

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

**ANALYSIS OF BOOLEAN N-CUBE INTERCONNECTION  
NETWORKS FOR MULTIPROCESSOR SYSTEMS**

**M.-Y. HORNG**

**March 1992  
CSD-920008**



UNIVERSITY OF CALIFORNIA  
Los Angeles

**Analysis of Boolean  $n$ -Cube Interconnection  
Networks for Multiprocessor Systems**

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

by

**Ming-yun Horng**

1992

© Copyright by  
Ming-yun Horng  
1992

The dissertation of Ming-yun Horng is approved.



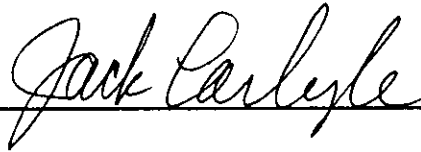
---

Kirby Baker



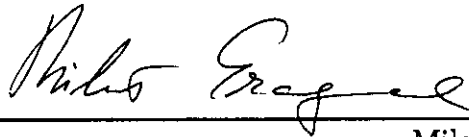
---

Christopher Tang



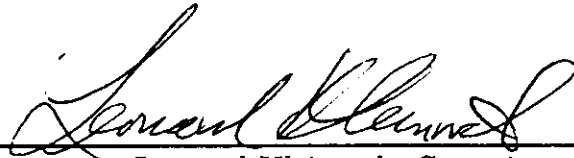
---

Jack Carlyle



---

Milos Ercegovac



---

Leonard Kleinrock, Committee Chair

University of California, Los Angeles

1991

To my family,  
for providing me with the love, support and freedom.

## TABLE OF CONTENTS

<b>1</b>	<b>Introduction</b>	<b>1</b>
1.1	Multiprocessor Interconnection Networks	1
1.2	Boolean $n$ -Cube Networks	6
1.2.1	Topological Properties	8
1.2.2	Routing Procedures	9
1.2.3	Switching Techniques	12
1.3	Design Consideration	13
1.3.1	Simplicity	13
1.3.2	Deadlock-free Routing in Finite-buffered Networks	14
1.3.3	Fault-tolerant Routing in Damaged Networks	15
1.3.4	Performance Analysis and Improvements	15
1.4	Summary of Dissertation	16
<b>2</b>	<b>Basic Models for Boolean <math>n</math>-Cube Networks</b>	<b>19</b>
2.1	Introduction	19
2.2	Assumptions of the Network Operation	19
2.3	Model 1: A Buffer at Each Outgoing Channel	21
2.3.1	Assumptions of the Model	21
2.3.2	Analysis of the Model	23
2.3.3	Validation of the Model	30
2.4	Model 2: A Shared Buffer at Each Node	31
2.4.1	Assumptions of the Model	31
2.4.2	Analysis of the Model	34
2.4.3	Validation of the Approximation	38
2.5	A lower Bound on the Mean Delay	39
2.5.1	The Optimistic Assumption	40
2.5.2	Analysis of the Model	42
2.5.3	Discussion: How good is the Random Assignment	46
2.6	Conclusions	48
<b>3</b>	<b>Analysis of a Deadlock-free Routing Algorithm with the Deflection Technique</b>	<b>49</b>
3.1	Introduction	49
3.2	A Deadlock-free Routing Algorithm in Boolean $n$ -Cube Networks	50
3.3	Analysis of the Deadlock-free Routing Algorithm	53
3.3.1	Assumptions of the Model	54
3.3.2	Imbedded Markov Chain Analysis	55

3.3.3	Calculating the Matrix $P$ . . . . .	56
3.3.4	Determining the Value for $P_t$ . . . . .	60
3.3.5	Delay and Throughput . . . . .	65
3.3.6	Effects of Deflection . . . . .	66
3.3.7	Validation of the Model and Discussion . . . . .	68
3.4	Optimization Issues . . . . .	71
3.5	Conclusions . . . . .	74
<b>4</b>	<b>Analysis of Deflection Routing in <math>k</math>-Ary <math>n</math>-Cube Networks . . . . .</b>	<b>75</b>
4.1	Introduction . . . . .	75
4.2	Deflection Routing in $k$ -Ary $n$ -Cube Networks . . . . .	78
4.3	Analysis of the Deflection Routing Algorithm . . . . .	81
4.3.1	Assumptions of the Model . . . . .	81
4.3.2	Analysis of the Model . . . . .	82
4.3.3	Validation of the Model . . . . .	92
4.4	Discussion . . . . .	95
4.4.1	Optimal Buffer Assignment . . . . .	95
4.4.2	Choice of Dimensions and Radixes . . . . .	96
4.5	Conclusions . . . . .	100
<b>5</b>	<b>Fault-tolerant Routing in Boolean <math>n</math>-Cube Networks . . . . .</b>	<b>102</b>
5.1	Introduction . . . . .	102
5.2	Degradation of Networks with Node Faults . . . . .	105
5.3	Routing in 1-Degraded Subnets . . . . .	110
5.3.1	The $k$ -Degraded Subnet . . . . .	111
5.3.2	The Optimal-path Routing Algorithm . . . . .	111
5.3.3	Discussion . . . . .	112
5.4	Routing in Convex Subnets . . . . .	114
5.4.1	The Convex Subnet . . . . .	114
5.4.2	The Routing Algorithm . . . . .	116
5.4.3	Discussion . . . . .	117
5.5	Routing in Two-level Hierarchical Networks . . . . .	118
5.5.1	Network Decomposition . . . . .	119
5.5.2	The Two-level Hierarchical Routing Algorithm . . . . .	121
5.5.3	Discussion . . . . .	123
5.6	Conclusions . . . . .	125
<b>6</b>	<b>Conclusions and Future Research . . . . .</b>	<b>128</b>
6.1	Future Work . . . . .	129
6.1.1	Virtual Cut-through with Deflection . . . . .	130



6.1.2	Communication Locality . . . . .	131
6.1.3	Dynamic Restoration of Regularity . . . . .	132
6.1.4	Fault-tolerant Routing for $k$ -Ary $n$ -Cube Networks . . . . .	133
6.2	Final Remarks . . . . .	133
<b>A</b>	<b>Search for Zeroes in the Optimistic Model . . . . .</b>	<b>134</b>
	<b>References . . . . .</b>	<b>139</b>

## LIST OF FIGURES

1.1	Architectures of multiprocessor systems. . . . .	3
1.2	A Boolean 4-cube network. . . . .	6
1.3	Basic routing algorithm for Boolean $n$ -cube networks. . . . .	10
2.1	Model 1: A queue at each outgoing channel. . . . .	22
2.2	Mean delay of a Boolean 6-cube network with a separate buffer at each outgoing channel. . . . .	31
2.3	Model 2: A shared buffer for a node. . . . .	32
2.4	An example of parallel and random message assignment. . . . .	33
2.5	Mean delay of Boolean 6, 8, and 10-cube networks. Lines corre- spond to the Poisson approximation and points are taken from a simulation. . . . .	38
2.6	Comparing the mean delays obtained from models 1 and 2. . . . .	39
2.7	Mean number of messages successfully transmitted. . . . .	41
2.8	Comparing the mean delay of the random assignment algorithm with an optimistic lower bound in a Boolean 6-cube network. . . . .	47
3.1	An example of two-phase message assignment. . . . .	53
3.2	A node with finite buffers. . . . .	55
3.3	Snapshot of queue fluctuations in cycle $m$ . . . . .	57
3.4	State transition diagram for the number of hops traversed by a message. . . . .	62
3.5	Input rate and output rate vs a given $P_t$ . . . . .	66
3.6	Channel utilization and probability of forwarding. . . . .	68
3.7	Probability of acceptance of an input message. . . . .	69
3.8	Throughput per node vs new message generation rate. . . . .	70
3.9	Mean message delay vs the throughput of each node. . . . .	71
3.10	Power vs applied input rates. . . . .	72
3.11	The buffer sizes which can deliver the percentage of the maximal achievable power. . . . .	73
4.1	The examples of $k$ -ary $n$ -cube networks with 16 nodes. . . . .	77
4.2	Transition diagram for calculating the mean path length. . . . .	86
4.3	State transition diagram for calculating mean path length. . . . .	90
4.4	Throughput per node for a 8-ary 2-cube network. . . . .	93
4.5	Mean delay vs throughput per node for a 8-ary 2-cube network. . . . .	94
4.6	The buffer size in a node which delivers 95% of the maximum achievable power in a 8-ary 2-cube network. . . . .	95
4.7	The buffer size in a node which delivers 95% of the maximum achievable power in a 4-ary 3-cube network. . . . .	96

