Probabilistic Analysis of Multistage Interconnection Network Performance

Patrick G. Sobalvarro
S.B. Massachusetts Institute of Technology (1988)

**Submitted in Partial Fulfillment
of the Requirements for the Degree of
Master of Science
in
Electrical Engineering and Computer Science
at the
Massachusetts Institute of Technology
April, 1992**

Signature of Author______________________________________
Department of Electrical Engineering and Computer Science
April 21, 1992

Approved by______________________________________
Neil A. Brock
Senior Engineer, Draper Laboratory
Technical Supervisor

Certified by______________________________________
Thomas F. Knight, Jr.
Principal Research Scientist, Artificial Intelligence Laboratory
Thesis Supervisor

Accepted by______________________________________
Arthur C. Smith
Chairman, Departmental Committee on Graduate Students

Probabilistic Analysis of Multistage Interconnection Network Performance

Patrick G. Sobalvarro

Submitted on April 21, 1992 to the
Department of Electrical Engineering and Computer Science
at the Massachusetts Institute of Technology
in partial fulfillment of the requirements for the degree of

Master of Science

Abstract

Low-latency communication in large-scale multiprocessors requires high-performance interconnection schemes. Multistage interconnection networks with redundant paths combine high performance with fault-tolerance, but exact evaluation of the blocking probability of interconnection networks with redundant paths is expensive. Equations for the blocking probability and throughput of multistage, multipath interconnection networks are derived. A method of approximate solution of the equations is presented, with a derivation of error bounds on the estimated solution. A program that solves the equations exactly and approximately is presented.

Technical Supervisor: Neil A. Brock
Title: Senior Engineer, Draper Laboratory

Thesis Supervisor: Thomas F. Knight, Jr.
Title: Principal Research Scientist, Artificial Intelligence Laboratory

This work is dedicated in loving memory of

ELSIE STARK CROWLEY

1925 – 1992

Acknowledgments

I owe a debt of gratitude to my thesis advisor, Dr. Thomas F. Knight Jr., whose keen perception and lively interest in a wide array of technical areas have made working with him over the years a most rewarding experience. I am also indebted to Neil A. Brock, my technical supervisor at Draper Laboratories, for his support, friendship, and good advice, both technical and administrative. André DeHon's dedication to making the Transit network architecture a reality, and his attention to the hard day-to-day work of making a research group function, created an environment that helped me to bring this work to fruition; I am also indebted to him for many rewarding technical discussions, his comments on earlier versions of this work, and his comments on the thesis itself.

Michael Bolotski was kind enough to read the papers that preceded this work, and gave me incisive commentary, encouragement, and the example of his good humor. Many other members of the Artificial Intelligence Laboratory and the Laboratory for Computer Science provided me with helpful and clear comments on my oral qualifying exam presentation, which covered the material presented here in Chapter 3. Professor John N. Tsitsiklis generously spent an hour answering my questions about related work, and I thank him for his thoughtful comments and advice.

I am grateful most of all to my beloved wife, Marie, for her love, encouragement, and patience through all the time I have spent on this work.

This thesis describes research done in part at the Artificial Intelligence Laboratory of the Massachusetts Institute of Technology. Support for the laboratory's artificial intelligence research is provided in part by the Advanced Research Projects Agency of the Department of Defense under Office of Naval Research contracts N00014-88-K-0825 and N00014-85-K-0124, and in part by the Defense Advanced Research Projects Agency under contract N00014-87-K-0825.

Acknowledgement
April 16, 1992

This thesis was prepared in part at the Charles Stark Draper Laboratory, Inc., under funding provided by Draper.

Publication of this thesis does not constitute approval by Draper of the findings or conclusions contained herein. It is published for the exchange and stimulation of ideas.

Contents

List of Figures

Chapter 1

Introduction

1.1 Background

The realization of low-latency communication in large-scale multiprocessors requires high-performance interconnection schemes. Both direct and indirect networks are examples of these; here our focus is on self-routing, multistage networks, both with unique paths and with multiple (redundant) paths.

One popular measure of the performance of a multistage interconnection network is its *bandwidth* or *throughput* – that is, the expected number of messages it delivers in each cycle, where the inputs have some given probability of generating a message. A related measure, from which the bandwidth may be calculated in some models, is the *probability of successful message transmission* or the *normalized throughput* – the probability that an arbitrary message at an input is not blocked (and presumably queued for later service) by some other request in the course of delivery. The problem of calculating the probability of successful message transmission is more often referred to in the telephone switching literature by its complement – the *blocking probability*, and we shall do the same here.

The problem of computing blocking probabilities in regular variants of unique-path multistage interconnection networks has been extensively studied. These networks were called *Banyan* networks by Goke and Lipovsky [9]. Patel [21] and Kruskal and Snir [15] in particular presented expressions for the probability of successful message transmission of *delta* networks, which are a particular regular variant of Banyan networks. Multiprocessors have been built using such regular Banyan networks for interconnection [22, 24]. A later chapter presents a method for calculating the exact blocking probabilities of unbuffered Banyan networks that applies not only for delta networks, but in the general case of any unique-path network. The method applies where sources generate messages with different probabilities, as well as where different destinations have different probabilities of having messages addressed to them.

However, precisely because Banyan networks are unique-path networks, they are not inherently fault-tolerant. The failure of a switching element will necessarily cut off communication between at least one message source and one message sink in the network. A scheme that allows replacement of failed components with idle spares must be used to maintain connectivity. This is the approach used in, *e.g.,* the *extra-stage cube* network [1], or in the *dynamic redundancy* network [13], both of which emulate a (Banyan) indirect cube network and provide fault-tolerance by reconfiguring in the presence of faults.

An alternative to the maintenance of idle spares is to make active use of the "spares" to increase bandwidth, by building a multipath network. Here we mean that, in the course of normal (fault-free) communication, the redundant paths are used in routing packets to their destinations. Some examples of these are the *augmented delta* network [8], the *multibutterfly* network [26], and the *merged delta* network [23].

Both fault-tolerance and performance can be enhanced with the addition of multiple paths. Unfortunately, multipath networks create problems for the traffic theorist. In a Banyan network, if one assumes messages at the inputs are generated by independent processes, the presence or absence of messages at the inputs of any switch in the network is independent of the presence or absence of messages at the other inputs of that switch.[1] Thus the analysis of blocking probabilities in Banyan networks is simplified, and polynomial-time algorithms exist for calculating the exact blocking probability [14]. When multiple paths are allowed, independence is violated.

The author has found in the literature no polynomial-time algorithm that calculates the exact blocking probability of a multipath network, nor any proof that the problem is *NP*- or #*P*-hard. The method described in a later chapter, for synchronous, packet-switched multipath networks, requires the solution of a number of equations that is exponential in the number of communications channels entering a stage in the network. A program that automatically solves these equations exactly, given a description of the network, is presented in what follows; but it cannot be used on large

[1]As will be shown in Section 3.2.

networks, as the running time grows too quickly.

Thus an approximation method must be used to estimate the blocking probability of larger networks. The exact solution remains useful, not only because it is used in the approximation method, but because it allows some evaluation of approximation methods through comparison with exact solutions for small problems. We consider two approximation methods.

The first is direct simulation of the network, where sample input loads are selected and offered to the simulation, the fraction of messages blocked in each is calculated, and a blocking probability is estimated. The second, which is more satisfactory because it achieves the same error bounds in less running time than does direct simulation, is approximation of the solution to the equations, by a Monte Carlo method that we shall describe. This is similar to the approach taken by Harvey and Hills in [11]. Harvey and Hills were considering circuit-switched telephone networks with unique paths; but their approach, which was to find approximate solutions of exact equations, rather than exact solutions to approximate equations, can still be of use here.

1.2 Prior Work

The earliest work in analysis of the performance of interconnection networks was driven by the need to efficiently switch telephone traffic. Some of the earlier work on interconnection networks and their performance, by Clos [7] and Benes [3], concentrated on the design of *non-blocking* networks, networks for which a connection that constitutes a bijective mapping from sources to destinations can always be accomplished without blocking.

Non-isochronous applications such as shared-memory references in a multiprocessor can better tolerate blocking, and thus often use blocking variants of Banyan networks, as presented by Goke and Lipovski [9].

Patel [21] presented a probabilistic analysis of the blocking probability of delta networks, a subset of the more general class of Banyan networks. His work assumed that all sources transmit with uniform probability, and that all destinations are selected with uniform probability. Bhuyan [5] has extended Patel's work to include analysis of the case where each processor has a single favorite destination that is not the favorite destination of any other processor. Kruskal and Snir [15] have extended Patel's work by finding an asymptotic expression for the blocking probability in networks with large numbers of stages.

Analyses that model the buffering that must be used due to blocking in Banyan networks have also been developed; two recent examples include the work of Merchant [18] and Lin and Kleinrock [17]. These models cannot be used for multipath networks, however, due to the above-mentioned correlation of channel loads in a multipath network.

The literature on performance of multistage multipath networks is more sparse. Specific topologies are usually simulated, as in [2], [8], [23], and [16]. A similar problem has been studied in the context of telephone switching systems [12]. However, in telephone switching systems the model is one of a circuit-switched network where the holding time for circuits varies. Furthermore, in the methods described in [12], it is assumed that the networks modeled are symmetric; because there are classes of asymmetric networks that are of interest,[2] and because we are partly interested in calculating blocking probabilities in the presence of (asymmetric) faults, these methods are not satisfactory.

1.3 Motivation

Our goal in this work is to provide a tool that can be used by multiprocessor architects to easily compare the performance of competing multistage interconnection network structures. A secondary goal is to provide an analysis that highlights some of the aspects of interconnection network structure that have particular bearing on performance.

Almost all Banyan networks used in multiprocessors to date have been delta or omega networks, and the performance of these has been studied extensively. Our contribution is in providing a method of calculating the throughputs of Banyan networks of arbitrary interconnection structure and with unusual switching components. The method allows easy modeling of cases with general destination distributions and general source transmission probabilities.

Multipath, multistage network performance has been less widely studied. The correlation of channel loads can have significant effects on the performance of these networks. The methods we develop calculate the joint probability mass function of groups of channels between stages of the network to allow calculation of multipath network performance parameters. We hope that the multipath network designer who wishes to examine the results of a design decision will be able to achieve some insight from our model.

[2] E.g., the randomly-interwired butterflies of [16].

Random interwiring of multipath networks (as described in [16]) for fault-tolerance yields a large space of possible network structures; one might generate a number of these, insert faults randomly and select the one with the best performance. In [2], Chong *et al.* describe the use of simulation to evaluate different circuit-switched multipath networks, including randomly- and deterministically-interwired networks. The method we develop allows a quick measure of performance in fewer steps than does direct simulation of the network.

1.4 Approach

We use a simple model of offered traffic in our calculation of blocking probabilities for both Banyan and multipath networks. In our model, although different inputs in the network can have varying probabilities of transmission, we assume that the messages presented at the inputs to the network are produced by independent memoryless processes.

This model is known to be optimistic. The throughput calculated in an analysis using this model will be higher than the throughput calculated in simulations that include buffering, or in more detailed analyses that model buffering. We can understand one reason for the optimism of the model by considering that it cannot account for multiple conflicting requests presented to the network. In the case of, say, a three-way collision between requests competing for the same resource in one cycle, only one request can be serviced, and there will necessarily be a collision again at the next cycle between the remaining requests.

Patel has noted the optimism of this memoryless model and comments that in his simulations that took buffering into account, the probability of successful message transmission varied only slightly from that predicted by the memoryless model [21]. Nussbaum *et al.* examined the analogous assumption for circuit-switched interconnects and reported that the error in the memoryless model was at all loads less than 10%, and suggested that for most purposes the memoryless model should probably be preferred for its simplicity [20].

Bhandarkar examined in particular the probability that a memory element in a distributed-memory multiprocessor system would be busy, and compared his analysis, which did model buffering of blocked requests, with a memoryless model [4]. His conclusion was that where the ratio of the number of memories to the number of processors was greater than 0.75, the expected number of busy memories in the memoryless model is always within 6 to 8% of that in the buffering model.

The results reported by Chang *et al.* were similar: in examining the throughput of multiprocessor memories, they found that a memoryless model was always 6 to 8% more optimistic than the results they generated with an analysis that modeled queueing of memory requests [6].

Given that our primary goal is to provide multiprocessor architects with a means of comparing the performance of alternative network structures, we deemed the known optimism of the memoryless model to be worth the simplicity it affords, especially in view of the complexity of the problem of deriving blocking probabilities in multipath networks.

1.5 Outline

In the remainder of this document, we further define the problem of calculating the throughput and blocking probability of a multistage interconnection network and present methods for solving it.

In Chapter 2 we define the problem and our model specifically enough to allow the description of a method for analyzing the performance of Banyan networks. Chapter 3 presents that method, as well as a program that calculates the performance parameters numerically or symbolically.

Chapter 4 further defines our model to include multipath networks and presents equations for exactly analyzing the performance of multipath networks. Chapter 5 presents means of approximating the performance parameters of multipath networks. Finally, we have included a listing of our procedures for Banyan network performance evaluation in an appendix, as these were compact enough to make such presentation practical.

Chapter 2

Problem Statement

2.1 The Model

An *indirect* network is one in which the network switching elements are segregated from the inputs and outputs of the network, as in Figure 2.1. The message sources inject messages into the network at the inputs, which in Figure 2.1 are depicted on the left side and are labeled I_0 through I_7. Messages are routed through the network and arrive at the message sinks on the right side, labeled O_0 through O_7. In a multiprocessor, the network input channels might connect to 8 processing elements, and the output channels might connect to the same 8 processing elements.

The particular class of indirect networks that we model is the class of multistage, unbuffered, synchronous, packet-switched networks. Such a network might look like the one depicted in Figure 2.2. This network has multiple stages: if we consider the stage consisting of all the sources to be stage 0, then stage 1 consists of the column of switching elements connected directly to the sources; stage 2 the column of switches to the right of stage 1, etc.

The networks we consider are *self-routing:* each message contains the information necessary to route the message from the source where it is injected to the sink that is its destination. No global information is used in the routing process, so that the probability mass function of the loads on the output channels of a switch can be calculated from the probabilities of the loads on the input channels. As a simple example, in the indirect cube of Figure 2.2, 2×2 switching elements route on individual bits of the destination address, starting with the low-order bit. There are $\log_2 8 = 3$ address

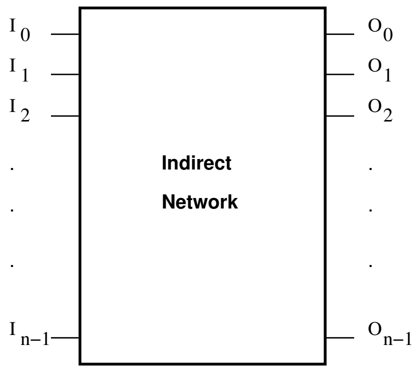

Figure 2.1: An indirect network is one in which switching elements are segregated from the inputs and outputs of the network. Messages enter the network through the input channels on the left side, and are routed to the output channels on the right side.

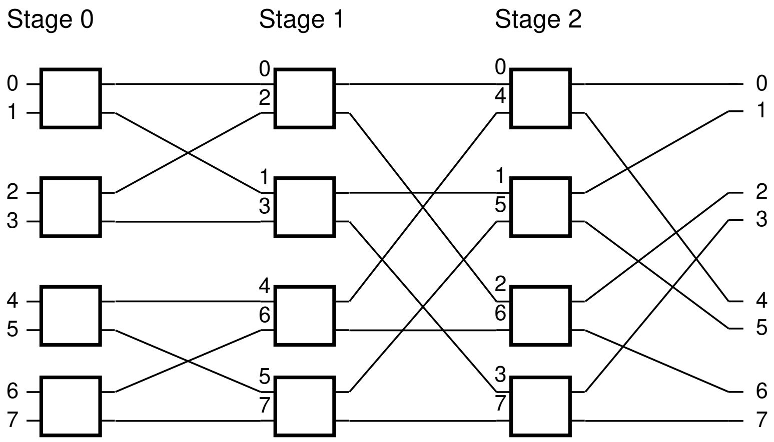

Figure 2.2: An 8×8 Banyan network. This is a unique-path network called an *indirect cube*; multipath networks will be treated in Chapter 4.

bits; the switching elements at stage 0 route on the 2^0's bit; those at stage 1 on the 2^1's bit, and those at stage 2 on the 2^2's bit. A cleared address bit indicates that the message should be routed through the upper channel; a set address bit indicates that it should be routed though the lower channel. Thus a message addressed to destination $6 = 110_2$ leaves stage 0 through an upper channel, stage 1 through a lower channel, and stage 2 through a lower channel.

Blocking occurs in the 2×2 crossbar when two messages arriving at the inputs are both to be routed through the same output channel. Both requests cannot be serviced, and so one of the messages is routed through the output channel, and the other is blocked. In our model, both messages have equal likelihood of being routed through the output channel, and many switching elements behave this way; but one might easily modify the analyses we present to change this assumption.

We also consider networks in which the channels between stages can carry more than one message. Kruskal and Snir referred to such networks as *dilated* networks [15], and we follow their lead here; furthermore we call switching elements in which the output ports can pass more than one message *dilated* switching elements. We refer to each of the dilated output ports as a *logical direction*. If a switch has N input ports, each of which can receive a single message, and M output ports, each of which can send up to K messages simultaneously, we call it an $N \times M$, dilation K switch.

We calculate the throughput of the network under the following assumptions:

- The processes generating messages at the sources are independent and memoryless. With some specified probability p_i, each source i generates or fails to generate a single message at the beginning of each cycle. Each generated message is directed to a stage 1 switch.

- The network is synchronous: at each cycle messages move from stage i to stage $i + 1$.

- The network is treated as unbuffered (as described in Section 1.4): if a message is blocked at some stage, it is considered to be lost, and does not in any way affect the future states of the system.

- If $A_0, A_1, \ldots$ are random variables representing the addresses of messages generated in some particular cycle by message sources $0, 1, \ldots$, then the A_i are independent and identically distributed; the distribution can be specified as a parameter of the model.

We define our model further in Chapter 4, extending it as necessary for multipath networks.

2.2 The Problem

We are interested in deriving the bandwidth, or throughput, of a multistage interconnection network – that is, the expected number of messages it delivers in a cycle. We calculate this number by finding the probability mass functions[1] of the loads on channels leading to sinks.

Suppose that the network has M sources. Call the probability that the ith source generates a message in a given cycle P_i. If we say that B is the bandwidth and P_S is the probability of successful message transmission, then we may calculate P_S as the ratio of B to the expectation of the input message loading. P_S will vary with the input loading, because of internal blocking in the network. B, too, will vary because of internal blocking and also directly with the number of messages entering the network. Thus we can better express P_S and B as functions of the P_i, giving us the relation:

$$P_S\left(P_0, P_1, \ldots, P_{M-1}\right) = \frac{B\left(P_0, P_1, \ldots, P_{M-1}\right)}{\sum_{i=0}^{M-1} P_i} \tag{2.1}$$

Thus our problem is finding the probability mass functions of the loads on channels leading to sinks. These probability mass functions can also be used to specify other information besides mean throughput; if the network is not symmetric, or if a non-uniform destination address distribution for injected messages is specified, or if different sources are specified to have different probabilities of message generation, then the loads on individual channels leading to sinks will be of interest in find the effects of the asymmetries on traffic to particular destinations.

