comput. Control* (1980).
- [Maly72] V. A. Malysev, "Positive Random Walks and Generalized Elliptic Integrals," *Soviet Math Dokl.* 12(1), pp.178-182 (1972).
- [Nels85] R. Nelson and D. Towsley, "On Maximizing the Number of Departures Before a Deadline on Multiple Processors," RC 11255 (#50703), IBM Watson Research Center, Yorktown Heights, NY (July 1985).
- [Ni81] L. M. Ni and K. Abani, "Nonpreemptive Load Balancing in a Class of Local Area Networks," in *Proceedings Comput. Networking Symp.* (Dec. 1981).

- [Peeb80] R. Peebles and T. Dopirak, "ADAPT: A Query System," in *Proceedings COMPCON* (Spring 1980).
- [Rama83] K. K. Ramakrishna, *The Design and Analysis of Resource Allocation Policies in Distributed Systems*, Ph.D Dissertation, Dept. of Computer Science, Univ. of Maryland (1983).
- [Sega79] Adrian Segall, "Optimal Distributed Routing for Virtual Line-Switched Data Networks," *IEEE Trans. Commun. COM-27*(1), pp.201-209 (Jan. 1979).
- [Silv84] E. de Souza e Silva and M. Gerla, "Load Balancing in Distributed Systems with Multiple Classes and Site constraints," pp. 17-33 in *Performance 84 Proceedings*, ed. E. Gelenbe (1984).
- [Silv87] E. de Souza e Silva and M. Gerla, "Queueing Network Models for Load Balancing in Distributed Systems," CSD-870069, Department of Computer Science, UCLA, Los Angeles, CA (December 1987).
- [Smit80] R. G. Smith, "The Contact Net Protocol: High Level Communication and Control in a Distributed Problem Solver," *IEEE Trans. Comput. C-29*, pp.1104-1113 (1980).
- [Ston78] H. S. Stone and S. H. Bokhari, "Control of distributed processes," *IEEE Computer C-11*, pp.97-106 (July 1978).
- [Tant85] Asser N. Tantawi and Don Towsley, "Optimal Static Load Balancing in Distributed Computer Systems," *J. ACM* 32(2), pp.445-465 (Apr. 1985).
- [Tayl88] B. A. Taylor and S. L. Hantler, "Boundary Value Problems Arising in Multi-Dimensional Queueing Systems," Special Session on Several Complex Variables and Their Applications, University of Maryland (May 1988).
- [Wang85] Yung-Terng Wang and Robert J. T. Morris, "Load Sharing in Distributed Systems," *IEEE Trans. Comput. C-34*(3), pp.204-216 (Mar. 1985).