4.8	Normalized delay as a function of the normalized throughput for a network with 64 nodes when $\beta = 0$ .	99
4.9	Normalized delay as a function of the normalized throughput for a network with 64 nodes when $\beta = 0.25$ .	100
4.10	Normalized delay as a function of the normalized throughput for a network with 64 nodes when $\beta = 1$ .	101
5.1	A Boolean 4-cube network with node faults.	103
5.2	A cut in dimension 3. (Surviving links crossing the cut are shown as heavy lines).	106
5.3	Minimal achievable delay of a Boolean 4-cube network with two different failure rates.	109
5.4	The optimal-path routing algorithm for 1-degraded subnets	112
5.5	Percentage of surviving nodes in the 1-degraded subnets.	113
5.6	A convex subnet constructed for the damaged Boolean 4-cube network as shown in Figure 5.1.	116
5.7	A convex subnet constructed for a network containing a faulty link.	117
5.8	Percentage of nodes remaining in the convex subnets.	118
5.9	Comparison of percentage of surviving nodes in a Boolean 8-cube network.	119
5.10	Mean delay for the convex subnets of the Boolean 6-cube networks with 6 faulty nodes.	120
5.11	The algorithm for decomposing a Boolean $n$ -cube network.	121
5.12	A two-level hierarchical structure of the Boolean 4-cube network as shown in Figure 5.1.	123
5.13	Mean number of clusters generated by our decomposition algorithm for a Boolean 6-cube network.	125
5.14	Comparison of the mean path length of the two-level hierarchical network and the original network.	126

## ACKNOWLEDGMENTS

I would like to express my appreciation to my doctoral committee consisting of Professors Leonard Kleinrock, Kirby Baker, Jack Carlyle, Milos Ercegovac, and Christopher Tang. I am particularly grateful to the committee chairman and my advisor, Dr. Leonard Kleinrock for providing enthusiasm, support, and advice during my graduate years.

This research has been supported by the Advanced Research Projects Agency of the Department of Defense under contract MDA 903-82-C0064, Advanced Teleprocessing Systems, and contract MDA 903-87-C0663, Parallel Systems Laboratory. I thank the projects managers for their supporting and believing in what we have been doing.

The ATS/PSL research group has been a wonderful environment in which to work with. To former students of the research group, Jau Huang, Willard Korfhage, Farid Mehovic, Joy Lin, Bob Felderman, Shioupyn Shen, and current students, Chris Ferguson, Jon Lu, Rajeev Gupta, Brian Tang, I offer my sincere thanks for helpful discussions. In particular, I would like to thank Bob, Jon, and Chris for their comments on earlier versions of my dissertation. I am also thankful to the following staff of the group: Lily Chien, Cheryle Childress, and Brenda Ramsey.

I dedicate this dissertation to Angela, my wife, for her constant encouragement and understanding and making my life joyful. My last and greatest gratitude goes to my Mom, Li-yu W. Horng, for her everlasting love and support.

## VITA

- 1958            Born, Taipei, Taiwan
- 1981            B. S., Computer Science and Information Engineering  
National Taiwan University
- 1987            M. S., Computer Science  
University of California, Los Angeles
- 1988–1992      Graduate Student Researcher, Parallel Systems Laboratory,  
Computer Science Department, University of California, Los  
Angeles

## PUBLICATIONS

Ming-yun Horng and Leonard Kleinrock, "Fault-tolerant Routing with Regularity Restoration in Boolean  $n$ -Cube Interconnection Networks," in the *Proceedings of the Third IEEE Symposium on Parallel and Distributed Processing*, pp. 458-465, December 1991.

Ming-yun Horng and Leonard Kleinrock, "On the Performance of a Deadlock-free Routing Algorithm for Boolean  $n$ -Cube Interconnection Networks with Finite Buffers," in the *Proceedings of 1991 International Conference on Parallel Processing*, pp. 228-235, August 1991.

Ming-yun Horng, "An Analytic Model for Packet Flow in a Boolean  $n$ -Cube Interconnection Network," Master Thesis, University of California, Los Angeles, December 1987.

ABSTRACT OF THE DISSERTATION

# Analysis of Boolean $n$ -Cube Interconnection Networks for Multiprocessor Systems

by

**Ming-yun Horng**

Doctor of Philosophy in Computer Science

University of California, Los Angeles, 1992

Professor Leonard Kleinrock, Chair

An effective interconnection architecture, which provides high-bandwidth and low-latency interprocessor communications, is the key to high performance multiprocessor systems. The Boolean  $n$ -cube network has become a popular interconnection architecture due to its topological properties. In this dissertation, we propose and analyze a set of routing algorithms for Boolean  $n$ -cube networks. Most of the analysis we give involves approximation which are shown to be excellent.

We first present several models for evaluating the performance of the Boolean  $n$ -cube network. Performance bounds are also examined. We then develop and analyze a deadlock-free routing algorithm with deflection for Boolean  $n$ -cube networks with finite buffers. We focus on the effect of the buffer size on performance. We show that the throughput of the network never degrades and that a small number of buffers in a node can deliver good performance. We further extend our model to a general class of networks called  $k$ -ary  $n$ -cubes and show that “ $2n$ ” buffers in a node are essential for efficient behavior of an  $n$ -dimensional network. We also analyze the networks with various combinations of dimension  $n$  and radix  $k$ .

We develop several fault-tolerant routing schemes based on the idea of restoring the regularity of a damaged Boolean  $n$ -cube network. One way to restore the regularity of a Boolean  $n$ -cube network in the presence of only node failures is to simply disable those nodes with more than one bad neighbor. The remaining network is called a “1-degraded subnet.” A very simple optimal-path routing algorithm is developed for such a subnet. This approach works well in the situation where a whole cluster of nodes have been “bombed out.” However, many nonfaulty nodes may have to be disabled. We further develop a heuristic algorithm to construct a “convex subnet” in an attempt to retain more nonfaulty nodes. This approach considers both node and link failures. We show that the optimal-path routing algorithm also works for the convex subnet, and that only a small number of nonfaulty nodes need be disabled. We also developed a two-level hierarchical fault-tolerant routing scheme without disabling any nodes. Here, a non-convex Boolean  $n$ -cube network is decomposed into a set of convex subcubes. A two-level hierarchical routing algorithm is developed. We show that the increase in the mean path length caused by hierarchical routing is very small.

# CHAPTER 1

## Introduction

### 1.1 Multiprocessor Interconnection Networks

A wide range of applications in science and technology such as weather forecasting, aerodynamic simulation, image analysis and military defense require enormous computational power that cannot be achieved with a serial computer. The current trend to solve the problem is to break the computation into subparts, build a multiprocessor system incorporating a number of processors, and then concurrently execute these subparts on the system. Many multiprocessor systems are commercially available, but many different multiprocessor architectures are available. In recent years, the advance of VLSI technology has given us an opportunity to build a large scale multiprocessor system consisting of hundreds of thousands of inexpensive processors, which can perform over  $10^9$  operations per second [Hil85].

Many efforts in developing parallel algorithms, parallel compilers, task partitioning and allocation schemes, and communication architectures, have been made to improve the overall performance of multiprocessor systems. Among these, the construction of an interconnection network which provides very high speed and fault-tolerant interprocessor communications is critical.