The quantities B and P_S will typically vary smoothly with the source transmission probabilities P_i. Let us consider a simple case. If the destination distribution is uniform, all sources i have equal probability of generating messages, and we vary P_i between 0 and 1 for the network of Figure 2.2, the resulting graphs for the bandwidth and probability of successful message transmission are as shown in Figures 2.3 and 2.4, respectively.

The probability of successful message transmission is close to 1 when there are very few messages injected into the network, because there is little

[1] Joint probability mass functions, in the case of multipath networks.

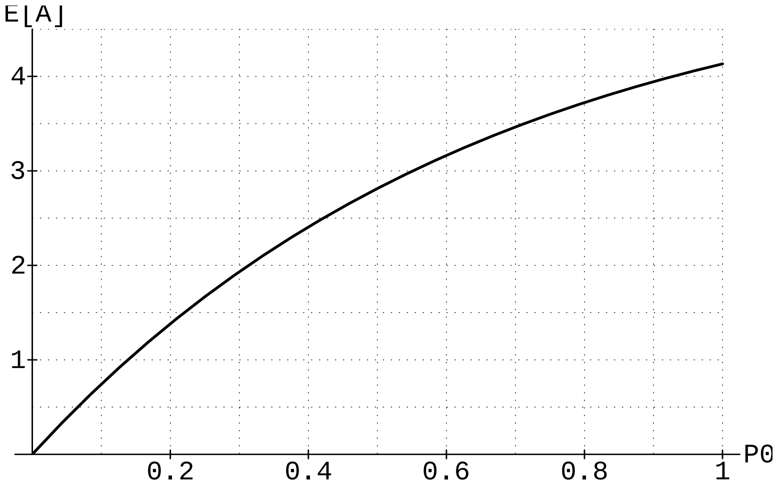

Figure 2.3: Bandwidth (labeled `E[A]`) plotted versus message generation probability (labeled `P0`) for the network of Figure 2.2. Here the destination address distribution is uniform.

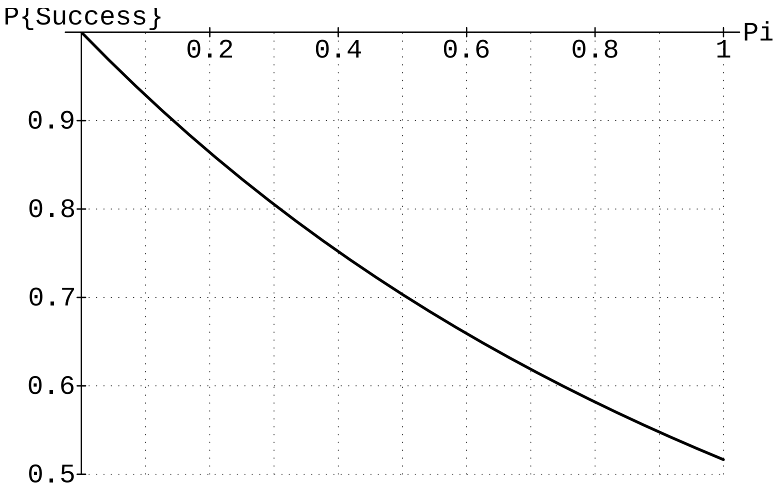

Figure 2.4: Probability of successful message transmission (labeled `P{Success}`) plotted versus message generation probability (labeled `P0`) for the network of Figure 2.2, with a uniform destination address distribution.

blocking in a nearly empty network. P_S decreases as the number of messages injected into the network increases. The bandwidth or throughput starts at 0, when no messages are being injected into the network, and, because of blocking, increases less than linearly as the probability of message transmission increases.

Chapter 3

Performance of Banyan Networks[1]

3.1 Introduction

In this chapter we present a method of calculating the throughput of a Banyan network. As described in Section 1.2, Patel [21] and Kruskal and Snir [15] have presented solutions to this problem for regular variants made up of crossbar switching devices, but we present a method that works for Banyan networks of arbitrary interconnection structure and allows modeling of some unusual switching devices.

Consider first the probability mass function of the message load on a single channel in a Banyan network. The channel may either be carrying a message, in which case its message load is one, or it may be idle, in which case its message load is zero. Let the random variable l denote the message load. The two values that l can take on partition the space of possible loading configurations for the network into two disjoint subsets. l is then a Bernoulli random variable, and we use the notation $p_l(l_0)$ to denote the value of its probability mass function at l_0.[2] We denote the value of the $\mathcal{Z}$-transform of l's probability mass function at z with the notation $p_l^T(z)$.

Our approach will be to define three operations on the probability mass functions of channel loads. These are called *bundling, switching,* and *concentration.* They are represented graphically as depicted in Figure 3.1. We

[1] The work described in this chapter was performed jointly with Dr. Thomas F. Knight, Jr. and has been described in [14].

[2] In later sections we will also use the notation $\mathrm{P}\{l = l_0\}$, when this is convenient.

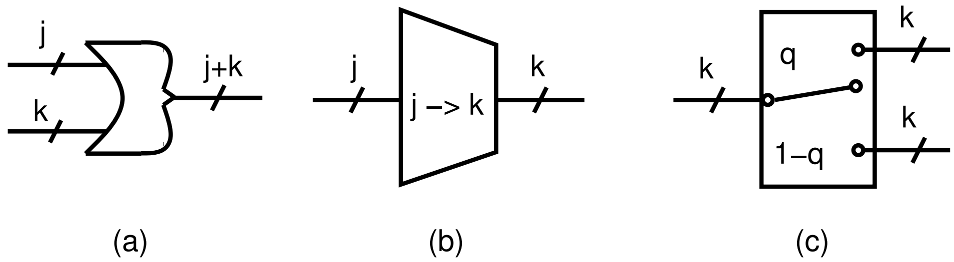

Figure 3.1: (a) The symbol for bundling two input bundles into one. (b) The symbol for concentrating j channels to k channels. (c) The symbol for switching with probability q to the top output channel, and $1 - q$ to the bottom output channel.

compose these operations to model switching elements, and further according to the interconnection structure of the network. The result is an operation that transforms the probability mass functions of the loads on input channels to the probability mass function of the load on an output channel.

3.2 Loads on Banyan Network Channels at a Single Stage are Independent

We require a simple proof to proceed. We will be forming the sum of the loads on distinct channels in a single stage in a Banyan network, and thus we need to understand how, if at all, the random variables we are summing are correlated. It turns out that these loads are in fact independent. A proof for the special case of delta networks is presented in [21]; here we present a different proof for the general case.

The proof is straightforward. Note first that, if messages are generated at source nodes by mutually independent random processes, and the sets of messages on distinct channels entering a switching node originate at disjoint sets of source nodes, then the loads on those channels are necessarily independent.

We now claim that the sets of messages on distinct channels entering any switching node in a Banyan network satisfy this criterion: i.e., they originate at disjoint sets of sources.

For, consider: if channel A and channel B are two channels entering a switching node, and a message on channel A and a message on channel B originate at a single source, then it must be the case that at least two paths exist from that source to any sinks accessible from the switching node: one path that uses channel A and one that uses channel B. But this is impossible in a Banyan network, as Banyan networks are in fact those in which there is exactly one path from each source to each sink.

Thus the sets of messages on distinct channels entering any switching node in a Banyan network must originate at disjoint sets of sources, and so the loads on the channels entering any switching node in a Banyan network must be mutually independent, as was to be proved.

3.3 Bundling

We call the operation of summing the loads on a group of channels *bundling.* We will call such a group of channels a *bundle.*

Because channel loads are independent, if we are summing loads a and b, then we form the convolution of their probability mass functions. We use the notation

$$\mathcal{B}\left[p_a(a_0)\,, p_b(b_0)\right] \equiv p_a(a_0) * p_b(b_0) \tag{3.1}$$

Of course, this operation can be performed on bundles, as well as on single channels. The result of bundling two bundles composed respectively of n and m single channels is a bundle whose load can take on values ranging from 0 through $n + m$. We depict in Figure 3.2 one possible loading probability mass function of a bundle composed of 8 single channels.

In the $\mathcal{Z}$-domain, bundling becomes multiplication of the $\mathcal{Z}$-transforms of the loading probability mass functions in question:

$$\mathcal{Z}\left[\mathcal{B}\left[p_a(a_0)\,, p_b(b_0)\right]\right] = p_a^T(z) \cdot p_b^T(z) \tag{3.2}$$

3.4 Concentration

Suppose in an $N \times M$, dilation K switch[3] more than K arriving messages are to be routed in a particular logical direction. Some of the messages are

[3] See Section 2.1 for an explanation of dilated switches.

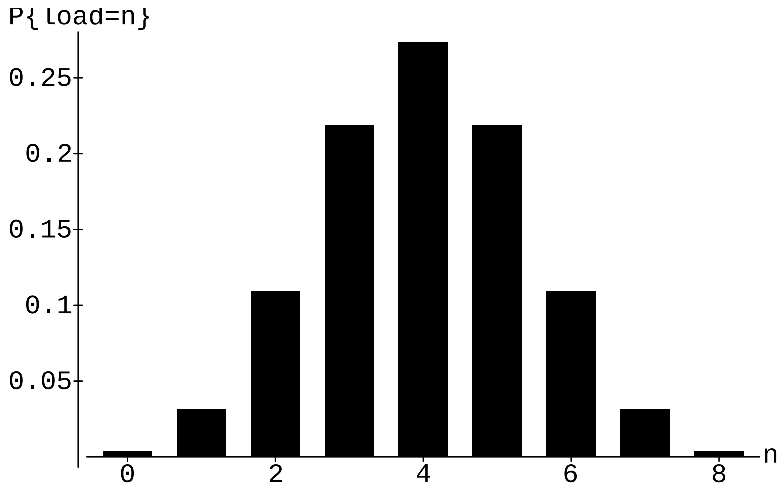

Figure 3.2: Loading probability mass function for an eight-channel bundle, where each channel carries a message with probability 1/2.

then blocked and must be dropped. We call the operation that corresponds to this situation *concentration.*

More specifically, suppose we have a bundle of N single channels, whose load we call a, and we wish to direct messages from it into a bundle of K single channels, whose load we call b. Of course if $N \leq K$, $p_b(l_0) = p_a(l_0)$ for all loads l_0, because in this case none of the messages on the input bundle will ever be blocked. If $N > K$, we calculate the probability mass function of b as follows:

- Because the output bundle carries fewer than K messages exactly when the input bundle carries fewer than K messages, we have that $p_b(l_0) = p_a(l_0)$ where $l_0 < K$.
- The output bundle will carry K messages whenever the input bundle carries *at least* K messages; thus we have that $p_b(K) = \sum_{i=K}^{N} p_a(i_0)$.
- Because the output bundle cannot carry more than K messages, $p_b(l_0) = 0$ for $l_0 > K$.

Intuitively, then, we can think of the effect of concentration on the input loading probability mass function as truncating it at K, by setting prob-

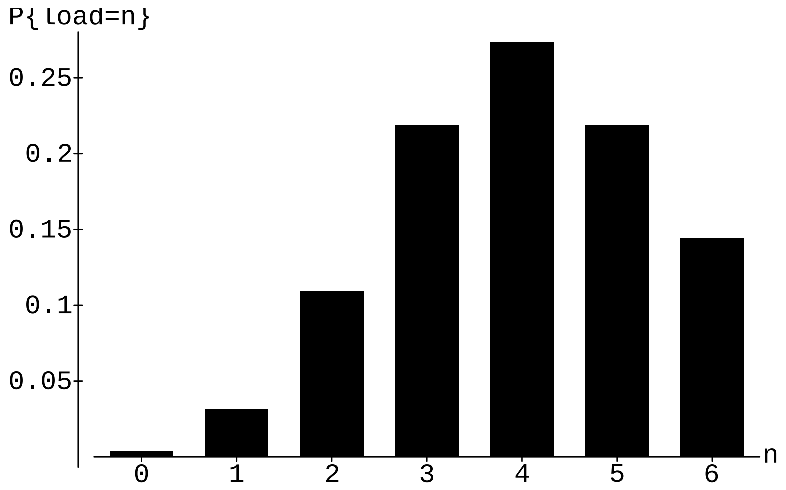

Figure 3.3: 6-concentration of the loading probability mass function of Figure 3.2.

abilities for loads greater than K to 0 and adding to the probability for K the probabilities for all greater input loads. In Figure 3.3 we show the result of concentrating to 6 channels the loading probability mass function of Figure 3.2.

If $\delta(n)$ denotes the value of the unit impulse function at n, we can express the loading probability mass function of an N-channel bundle as an impulse train with value k_i at i:

$$p_l(l_0) = \sum_{i=0}^{N} k_i \delta(l_0 - i)$$

If $u\,(n)$ the value of the unit step function at n, we can express concentration of this N-channel bundle to a K-channel bundle as follows:

$$\mathcal{C}_{N,K}\left[p_l(l_0)\right] \equiv p_l(l_0)\,u(K - l_0) + \left(\sum_{l_1=K+1}^{N} p_l(l_1)\right)\delta(l_0 - K) \tag{3.3}$$

In the $\mathcal{Z}$-domain, we have

$$\mathcal{Z}\left[\mathcal{C}_{N,K}\left[p_l(l_0)\right]\right] = p_l^T(z) - \sum_{l_1=K+1}^{N} p_l(l_1)\, z^{l_1} + \left(\sum_{l_1=K+1}^{N} p_l(l_1)\right) z^K$$

Combining the two summations, we get

$$\mathcal{Z}\left[\mathcal{C}_{N,K}\left[p_l(l_0)\right]\right] = p_l^T(z) + \sum_{l_1=K+1}^{N} p_l(l_1)\left(z^K - z^{l_1}\right) \tag{3.4}$$

3.5 Switching

We call the elementary operation of directing the messages on a bundle to two other bundles of the same width as the input bundle *switching*. Here we do not mean to use the term in precisely the sense that it is used when we speak of, *e.g.,* a 2×2 switch. In the elementary operation we call switching, no blocking is modeled; no messages can be lost. What we are modeling instead is the direction of messages to separate ports in routing.

We specify the probability that the messages on the input load are switched in the direction of the output load. This probability is calculated in accordance with the destination address distribution (as will be described in Section 3.6); but as an example, for the 2×2 crossbars in the network of Figure 2.2 under a uniform destination address distribution the probability specified for the switching operation will be 1/2.

Thus the switching operation is performed on an input loading probability mass function and a switching probability, and its result is an output loading probability mass function. Call the load on the input bundle a, and that on the output bundle b, and say that the input bundle (and perforce the output bundle) is composed of N single channels.

We form $p_b(b_0)$ by conditioning on the number of messages on the input bundle:

$$p_b(b_0) = \sum_{a_0=0}^{N} p_{b|a}\left(b_0 \mid a_0\right) p_a(a_0)$$

To evaluate the conditional probability, let q be the probability that an input message is switched to the output bundle. By independence of message destinations, each message is switched independently, and thus the number of messages switched to the output bundle is binomially distributed, because it is the number of successes in a_0 independent Bernoulli trials with probability q of success:

$$p_{b|a}(b_0 \mid a_0) = \binom{a_0}{b_0} q^{b_0}(1-q)^{a_0-b_0}$$

Substituting, we have

$$\mathcal{S}\left[p_a(a_0), S\right] \equiv p_b(b_0) = \sum_{a_0=0}^{N} p_a(a_0) \binom{a_0}{b_0} q^{b_0}(1-q)^{a_0-b_0} \tag{3.5}$$

In the $\mathcal{Z}$-domain, we take an analogous approach. Note that the number of messages routed to the output channel is the sum of a random number of identically distributed random variables. The number of summands is the number of messages on the input load. The summands themselves are Bernoulli random variables that are 1 when the message in question is routed to the output bundle and 0 when it is not.

If we use the random variable c to denote one of the summands, its probability mass function is given by

$$p_c(c_0) = (1-q)\delta(c_0) + q\delta(c_0 - 1)$$

with $\mathcal{Z}$-transform

$$p_c^T(z) = 1 - q + qz$$

Thus we have

$$\begin{aligned} \mathcal{Z}\left[\mathcal{S}\left[p_a(a_0), q\right]\right] &= p_a^T\left(p_c^T(z)\right) \\ &= p_a^T(1 - q + qz) \end{aligned} \tag{3.6}$$

Of course, if we cascade K switching operations whose probabilities are $q_1, q_2, \ldots, q_K$, the effect on the probability that an individual message is routed to the output bundle is the same as if we performed one switching operation with $q = \prod_{i=1}^{K} q_i$.

The (predictable) effect of switching upon a loading probability mass function is to decrease the mean. The effect on the distribution of Figure 3.2 of switching with $q = 1/2$ is shown in Figure 3.4.

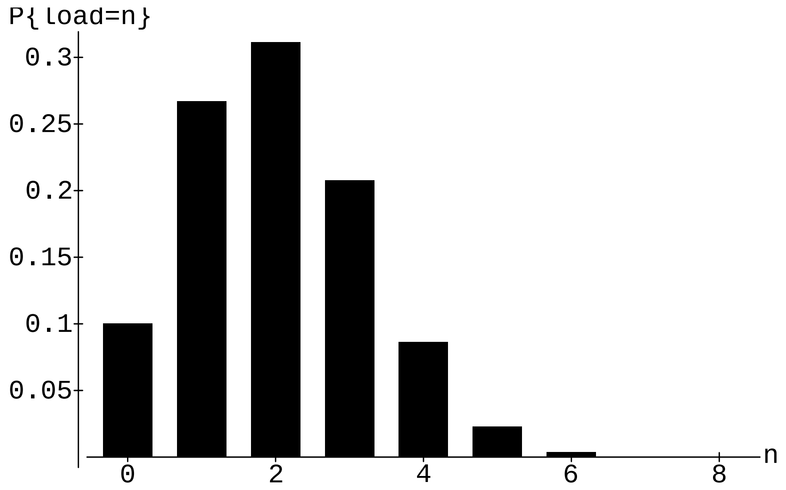

Figure 3.4: The effect of switching the loading probability mass function of figure 3.2 with probability 1/2.

3.6 Deriving Switching Probabilities from Message Destination Distributions

The technique we use for deriving switching probabilities from message destination distributions has also been used by Lin and Kleinrock in [17].

As described in Section 2.1, the addresses of distinct messages injected into the network are independent and identically distributed. Suppose that the message sinks are numbered $0, 1, \ldots, N-1$, and consider a switch X for which the set of accessible message sinks is S. Suppose that X has M output ports. By uniqueness of paths in a Banyan network, the ports must have disjoint sets $S_1, S_2, \ldots, S_M$ of accessible destinations, and because the destinations accessible through the output ports are all the destinations, we must have that $\bigcup_{i=1}^{M} S_i = S$. An example is depicted in Figure 3.5.

We wish to know the probability that an arbitrary message arriving at switch X is directed in direction i. Suppose that some message W with destination given by the random variable D is injected into the network. The value we are looking for is the conditional probability that W is addressed to a destination in the set S_i, given that it has arrived at switch X. We have

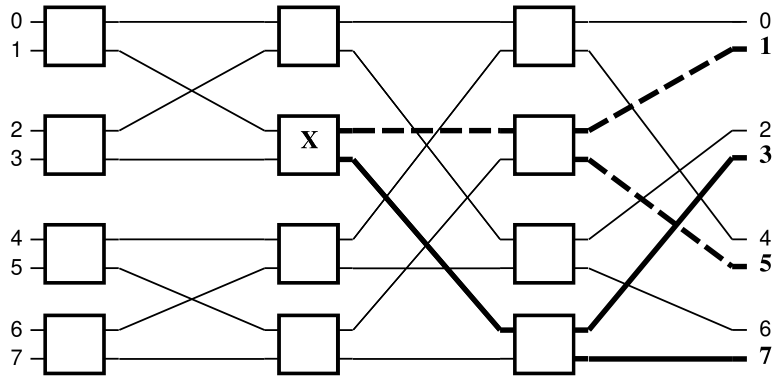

Figure 3.5: The destination set of the switch X is $\{1,3,5,7\}$. The destination set of the upper channel is $\{1,5\}$; that of the lower channel is $\{3,7\}$.

$$\begin{aligned} \mathrm{P}\{D \in S_i \mid D \in S\} &= \frac{\mathrm{P}\{(D \in S_i) \cap (D \in S)\}}{\mathrm{P}\{D \in S\}} \\ &= \frac{\mathrm{P}\{D \in S_i\}}{\mathrm{P}\{D \in S\}} \\ &= \frac{\sum_{s \in S_i} \mathrm{P}\{D = s\}}{\sum_{s \in S} \mathrm{P}\{D = s\}} \end{aligned} \tag{3.7}$$

where the last expression follows from mutual exclusivity of destinations.

3.7 Example: the $2^k \times 2^k$ Crossbar

As an example of both the symbolic and numeric use of our method, we derive a well-known expression for the throughput of the $2^k \times 2^k$ crossbar.

We form a schematic representation of the crossbar with a combination of our operators. First we construct a bundle of 2^k channels by bundling the single-channel inputs k times. Then we switch the messages on the bundle k times, to form 2^k bundles, each of which can hold 2^k messages. Finally we concentrate these 2^k-wide bundles to single channels, thereby modeling the blocking that takes place in the crossbar. Figure 3.6 shows the result for an 8×8 crossbar.

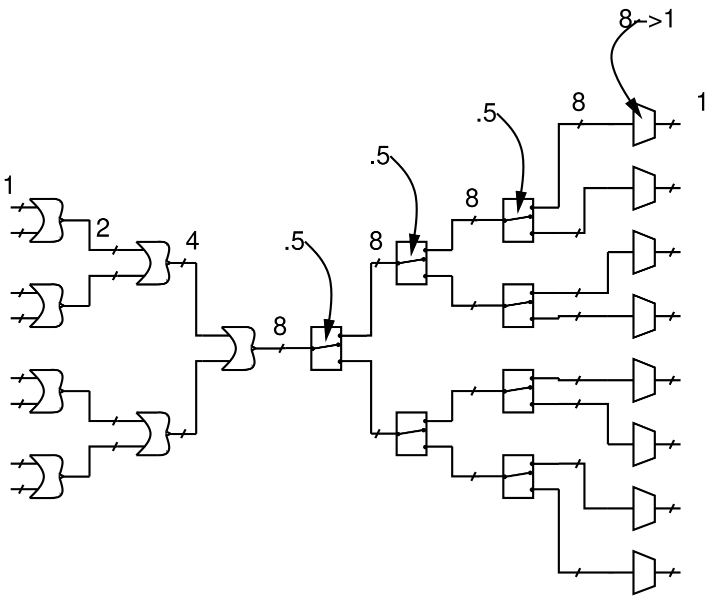

Figure 3.6: Schematic representation of an 8×8 crossbar network. Here we show the switching probabilities set for a uniform destination address distribution.

For brevity's sake, in our analysis we assume that all input channels have a single probability Q of transmitting, and that the destination address distribution is uniform. It will be evident that the derivation would otherwise proceed in the same fashion, but would be more lengthy.

Suppose that the input channels have probability Q of transmitting during a cycle. If we call the load on an input channel y, the loading probability mass function for an input channel will then be

$$p_y(y_0) = Q\delta(y_0 - 1) + (1 - Q)\,\delta(y_0)$$

with $\mathcal{Z}$-transform

$$p_y^T(z) = Qz + (1 - Q)$$

Bundling k times, we get for the transform of the probability mass function of the load x_c on the bundle entering the switches

$$p_{x_c}^T(z) = \left(p_y^T(z)\right)^{2^k}$$

Let x_s be the load on a channel after the stages of switching, but before concentration. We switch k times with probability $1/2$ at each stage, the result being the same as switching once with probability $1/2^k$:

$$\begin{aligned}
p_{x_s}^T(z) &= \left(p_y^T\left(\frac{z + 2^k - 1}{2^k}\right)\right)^{2^k} \\
&= \left(Q\left(\frac{z + 2^k - 1}{2^k}\right) + (1 - Q)\right)^{2^k} \\
&= \left(\frac{Q}{2^k}z + \left(1 - \frac{Q}{2^k}\right)\right)^{2^k} \\
&= \sum_{l=0}^{2^k} \binom{2^k}{l} \left(1 - \frac{Q}{2^k}\right)^l \left(\frac{Q}{2^k}z\right)^{2^k - l}
\end{aligned}$$

To make the expression clearer, we substitute $M = 2^k$, rearrange, and invert the $\mathcal{Z}$-transform:

$$\begin{aligned}
p_{x_s}^T(z) &= \left(\frac{Q}{M}\right)^M \left(\sum_{l=0}^{M} \binom{M}{l} \left(\frac{M}{Q} - 1\right)^l z^{M-l}\right) \\
p_{x_s}(x_{s0}) &= \left(\frac{Q}{M}\right)^M \left(\sum_{l=0}^{M} \binom{M}{l} \left(\frac{M}{Q} - 1\right)^l \delta(x_{s0} - (M - l))\right)
\end{aligned}$$

We can save ourselves some work in performing the concentration from 2^k (that is, M) channels to one channel by making use of the following device. We note that, because we are concentrating to a single channel, the only possible loads for the channel are 0 and 1. We recall from Section 3.4 that concentration will retain the probability for a load of 0, as 0 is less than the maximum load on the channel. The probability for a load of 1 will necessarily be the complement of that for 0. First we take the probability that $x_s = 0$:

$$p_{x_s}(0) = \left(\frac{Q}{M}\right)^M \left(\sum_{l=0}^{M} \binom{M}{l} \left(\frac{M}{Q} - 1\right)^l \delta(-(M-l))\right)$$

We simplify the expression by noting that the terms where $l \neq M$ will all be 0:

$$\begin{aligned} p_{x_s}(0) &= \left(\frac{Q}{M}\right)^M \left(\frac{M}{Q} - 1\right)^M \\ &= \left(1 - \frac{Q}{M}\right)^M \end{aligned}$$

If we call the load on an output channel l, the loading probability mass function for an output channel is then given by

$$p_l(l_0) = \left(1 - \frac{Q}{M}\right)^M \delta(l_0) + \left(1 - \left(1 - \frac{Q}{M}\right)^M\right) \delta(l_0 - 1) \tag{3.8}$$

The expected load on an output channel is then

$$\mathrm{E}[l] = \left(1 - \left(1 - \frac{Q}{M}\right)^M\right)$$

There are M output channels, so the expected load on all of them, or the throughput of the crossbar, is

$$M\mathrm{E}[l] = M\left(1 - \left(1 - \frac{Q}{M}\right)^M\right)$$

The expected load on an input channel was Q, so that the total expected input load is MQ. We can now use Equation (2.1) to derive the probability of successful message transmission in $M \times M$ crossbar:

$$P_S = \frac{\left(1 - \left(1 - \frac{Q}{M}\right)^M\right)}{Q}$$

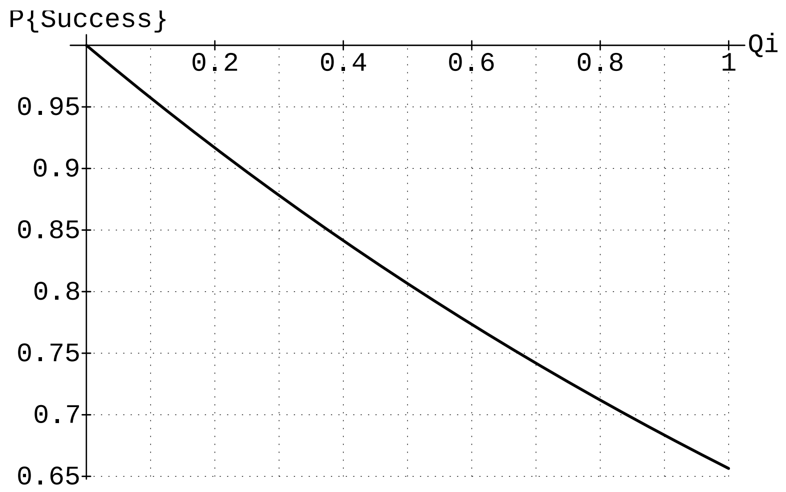

Figure 3.7: The probability of successful message transmission (labeled `P{Success}`) as a function of the probability that a source is transmitting (labeled `Qi`), in an eight-by-eight crossbar network with a uniform destination distribution.

We plot the probability of successful message transmission against the source transmission probability in Figure 3.7.

3.8 Automatic Calculation of Numerical Values for Performance Parameters

We present in Appendix A a package of Mathematica procedures that implement the elementary operations we have described. Of course the operations are easily implemented in other languages, but it is advantageous to use a symbolic algebra package if one wishes to derive symbolic expressions for performance.

We can use this package to implement procedures that operate on source loading probability mass functions and return the loading probability mass functions for channels leading to sinks. As an example, we turn again to the network of Figure 2.2. The tree whose root is one of the sinks and whose leaves are the sources is depicted in Figure 3.8.

Assuming for clarity's sake that the destination address distribution is

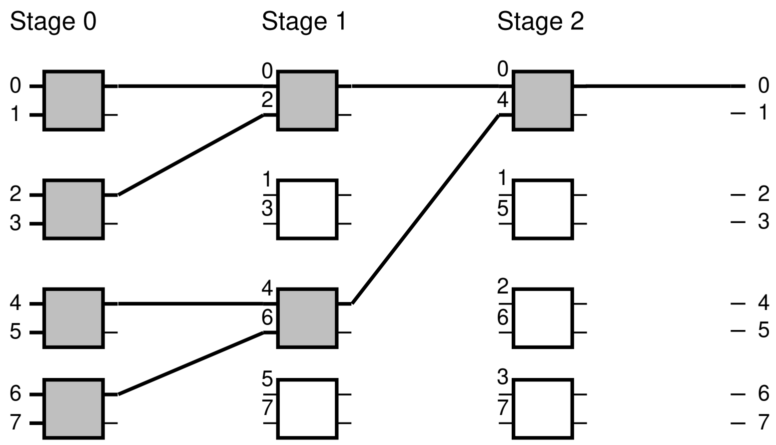

Figure 3.8: The tree of channels leading to a sink in the network of Figure 2.2.

uniform, we might use our package to model the 2×2 crossbar as follows:

```
crossbar2x2[PMF1_, PMF2_] :=
  concentrate[switch[bundle[PMF1, PMF2],
                     1/2],
              1]
```

If we also assume (again, in the interests of brevity; it will be clear that the calculation in the general case is no more complex) that all sources transmit with equal probability, we can take advantage of the symmetry of the network to calculate the loading probability mass function of a channel leading to a sink as follows:

```
threeStageDelta[q_] :=
  Block[{inputPMF, stage1PMF, stage2PMF},
    inputPMF := [(1-q), q];
    stage1PMF := crossbar2x2[inputPMF, inputPMF];
    stage2PMF := crossbar2x2[stage1PMF, stage1PMF];
    crossbar2x2[stage2PMF, stage2PMF]
        ]
```

Here the probability that a source is transmitting is specified as the input parameter; three levels of switching are performed; the result of the last is returned.

We may calculate the resulting bandwidth and probability of successful message transmission from Equation (2.1), as is done in Section 3.7. The results are plotted in Figures 2.3 and 2.4, on page 11.

3.9 Modeling an Unusual Switching Component

We use an example to illustrate the modeling of an unusual switching component – an 8×4, dilation 2 switch.[4] Such switches are more usually used in multipath networks, as we shall see in Chapter 4, but Banyan networks with replicated links are not unknown, and Kruskal and Snir have analyzed regular variants in [15].

3.9.1 An Application for an 8×4, dilation 2 Switch

We can use standard 4×4 crossbars to build a 16×16 indirect binary cube network, as depicted in Figure 3.9. The methods of analysis of the performance of this network follow directly those of Sections 3.7 and 3.8.

As an alternative, we might choose to use a different sort of switching element in the first stage, to improve performance. This switching element – an 8×4, dilation 2 switch – has eight input channels, but switches messages in only four logical directions, with two output ports for each of these logical directions. If only one message is switched in a particular direction, the output port is picked randomly. If two messages are switched in the direction, both ports are used; if more than two messages are switched in the direction, the excess messages are blocked.

In Figure 3.10, we show how we might modify the first stage of the 16×16 indirect cube network to make use of the dilated switching component. The second stage must still use 4×4 crossbars, to select the particular output channel to which the message is directed.

Although it might appear that we have constructed a multipath network here, in fact we have not. The numbers appearing next to output ports on the dilated components in Figure 3.10 are logical direction numbers, and it is to be noted that both outputs for a particular logical direction lead to

[4] See Section 2.1 for an explanation of this terminology.

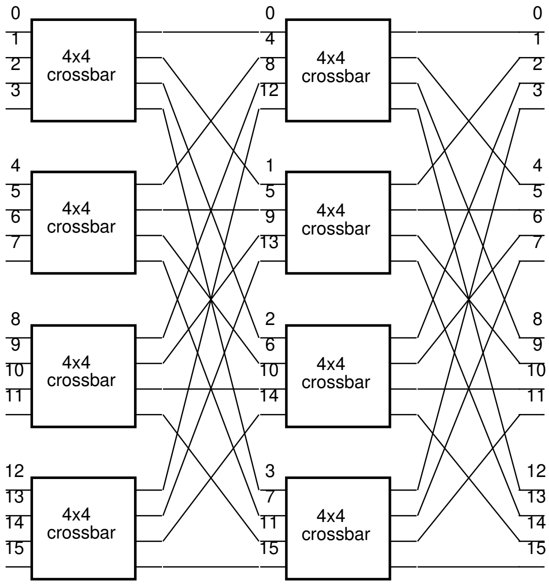

Figure 3.9: A 16×16 indirect binary cube network built from standard 4×4 crossbars.

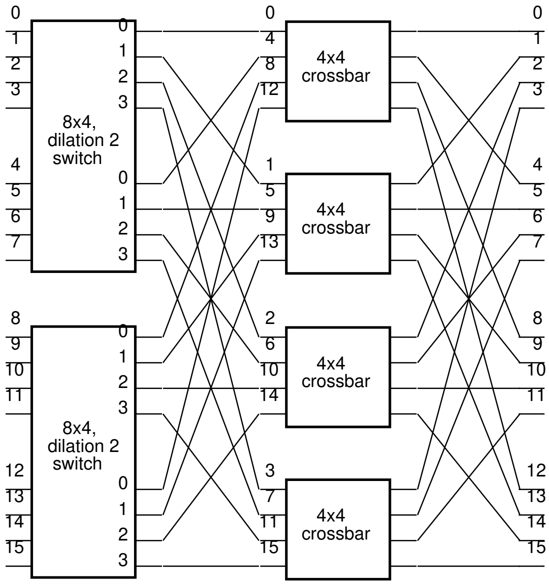

Figure 3.10: A 16×16 indirect binary cube network with the first stage built from 8×4, dilation 2 switches, and the second stage from standard 4×4 crossbars.

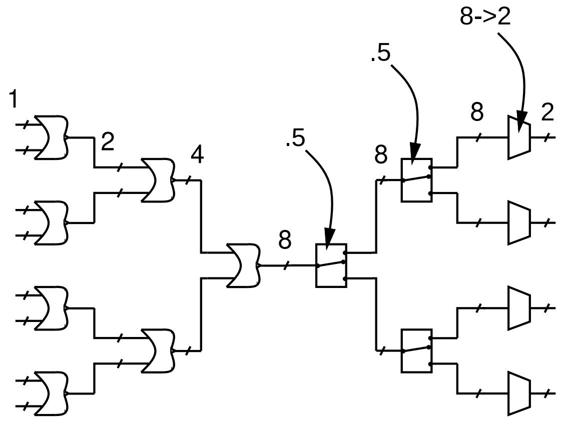

Figure 3.11: Schematic representation of an eight-by-four, dilation two switching component. The switching probabilities are for a uniform destination address distribution.

the same second-stage switch; the model will reflect this. Thus we have four two-channel bundles leading from each first-stage component.

3.9.2 Deriving Expressions for the Performance of the 8×4, Dilation 2 Switch

A schematic model of the 8×4, dilation 2 switch is shown in Figure 3.11. Note that, in our model, the only difference between this component and the 8×8 crossbar of Figure 3.6 is that there are only two stages of switching, and the final concentration is to two channels, rather than to one. Here we gain an intuition from our model: we noted that concentration was where blocking occurred. Because there is less concentration in the new network, there will be less blocking.

The derivation follows that of the $2^k \times 2^k$ crossbar in Section 3.7. Again, for brevity's sake we assume a uniform destination distribution. Call the load on an input channel y. Assuming all inputs transmit with probability Q, the loading probability mass function for an input channel is

$$p_y(y_0) = Q\delta(y_0 - 1) + (1 - Q)\,\delta(y_0)$$

with $\mathcal{Z}$-transform

$$p_y^T(z) = Qz + (1 - Q)$$

Call the load on the bundle entering the switches x_c. The transform of the probability mass function of x_c is then

$$p_{x_c}^T(z) = \left(p_y^T(z)\right)^8$$

We switch twice with probability 1/2 each time, the result being the same as switching once with probability 1/4. If x_s is the load on a channel after the two stages of switching, we have

$$\begin{aligned} p_{x_s}^T(z) &= \left(p_y^T\left(\frac{z+3}{4}\right)\right)^8 \\ &= \left(Q\left(\frac{z+3}{4}\right) + (1-Q)\right)^8 \\ &= \left(\frac{Q}{4}z + \left(1 - \frac{Q}{4}\right)\right)^8 \\ &= \sum_{l=0}^{8} \binom{8}{l} \left(1 - \frac{Q}{4}\right)^l \left(\frac{Q}{4}z\right)^{8-l} \end{aligned}$$

Now we invert the transform:

$$p_{x_s}(x_{s_0}) = \left(\frac{Q}{4}\right)^8 \left(\sum_{l=0}^{8} \binom{8}{l} \left(\frac{4}{Q} - 1\right)^l \delta(x_{s_0} - (8 - l))\right)$$

We concentrate to two channels here, so that it still saves us some work to use the technique we did for the crossbar, but it will be a little more complicated to do so.