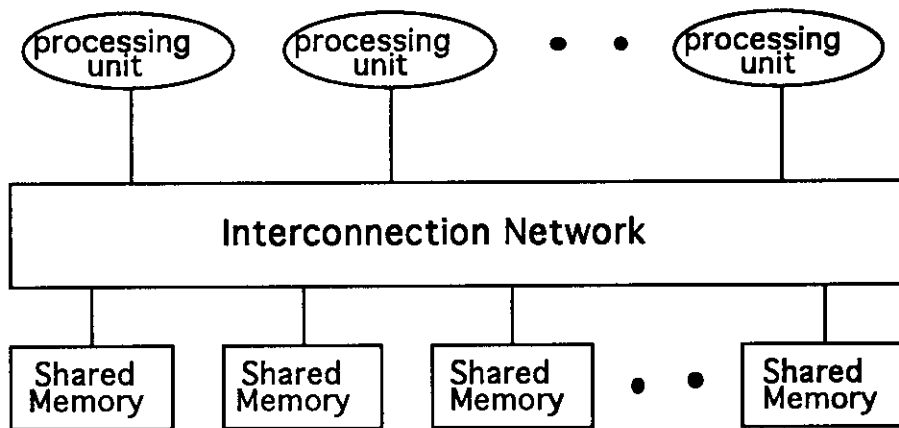
In general, there are two ways of using an interconnection network in a



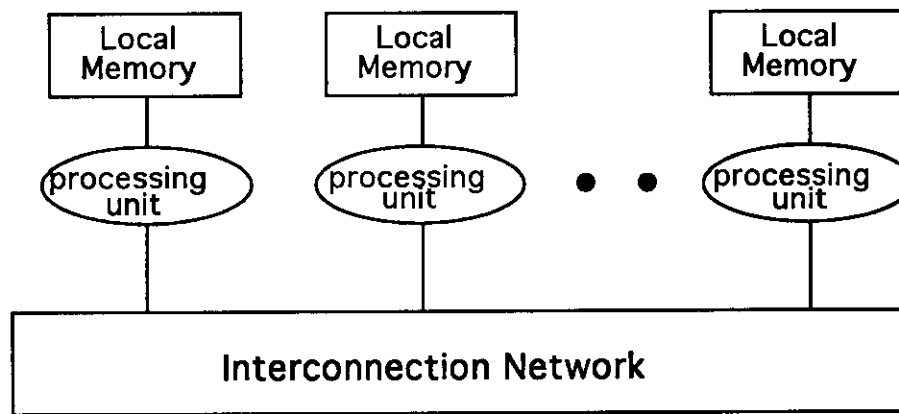
multiprocessor system. (See Figure 1.1.) A multiprocessor can be constructed as a *shared memory architecture* or a *distributed memory architecture*. In the shared memory architecture, an interconnection network is interposed between the processors and the memory modules. Each memory module is accessible to all processors. The problem of the shared memory architecture is that, as the number of processors grows, contention for the memory modules can result in serious performance degradation. The other approach is known as a distributed memory architecture, where direct communications between processors are provided. In this architecture, each processor has its own local memory, which it can access very fast. To access the memory of another processor, a processor needs to communicate with that processor by exchanging messages through the interconnection network. The distributed memory architecture is becoming increasingly popular for large-scale multiprocessor systems because its communication locality is easier to exploit.

Various network topologies for interconnecting the processors and/or memory modules have been proposed. For a review of these networks, see [Fen81], [HB84], and [Sie85]. There is no single network that is generally considered the best for all applications. For example, one very simple communication architecture is the bus architecture where a single bus is shared by all processors. However, the bandwidth of the bus limits the total communication traffic on the network. Although this problem can be alleviated by increasing the bandwidth of the bus, the cost of building faster and faster buses can go unbounded. Besides, the communication latency will become intolerable if the bus is heavily loaded. Thus, the number of processors supportable by a shared bus is limited.

Based on the ways in which processors communicate with each other, inter-



(a) Shared Memory Architecture



(b) Distributed Memory Architecture

Figure 1.1: Architectures of multiprocessor systems.

connection networks can be classified as *direct networks* or *indirect networks*. In an indirect network, there are no direct links between processors. Communications between processors must go through a set of intervening switches. Typical examples of the indirect networks include crossbar networks, multistage networks (e.g., the Omega network), etc [Fen81]. The crossbar is a simple architecture in which a connection between any two processors can be established in a cycle. Let  $N$  be the number of processors. The number of switches required in a fully connected crossbar network is  $O(N^2)$ , which becomes prohibitively large for large  $N$ .

A multistage network consists of several stages of switches. In general, each switch is of the same size and constructed as a small crossbar switch. One typical network of this family is the Omega network which consists of  $\log_2 N$  stages; each stage has  $N/2$  switches. Each switch has two input ports and two output ports. Between two adjacent stages is a perfect-shuffle interconnection. The hardware cost of the Omega network with  $N$  processors is proportional to  $N \log_2 N$ . However, in the Omega network, there is only one single path between any two nodes. This increases the probability of a traffic jam developing and makes fault-tolerant routing more difficult.

In the direct network, processors communicate directly with each other over a set of point-to-point links. Typical point-to-point topologies include rings, trees, stars (special cases of trees), meshes, complete connections, and Boolean  $n$ -cubes. To support a large-scale multiprocessor system, a good point-to-point interconnection network should exhibit the following properties:

1. The diameter of the network should grow slowly with an increasing number of processors.

2. The processors should be topologically equivalent.
3. The addressing scheme should provide a simple routing mechanism.
4. There should be redundant paths between each pair of processors such that congestions can be avoided. Also, by carefully designing the routing algorithms (as shown in Chapter 5), we should be able to continue to use a network in the presence of node and/or link failures.
5. The network should be cost-effective. Links and node hardware to support the physical connections cost money. We should keep the number of links in a network reasonably small.

Thus, for a large-scale interconnection network, some topologies, such as rings and meshes, are immediately ruled out due to their limitations on communication bandwidths and long latencies. Trees become impractical because of the unbalanced heavy load in the root of the tree. This problem of root congestion can be remedied by adding extra links at the higher levels of the tree. Complete connections give the minimal latency time, but their cost is very high. The Boolean  $n$ -cube network is an architecture which satisfies all those requirements. We also note that the considerations of performance and cost are very dependent on the currently available technologies. For example, if a chip can only have at most one hundred pins based on the current technologies, a topology which requires 101 pins per chip is absolutely undesirable. In the rest of this dissertation, we will concern ourselves with the design and performance of the routing algorithms for interconnection networks. The detailed implementation in hardware is beyond our discussion.

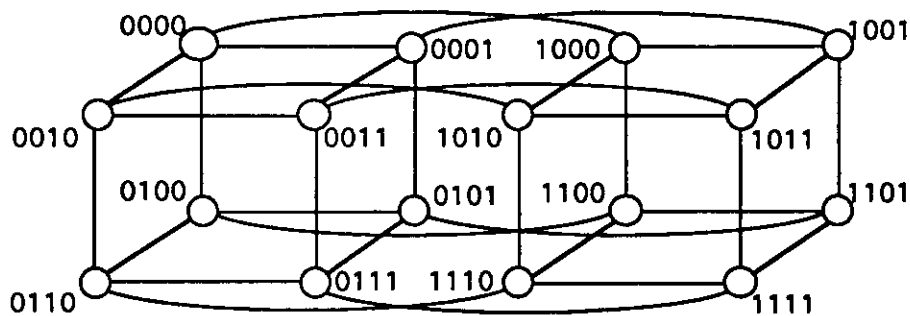


Figure 1.2: A Boolean 4-cube network.

## 1.2 Boolean $n$ -Cube Networks

The Boolean  $n$ -cube network, also known as the hypercube network, has certain topological properties which are discussed in [SS88]. In the following discussion, we simply use *node* to indicate a processor or a communication co-processor which handles communications for several processors. A Boolean  $n$ -cube network consists of  $2^n$  nodes, each addressed by an  $n$ -bit binary number from 0 to  $2^n - 1$ . Nodes are interconnected in such a way that there is a link between two nodes if and only if their addresses differ in exactly one bit position. We say two neighboring nodes with addresses  $i$  and  $j$  are connected together with a link in dimension  $k$  if and only if  $|i - j| = 2^k$ , where  $k = 0, 1, \dots, n - 1$ . Thus, every node has exactly  $n$  neighbors; each is linked to the node with a different dimension from 0 to  $n - 1$ . A Boolean 4-cube network is shown in Figure 1.2, where, for example, nodes 1000 and 1010 are connected together in dimension 1. Here, we assume that every link is bidirectional so that messages can be sent in both directions simultaneously. There are a total of  $n2^{n-1}$  links in a Boolean  $n$ -cube network.

Each node in a Boolean  $n$ -cube network is assumed to be able to serve multiple local processors. A message is injected into the node from its local processor(s) if that processor needs to communicate with some other processor served by another node. The message is then routed to its destination node. When a node receives a message which is destined to one of its local processors, the node removes the message from the network and delivers it to the processor.

Many commercial and research multiprocessor systems based on the Boolean  $n$ -cube interconnection network have been built since 1980s. A survey of the such systems can be found in [HM89]. The 64-node Cosmic Cube was the first such machine and was built at Caltech; it became operational in 1983. Each node of this machine is an Intel 8086/8087 microprocessor. The Cosmic Cube has been applied to a variety of computational tasks, and significant speedups have been achievable compared to conventional computers of similar cost.

The Connection Machine [Hil85] is another good example of a machine which exploits the properties of the Boolean  $n$ -cube network. The Connection Machine CM-2 [TMC87] consists of 64K processors, each associated with 4K bits of memory and a simple serial arithmetic logic unit. The processors are interconnected in a packet-switched Boolean 12-cube network, where 16 processors are attached to each node in the Boolean 12-cube network as its local processors. Other examples of Boolean  $n$ -cube network-based multiprocessors include the NCUBE/ten [NC85], the Intel iPSC Hypercube [Int85], and the Caltech Mark Series [PTLP85].

Of course, the Boolean  $n$ -cube network has its own drawbacks. For example, to exploit the rich connections of the network, a node must be equipped with the capability of sending multiple messages simultaneously. Moreover, as the

dimension of a Boolean  $n$ -cube network increases, the cost and difficulty of designing and fabricating also increases significantly. This architectural limitation on its dimension has become the most serious drawback of the Boolean  $n$ -cube network. In Chapter 4, we will discuss the possibility of building a machine with a network of small dimensions.

### 1.2.1 Topological Properties

Topological properties of the Boolean  $n$ -cube network which are important to the discussion of this dissertation are briefly discussed in this section. A Boolean  $n$ -cube can be recursively constructed with lower dimensional cubes. For example, consider two Boolean  $(n - 1)$ -cubes whose nodes are addressed from 0 to  $2^{n-1}$ . By connecting every node of the first  $(n - 1)$ -cube to the node of the second with the same address, and adding  $2^n$  to the address of each node in the second  $(n - 1)$ -cube, we can construct a Boolean  $n$ -cube network. A Boolean  $n$ -cube network can be separated into two  $(n - 1)$ -cube networks by removing all of the links in a particular dimension.

A subcube in a Boolean  $n$ -cube network is uniquely identified by a string of  $n$  symbols in the ternary symbol set  $\{0, 1, X\}$ , where  $X$  is a *don't-care* symbol. For example, in Figure 1.2, the 4-node subcube containing nodes 0010, 0011, 0110, and 0111 is addressed as  $0X1X$ . We note that a node itself is essentially a subcube where each of the  $n$  symbols is either 0 or 1. The Hamming distance between two subcubes with addresses  $a = a_{n-1}a_{n-2}\dots a_1a_0$  and  $b = b_{n-1}b_{n-2}\dots b_1b_0$  is defined as follows:

$$D(a, b) = \sum_{i=0}^{n-1} d(a_i, b_i),$$

where

$$d(a_i, b_i) = \begin{cases} 1 & \text{if } a_i = 0 \text{ and } b_i = 1, \text{ or } a_i = 1 \text{ and } b_i = 0 \\ 0 & \text{otherwise.} \end{cases}$$

In a direct network, communication between any two non-neighboring nodes must take place via message passing through intermediate nodes. The path from one node to another can be represented by a sequence of such intermediate nodes. The length of a path is equal to the number of links on the path. We say a message is transmitted over an “optimal path” if the length of the path is equal to the Hamming distance between the message’s source node and destination node. In a Boolean  $n$ -cube network, messages are normally routed to their destination nodes via an optimal path. However, in order to avoid congestion, a message can be routed through a longer path. Also, in Chapter 5, we will see that in a damaged network where some nodes or links are down, a node might not be able to communicate with another via an optimal path.

### 1.2.2 Routing Procedures

The key function of an interconnection network is to effectively route messages among nodes. Many routing schemes for computer networks have been extensively studied in the past decade. Most of them have been developed for long-haul communication networks. A typical routing algorithm was the one used in the ARPANET [Kle76]. Those routing algorithms are usually complicated due to the irregular topologies of the underlying networks. Moreover, most of these algorithms require each node to maintain a routing table whose size is proportional to the number of nodes in the network. These algorithms



```
When a message is received,  
if (header = 0)  
    Send the message to the local processor.  
else  
    Select a 1-bit from the header.  
    Change the selected 1-bit to 0.  
    Send the message over the corresponding channel.
```

Figure 1.3: Basic routing algorithm for Boolean  $n$ -cube networks.

are impractical for large-scale interconnection networks; one solution to this problem is to use hierarchical routing [KK77].

The routing algorithm for an interconnection network should take advantage of the network's topological regularity. Very simple routing algorithms for the Boolean  $n$ -cube network can be found, for example, in [SB77] and [Lan82]. The header (address portion) of an input message is computed as the exclusive-OR of the message's source and destination addresses. Every one-bit in the header corresponds to a "valid" channel (or valid dimension) over which the message can be sent one hop closer to its destination. When a message is sent over a valid channel, the corresponding bit in the header is changed to zero. A message reaches its destination when its header contains all zeroes. A simple routing algorithm is shown in Figure 1.3. The selection of the valid channels can be deterministic, random, or adaptive to traffic. Note that no matter which criterion is used to select valid channels, all messages are routed to their destinations via an optimal path.

The e-cube routing algorithm [Lan82] is a deterministic algorithm used in

some commercial systems such as Intel iPSC Hypercube. In this algorithm, a message is always sent over the channel corresponding to the least significant non-zero bit in its header. Thus, every message can only be transmitted along a single path from its source to its destination. The e-cube routing algorithm has been proven to be deadlock-free. However, this routing scheme does not exploit the property of multiple paths from one node to another.

Adaptive algorithms make their routing decisions based on the measurement of current traffic. It is not surprising that adaptive routing algorithms perform better than nonadaptive ones in a dynamic network environment [CBN81], [FK71]. A simple adaptive routing algorithm for Boolean  $n$ -cube networks is always to send a message over the valid channel with the lightest traffic load in hopes of balancing the traffic. However, this scheme is more suitable for asynchronous networks where messages arrive at a node at different times, and the routing decision is made asynchronously for each message. In synchronous networks where messages arrive at nodes at the same time, if the routing decisions are made simultaneously for all messages, many of these messages can end up being assigned to the channel with the shortest queue, making it a much larger queue. As a result, the traffic may still be unbalanced. An even simpler routing algorithm is to randomly select a valid channel for each message. Although this random routing may not be able to deliver optimal performance, the routing decisions can be made in a very simple way, and, hence, can be made in a very short time.

Errors do occur during transmission. Transmission error detection and correction can be done at many levels. For an interconnection network which is usually operated in a well protected environment, the transmission error rate

across a link between two nodes is expected to be extremely small. In fact, the transmission error rate in an interconnection network is about the same as the memory fetch error rate in a processor. With this extremely small error rate, we can leave the error detection and correction to the destination processors. This is commonly referred as the end-to-end error detection and correction. Moreover, given that no messages can be lost due to a full buffer and the network is synchronous, a node can simply keep pumping its messages into its outgoing channels without waiting for the acknowledgements from the nodes on the other ends of the links. In this dissertation, we assume that the communication error rate is negligible.

### 1.2.3 Switching Techniques

There are several principal switching techniques: circuit switching, message switching, and some hybrid switching techniques such as virtual cut-through switching [KK79]. In circuit switching, an end-to-end physical path is set up before any data is transmitted. Circuit switching eliminates the need for buffering in the intermediate nodes of a multi-hop path. However, circuit switching is considered a poor choice for an environment where moderate to heavy bursts of traffic are likely to occur and where messages are short.