We take the probability that $x_s = 0$ first. The sum will be zero whenever $l \neq 8$, giving us:

$$\begin{aligned} p_{x_s}(0) &= \left(\frac{Q}{4}\right)^8 \left(\frac{4}{Q} - 1\right)^8 \\ &= \left(1 - \frac{Q}{4}\right)^8 \end{aligned}$$

For $x_s = 1$, the sum will be zero whenever $l \neq 7$, giving us:

$$\begin{aligned} p_{x_s}(0) &= \left(\frac{Q}{4}\right)^8 (8) \left(\frac{4}{Q} - 1\right)^7 \\ &= 2Q\left(1 - \frac{Q}{4}\right)^7 \end{aligned}$$

Call the load on a two-channel output bundle l. We know that $p_l(0) = p_{x_s}(0)$ and $p_l(1) = p_{x_s}(1)$. The only other case for a two-channel bundle is $l = 2$, so the probability for $l = 2$ must be the complement of the other two cases, so we have for the probability mass function of l:

$$\begin{aligned} p_l(l_0) = \left(1 - \frac{Q}{4}\right)^8 \delta(l_0) \quad &+ \quad 2Q\left(1 - \frac{Q}{4}\right)^7 \delta(l_0 - 1) \\ &+ \quad \left(1 - \left(1 - \frac{Q}{4}\right)^7 \left(1 + \frac{7Q}{4}\right)\right) \delta(l_0 - 2) \end{aligned}$$

By our assumptions of uniformity, all four output bundles have the same loading probability mass function, and so the throughput of the switch is E[$4l$]:

$$\mathrm{E}[4l] = 4\left(1 \cdot \left(2Q\left(1 - \frac{Q}{4}\right)^7\right) + 2 \cdot \left(1 - \left(1 - \frac{Q}{4}\right)^7 \left(1 + \frac{7Q}{4}\right)\right)\right)$$

The expected load on an input channel was Q, so that the total expected input load is $8Q$. As for the crossbar, we use Equation (2.1) to derive the probability of successful message transmission:

$$\begin{aligned} P_S &= \frac{Q\left(1 - \frac{Q}{4}\right)^7 + 1 - \left(1 - \frac{Q}{4}\right)^7 \left(1 + \frac{7Q}{4}\right)}{Q} \\ &= \frac{1 + \left(Q - \left(1 + \frac{7Q}{4}\right)\right)\left(1 - \frac{Q}{4}\right)^7}{Q} \\ &= \frac{1 - \left(1 + \frac{3Q}{4}\right)\left(1 - \frac{Q}{4}\right)^7}{Q} \end{aligned}$$

The probability of successful message transmission is plotted against the source transmission probability in Figure 3.12.

3.9.3 Performance of the 8×4, Dilation 2 Switch

We can use our package of Mathematica procedures to write a procedure for the 8×4, dilation 2 switch, as follows:

```
eightXfourD2[q_] :=
    Block[{bundled, switched, instages, outstages},
```

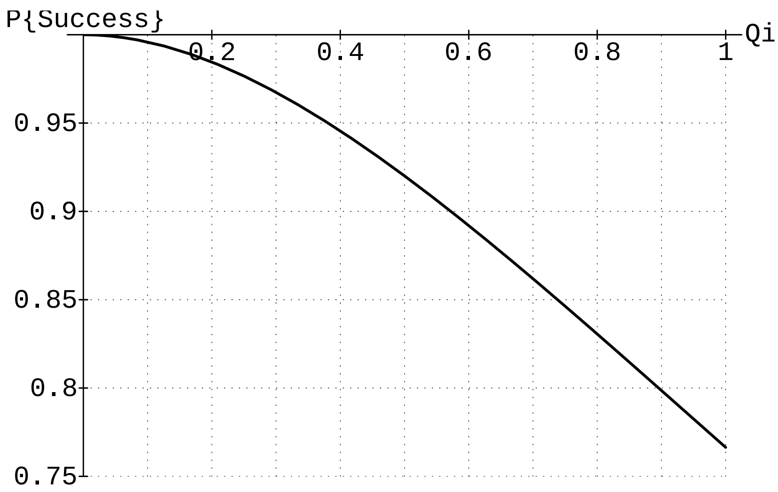

Figure 3.12: Probability of successful message transmission plotted against source transmission probability for an 8×4, dilation 2 switch, under a uniform destination address distribution.

```
instages = 3;
outstages = 2;
bundled = {(1-q), q};
Do[bundled = bundle[bundled, bundled], {instages}];
switched = bundled;
Do[switched = switch[switched, .5], {outstages}];
concentrate[switched, 2]]
```

We will need a four-by-eight, input dilation two crossbar for the second stage:

```
crossbar2x4in2[stageTwoPMF_] :=
    Block[{bundled, switched},
          bundled = stageTwoPMF;
          bundled = bundle[bundled, bundled];
          switched = bundled;
          Do[switched = switch[switched, .5], {2}];
          concentrate[switched, 1]]
```

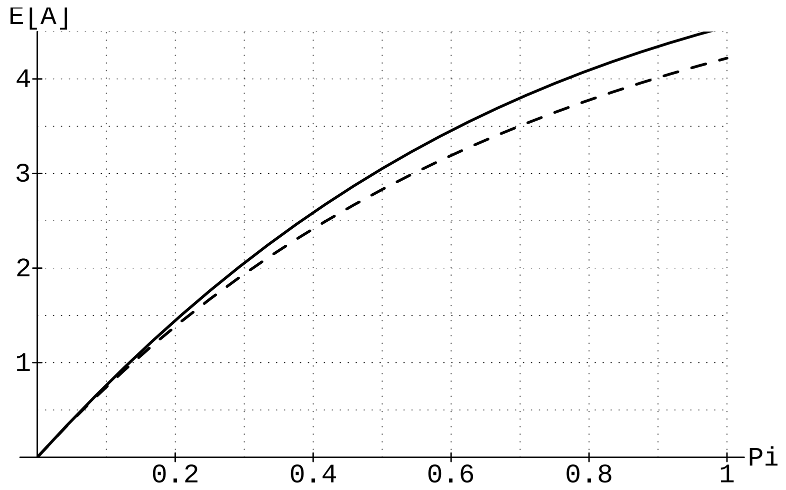

Figure 3.13: The bandwidth of the 16×16 indirect cube made from 4×4 crossbars, as depicted in Figure 3.9, is shown dashed. The bandwidth of the variant with a first stage made from 8×4, dilation 2 switches, as depicted in Figure 3.10, is shown in solid black. Both are plotted against the source transmission probability, for a uniform destination address distribution.

Now we can specify a procedure that yields as output the probability mass function of the load on a channel leading to a sink, given the probability that a source is transmitting:

```
eightXfourD2indirect16[q_] :=
  Block[{firstStageOut},
        (* input of first stage is just q *)
        (* returns LPMF for 2-wide channel *)
        firstStageOut = eightXfourD2[q];
        (* now feed to 4x4 crossbars and return result *)
        crossbar2x4in2[firstStageOut]]
```

We plot in Figure 3.13 the bandwidth for the 16×16 indirect cube made from 4×4 crossbars, and that for the variant with a first stage made from 8×4, dilation 2 switches. It will be seen that, as predicted, the performance of the network built with the dilated part is better.

Chapter 4

Analyzing the Performance of Multipath Networks

4.1 Introduction

In the previous chapter we have presented a method of analysis of Banyan network performance. But as we discussed in the introduction, Banyan networks, while amenable to analysis, are not intrinsically fault-tolerant.

We present in this chapter a method of analysis of multipath networks. The performance parameters, and the model, are much the same as for Banyan networks; but the requirement of unique paths and thus independence of channel loads is removed.

We leave behind the scheme of using elementary operations to build descriptions of switching elements, and instead directly derive the joint loading probability mass function for a set of channels leading from a switch.

We also present a program that solves these equations exactly. As was mentioned in the introduction, the program cannot be used for large networks, as its running time grows too quickly. We have found in the literature no polynomial-time program that computes the exact blocking probability of a multipath network. It may be that the problem is intractable, although we know of no proof of NP- or $\#P$-completeness for it.

Chapter 5 describes an approximation method for estimating solutions to the equations, by making use of the exact solution for subproblems.

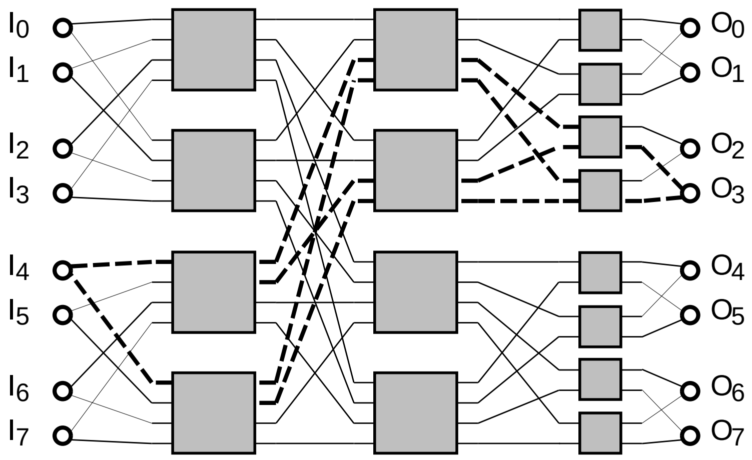

Figure 4.1: An 8×8 deterministically-interwired network with redundant paths. There are a number of different paths from any source to any sink, to increase fault-tolerance; redundant paths from message source 4 to sink 3 are highlighted. Routing is oblivious, with stochastic concentration. This wiring scheme is from [2].

4.2 Extensions to the Model

Figure 4.1 depicts a multipath network. We extend our model so that sources can have more than one channel to the network. A source still generates at most one message per cycle, which is directed to a stage 1 switch via one of the channels connecting the source to the network. The particular channel is selected randomly and with uniform probability.

As before, the processes generating messages at the sources are independent and memoryless. With some specified probability p_i, each source i generates or fails to generate a single message at the beginning of each cycle. The network is synchronous: at each cycle messages move from stage i to stage $i+1$. It is also unbuffered: if a message is blocked at some stage, it is considered to be lost, and does not in any way affect the future states of the system.

We use dilated switches, as described in Section 2.1, so that the set of output channels of a switching element is divided into nonempty disjoint subsets called *logical directions.* At each cycle, the switching element directs each incoming message in one logical direction. As for Banyan networks, we can choose the switching probabilities to model any single destination address distribution. When we route messages in a logical direction, we use *stochastic concentration*:

- If there are fewer messages or exactly the same number of messages directed in the logical direction as there are channels in that logical direction, then the channels that will carry the messages are chosen randomly, with uniform probability.
- If there are more messages directed in a logical direction than there are channels in that direction, the messages that can be carried are chosen with uniform probability, and the other messages are blocked and lost.

We note again that our network is self-routing: each message contains the information necessary to route the message from the source where it is injected to the sink that is its destination. No global information is used. In particular, this means that if we have several switches at a single stage, then given the loads on their input channels, the loads on the output channels of each switch are independent of the loads on the output channels of the other switches. This fact will be important in allowing us to factor joint loading probability mass functions.

Having extended our model, let us return to the network of Figure 4.1. The switches here are 4×2, dilation 2 switches, except at the last stage, where they are simply 2×2 (dilation 1) switches. In the 4×2, dilation 2 switches, the top two output channels constitute one logical direction, and the bottom two constitute another.

As with Banyan networks, we wish to find the bandwidth and the probability of successful message transmission of the networks we model. We find these parameters by finding the probability mass functions of the loads on channels leading to sinks.

4.3 The Joint Probability Mass Function of an Aggregate of Channels

Suppose that the input channels of a switch S, depicted in Figure 4.2, are connected to several switches $R_1, R_2, \ldots, R_i$. Let us use the random variable L to denote the entire output loading configuration of S at some specified

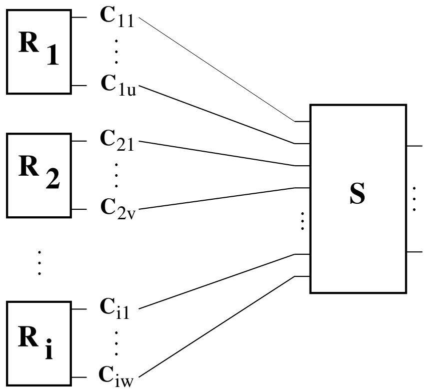

Figure 4.2: Interstage wiring. Note that no subset of the channels depicted need have mutually independent loads in a network with redundant paths. The output channels on the right of the switch marked S are those whose loads are referred to collectively in the text with the random variable L.

discrete time t, so that $\mathrm{P}\{L = l\}$ is the probability that the output channels of the switch have some particular loads designated in their aggregate by l during cycle t.

Now consider the loads on the input channels $C_{11}, \ldots, C_{iw}$ at cycle $t-1$. (Because we assume a synchronous, unbuffered network with memoryless processes generating the messages at the inputs, only the cycle before cycle t is of interest.) Let us denote the loads on the input channels at cycle $t-1$ with the random variables $L_{C_{11}}, \ldots, L_{C_{iw}}$.

In order to find the joint probability mass function of the loads on the output channels of S, we condition on the loads on the input channels:

$$\mathrm{P}\{L = l\} = \sum_{l_{C_{11}}, \ldots, l_{C_{iw}}} \mathrm{P}\{L = l \mid L_{C_{11}} = l_{C_{11}}, \ldots, L_{C_{iw}} = l_{C_{iw}}\} \cdot \mathrm{P}\{L_{C_{11}} = l_{C_{11}}, \ldots, L_{C_{iw}} = l_{C_{iw}}\} \tag{4.1}$$

where the sum is over all tuples $l_{C_{11}}, \ldots, l_{C_{iw}}$ with elements in $\{0, 1\}$.

Suppose that we can compute $\mathrm{P}\{L = l \mid L_{C_{11}} = l_{C_{11}}, \ldots, L_{C_{iw}} = l_{C_{iw}}\}$.[1] In order to compute the probability of an output loading configuration of S we will still need to find the joint probability mass function of the channel

[1] An expression for this conditional probability is derived in Section 4.4.

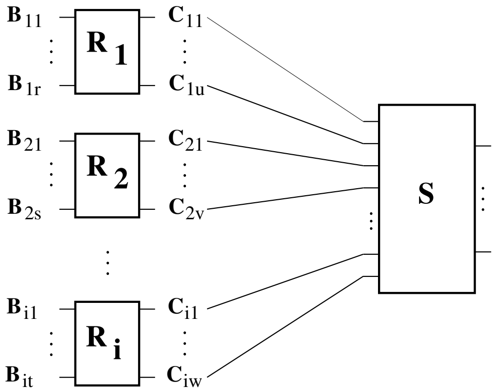

Figure 4.3: Channels referred to in Equation (4.2). Although the probabilities of the message loads on the channels $C_{11}, \ldots, C_{iw}$ are not in general independent, the loads on the subset of channels from each switching element are independent given the message loads on the input channels $B_{11}, \ldots, B_{it}$.

loads $L_{C_{11}}, \ldots, L_{C_{iw}}$. In a Banyan network, it would be easy to compute this function; it would simply be the product of the probability mass functions of the loads on the individual channels, as channel loads in a Banyan network are independent.[2] In a network with redundant paths, however, the loads on these channels are not in general independent, as they may derive from the same sources, and a message from a single source that has traveled one path in the network cannot be traveling along another path. Thus another method must be used.

In Figure 4.3, we see that the input channels $C_{11}, \ldots, C_{iw}$ of switch S are the output channels of switches $R_1, \ldots, R_i$. Let us call the loads on the input channels to these switches $L_{B_{11}}, \ldots, L_{B_{it}}$. We may now calculate $\mathrm{P}\{L_{C_{11}} = l_{C_{11}}, \ldots, L_{C_{iw}} = l_{C_{iw}}\}$ by conditioning on the values of the variables $L_{B_{11}}, \ldots, L_{B_{it}}$. We have

$$\begin{aligned}
&\mathrm{P}\{L_{C_{11}} = l_{C_{11}}, \ldots, L_{C_{iw}} = l_{C_{iw}}\} = \\
&\quad \sum_{l_{B_{11}}, \ldots, l_{B_{it}}} \mathrm{P}\{L_{C_{11}} = l_{C_{11}}, \ldots, L_{C_{iw}} = l_{C_{iw}} \mid L_{B_{11}} = l_{B_{11}}, \ldots, L_{B_{it}} = l_{B_{it}}\} \cdot \\
&\qquad \mathrm{P}\{L_{B_{11}} = l_{B_{11}}, \ldots, L_{B_{it}} = l_{B_{it}}\} \qquad (4.2)
\end{aligned}$$

[2] See Section 3.2.

where the sum is over all tuples $l_{B_{11}}, \ldots, l_{B_{it}}$ with elements in $\{0, 1\}$.

The loads on the output channels of these switches are not in general mutually independent. However, let us partition them into subsets according to the switch at which they originate, so that for the channels shown in Figure 4.3 we would have the subsets

$$\{C_{11}, \ldots C_{1u}\}, \{C_{21}, \ldots C_{2v}\}, \ldots, \{C_{i1}, \ldots, C_{iw}\}$$

Note that, under the assumption of independence of message destinations, and given the loads on the channels $B_{11}, \ldots, B_{it}$, the loads on the switch output channel subsets *are* mutually independent. As mentioned in Section 4.2, this is a consequence of the fact that the networks we model are self-routing. No global information is used in routing messages through the network.