In message switching, no physical path between the source and the destination is established in advance. Messages are transmitted in a *store-and-forward* manner, where each message must be completely received and buffered in an intermediate node before it can be transmitted to the next node. For a high speed network, in order to simplify the operation of the network and improve the performance of the network, we require all messages to be of a fixed length.

One may further divide a message into several packets and then route these packet individually; this scheme is known as packet switching.

Another switching technique is virtual cut-through [KK79]. In virtual cut-through, whenever a message's header is received by an intermediate node and its selected output channel is free, the message is transmitted to the next node before it is completely received in this node. Thus, unnecessary buffering in front of an idle channel can be avoided. A similar technique called *wormhole routing* is described in [Sei85b]. Wormhole routing differs from virtual cut-through in that, if the header of a message is blocked, the message stops advancing and blocks the progress of any other message requiring the channels it occupies.

The detailed comparison of switching techniques is beyond our discussion. Instead, we are more interested in the effect of limited buffer space on performance. We assume message switching in this dissertation.

### 1.3 Design Consideration

In this section, we discuss some issues which are important to the design of an interconnection network.

#### 1.3.1 Simplicity

With current VLSI technology, the channel speed of an interconnection network is approaching one gigabit per second [AS88]. At this high speed, the amount of time that the routing algorithm can afford to spend in making routing decisions is severely constrained. By taking advantage of the topological regularity of the

interconnection network, we should simplify the routing algorithms so that they can be implemented by hardware.

### 1.3.2 Deadlock-free Routing in Finite-buffered Networks

Nodes are hardware limited to finite buffer space in the real world. Routing algorithms must be carefully designed in a finite-buffered network in order to be deadlock-free. One way deadlocks can occur in a network is when each node in a cycle is trying to send a message to its neighbor, but no node has any buffers available to accept messages. No message can advance toward its destination; thus the cycle is deadlocked. This locked state cannot be resolved until exceptional action is taken.

Several deadlock prevention techniques based on the concepts of removing or avoiding the cycles of channel dependency have been developed. One well-known technique is the *structured buffer pool* [MS80], which is used in store-and-forward networks where a directed graph is constructed and all messages are moved from one buffer to another along the arcs of the graph. The other technique known as *virtual channels*, is used with wormhole routing [DS87]. This scheme splits physical channels on a cycle into multiple virtual channels and then restricts the routing such that the dependency between the virtual channels is acyclic. These algorithms are too complicated to be used in a very high speed interconnection network because the time spent in selecting paths for forwarding messages is prohibitive.

In this dissertation, we consider a simple deadlock-free routing algorithm which exploits the homogeneity of the network and applies a technique called deflection [Max85]. In this routing scheme, when a node has no buffers available

on any of the preferred outgoing channels, messages are forced to take alternate paths which are usually longer than the preferred ones.

### 1.3.3 Fault-tolerant Routing in Damaged Networks

The issues involved in building a reliable multiprocessor system are complicated and require much effort from the lowest hardware level up to the highest user application level. Building a reliable interconnection network which provides high performance and fault-tolerant interprocessor routing in the presence of node and/or link failures is obviously an important step to achieve this goal.

We note that the success of the simple routing algorithm in Boolean  $n$ -cube networks is based on the networks' regularity property. As processors or links fail, the regularity of the network is destroyed and the original routing algorithm may no longer be applicable. There are several approaches to ensure successful routing between any pair of surviving nodes. To make our routing algorithms as simple as possible, we choose to partially restore the regularity of a damaged Boolean  $n$ -cube network so that the original routing algorithm for nonfaulty Boolean  $n$ -cube networks can still be used with only a minor change.

### 1.3.4 Performance Analysis and Improvements

Results regarding the communication behavior of Boolean  $n$ -cube networks have been reported in the literature. However, some authors report the performance values obtained from experiments or simulations, but do not form mathematical models. Other authors develop queueing models [BA84], but do not consider the effect of buffer size on performance. In a real interconnection network, nodes can be equipped with very few buffers for handling messages. Thus, a

model simply assuming infinite buffers in a node cannot accurately reflect the communication behavior of interconnection networks. Abraham and Padmanabhan [AP89] developed a model for finite-buffered Boolean  $n$ -cube networks. In this work, however, messages can be dropped from the network if they are assigned to a full outgoing channel. Thus, their models are based on a loss system, which is not desirable for high performance interconnection networks.

In this dissertation, we develop mathematical models for evaluating deflection routing in a Boolean  $n$ -cube network with finite buffers. We study the effect of buffer size on performance and try to find the optimal buffer size assignment. We also extend our model to analyze deflection routing in a very general class of networks called  $k$ -ary  $n$ -cube networks. A  $k$ -ary  $n$ -cube network has  $n$  dimensions with  $k$  nodes in each dimension. Given  $N$  nodes in the network, the relationship  $N = k^n$  must hold between the dimension  $n$  and the radix  $k$ . Examples of  $k$ -ary  $n$ -cube networks include rings, tori, and Boolean  $n$ -cube networks. We also study the degradation of the performance of a damaged Boolean  $n$ -cube network.

## 1.4 Summary of Dissertation

We now give a summary of this dissertation. In Chapter 2, we present several mathematical models for evaluating the performance of Boolean  $n$ -cube networks. We consider both the cases where all the outgoing channels share a buffer pool and where every channel has its own buffers. Analytic bounds on the performance are also given. In this chapter, the buffer size is assumed to be unlimited.

In Chapter 3, we develop a deadlock-free routing algorithm by using the

deflection technique. With this algorithm, all messages entering the network are delivered to their destinations without deadlocks. We analyze the effect of the buffer size on performance. We show that the throughput of the network never degrades. We use the function, *power* (the ratio of the throughput of the network over the mean delay of messages in the network) as our objective function. We say that the performance of a network is optimized when its power is maximized. We show that, for a wide range of input rates, the performance of a Boolean  $n$ -cube network is near optimal if each node is equipped with “ $2n$ ” buffers.

We further analyze our deadlock-free routing algorithm in a very general class of networks called  $k$ -ary  $n$ -cubes [SB77] in Chapter 4. We show that in an  $n$ -dimensional network, “ $2n$ ” buffers are essential to optimize the performance. We also examine the effect of different combinations of dimension  $n$  and radix  $k$  on performance.

Chapter 5 presents a set of algorithms to restore the regularity of a Boolean  $n$ -cube network in the presence of node and/or link failures. We first construct a *1-degraded* subnet for a damaged Boolean  $n$ -cube network by disabling certain nonfaulty nodes of the network in a distributed manner. An optimal-path routing algorithm for such degraded networks is then developed. Though the construction of 1-degraded subnets is very simple, a number of nodes can be disabled. We then construct a *convex* subnet where every pair of surviving nodes is connected by at least one optimal path. We show that very few nodes are disabled in constructing convex subnets. We also develop a two-level hierarchical fault-tolerant routing scheme without disabling a single node. With this approach, a damaged network is decomposed into a set of clusters; each clus-



ter is essentially a convex subcube. Routing is also separated into two levels. Each node maintains a small routing table. Messages are first routed to their destination clusters by the table. After a message has arrived at its destination cluster, it is further routed to its destination node using the optimal-path routing algorithm we have developed for 1-degraded subnets. We show that the increase in the mean path length caused by the hierarchical routing is very small.

We conclude this dissertation in Chapter 6 and suggest some potential areas for future research.

## CHAPTER 2

### Basic Models for Boolean $n$ -Cube Networks

#### 2.1 Introduction

This chapter presents mathematical models for predicting the performance of the Boolean  $n$ -cube networks where the buffer size of each node is assumed to be unlimited (or infinite). Routing algorithms in finite-buffered Boolean  $n$ -cube networks are analyzed in the next chapter.

We first assume that every outgoing channel of a node has its own buffer. Every message which needs further transmission is randomly assigned to a valid outgoing channel. We solve for the mean delay of a message in the network. We next develop an approximation model in which every node is assumed to have a shared buffer for its all outgoing channels. We show that the shared-buffered model performs better than the separate-buffered model. An optimistic lower bound on the mean delay is also given.