That is, if we know the input loads for the switches $R_1, \ldots, R_i$, then the loading probabilities for the output channels of each of the switches do not depend on the output loads of any other switch. We may use this fact to derive the joint probability mass function of the loads on the output channels $C_{11}, \ldots, C_{iw}$ by conditioning on the input channel loads. We have then

$$\begin{aligned}
&\mathrm{P}\{L_{C_{11}} = l_{C_{11}}, \ldots, L_{C_{iw}} = l_{C_{iw}}\} = \\
&\quad \sum_{l_{B_{11}}, \ldots, l_{B_{it}}} \mathrm{P}\Big\{L_{C_{11}} = l_{C_{11}}, \ldots, L_{C_{1u}=l_{C_{1u}}} \mid L_{B_{11}} = l_{B_{11}}, \ldots, L_{B_{1r}} = l_{B_{1r}}\Big\} \cdot \\
&\qquad\qquad \mathrm{P}\Big\{L_{C_{21}} = l_{C_{21}}, \ldots, L_{C_{2v}=l_{C_{2v}}} \mid L_{B_{21}} = l_{B_{21}}, \ldots, L_{B_{2s}} = l_{B_{2s}}\Big\} \cdot \\
&\qquad\qquad \ldots \quad \cdot \\
&\qquad\qquad \mathrm{P}\{L_{C_{i1}} = l_{C_{i1}}, \ldots, L_{C_{iw}} = l_{C_{iw}} \mid L_{B_{i1}} = l_{B_{i1}}, \ldots, L_{B_{it}} = l_{B_{it}}\} \cdot \\
&\qquad\qquad \mathrm{P}\{L_{B_{11}} = l_{B_{11}}, \ldots, L_{B_{it}} = l_{B_{it}}\}
\end{aligned} \tag{4.3}$$

where the sum is once again over all tuples $l_{B_{11}}, \ldots, l_{B_{it}}$ with elements in $\{0, 1\}$.

The subexpression $\mathrm{P}\{L_{B_{11}} = l_{B_{11}}, \ldots, L_{B_{it}} = l_{B_{it}}\}$ can be evaluated recursively by means of Equation (4.3), until the channels $B_{11}, \ldots, B_{it}$ correspond to sources. If these channels originate at message sources, then we substitute instead the probability mass functions corresponding to sources. We may simply take the product of these functions for the sources in question, as in our model the processes generating messages at the sources are mutually independent.

If source i, depicted in Figure 4.4, generates a message with probability p_i and has k channels into the network, then we have for the loads on the

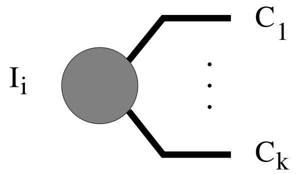

Figure 4.4: The source I_i generates a single message at each cycle with probability p_i. The message is transmitted with uniform probability over a randomly picked channel in the set $\{C_1, \ldots, C_k\}$.

channels $C_1, \ldots, C_k$ the joint probability mass function

$$\mathrm{P}\{L_{C_1} = l_{C_1}, \ldots, L_{C_k} = l_{C_k}\} = \begin{cases} 1 - p_i & \text{if all the } l_{C_j} \text{ are } 0 \\ \frac{p_i}{k} & \text{if exactly one } l_{C_j} \text{ is } 1, \\ & \text{and the rest are } 0 \\ 0 & \text{otherwise} \end{cases} \tag{4.4}$$

It remains now to evaluate the conditional probabilities in Equation (4.3). Recall that these are the joint conditional probability that some subset of the output channels of a dilated switch have a particular load, given that the input channels have a particular load. We derive an equation for these conditional probabilities in the next section.

4.4 Joint Probability Mass Functions of Dilated Switch Output Channels

Suppose we have an $M \times N$, dilation K switch. We may form the conditional probability mass function of the loads on the output channels, given the input load, by conditioning. Say that the random variable $L_{f,g}$ represents the load on the g^{th} channel in the f^{th} logical direction.

We wish to evaluate the expression

$$\mathrm{P}\{L_{1,1} = l_{1,1}, \ldots, L_{N,k} = l_{N,k} \mid L_{C_1} = l_{C_1}, \ldots, L_{C_M} = l_{C_M}\}$$

For an event E, define

$$\mathrm{Q}\{E\} = \mathrm{P}\{E \mid L_{C_1} = l_{C_1}, \ldots, L_{C_M} = l_{C_M}\} \tag{4.5}$$

Of course $\mathrm{Q}\{E\}$, like $\mathrm{P}\{E \mid L_{C_1} = l_{C_1}, \ldots, L_{C_M} = l_{C_M}\}$, is a probability in the usual sense; the definition is used to make completely clear the significance of the further conditioning we perform below. We will condition on the number of messages directed in each logical direction. If the random variable C_i represents the number of messages routed in logical direction i, we have:

$$\begin{aligned}
&\mathrm{Q}\{L_{1,1} = l_{1,1}, \ldots, L_{N,k} = l_{N,k}\} = \\
&\qquad \sum_{d_1,\ldots,d_N} \mathrm{Q}\{L_{1,1} = l_{1,1}, \ldots, L_{N,k} = l_{N,k} \mid D_1 = d_1, \ldots, D_N = d_n\} \cdot \\
&\qquad\qquad \mathrm{Q}\{D_1 = d_1, \ldots, D_N = d_n\} \qquad (4.6)
\end{aligned}$$

where the sum is over all N-tuples $d_1, \ldots, d_N$ such that each $d_i \geq 0$ and $\sum_{i=1}^{N} d_i = j$.

Now we consider the switching probability. We calculate the probabilities for the N logical directions using Equation (3.7) of Section 3.6 (of course, under uniform addressing each of these probabilities would be $1/N$). Suppose that these probabilities are $q_1, q_2, \ldots, q_N$. By our assumption of independence of message addresses, the probability that of the $\sum_{i=1}^{M} l_{C_i}$ arriving messages, d_1 are directed in direction 1, d_2 in direction 2, and so on, is simply multinomial, so that

$$\mathrm{Q}\{D_1 = d_1, \ldots, D_N = d_N\} = \binom{\sum_{i=1}^{M} l_{C_i}}{d_1, \ldots, d_N} q_1^{d_1} q_2^{d_2} \cdots q_N^{d_N} \qquad (4.7)$$

Now let us evaluate $\mathrm{Q}\{L_{1,1} = l_{1,1}, \ldots, L_{N,k} = l_{N,k} \mid D_1 = d_1, \ldots, D_N = d_n\}$. Say that b_i is the number of messages output in direction i; that is, $b_i = \sum_{g=1}^{K} l_{i,g}$. This number is not the same as d_i, because if there are more than K messages to be output in a K-wide direction, some messages are dropped and lost. If b_i messages are output, then under stochastic concentration the channels are picked with uniform probability, and so the probability of any single configuration will be $1/\binom{K}{b_i}$. Thus

$$\begin{aligned}
&\mathrm{Q}\{L_{1,1} = l_{1,1}, \ldots, L_{N,k} = l_{N,k} \mid D_1 = d_1, \ldots, D_N = d_N\} = \\
&\qquad \begin{cases} 0 & \text{if for any } i, \quad b_i \neq \min(d_i, K) \\ \displaystyle\prod_{i=1}^{N} \frac{1}{\binom{K}{b_i}} & \text{otherwise} \end{cases} \qquad (4.8)
\end{aligned}$$

where $b_i = \sum_{g=1}^{K} l_{i,g}$.

Combining Equations (4.5), (4.6), (4.7), and (4.8), we have

$$\mathrm{P}\{L_{1,1} = l_{1,1}, \ldots, L_{N,k} = l_{N,k} \mid L_{C_1} = l_{C_1}, \ldots, L_{C_M} = l_{C_M}\} =$$
$$\left(\prod_{i}^{N} \frac{1}{\binom{K}{b_i}}\right) \sum_{d_1,\ldots,d_N} \binom{\sum_{i=1}^{M} l_{C_i}}{d_1, \ldots, d_N} q_1^{d_1} q_2^{d_2} \cdots q_N^{d_N} \tag{4.9}$$

where $b_i = \sum_{g=1}^{K} l_{i,g}$, and the sum is over the N-tuples $d_1, \ldots, d_N$ such that for each d_i, $\min(d_i, K) = b_i$, and $\sum_{i=1}^{N} d_i = \sum_{i=1}^{M} l_{C_i}$.

Of course, if the conditional joint probability of the load on a subset of the switch's output channels is desired, as opposed to all of the switch's channels, we can simply sum this expression over all the possible loads on the complement of the subset of channels whose loads are required, as the different configurations of the output channels are mutually exclusive events.

4.5 Automatic Calculation of Blocking Probabilities

It will be clear that the automatic calculation of blocking probabilities by this means will require a great deal of time. Suppose we have a computer program that calculates the blocking probabilities for a network in the most obvious way, by finding the joint probability mass function of the channels leaving the final stage, using Equation (4.3) recursively. In the worst case, we can imagine a network where there are N stages and M dependent channels between each of the N stages, and the joint probability mass function of all of the channels between each of the stages must be formed. The domain of the joint probability mass function for each stage then is of size 2^M, each value being calculated as a sum over 2^M terms. Assuming the time to calculate each of the terms summed over in Equation (4.3) is $\mathrm{O}(M)$, we have then $\mathrm{O}\left(NM2^{2M}\right)$ for the worst-case performance.

The performance on some networks can be better than this, however. Suppose that we need to calculate $\mathrm{P}\{L_{C_1} = l_{C_1}, \ldots, L_{C_n} = l_{C_n}\}$. Let $\mathcal{S}(c)$ denote the set of source nodes from which messages can reach channel c. If we can partition the set of channels $\{C_1, \ldots, C_n\}$ into disjoint subsets $S_1, \ldots, S_m$ such that for any $C_1 \in S_i$ and $C_2 \in S_j$, $i \neq j$, $\mathcal{S}(C_1) \cap \mathcal{S}(C_2)$ is empty, then the loads on the channels in each subset S_i are independent of the loads on the channels in any and all of the other subsets in the partition.[3]

[3] As can be seen from the argument in Section 3.2.

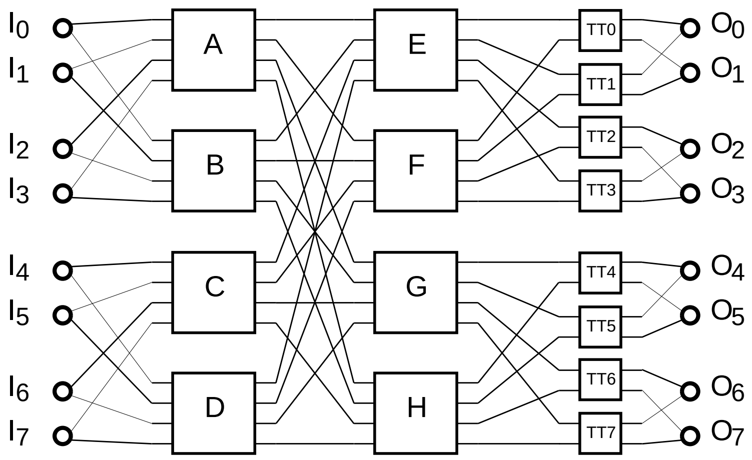

Figure 4.5: The network of Figure 4.1, with switches labeled.

Then the expression $\mathrm{P}\{L_{C_1} = l_{C_1}, \ldots, L_{C_n} = l_{C_n}\}$ can be factored into the product of m joint probability mass functions, one for each subset S_i. In the limiting case of a Banyan network, a complete factoring will be possible for every set of channels, and the summation itself can be factored, so that the worst case performance for a Banyan network of N stages with M channels between the stages becomes $\mathrm{O}(NM)$.

A Common LISP program has been written to evaluate the joint probability mass function of the loads on specified channels in a multistage interconnection network. The program is given a symbolic description of the interconnection network; this requires labeling the switching nodes of the network. We show a labeling of the network of Figure 4.1 nodes in Figure 4.5. The input description for this network is shown in Figure 4.5.

The program uses the network representation to build an internal structure in which (for example) information about independence of channel loads has been pre-computed, and channels have been assigned names generated from the names of the their nodes of origin and destination. One can then query the program for the probability mass function of interest. The result is numerical, as in the example below:

```
> (setq d8x8 (parse-multistage-network
              deterministically-interwired-8x8-rep))
```

```
(defparameter deterministically-interwired-8x8-rep
  ;; inputs first -- these don't get a conditional probability
  ;; function.
  '((i0 (a b) nil 1/2) (i1 (a b) nil 1/2)
    (i2 (a b) nil 1/2) (i3 (a b) nil 1/2)
    (i4 (c d) nil 1/2) (i5 (c d) nil 1/2)
    (i6 (c d) nil 1/2) (i7 (c d) nil 1/2)
    ;; stage 1 4x4's
    (a (e f g h) 4x2d2-cp-fun) (b (e f g h) 4x2d2-cp-fun)
    (c (e f g h) 4x2d2-cp-fun) (d (e f g h) 4x2d2-cp-fun)
    ;; stage 2 4x4's
    (e (tt0 tt1 tt2 tt3) 4x2d2-cp-fun)
    (f (tt0 tt1 tt2 tt3) 4x2d2-cp-fun)
    (g (tt4 tt5 tt6 tt7) 4x2d2-cp-fun)
    (h (tt4 tt5 tt6 tt7) 4x2d2-cp-fun)
    ;; stage 3 2x2's
    (tt0 (o0 o1) 2x2d1-cp-fun) (tt1 (o0 o1) 2x2d1-cp-fun)
    (tt2 (o2 o3) 2x2d1-cp-fun) (tt3 (o2 o3) 2x2d1-cp-fun)
    (tt4 (o4 o5) 2x2d1-cp-fun) (tt5 (o4 o5) 2x2d1-cp-fun)
    (tt6 (o6 o7) 2x2d1-cp-fun) (tt7 (o6 o7) 2x2d1-cp-fun)
    ;; outputs
    (o0) (o1) (o2) (o3) (o4) (o5) (o6) (o7)))
```

Figure 4.6: Symbolic description of the network of Figure 4.5. The description specifies that during each cycle each source node generates a message with probability 1/2.

```
#<MULTISTAGE-NETWORK 8x8>
> (jlpmf '(tt6-o7-0 tt7-o7-0) d8x8)
(#S(JLPMF-PART CHANNELS (#<CHANNEL TT6-07-0>
                          #<CHANNEL TT7-07-0>)
               NUMBER-OF-CHANNELS 2
               VECTOR #(10321939817/17179869184
                        2931771091/17179869184
                        2931771091/17179869184
                        994387185/17179869184)))
```

Here we have calculated the joint probability mass function of the loads on two channels leading from two 2×2 switches to sink $O7$ in the network of Figure 4.1, given a probability of transmission in each message source of 1/2, and under a uniform destination address distribution. The vector component of the structure result above is indexed by integers in which the bit with weight 2^i specifies the load of the ith channel (starting with $i = 0$) in the vector of channels whose joint loading probability mass function was required. Thus in the example above, the probability that no messages are transmitted to sink $O7$ is $10321939817/17179869184 \approx 0.601$; the probability that 1 message is transmitted along the channel from switch $TT7$ to $O7$ is $2931771091/17179869184 \approx 0.171$, as is the probability that 1 message is transmitted along the channel to $O7$ from switch $TT6$. Finally, the probability that both channels carry a message is $994387185/17179869184 \approx 0.058$; we assume here, as in [2], that a message sink can receive two messages during a single cycle.[4]

To find the blocking probability of the network, we use Equation (2.1); we form the probability of successful message transmission as the ratio of the expected number of messages entering the network to the expected number of messages arriving at sinks. Because of the symmetry of the network, all the channels leading to sinks have identical loading probabilities, and so we can simply sum the expectations of their loads. We have then that the expected number of messages arriving at a single sink is

$$1 \cdot \frac{2931771091}{17179869184} + 1 \cdot \frac{2931771091}{17179869184} + 2 \cdot \frac{994387185}{17179869184} = \frac{981539569}{2147483648} \approx 0.457$$

[4]If a sink can receive only one message during a cycle, then the expected number of messages received by a sink during a cycle will be

$$1 - \frac{10321939817}{17179869184} = \frac{6857929367}{17179869184} \approx 0.40$$

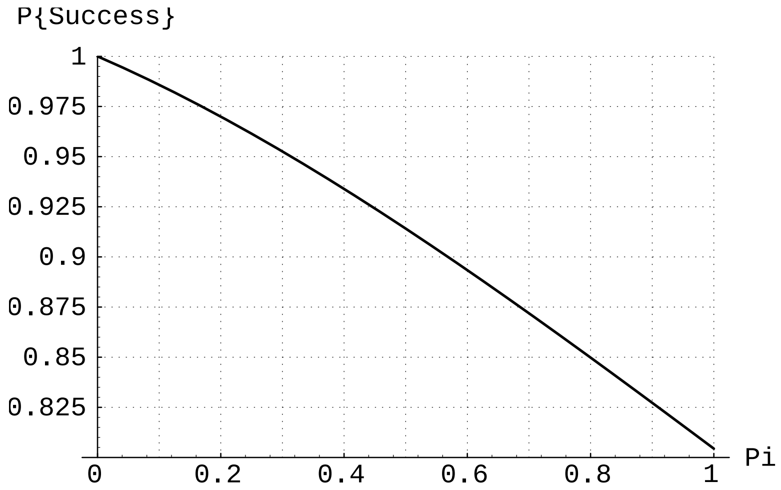

Figure 4.7: The probability of successful message transmission (`P{Success}`) plotted against the the source transmission probability (`Pi`) for the network of Figure 4.1, under a uniform destination address distribution.

and the expected number of messages arriving at all sinks during any cycle is

$$8 \cdot \frac{981539569}{2147483648} = \frac{981539569}{268435456} \approx 3.66$$

Because the expected number of messages entering the network is $8 \cdot \frac{1}{2} = 4$, we have that the aggregate probability of successful message transmission in this network at a loading factor of $1/2$ is

$$\frac{\text{E[messages arriving at sinks]}}{\text{E[messages injected by sources]}} = \frac{981539569}{1073741824} \approx 0.914$$

and thus the blocking probability is approximately 0.086.

We plot for the network of Figure 4.1 the probability of successful message transmission versus the probability that a source transmits in Figure 4.7.

The Common LISP implementation internally records joint probability mass functions so that they need not be recomputed. The implementation has been coded with some attention to performance, because, although the asymptotic performance is pessimal, the same code is used on subnetworks of larger networks in an approximation scheme.

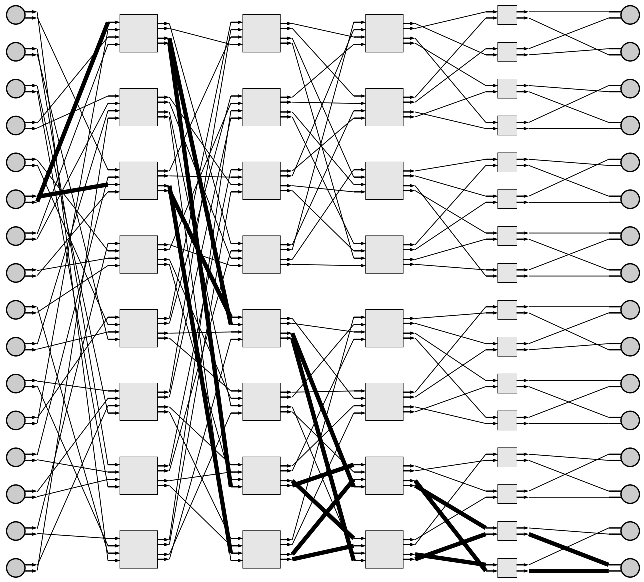

Figure 4.8: A 16×16 network with random interwiring in the first and second stages. The figure is from [2].

4.6 Applicability of Exact Calculation of Blocking Probabilities

We have presented a means of exact calculation of the blocking probability of a multistage network with redundant paths, and demonstrated its use in a program that automatically calculates blocking probabilities and exploits independence of channel loading probabilities where this is possible.

The implementation described cannot be used to calculate the blocking probabilities of networks with much more path redundancy than the one of Figure 4.1. We might consider an implementation that could exploit the symmetry exhibited by some multistage networks, but such an implementation could still not be used on a network like that in Figure 4.8, in which the wiring in the first and second stages is not symmetric and is in fact randomly generated. That such networks are of interest is demonstrated in [16].

Thus we must seek approximate solutions to the problem. This we do in the next chapter, where we will see that the exact equations and our algorithm for solving them can be used to realize a faster approximation method.

Chapter 5

Approximating Performance Parameters for Multipath Networks

5.1 Introduction

We saw in the previous chapter that exact calculation of the probability mass functions of channels leading to sinks in a multipath network could be very expensive. In this chapter we seek a method of approximation of performance parameters that will allow us to estimate to within a given error the loading probability of a channel leading to a sink. We will do this by using Monte Carlo methods, attempting both direct simulation of the network and also approximation of Equation (4.3), and compare the expense and error of the two methods.

Our approximations use exactly the model we described in Sections 2.1 and 4.2. We will find that this exact correspondence is important as we develop a method of approximating solutions to the equations by a combination of simulation and exact methods.

5.2 Direct Simulation

In direct simulation, we simulate the transition through the network of a group of messages generated in a single cycle as follows:

1. Messages are generated for the cycle being simulated by sources in accordance with the source transmission probabilities p_i.
2. Addresses are picked according to the destination address distribution.
3. The messages arrive at switching elements and are directed in logical directions in accordance with their addresses.
   - The direction of more messages in a logical direction than there are channels in the direction is resolved by randomly choosing messages are blocked.
   - Output channels within a logical direction are selected randomly, with each channel having the same probability of being selected to carry a message.
4. Step 3 is repeated until we have calculated the loads of the channels whose states we are examining in the simulation.

Note that, using the results of Section 3.6, we can generate the same distribution of messages as we do in step 2 by modifying step 3 to randomly pick, for each message, a logical direction in accordance with the switching probabilities of the switch. Our simulation algorithm then becomes:

1. Messages are generated for the cycle being simulated by sources in accordance with the source transmission probabilities p_i.
2. The messages arrive at switching elements and are directed in logical directions in accordance with the switching probabilities of the switch.
   - The direction of more messages in a logical direction than there are channels in the direction is resolved by truncating the number of messages to the dilation of the logical direction.
   - Output channels within a logical direction are selected randomly, with each channel having the same probability of being selected to carry a message.
3. Step 2 is repeated until we have calculated the loads of the channels whose states we are examining in the simulation.

Simulation procedures for the random selections described above are straightforward. We describe them briefly here; details of these techniques can be found in introductory texts on probability models (*e.g.,* [25]).

Message generation is performed by simulating the generation of a Bernoulli random variable with the source transmission probability. Selection of logical directions for a message can be performed by subdividing the half-open interval $[0, 1)$ into as many segments as there are logical directions, the length of the segment for a logical direction being the same as the probability of selecting that direction. A uniform random variable U is generated and the segment into which U falls is taken as corresponding to the selected logical direction. Finally, the random selection of output channels within a logical direction can be performed in many ways; we do so by considering the k channels to correspond to bits in a k-bit vector. If there are n messages to be directed in the logical direction, we set only the low n bits in the vector and then randomly permute the vector, which can be done in $O(k)$ steps.[1] The bits that are set after the permutation correspond to the channels that carry messages.

5.3 Approximation of Performance Parameters Using Direct Simulation

Repeated simulations can be used to approximate the parameter of interest by the Monte Carlo method. Suppose that what we are interested in is the probability that some set of channels C has a particular loading configuration l. We run some number N of simulations, examining after each simulation the loads on the channels C. If the channels have the loading configuration l, the experiment is considered a “hit” and has value 1. If the channels do not have the loading configuration l, the experiment is a “miss” with value 0. The mean of the values of the experiment is taken as an approximation of the expected load.

Now we describe direct simulation using standard notation, as the textual description above would prove too unwieldy later in the chapter.[2]

Let $r_1, r_2, \ldots, r_k$ denote all the random variates that might be required to perform a single direct simulation by the algorithm described above.[3] Then let $\mathbf{R} = (r_1, r_2, \ldots, r_k)$ be a vector of these random variates. Now let $\mathbf{R}_1, \mathbf{R}_2, \ldots, \mathbf{R}_n$ be a sequence of such vectors, identically and independently distributed.

[1] Using an algorithm on *pp.* 474–476 of [25].

[2] We use the notation of [10].

[3] That the number of random variates that might be required is finite will be clear when we consider that only a finite number of outcomes from each experiment is possible.

If $f(\mathbf{R})$ is a function whose value is 1 where the channels C whose states we are examining in simulation have the load l, and 0 where they do not, then the variables

$$f_i = f(\mathbf{R_i})$$

are identically and independently distributed. If $\mathrm{E}[f(\mathbf{R})] = \mu$, then

$$\tilde{f} = \sum_{i=1}^{n} f_i$$

is an unbiased estimator of $\mathrm{P}\{L_C = l\} = \mu$.

In order to calculate error bounds on our approximations, we will need to know the variance of $\tilde{f}$. Because the f_i are Bernoulli,

$$\mathrm{Var}\left(\tilde{f}\right) = \frac{1}{n}\mu(1-\mu)$$

because $\mu(1-\mu)$ is the variance of $f(\mathbf{R})$. Unfortunately, this expression will not be very useful in practice, as we do not *a priori* know μ, or there would be no need to estimate it. Thus we estimate the variance of $f(\mathbf{R})$, using the unbiased estimator

$$s^2 = \frac{1}{n-1}\sum_{i=1}^{n}\left(f_i - \tilde{f}\right)^2 \quad \approx \mathrm{Var}(f(\mathbf{R}))$$

There are means of estimating the variance of s^2, but we will not use these, as in practice the variance is small, and our error bounds are conservative.

Given the estimate s^2 for $\mathrm{Var}(f(\mathbf{R}))$, we may estimate the variance of $\tilde{f}$ as

$$\mathrm{Var}\left(\tilde{f}\right) \approx \frac{s^2}{n}$$

yielding a standard error of $s/\sqrt{n}$, which shows clearly that the error will vary as the inverse root of the number of trials.

5.4 Bounding the Number of Iterations

To bound the number of iterations for which our simulation must run in order to achieve a specified level of precision at a specified confidence, we can use the Chebyshev Inequality, which states that if X is a random variable with mean μ and variance ν^2, then

$$\mathrm{P}\{|X-\mu| \geq k\} \leq \frac{\nu^2}{k^2}$$

Call the number of iterations performed n. Suppose that we wish to bound by c the probability that our estimate $\tilde{f}$ deviates from the value μ being estimated by more than some fraction d of μ. Because the variance of $\tilde{f}$ is $\frac{\sigma^2}{n}$ we have

$$\begin{aligned} \mathrm{P}\left\{\left|\tilde{f}-\mu\right| \geq d\mu\right\} &\leq \frac{\sigma^2}{nd^2\mu^2} \\ \mathrm{P}\left\{\frac{\left|\tilde{f}-\mu\right|}{\mu} \geq d\right\} &\leq \frac{\sigma^2}{nd^2\mu^2} \qquad (5.1) \end{aligned}$$

Now we can estimate the number of iterations we require by considering c, the complement of our desired confidence level:

$$\begin{aligned} c &= \frac{\sigma^2}{nd^2\mu^2} \\ n &= \frac{\sigma^2}{cd^2\mu^2} \qquad (5.2) \end{aligned}$$

In practice we use the estimate s^2 for σ^2 and the estimate $\tilde{f}$ for μ in calculating a projected number of iterations. We repeat the calculation after each iteration of the algorithm and check to see whether we have performed enough iterations to bound the error as desired.

The Chebyshev Inequality provides a conservative bound on the number of iterations required. For large numbers n of simulations, we expect from the Central Limit Theorem that the distribution of $\tilde{f}$ is approximately normal. Thus, for example, we may have 95% confidence that $\tilde{f}$ is in the interval $\left[\mu-\frac{2s}{\sqrt{n}}, \mu+\frac{2s}{\sqrt{n}}\right]$. We can use the Central Limit Theorem in the same way that we did the Chebyshev Inequality, to calculate a projected number of iterations required to bound the error as desired.

By the Central Limit Theorem we have that

$$\mathrm{P}\left\{\frac{f_1+f_2+\ldots+f_n-n\mu}{\sigma\sqrt{n}} \leq a\right\} \rightarrow \Phi(a) \quad \text{as} \quad n \rightarrow \infty$$

so that

$$\begin{aligned} \mathrm{P}\left\{\tilde{f} - \mu \leq \frac{a\sigma}{\sqrt{n}}\right\} &\rightarrow \Phi(a) \\ \mathrm{P}\left\{\frac{\left|\tilde{f} - \mu\right|}{\mu} \leq \frac{a\sigma}{\mu\sqrt{n}}\right\} &\rightarrow \Phi(a) - \Phi(-a) \\ &= 2\Phi(a) - 1 \quad \text{as} \quad n \rightarrow \infty \end{aligned} \tag{5.3}$$

Substituting d for $\frac{a\sigma}{\mu\sqrt{n}}$ and taking the complement, we have then

$$\mathrm{P}\left\{\frac{\left|\tilde{f} - \mu\right|}{\mu} > d\right\} \rightarrow 2\left(1 - \Phi\left(\frac{\mu d\sqrt{n}}{\sigma}\right)\right) \quad \text{as} \quad n \rightarrow \infty \tag{5.4}$$

where as before we use the estimate s for σ and the estimate $\tilde{f}$ for μ in practice. If we wish to bound by c the probability that $\tilde{f}$ varies from the desired result by more than d, we may use our formula by calculating after each iteration of the simulation the quantity $2\left(1 - \Phi\left(\frac{\mu d\sqrt{n}}{\sigma}\right)\right)$ and halting when it is less than c.[4]

5.5 An Example of Direct Simulation

A program has been written to estimate the probability that a set of channels in a network will have a particular load, using the simulation algorithm of Section 5.2. Although simulation will let us estimate blocking probabilities for larger networks, and we will use a larger network later in the chapter, here we use the network of Figure 4.1, reproduced here in Figure 5.1. We do so because we know an exact result for this network (as shown in Section 4.5), and thus we can verify that in this example the simulation algorithm achieves the error bounds it should.

We will estimate the probability that both of the channels leading to sink 7 in this network carry no messages. We had determined in Section 4.5 that this probability (under uniform addressing, with each source having a probability of 0.5 of generating a message at each cycle) was

$$\mathrm{P}\{L_{TT6-O7-0} = 0, L_{TT7-O7-0} = 0\} = \frac{10321939817}{17179869184} \approx 0.6008$$

[4] Of course, we could also use the inverse function Φ^{-1} to allow us to project a number of iterations; but we have not done this here.

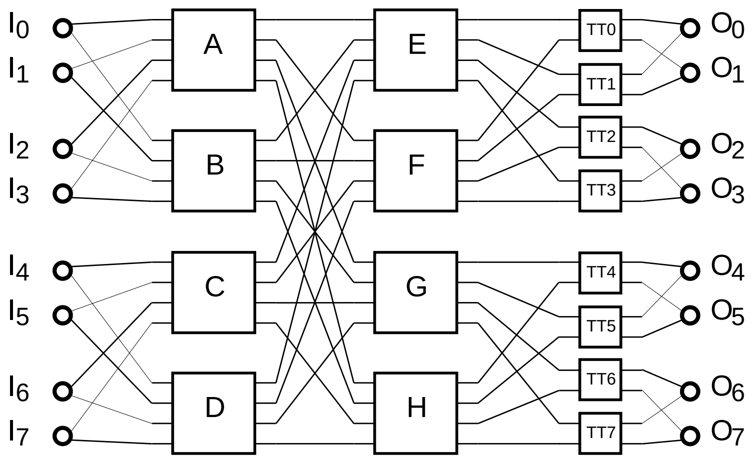

Figure 5.1: The 8×8 deterministically-interwired network of Figure 4.1. In the example we estimate the probability that the channels leading to sink 7 carry no messages, under uniform addressing.

We see in Figure 5.2 the result of running the program to estimate the required probability, using Equation (5.4) to calculate the number of iterations necessary to achieve an estimate that lies within 1% of the actual value with 95% confidence.

We see that approximately $25,000$ iterations are required to estimate the value

$$\frac{15222}{25211} \approx 0.604$$

which is indeed within 1% of the exact solution. Using the more conservative bound of Equation (5.2), the simulation runs for about $133,000$ iterations, yielding a result of $79825/132847 \approx 0.6009$.

5.5.1 The Expense of Direct Simulation

For a network with N stages with M channels between each stage, an iteration of the direct simulation algorithm of Section 5.2 runs in time $\mathrm{O}(NM)$. We can use Equation (5.2) to bound the total cost of estimating μ with a

```
> (setq o7-channels (elements-named '(tt6-o7-0 tt7-o7-0) d8x8))
(#<CHANNEL TT6-O7-0> #<CHANNEL TT7-O7-0>)
> (simulate-multi-channel-loading-probability d8x8 o7-channels '(0 0)
       (make-clt-stopping-function .01 .05 5000))
Iteration 15; mean: .667; variance: .238; current confidence .042
Iteration 5000; mean: .607; variance: .239; current confidence 0.62
Iteration 10000; mean: .603; variance: .239; current confidence .782
Iteration 15000; mean: .605; variance: .239; current confidence .871
Iteration 20000; mean: .603; variance: .239; current confidence .919
Iteration 25000; mean: .604; variance: .239; current confidence .949
15222/25211
76026279/317784655
25211
```

Figure 5.2: Estimating by direct simulation the probability that both channels leading to sink 7 in the network of Figure 5.1 carry no messages, under uniform addressing and with a source transmission probability of 1/2.

deviation factor of d and at a confidence of $1 - c$ as

$$\mathrm{O}\left(NM\frac{\sigma^2}{cd^2\mu^2}\right) = \mathrm{O}\left(NM\frac{(1-\mu)}{cd^2\mu}\right)$$

because in the Bernoulli trials that make up the iterations of a direct simulation we have $\sigma^2 = \mu(1-\mu)$.

5.6 Approximating a Solution to the Exact Equations

5.6.1 Approximating Equation (4.2) Across a Single Stage

We saw in Chapter 4 that our method of exact calculation of blocking probabilities suffered from exponential increase in the expense of calculation as the number of dependent paths between stages increased. Equation (4.2) specified the probability that a set of output channels of switches, depicted in Figure 5.3, carried a particular load:

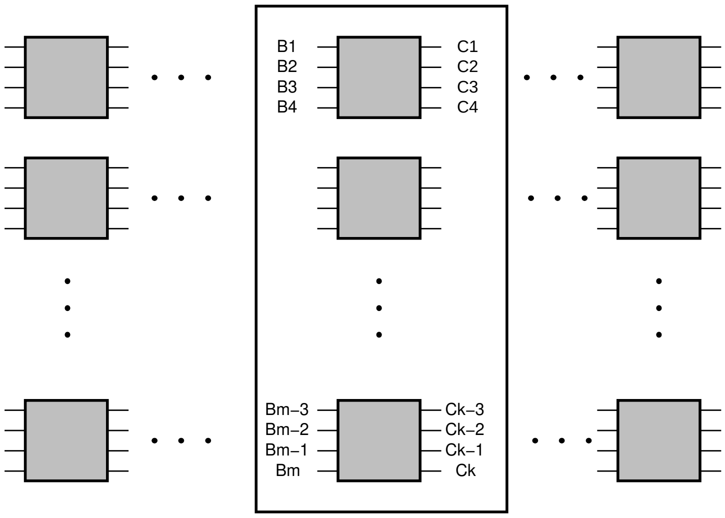

Figure 5.3: The network stage referred to by Equation (5.5).

$$\begin{aligned}
&\mathrm{P}\{L_{C_1} = l_{C_1}, \ldots, L_{C_k} = l_{C_k}\} = \\
&\qquad \sum_{l_{B_1},\ldots,l_{B_m}} \mathrm{P}\{L_{C_1} = l_{C_1}, \ldots, L_{C_k} = l_{C_k} \mid L_{B_1} = l_{B_1}, \ldots, L_{B_m} = l_{B_m}\} \cdot \\
&\qquad\qquad \mathrm{P}\{L_{B_1} = l_{B_1}, \ldots, L_{B_m} = l_{B_m}\} \qquad (5.5)
\end{aligned}$$

where the sum is over all tuples $l_{B_1}, \ldots, l_{B_m}$ with elements in $\{0, 1\}$.

A method of approximate solution of this equation that suggests itself immediately is one of the following form:

> Rather than calculating this sum over all tuples $l_{B_1}, \ldots, l_{B_m}$, calculate it exactly for only some of the tuples.

To be more precise, suppose that we define

$$\begin{aligned}
&g\,(l_{B_1}, \ldots, l_{B_m}) = \\
&\qquad \mathrm{P}\{L_{C_1} = l_{C_1}, \ldots, L_{C_k} = l_{C_k} \mid L_{B_1} = l_{B_1}, \ldots, L_{B_m} = l_{B_m}\}
\end{aligned}$$

and we generate tuples $l_{B_1}, \ldots, l_{B_m}$ randomly in accordance with the probability mass function $\mathrm{P}\{L_{B_1} = l_{B_1}, \ldots, L_{B_m} = l_{B_m}\}$.

Now $g\,(l_{B_1}, \ldots, l_{B_m})$ is a random variable, and its expectation is

$$\begin{aligned}
&\mathrm{E}[g\,(l_{B_1}, \ldots, l_{B_m})] \\
&\quad = \sum_{l_{B_1},\ldots,l_{B_m}} g\,(l_{B_1}, \ldots, l_{B_m})\,\mathrm{P}\{L_{B_1} = l_{B_1}, \ldots, L_{B_m} = l_{B_m}\} \\
&\quad = \sum_{l_{B_1},\ldots,l_{B_m}} \mathrm{P}\{L_{C_1} = l_{C_1}, \ldots, L_{C_k} = l_{C_k} \mid L_{B_1} = l_{B_1}, \ldots, L_{B_m} = l_{B_m}\} \cdot \\
&\qquad\qquad \mathrm{P}\{L_{B_1} = l_{B_1}, \ldots, L_{B_m} = l_{B_m}\} \\
&\quad = \mathrm{P}\{L_{C_1} = l_{C_1}, \ldots, L_{C_k} = l_{C_k}\}
\end{aligned}$$

Thus we see that $g\,(l_{B_1}, \ldots, l_{B_m})$ is an unbiased estimator of the probability we wished to estimate: $\mathrm{P}\{L_{C_1} = l_{C_1}, \ldots, L_{C_k} = l_{C_k}\}$.