#### 2.2 Assumptions of the Network Operation

The necessary assumptions about the operation of the network are presented in this section. We assume that messages are of a fixed length and that the time of the network is divided into cycles (or slots) with a duration which

corresponds to the transmission time of a message from one node to its neighbor. The processing time of a node to make its routing decisions is assumed to be negligible.

We further assume that nodes are synchronized in transmitting their messages in each cycle. In fact, it is possible to operate a tightly-coupled interconnection network with a systemwide clock. For the network where a single clock is difficult to manage, the synchronization can be achieved as follows. At the beginning of a cycle, each node makes its routing decisions by assigning some messages to its outgoing channels. When a node finishes its routing decisions, it is “ready” to exchange its messages with its neighbors. The node then informs its neighbors by sending them a READY signal. When a “ready” node receives a READY signal from a neighbor, it starts transmitting the message which has been assigned to the corresponding outgoing channel to the neighbor. At the same time, the node begins receiving a message from the neighbor. If a node has no message to send to a neighbor, the node can just send a signal to the neighbor. When a node finishes exchanging messages with its all neighbors, it admits some input messages from its local processors and then continues to the next cycle. (The reason why input messages can only be admitted to a node at the end of a cycle will become clear in the next chapter when we discuss the deflection routing in the finite-buffered networks.) Another distributed synchronization technique for loosely-coupled systems can also be found in [Max88]. However, the detailed implementation of the synchronization is beyond the discussion of this dissertation. In the following discussion, we simply assume that nodes are synchronized in each cycle.

Here, we have assumed that a node is capable of sending multiple messages

along its  $n$  outgoing channels simultaneously. Clearly, without this capability, the rich connections and the high bandwidth of a Boolean  $n$ -cube network cannot be fully exploited. For simplifying the analysis, we further assume that when a message arrives at its destination node, it is delivered to the local processor immediately. Only the messages which need further transmission are stored in the buffer. We call these messages the *transit* messages.

Many routing algorithms can be used to decide which message is to be transmitted on each channel in a cycle. In this dissertation, instead of using complicated algorithms, we choose very simple algorithms which randomly assign messages to their valid outgoing channels. The performance of these algorithms is also compared with a bound.

## 2.3 Model 1: A Buffer at Each Outgoing Channel

### 2.3.1 Assumptions of the Model

This model assumes that every outgoing channel has a separate buffer, as shown in Figure 2.1. Each message received from a neighboring node or from a local processor is randomly assigned to a valid outgoing channel. Each outgoing channel can only transmit one message in a cycle. We assume that every outgoing channel is statistically identical and modeled as a bulk-arrival and single-server system. We further assume that there exists an equilibrium state for each outgoing channel. (Clearly, one should not use this assumption in the case of unbalanced traffic.)

For purposes of analysis, we assume that a message's destination is uniformly distributed over the network and that a local processor does not inject a message

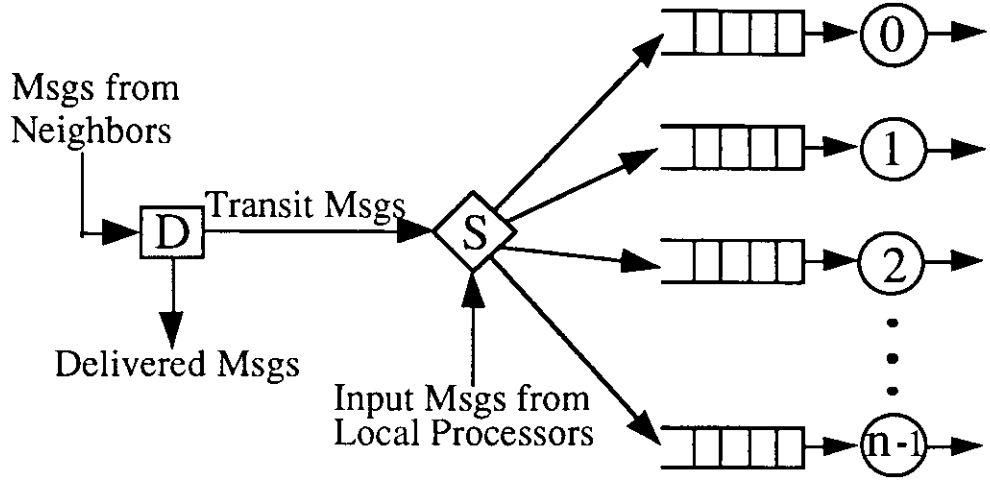


Figure 2.1: Model 1: A queue at each outgoing channel.

into the node if the message is destined for another processor of the same node. Let  $d_i$  be the probability of an input message being  $i$  hops away from its destination. Thus,

$$d_i = \binom{n}{i} / (2^n - 1) \quad \text{for } i = 1, 2, \dots, n. \quad (2.1)$$

The expected number of hops a message needs to travel in the network is easily calculated as  $\bar{d} = \sum_{i=1}^n i d_i = n 2^{n-1} / (2^n - 1)$ , which is equal to the mean number of one-bits in the header of an input message. We note that  $\bar{d} > n/2$ , but approaches  $n/2$  when  $n$  is large. It is not difficult to extend our model to include the effect of communication locality (where messages are more likely to be transmitted to nearby nodes) as long as the traffic is uniformly loaded on all directions.

We further assume that the arrival of input messages from local processors follows a geometric distribution with a generation rate of  $\lambda$  messages per cycle.

Let  $g_i$  be the probability of a node receiving  $i$  input messages from its local processors in a cycle. We have

$$g_i = (1 - \alpha) \alpha^i. \quad (2.2)$$

The generation rate of the geometric distribution is given by

$$\begin{aligned} \lambda &= \sum_{i=0}^{\infty} g_i i \\ &= (1 - \alpha) \alpha^i i \\ &= \frac{\alpha}{1 - \alpha} \end{aligned}$$

That is,  $\alpha = \lambda/(1 + \lambda)$ . We require that  $0 \leq \alpha < 1$ . Clearly, the communication load of the network is determined by the set of probabilities  $d_i$ 's and  $\lambda$ .

### 2.3.2 Analysis of the Model

Let us first determine the number of messages a node receives from its neighbors in a cycle. Let  $P_e$  be the probability that a message which is received from a neighbor will exit the network at this node. Since, on the average, a message will exit from the network after moving  $\bar{d}$  hops, we have

$$P_e = 1/\bar{d}. \quad (2.3)$$

We further let  $P_c$  be the probability that a message received from a neighbor needs further transmission. Clearly,

$$\begin{aligned} P_c &= 1 - P_e \\ &= 1 - 1/\bar{d}. \end{aligned} \quad (2.4)$$

From the description of the system, it follows that every channel has the same probability of sending a message in every cycle. We define

$$\rho \triangleq \text{Channel Utilization}$$

$$= \text{Prob}[A \text{ channel transmits a message in a cycle.}] \quad (2.5)$$

Thus, on the average a node delivers  $n\rho P_e$  messages to its local processors in a cycle. In equilibrium the network flow must be conserved in the sense that the input flow equals the output flow. Thus,

$$N\lambda = Nn\rho P_e \quad (2.6)$$

where  $N$  is the number of nodes in the network. Replacing  $P_e$  by  $1/\bar{d}$ , we obtain

$$\rho = \frac{\lambda}{n} \bar{d}. \quad (2.7)$$

To insure stability, we require  $\rho < 1$ . In the case where messages are uniformly distributed over the network, since  $\bar{d} > n/2$ , we require that  $\lambda < 2$ .