We can readily calculate

$$\mathrm{P}\{L_{C_1} = l_{C_1}, \ldots, L_{C_k} = l_{C_k} \mid L_{B_1} = l_{B_1}, \ldots, L_{B_m} = l_{B_m}\}$$

by factoring the expression as in Equation (4.3). As we observed in Chapter 4, although the loads on the channels $C_1, \ldots, C_k$ are not in general independent, the loads on the subset of channels originating at each switching element are independent given the message loads on the input channels $B_1, \ldots, B_m$.

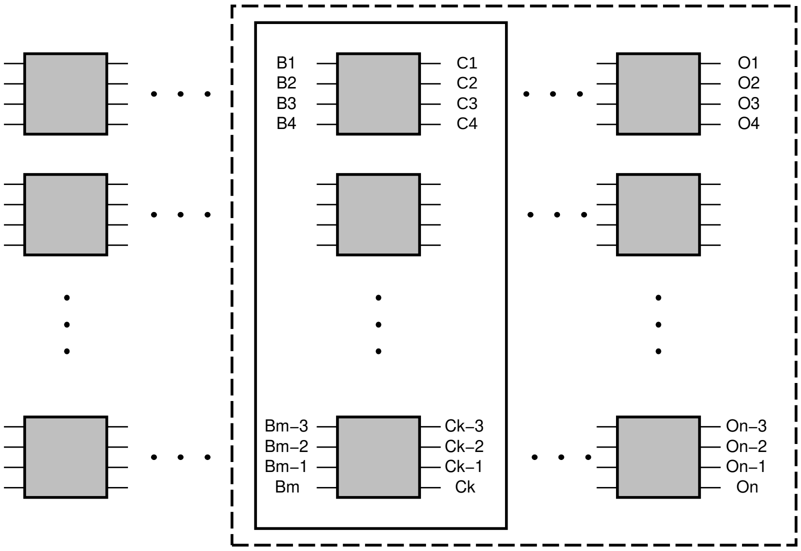

Figure 5.4: The solid box shows the stages of the network for which the estimator g performs an exact calculation; the dotted box shows the stages of the network for which h will perform an exact calculation.

5.6.2 Approximating Equation (4.2) Across Multiple Stages

We have then an estimator for the probability that a set of channels at some stage in the network carries a particular load. We may estimate the value of this probability by generating, in accordance with the appropriate probability distribution, input loads for the switches at which the channels originate. It occurs now to ask whether we might be able to extend the estimation technique to cover more than one stage of the network.

The situation is as depicted in Figure 5.4. We have an estimator g that will allow us to estimate the probability of loads on the channels $C_1, \ldots, C_k$, if we generate the loads on the input channels $B_1, \ldots, B_m$. We require an estimator h that will allow us to estimate the probability of loads on the channels $O_1, \ldots, O_n$, by generating the loads $B_1, \ldots, B_m$.

The estimator $h\,(l_{B_1}, \ldots, l_{B_m})$ will in fact simply be

$$h\,(l_{B_1}, \ldots, l_{B_m}) = \\ \mathrm{P}\{L_{O_1} = l_{O_1}, \ldots, L_{O_n} = l_{O_n} \mid L_{B_1} = l_{B_1}, \ldots, L_{B_m} = l_{B_m}\} \quad (5.6)$$

which, by an argument identical to that for g, will be an unbiased estimator of $\mathrm{P}\{L_{O_1} = l_{O_1}, \ldots, L_{O_n} = l_{O_n}\}$.

To evaluate the conditional probability, we define

$$\mathrm{Q}\{E\} = \mathrm{P}\{E \mid L_{B_1} = l_{B_1}, \ldots, L_{B_m} = l_{B_m}\}$$

If the input channels to the final-stage switches are called $D_1, \ldots, D_j$, we now have

$$\begin{aligned} &\mathrm{Q}\{L_{O_1} = l_{O_1}, \ldots, L_{O_n} = l_{O_n}\} = \\ &\quad \sum_{l_{D_1}, \ldots, l_{D_j}} \mathrm{Q}\Big\{L_{O_1} = l_{O_1}, \ldots, L_{O_n} = l_{O_n} \mid L_{D_1} = l_{D_1}, \ldots, L_{D_j} = l_{D_j}\Big\} \cdot \\ &\quad\quad \mathrm{Q}\Big\{L_{D_1} = l_{D_1}, \ldots, L_{D_j} = l_{D_j}\Big\} \end{aligned} \tag{5.7}$$

which is similar to Equation (4.2). Note that

$$\begin{aligned} &\mathrm{Q}\Big\{L_{O_1} = l_{O_1}, \ldots, L_{O_n} = l_{O_n} \mid L_{D_1} = l_{D_1}, \ldots, L_{D_j} = l_{D_j}\Big\} = \\ &\quad \mathrm{P}\Big\{L_{O_1} = l_{O_1}, \ldots, L_{O_n} = l_{O_n} \mid L_{D_1} = l_{D_1}, \ldots, L_{D_j} = l_{D_j}\Big\} \end{aligned}$$

because, given the loads on the input channels $D_1, \ldots, D_j$, the loading probabilities on the channels $O_1, \ldots, O_n$ are independent of the loads on $B_1, \ldots, B_m$, so long as these are distinct from $D_1, \ldots, D_j$. Thus the conditional probability inside the summation can be factored in the same fashion as that in Equation (4.2).

We can evaluate the term

$$\mathrm{Q}\Big\{L_{D_1} = l_{D_1}, \ldots, L_{D_j} = l_{D_j}\Big\}$$

using Equation (5.7) recursively, just as we did with Equation (4.2). In fact, the only point at which the evaluation of $h(l_{B_1}, \ldots, l_{B_m})$ will differ from that of a network comes when the channels $D_1, \ldots, D_j$ correspond to the channels $B_1, \ldots, B_m$. At this point we will be evaluating

$$\begin{aligned} &\mathrm{Q}\{L_{B_1} = l'_{B_1}, \ldots, L_{B_m} = l'_{B_m}\} = \\ &\quad \mathrm{P}\{L_{B_1} = l'_{B_1}, \ldots, L_{B_m} = l'_{B_m} \mid L_{B_1} = l_{B_1}, \ldots, L_{B_m} = l_{B_m}\} \end{aligned}$$

which will be 1 only when

$$l_{B_1} = l'_{B_1}, \ldots, l_{B_m} = l'_{B_m}$$

and will be 0 otherwise.

It is interesting to note that this last expression can thus be factored as

$$\begin{aligned} &\mathrm{Q}\{L_{B_1} = l'_{B_1}, \ldots, L_{B_m} = l'_{B_m}\} = \\ &\qquad \mathrm{P}\{L_{B_1} = l'_{B_1} \mid L_{B_1} = l_{B_1}\} \cdot \ldots \cdot \mathrm{P}\{L_{B_m} = l'_{B_m} \mid L_{B_m} = l_{B_m}\} \end{aligned}$$

which demonstrates that, given the input loads, the individual channel probabilities are independent. In particular, we see that in evaluating $h\,(l_{B_1}, \ldots, l_{B_m})$ we may treat the channels $B_1, \ldots, B_m$ as the source channels of a network the sources of which have transmission probability 0 when $l_{B_j} = 0$, and transmission probability 1 when $l_{B_j} = 1$.

That is, we see from Equation (4.4) that a channel leading from a source that transmits with probability l_{B_i} has a loading probability mass function

$$\mathrm{P}\Big\{L_{B_i} = l'_{B_i}\Big\} = \begin{cases} 1 - l_{B_i} & \text{if } l'_{B_i} \text{ is } 0 \\ l_{B_i} & \text{if } l'_{B_I} \text{ is } 1 \end{cases} \tag{5.8}$$

which, because l_{B_i} and l'_{B_i} can only be 0 or 1, is the same as

$$\mathrm{P}\Big\{L_{B_i} = l'_{B_i} \mid L_{B_i} = l_{B_i}\Big\}$$

Therefore we see that we may evaluate the conditional probability that is the definition of the estimator h by means of recursive application of Equation (4.3) with a network whose sources $I_{B_1}, \ldots, I_{B_m}$ are connected to channels $B_1, \ldots, B_m$. Source I_{B_j} has source transmission probability 0 when $l_{B_j} = 0$, and source transmission probability 1 when $l_{B_j} = 1$.

Thus a scheme for approximating Equation (4.2) is to pick a stage at which to divide the network, and solve the network to the right of it exactly, given source transmission probabilities equal to loads that we generate with probabilities given by the joint probability mass function of the channels where the division was made. This yields a sample value of the unbiased estimator $h\,(l_{O_1}, \ldots, l_{O_n})$, whose expectation we may evaluate by a Monte Carlo method.

5.6.3 Generating Random Variates from the Joint Probability Mass Function $\mathrm{P}\{L_{B_1} = l_{B_1}, \ldots, L_{B_m} = l_{B_m}\}$

It remains to describe a method of generating random tuples $l_{B_1}, \ldots l_{B_m}$ in accordance with the probability mass function $\mathrm{P}\{L_{B_1} = l_{B_1}, \ldots, L_{B_m} = l_{B_m}\}$.

The method is straightforward: we simply simulate the network using the method of Section 5.2, and use the channel loads generated by the simulation. Because we were careful that our simulation would correspond exactly to the equations, the random variates generated this way will have the mass function $\mathrm{P}\{L_{B_1} = l_{B_1}, \ldots, L_{B_m} = l_{B_m}\}$.

Thus we see that one method of approximate solution of the exact equations corresponds to combining simulation and exact calculation. In fact, looked at another way, solving for the loading probabilities of the subnetwork made up of the later stages is simply a means of reducing the variance of the simulation, because, as we shall see, $h(l_{O_1}, \ldots, l_{O_n})$ will always have lower variance than the corresponding Bernoulli variable in direct simulation.

5.7 Examples of Approximation of the Exact Equations

A program has been written to use the approximation method described in the previous section. We will first examine some details of the performance of the method by considering some examples in detail. Then we will use the techniques we have described to compare the performance of three networks.

5.7.1 Performance of the approximation method on some simple examples

For a first example, let us consider the familiar network depicted in Figure (5.5). Here the estimator h is used for only the final stage of the network. The resulting run is shown in Figure 5.6.

We see that about $11,000$ iterations are required to estimate a loading probability of

$$\frac{100811}{168112} \approx 0.5997$$

as compared to about $25,000$ iterations for the same error bound by direct simulation. The reason for the difference is directly evident when we compare the variance of $f(\mathbf{R})$ to that of $h(l_{O_1}, \ldots, l_{O_n})$:

$$\mathrm{Var}(f(\mathbf{R})) \approx 0.239 \quad \text{but} \quad \mathrm{Var}(h(l_{O_1}, \ldots, l_{O_n})) \approx .098$$

so that the variance has been reduced by a factor of about 2.43.

This is in fact a general result; the variance of h will always be lower than that of f, as we shall see in Section 5.9.

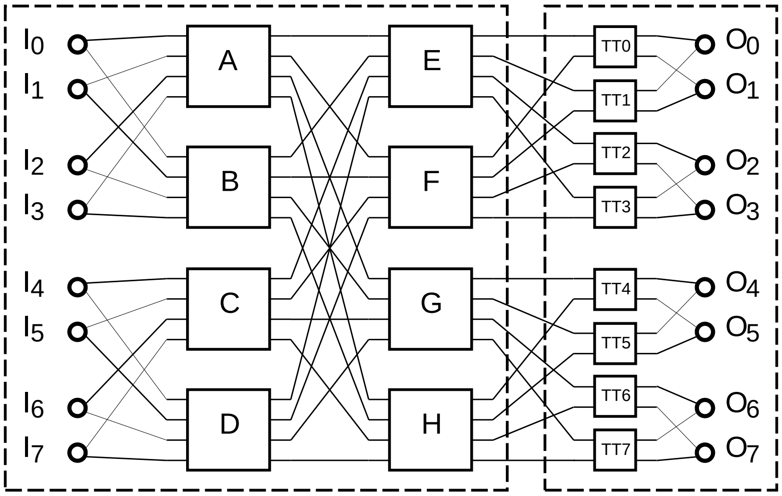

Figure 5.5: The 8×8 deterministically-interwired network of Figure 4.1. The left box contains the two stages simulated in the first example of Section 5.7.1; the right box contains the stage solved for exactly.

```
> (estimate-loading-probability d8x8-left-2 d8x8-right-1
         '((tt6-o7-0 0) (tt7-o7-0 0))
         (make-clt-stopping-function .01 .05 5000)
         '(g-tt6-sink h-tt6-sink g-tt7-sink h-tt7-sink))
Iteration 15; mean:  0.7; variance: .151; current confidence .056
Iteration 5000; mean: .597; variance:  0.1; current confidence .819
Iteration 10000; mean: .601; variance: .098; current confidence .944
100811/168112
231602841/2354912896
10507
```

Figure 5.6: Estimation by approximation method of the probability that both channels leading to sink 7 in the network of Figure 5.5 carry no messages, under uniform addressing and with a source transmission probability of 1/2. Compare to Figure 5.2.

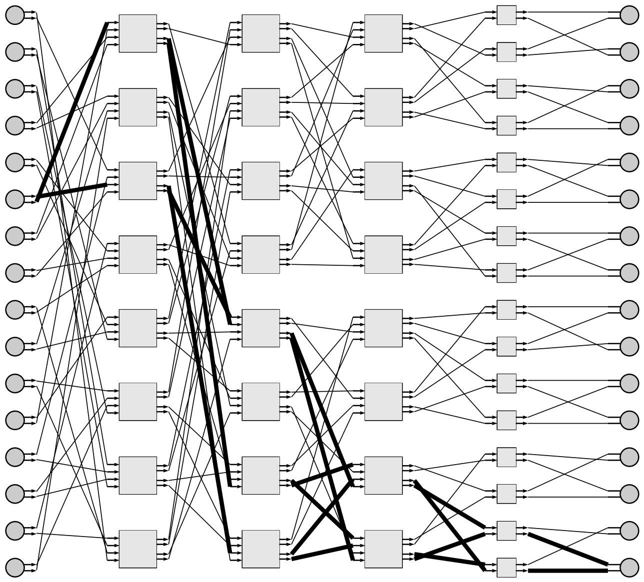

Figure 5.7: The 16×16 randomly-interwired network of Figure 4.8. The network is from [2].

It will be clear from Equations (5.2) and (5.3) that, all other factors remaining equal, the number of iterations required to achieve an error bound is a linear function of the variance of the random variable whose expectation is being estimated.

Now we try direct simulation and our approximation method on the four-stage 16×16 network of Figure 4.8, reproduced in Figure 5.7.

We see in Figure 5.8 the results of using direct simulation to estimate the probability that the top channel leading to sink 0 in the network of Figure 5.7 carries no messages. In Figure 5.9 we see the results of using the approximation method where exact calculation is used for only the final stage of the network. Finally, in Figure 5.10 we see the results of using the approximation method where exact calculation is used for the final two stages of the network. In all three cases, uniform addressing was used, with sources having transmission probabilities of $1/2$.

Where direct simulation was used, the variance was ≈ 0.171; where exact calculation was used for only the final stage, the variance was ≈ 0.072; where exact calculation was used for the final two stages, the variance was ≈ 0.018.

```
> (simulate-multi-channel-loading-probability rnd16x16
       (elements-named '(tt0-o0-0) rnd16x16) '(0)
       (make-clt-stopping-function .01 .05 5000))
Iteration 15; mean:  0.8; variance: .171; current confidence 0.06
Iteration 5000; mean: .778; variance: .173; current confidence .815
Iteration 10000; mean: 0.78; variance: .172; current confidence 0.94
8392/10737
2459905/14409054
10737
```

Figure 5.8: Estimating the probability that the first channel leading to sink 0 in the network of Figure 5.7 carries no messages, by direct simulation. $10,737$ iterations were required to achieve the error bound of $\pm 1\%$ with 95% confidence. Here uniform addressing was used, with sources having transmission probabilities of $1/2$.

```
> (estimate-loading-probability rnd16x16-left-3 rnd16x16-right-1
         '((tt0-o0-0 0))
         (make-clt-stopping-function .01 .05 5000)
         '(s-tt0-sink t-tt0-sink))
Iteration 15; mean: 0.75; variance: 0.08; current confidence .082
7101/9076
2976821/41177812
4538
```

Figure 5.9: Estimating the probability that the first channel leading to sink 0 in the network of Figure 5.7 carries no messages, by approximation where exact calculation is used for only the final stage of the network. $4,538$ iterations were required to achieve the error bound of $\pm 1\%$ with 95% confidence. Uniform addressing was used, with sources having transmission probabilities of $1/2$.

```
> (estimate-loading-probability rnd16x16-left-2 rnd16x16-right-2
         '((tt0-o0-0 0))
         (make-clt-stopping-function .01 .05 5000)
         '(j-s-sink k-s-sink l-s-sink m-s-sink
           j-t-sink k-t-sink l-t-sink m-t-sink))
Iteration 15; mean: .762; variance: 0.02; current confidence .164
16577/21376
935754207/51132760064
1169
```

Figure 5.10: Estimating the probability that the first channel leading to sink 0 in the network of Figure 5.7 carries no messages, by approximation where exact calculation is used for the final two stages of the network. $1,169$ iterations were required to achieve the error bound of $\pm 1\%$ with 95% confidence. Uniform addressing was used, with sources having transmission probabilities of $1/2$.

We see then that by using exact calculation for two stages of this network, we reduce the number of iterations necessary by a factor of about 9. In the next section we will see why we can always expect lower variance from h than from f.

5.7.2 A comparison of the performance of three networks

We present three example networks, all taken from [2]. The first network, shown in Figure 5.11, is constructed from two non-dilated four-stage networks connecting 16 endpoints. Because the degree of path-redundancy is small (there are only two paths connecting any two endpoints), automatic calculation of the exact probability of successful message transmission is feasible.

The second network, shown in Figure 5.12, is a deterministically-interwired multipath network constructed from 4×2, dilation 2 crossbars, and 2×2 crossbars. As can be seen in the figure, multiple paths connect any two endpoints, and calculation of the exact probability of successful message transmission is not quickly feasible on current uniprocessor workstations.