Recall that only transit messages are stored in the buffers. Let  $P_t$  be the probability of a node receiving a transit message from an incoming channel in a cycle. We have

$$\begin{aligned} P_t &= \rho P_e \\ &= \frac{\lambda}{n} (\bar{d} - 1). \end{aligned} \quad (2.8)$$

Let  $r_i$  be the probability of a node receiving exactly  $i$  transit messages in a cycle. Since every channel is assumed to be statistically identical, the arrival of transit messages at a node follows a binomial distribution, which is given by

$$r_i = \begin{cases} \binom{n}{i} P_t^i (1 - P_t)^{n-i} & \text{for } i = 0, 1, 2, \dots, n \\ 0 & \text{otherwise.} \end{cases} \quad (2.9)$$

We further define  $a_k$  as the probability of a node receiving exactly  $k$  messages (transit and input) in a cycle. Thus,

$$a_k = \sum_{i=0}^k r_i g_{k-i} \quad \text{for } k \geq 0. \quad (2.10)$$

Define  $G(z)$ ,  $R(z)$ , and  $A(z)$  as the  $z$ -transforms for probability mass functions  $g_i$ ,  $r_i$ , and  $a_i$ , respectively. We have the following equations:

$$G(z) \triangleq \sum_{i=0}^{\infty} g_i z^i = \frac{1 - \alpha}{1 - \alpha z} \quad (2.11)$$

$$R(z) \triangleq \sum_{i=0}^{\infty} r_i z^i = (1 - P_t + P_t z)^n. \quad (2.12)$$

Also, we have

$$\begin{aligned} A(z) &\triangleq \sum_{i=0}^{\infty} a_i z^i \\ &= \sum_{k=0}^{\infty} \sum_{i=0}^k r_i g_{k-i} z^k \\ &= \sum_{k=0}^{\infty} \sum_{i=0}^k (r_i z^i) (g_{k-i} z^{k-i}) \\ &= \left( \sum_{i=0}^{\infty} r_i z^i \right) \left( \sum_{k=i}^{\infty} g_{k-i} z^{k-i} \right) \\ &= R(z) G(z). \end{aligned} \quad (2.13)$$

For any  $z$ -transform,  $X(z)$ , we define  $X^{(1)}(1)$  and  $X^{(2)}(1)$  as the first and the second derivatives evaluated at  $z = 1$ . After performing some algebraic manipulation, we obtain the following equations:

$$A^{(1)}(1) = \lambda \bar{d} \quad (2.14)$$

$$A^{(2)}(1) = \lambda^2 \left[ 2\bar{d} + \left( 1 - \frac{1}{n} \right) (\bar{d} - 1)^2 \right]. \quad (2.15)$$

We now determine the number of messages assigned to an outgoing channel per cycle. Let  $b_j$  be the probability of  $j$  messages being assigned to an outgoing



channel in a cycle. Assuming that outgoing channels are uniformly loaded, we immediately have

$$b_j = \sum_{i=j}^{\infty} a_i \binom{i}{j} \left(\frac{1}{n}\right)^j \left(1 - \frac{1}{n}\right)^{i-j}. \quad (2.16)$$

The  $z$ -transform for  $b_j$  is given as follows:

$$\begin{aligned} B(z) &= \sum_{j=0}^{\infty} b_j z^j \\ &= \sum_{j=0}^{\infty} \sum_{i=j}^{\infty} a_i \binom{i}{j} \left(\frac{1}{n}\right)^j \left(1 - \frac{1}{n}\right)^{i-j} z^j \\ &= \sum_{i=0}^{\infty} a_i \sum_{j=0}^i \binom{i}{j} \left(\frac{z}{n}\right)^j \left(1 - \frac{1}{n}\right)^{i-j} \\ &= \sum_{i=0}^{\infty} a_i \left(1 - \frac{1}{n} + \frac{z}{n}\right)^i \\ &= A \left(1 - \frac{1}{n} + \frac{z}{n}\right). \end{aligned} \quad (2.17)$$

The first and the second derivatives of  $B(z)$  evaluated at  $z = 1$  are then given by

$$B^{(1)}(1) = \frac{\lambda}{n} \bar{d} \quad (2.18)$$

$$B^{(2)}(1) = \frac{\lambda^2}{n^2} \left[ 2\bar{d} + \left(1 - \frac{1}{n}\right) (\bar{d} - 1)^2 \right]. \quad (2.19)$$

We now calculate the mean queue length of an outgoing channel. Below we essentially repeat the material from Kleinrock's Queueing Systems, Volume I: Theory [Kle75], in which the author derives the Pollaczek-Khinchin formula for the mean queue length of the M/G/1 system. We define the following two random variables:

$q_k \triangleq$  number of messages queueing at an outgoing  
channel at the beginning of cycle  $k$

$v_k \triangleq$  number of messages arriving at the outgoing  
channel during the cycle  $k$

Clearly, since an outgoing channel can only transmit a message in a cycle, we have

$$q_{k+1} = \begin{cases} q_k - 1 + v_k & \text{for } q_k > 0 \\ v_k & \text{for } q_k = 0. \end{cases} \quad (2.20)$$

We define  $\Delta_k$ , the shifted discrete step function, as follows:

$$\Delta_k \triangleq \begin{cases} 1 & \text{for } k = 1, 2, \dots, n \\ 0 & \text{for } k \leq 0. \end{cases} \quad (2.21)$$

Applying this definition to Equation (2.20), we may rewrite the equation for  $q_{k+1}$  as follows:

$$q_{k+1} = q_k - \Delta_k + v_k \quad (2.22)$$

We are concerned with the limiting distribution for the random variable  $q_k$  as  $k \rightarrow \infty$ , which is denoted by  $\tilde{q}$ . We assume that the  $j$ th moment of  $q_k$  exists in the limit. That is,

$$\lim_{k \rightarrow \infty} E [q_k^j] = E [\tilde{q}^j]. \quad (2.23)$$

We are interested in finding,  $E[\tilde{q}]$ , the mean value for  $\tilde{q}$ . We hope that by forming the expectation of both sides of Equation (2.22) and taking the limit as  $k \rightarrow \infty$  will yield the mean value. Following the description, we have

$$E [q_{k+1}] = E [q_k] - E [\Delta_{q_k}] + E [v_k].$$

In the limit as  $k \rightarrow \infty$ , we have

$$E[\tilde{q}] = E[\tilde{q}] - E[\Delta_{\tilde{q}}] + E[\tilde{v}].$$

The expectation we were seeking drops out of this equation. However, we obtain

$$E[\Delta_{\tilde{q}}] = E[\tilde{v}]. \quad (2.24)$$

By definition, the left-hand side can be written as

$$\begin{aligned} E[\Delta_{\tilde{q}}] &= \sum_{k=0}^{\infty} \Delta_k \text{Prob}[\tilde{q}=k] \\ &= \Delta_0 \text{Prob}[\tilde{q}=0] + \Delta_1 \text{Prob}[\tilde{q}=1] + \dots \\ &= \text{Prob}[\tilde{q} > 0] \\ &= \text{Prob}[\text{A channel is busy.}] \end{aligned}$$

Recalling that  $\rho$  is defined as the probability that a channel is busy in transmitting a message in a cycle, we have

$$E[\Delta_{\tilde{q}}] = \rho. \quad (2.25)$$

Thus,

$$E[\tilde{v}] = \rho \quad (2.26)$$

which also concludes our previous calculation in Equation (2.18) since

$$\begin{aligned} E[\tilde{v}] &= \sum_{i=0}^{\infty} b_i i \\ &= B^{(1)}(1) \\ &= \frac{\lambda}{n} \bar{d} \\ &= \rho. \end{aligned}$$

We now square both sides of Equation (2.22) and then take the expectation in hopes of finding the mean queue length as follows:

$$q_{k+1}^2 = q_k^2 + \Delta_{q_k}^2 + v_k^2 - 2q_k \Delta_{q_k} + 2q_k v_k - 2\Delta_{q_k} v_k. \quad (2.27)$$

By the definition of  $\Delta_k$ , we have

$$(\Delta_{q_k})^2 = \Delta_{q_k},$$

and

$$q_k \Delta_{q_k} = q_k.$$

Taking the expectation of Equation (2.27) and letting  $k$  go to infinity, we have

$$\begin{aligned} E[\tilde{q}^2] &= E[\tilde{q}^2] + E[\Delta_{\tilde{q}}] + E[\tilde{v}^2] - 2E[\tilde{q}] + \\ &\quad 2E[\tilde{q}]E[\tilde{v}] - 2E[\Delta_{\tilde{q}}]E[\tilde{v}]. \end{aligned}$$