The third network is the randomly-interwired multipath network of Figure 5.7. Recall that, as with the deterministically-interwired network, multiple paths connect any two endpoints, and, again, exact calculation of per-

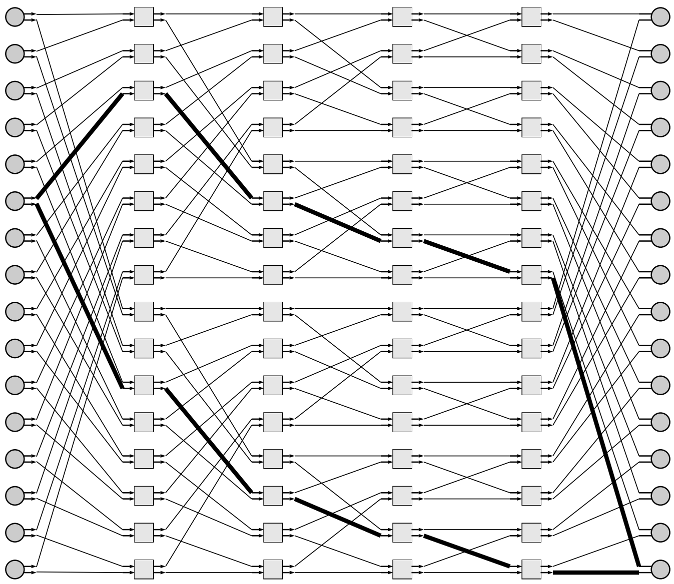

Figure 5.11: A 16×16 network constructed from two non-dilated networks each connecting 16 endpoints. Redundant paths between an input and an output are shown. The figure is from [2].

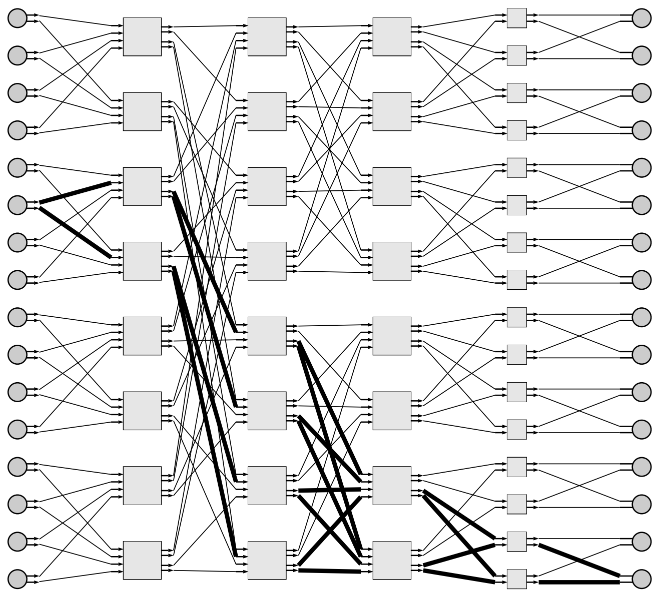

Figure 5.12: A 16×16 network with deterministic interwiring in the first and second stages. Redundant paths between an input and an output are shown. The figure is from [2].

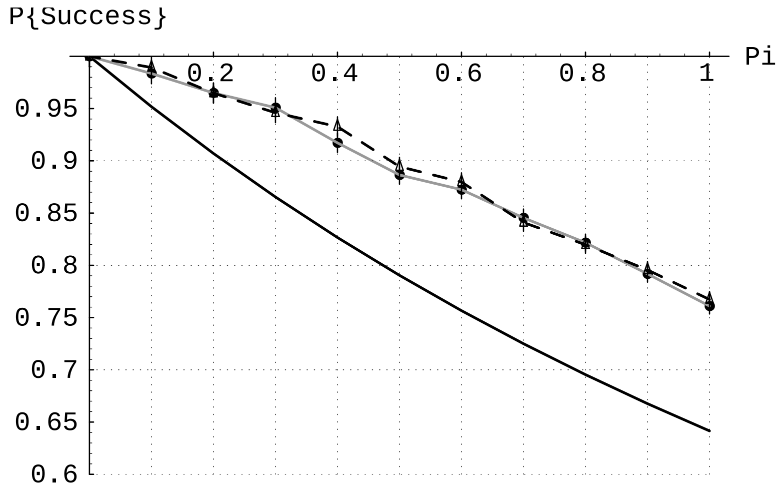

Figure 5.13: The probability of successful message transmission is shown for each of the three networks in Figures 5.11, 5.12, and 5.7. The results for the replicated network are shown in black, and are exact; the results for the deterministically-interwired network are shown in grey, and those for the randomly-interwired network are shown dashed. See the text for a discussion of the results.

formance parameters is too expensive to be feasible.

The performance of the three networks can nonetheless be compared effectively using the exact method for the first and the approximation method for the second and third. In the cases where the approximation method was used, we have specified that the solution must lie within $\pm 1\%$ of the actual value with 95% confidence.

We see in Figure 5.13 the probability of successful message transmission for each of the three networks, and in Figure 5.14 the bandwidth, or throughput, for each of the three networks. As was also found in [2] (although using a much more complex model), the deterministically- and randomly-interwired networks perform identically to within the resolution of the approximation; and the replicated network performs considerably worse than either.

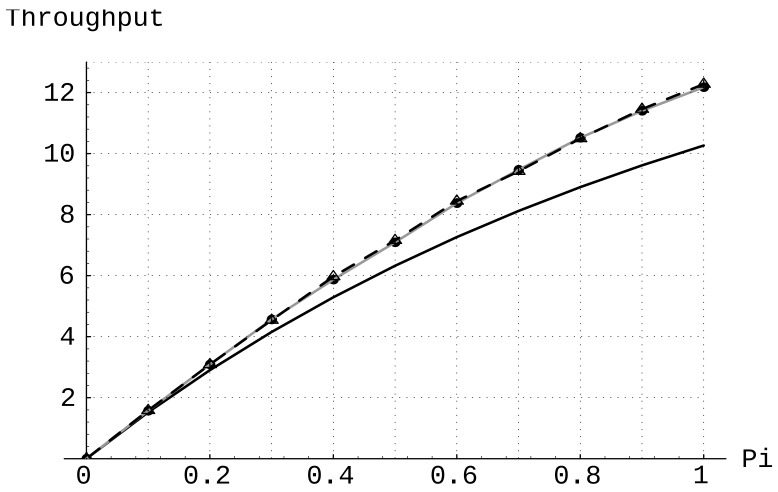

Figure 5.14: The bandwidth, or throughput, is shown for each of the three networks in Figures 5.11, 5.12, and 5.7. The results for the replicated network are shown in black, and are exact; the results for the deterministically-interwired network are shown in grey, and those for the randomly-interwired network are shown dashed. See the text for a discussion of the results.

5.8 Variance of Estimators in the Approximation Method and in Direct Simulation

A simple and well-known theorem in Monte Carlo methods confirms what we have seen in our examples: the estimator h will always have lower variance than will the estimator f. In [10], the theorem is paraphrased as, "if, at any point of a Monte Carlo calculation, we can replace an estimate by an exact value, we shall reduce the sampling error in the final result." This is why we can see our method of approximating the exact equations as a means of reducing the variance of the simulation. The exact equations are too expensive to solve exactly for large networks with many dependent paths, but knowledge and use of the exact equations on a subproblem makes it possible for us to realize in simulation the reduced sampling error promised by the theorem.

The argument in [10] is short enough that we reproduce it here, adapted to our particular estimators.

We note that $f(\mathbf{R})$ and $h(l_{B_1}, \ldots, l_{B_m})$ have the same mean, μ. Because f is binomial, it has variance $\mu(1-\mu)$. The variance of h is given by

$$\mathrm{Var}(h) = \mathrm{E}\left[h^2\right] - \mathrm{E}[h]^2$$

Thus

$$\begin{aligned} \mathrm{Var}(f) - \mathrm{Var}(h) &= \mu - \mu^2 - \left(\mathrm{E}\left[h^2\right] - \mu^2\right) \\ &= \mu - \mathrm{E}\left[h^2\right] \\ &= \mathrm{E}\left[h - h^2\right] \end{aligned}$$

Now, h, being a loading probability, lies in the interval $[0,1]$, so that everywhere $h \geq h^2$. But h takes on with nonzero probability at least some values that are not 0 or 1, because h is not Bernoulli, so that for some tuples $\langle l_{B_1}, \ldots, l_{B_m} \rangle$, $h - h^2 > 0$. Thus $\mathrm{E}[h - h^2] > 0$ and $\mathrm{Var}(f) > \mathrm{Var}(h)$, as we desired to show.

5.9 Expense of the Approximation Method

One is tempted by the results of Section 5.7.1 to ask what happens if we again increase the number of stages for which h performs an exact calculation. Although it seems likely that the variance would be reduced further, the experiment is not likely to be worth our while, as the network for which we would be calculating exactly the loading probabilities would now have a much larger number of redundant paths leading from its sources to sink 0. Thus we would be faced with the same problem of exponential growth as before.

Our method can only reduce the expense of simulation so much, until the exponential growth of the running time of each iteration with the number of dependent channels dominates the savings in number of iterations. In fact, the final stage of a network, considered by itself, will always constitute a Banyan network, and so we can can always calculate loading probabilities across it at the same asymptotic expense as simulation – there are no redundant paths, and the reduction of the number of iterations with the variance will be realized in reduced running time.

The final two stages of the network of Figure 5.7 do not constitute a Banyan network, but the number of redundant paths between a source and a sink is small (two), and so in this case the running time is also significantly reduced. In many types of multipath networks larger final subnetworks constitute Banyan networks or have small numbers of dependent channels; in these networks it will be profitable to use exact calculation for more than one final stage.

In a network with N stages with M channels between each stage, if exact calculation is used for the final K stages, then in the worst case, where the load on every channel between two stages of switches in the final K stages is dependent on the loads on the other channels between those two stages of switches, the running time of exact calculation for the final stages will be $\mathrm{O}\left(KM2^{2M}\right)$. There will be $N-K$ stages simulated, at an expense of $\mathrm{O}((N-K)\,M)$ steps per simulation, so that the worst-case performance will be

$$\mathrm{O}\left(\frac{\sigma^2 M}{cd^2\mu^2}\left(K2^{2M}+(N-K)\right)\right)=\mathrm{O}\left(\frac{\sigma^2 M}{cd^2\mu^2}\left(2^{2M}K+N\right)\right)$$

where c is the complement of the desired confidence; d is the deviation factor, μ is the mean and σ^2 the variance of h, the result of exact calculation.

The worst-case result is misleading, however, because in networks built in practice, the subnetworks constituted by final stages have smaller numbers of dependent channels than does the entire network. In fact, if the final stages for which exact calculation is performed constitute a Banyan network, then the running time of exact calculation is $\mathrm{O}(KM)$, and the asymptotic running time of the approximation is simply

$$\mathrm{O}\left(NM\frac{\sigma^2}{cd^2\mu^2}\right)$$

where once again c is the confidence complement, d the deviation factor, μ the mean and σ^2 the variance of h.

5.10 Conclusions

We have developed methods of calculating the value of some performance parameters for multistage networks – the normalized throughput and probability of successful message transmission – by computing the loading probabilities of channels leading to sinks.

We showed initially that independence of loads on channels in a Banyan network allows a simple means of calculating channel loading probabilities for these networks, and described a way of composing operations on loading probability mass functions to derive expressions for the performance parameters. We presented a program that derived such expressions and could be used for numerical calculation of performance parameters.

We then saw that independence of loads on channels will not hold in multipath networks, and developed equations for channel loading probabilities in these networks. We showed that the number of equations that must be solved by this method is exponential in the number of dependent paths in the network, rendering the method impractical for large networks. We presented a program that could be used to calculate channel loading probabilities exactly for small networks, and discussed its performance in the cases of multipath networks and Banyan networks.

We developed a method of approximate solution of the exact equations, and compared its performance to that of direct simulation. We developed programs for both our approximation method and direct simulation. We saw that use of the exact equations will always afford some improvement in performance, by reducing the variance of the estimator in question; and we discussed cases where the reduction in running time will be quite substantial.

5.11 Future Work

The literature on Monte Carlo methods contains many techniques for reducing the variance of estimators. Some of these are particularly promising for our application. For example, the use of stratified sampling, where the strata are segregated by the number of messages generated by sources in a particular cycle, should be easy to implement and promises a significant reduction in variance.

We look forward to comparing more results of the application of these methods to the results of more faithful and complex simulations performed at M.I.T.'s Transit Group. The aim of the Transit Group's simulations is to select a network structure for implementation in a large-scale multiprocessor. We expect from the results cited in Section 1.4 that our model will be useful in comparing candidate networks.

Appendix A

Mathematica Procedures for Modelling Banyan Networks

```
concentrate::usage =
    "concentrate[x, n] concentrates the LPMF x to n channels."

concentrate[x_, n_] :=
    (* get distribution for 0 through n-1 channels, and add
       as last element the sum of the rest of the channels. *)
    Append[Take[x, n], Apply[Plus, Drop[x, n]]]

discreteconvolution::usage =
    "discreteconvolution[x, y] treats x and y as 0-based
     vectors and returns their discrete convolution."

discreteconvolution[x_, y_] :=
    Block[{xlgth, ylgth, lgth},
          xlgth = Length[x];
          ylgth = Length[y];
          lgth = xlgth + ylgth - 1;
          (* in summation, portions of sequence with indices
             out of range for sequences must be treated as
             0. *)
          Table[Sum[If[k < 1 || k > xlgth ||
                       (n-k+1) < 1 || (n-k+1) > ylgth,
                       0,
                       (* because of the 0->1 index
```

```
                          translation, we increase the y-index
                          to shift the result sequence back
                          down to begin at 1. *)
                       x[[k]] y[[n-k+1]]],
                    {k, xlgth}],
                {n, lgth}]]

bundle::usage =
    "bundle[x, y] forms the LPMF that results from bundling
     two input bundles with LPMFs x and y."

bundle[x_, y_] :=
    discreteconvolution[x, y]


switch::usage =
    "switch[x, p] returns the LPMF of an output bundle to
     which x is switched with probability p."

switch[x_, p_] :=
    Block[{lgth},
          lgth = Length[x];
          Table[Sum[x[[i+1]] Binomial[i, n] p^n (1-p)^(i-n),
                    {i, n, lgth-1}],
                {n, 0, lgth-1}]]
```

Bibliography

[1] Adams, G. B. III, and Siegel, H. J. "The Extra Stage Cube: A Fault-Tolerant Interconnection Network for Supersystems," in *IEEE Transactions on Computers*, Vol. C-31, No. 5, pp. 443-454, May 1982.

[2] Chong, F., Egozy, E., and DeHon, A. "Fault Tolerance and Performance of Multipath Multistage Interconnection Networks," in *Advanced Research in VLSI: MIT/Brown Conference 1992,* edited by Thomas F. Knight Jr. and John Savage, MIT Press, to be published March 1992.

[3] Beneš, V. E. *Mathematical Theory of Connecting Networks and Telephone Traffic.* Academic Press, New York, 1965.

[4] Bhandarkar, D. P. "Analysis of Memory Interference in Multiprocessors," in *IEEE Transactions on Computers,* Vol. C-24, No. 9, September 1975.

[5] Bhuyan, L. N. "An Analysis of Processor-Memory Interconnection Networks," in *IEEE Transactions on Computers*, Vol. C-34, No. 3, March 1985.

[6] Chang, D. Y., Kuck, D. J., and Lawrie, D. H. "On the Effective Bandwidth of Parallel Memories," in *IEEE Transactions on Computers,* Vol. C-26, No. 5, May 1977.

[7] Clos, C. "A Case Study of Non-blocking Switching Networks," in *Bell System Technical Journal,* Vol. 32, pp. 406-424, March 1953.

[8] Dias, D. M., and Jump, J. R. "Augmented and Pruned N Log N Multistage Networks: Topology and Performance," in *Proceedings of the 1982 International Conference on Parallel Processing.* IEEE Computer Society Press, August, 1982.

[9] Goke, L. R., and Lipovski, G. J. "Banyan Networks for Partitioning Multiprocessor Systems," in *Proceedings of the First Annual Symposium on Computer Architecture*, 1973.

[10] Hammersley, J. M., and Handscomb, D. C. *Monte Carlo Methods.* Methuen's Monographs on Applied Probability and Statistics, Methuen and Company, London, 1965.

[11] Harvey, C., and Hills, C. R. "Determining Grades of Service in a Network," in *Ninth International Teletraffic Conference*, Torremolinos, Spain, October, 1979.

[12] Hui, J. Y. *Switching and Traffic Theory for Integrated Broadband Networks,* pp. 246-270. Kluwer Academic Publishers, Boston, 1990.

[13] Jeng, M., and Siegel, H. J. "A Fault-tolerant Multistage Interconnection Network for Multiprocessor Systems Using Dynamic Redundancy," in *The 6th International Conference on Distributed Computing Systems.* Cambridge, Massachusetts, May, 1986.

[14] Knight, T. F., and Sobalvarro, P. G. "Routing Statistics for Unqueued Banyan Networks." M.I.T. Artificial Intelligence Laboratory Memo No. 1101, September, 1990.

[15] Kruskal, C. P., and Snir, Marc. "The Performance of Multistage Interconnection Networks for Multiprocessors," in *IEEE Transactions on Computers,* Vol. C-32, No. 12, December 1983.

[16] Leighton, T., and Maggs, B. "Expanders Might be Practical: Fast Algorithms for Routing Around Faults in Multibutterflies," in *30th Annual Symposium on Foundations of Computer Science,* IEEE Computer Society Press, November, 1989.

[17] Lin, T., and Kleinrock, L. "Performance Analysis of Finite-buffered Multistage Interconnection Networks with a General Traffic Pattern," in *Proceedings of the 1991 ACM SIGMETRICS Conference on Measurement and Modelling of Computer Systems*, May 1991.

[18] Merchant, A. *Analytical Models of the Performance Analysis of Banyan Networks.* Ph.D. Thesis, Stanford University Department of Computer Science Report No. STAN-CS-90-1347, December, 1990.

[19] Minsky, H., DeHon, A., and Knight, T. F. "RN1: Low-latency, Dilated, Crossbar Router," in *Hot Chips Symposium III*, 1991.

[20] Nussbaum, D., Vuong-Adlerberg, I., and Agarwal, A. "Modeling a Circuit-Switched Multiprocessor Interconnect," in *Proceedings of of the 1990 ACM SIGMetrics Conference on Measurement and Modeling of Computer Systems*, May, 1990.

[21] Patel, J. H. "Performance of Processor-Memory Interconnections for Multiprocessors," in *IEEE Transactions on Computers,* Vol. C-30, No. 10, October 1981.

[22] Pfister, G. F. *et al.,* "The IBM Research Parallel Processor Prototype (RP3): Introduction and Architecture," in *Proceedings of the 1985 International Conference on Parallel Processing*, IEEE Computer Society Press, August, 1985.

[23] Reddy, S. M., and Kumar, V. P. "On Fault-Tolerant Multistage Interconnection Networks," in *Proceedings of the 1984 International Conference on Parallel Processing*, IEEE Computer Society Press, August, 1984.

[24] Rettberg, R., and Thomas, R. "Contention is no Obstacle to Shared-Memory Multiprocessing," in *Communications of the ACM,* Vol. 29, No. 12, December, 1986.

[25] Ross, Sheldon M. *Introduction to Probability Models,* Fourth Edition. Academic Press, San Diego, California, 1989.

[26] Upfal, E. "An $O(\log N)$ Deterministic Packet Routing Scheme," in *Proceedings of the 21st Annual ACM Symposium on the Theory of Computing,* May, 1989.