Making use of Equations (2.25) and (2.26) and letting  $\bar{q} = E[\tilde{q}]$ , we obtain the following equation:

$$\bar{q} = \frac{E[\tilde{v}^2] - \rho}{2(1 - \rho)} + \rho. \quad (2.28)$$

However,

$$\begin{aligned} E[\tilde{v}^2] &= \sum_{i=0}^{\infty} \text{Prob}[\tilde{v}=i] i^2 \\ &= B^{(2)}(1) + B^{(1)}(1) \\ &= \frac{\lambda^2}{n^2} \left[ 2\bar{d} + \left(1 - \frac{1}{n}\right) (\bar{d} - 1)^2 \right] + \frac{\lambda \bar{d}}{n} \end{aligned} \quad (2.29)$$

We finally have the mean queue length at an outgoing channel as follows:

$$\bar{q} = \frac{\frac{\lambda^2}{n^2} \left[ 2\bar{d} + \left(1 - \frac{1}{n}\right) (\bar{d} - 1)^2 \right]}{2 \left(1 - \frac{\lambda \bar{d}}{n}\right)} + \frac{\lambda \bar{d}}{n} \quad (2.30)$$

Applying Little's result [Lit61], we obtain the mean delay of a message at a given node as

$$\begin{aligned}
 T_{node} &= \frac{\bar{q}}{E[\tilde{v}]} \\
 &= \frac{\lambda \left[ 2\bar{d} + \left(1 - \frac{1}{n}\right) (\bar{d} - 1)^2 \right] / \bar{d}}{2(n - \lambda\bar{d})} + 1.
 \end{aligned} \tag{2.31}$$

On the average a packet needs to travel  $\bar{d}$  hops through the network. Thus, the mean delay is

$$T = \frac{\lambda \left[ 2\bar{d} + \left(1 - \frac{1}{n}\right) (\bar{d} - 1)^2 \right]}{2(n - \lambda\bar{d})} + \bar{d}. \tag{2.32}$$

### 2.3.3 Validation of the Model

Our model assumes that every outgoing channel is statistically identical. We validated this approximation model through simulations against several network sizes with a variety of workloads. In any given cycle, each node in the simulator generated input messages based on a geometric distribution with an input rate of  $\lambda$  messages per cycle. The simulator actually routed messages through the network by our routing algorithm, and generated statistics such as the channel utilization, the mean queue length, and the mean delay. We ran the simulations long enough for the network to be in a steady state. Figure 2.2 shows the mean delay in a Boolean 6-cube network for various input rates. Note that when  $\lambda$  approaches zero, the mean delay is essentially the mean number of hops a message needs to travel in the network. The match between the model and the simulation results is extremely good.

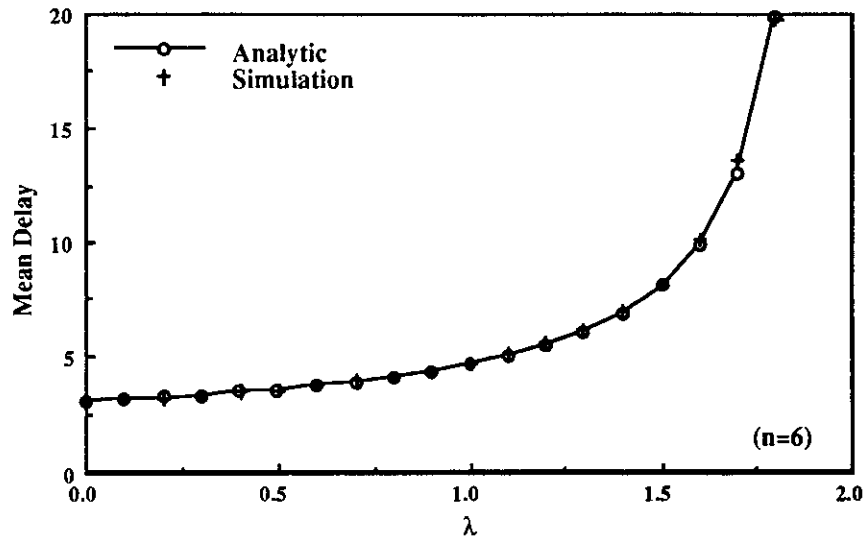


Figure 2.2: Mean delay of a Boolean 6-cube network with a separate buffer at each outgoing channel.

## 2.4 Model 2: A Shared Buffer at Each Node

### 2.4.1 Assumptions of the Model

This section presents an approximation for evaluating a routing algorithm in Boolean  $n$ -cube networks where every node is assumed to have a shared buffer of unlimited size. All messages in a node which need further transmission are stored in the shared buffer. The network is assumed to be decomposed into a set of statistically identical nodes. Considering an arbitrary node in isolation, we have a bulk-arrival and bulk-service system as shown in Figure 2.3. (Recall that each outgoing channel is able to transmit a message in each cycle.) We assume there exists an equilibrium state for the node.

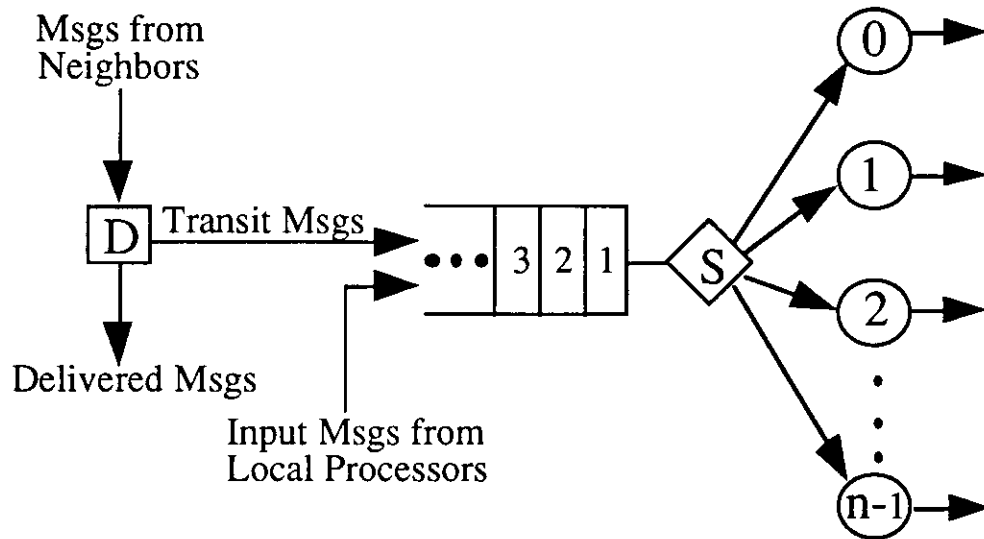


Figure 2.3: Model 2: A shared buffer for a node.

Our routing algorithm works as follows: At the beginning of a cycle, every node randomly selects a valid channel for every message the node currently has. To lessen the node processing time, we assume that the selection is done in parallel for all messages. If more than one message is selected with the same bit position, the one having the highest priority (i.e., having been in the network the longest) is successfully assigned to the corresponding outgoing channel. Unsuccessful messages are saved in the shared buffer for further assignment in the next cycle. We assume that the assignment is non-persistent. That is, unsuccessful messages will be reassigned to channels independently as if they had not been assigned before. A node begins sending the successfully assigned messages to its neighbors after the node has finished its message assignment. At the end of a cycle, the node admits some input messages from its local processors and then continues to the next cycle.

		Channel No.					
		5	4	3	2	1	0
msg. 1	→	0	1	①	0	1	0
msg. 2	→	①	0	1	0	0	0
msg. 3	→	1	0	0	0	0	①
msg. 4	→	①	1	0	0	1	0
msg. 5	→	0	0	0	0	①	1
msg. 6	→	1	1	①	0	1	0
		⋮					

○ : Random Assignment    ⊙ : Successful Trans.

Figure 2.4: An example of parallel and random message assignment.

An example of this parallel and random assignment for a node in a Boolean 6-cube network is given in Figure 2.4. The node is shown with 6 buffered messages. After the random assignment, messages 1, 2, 3 and 5 are successfully assigned to channels 3, 5, 0, and 1, respectively. Note that in this example, since no message has a one-bit in dimension 2, channel 2 must be idle in this cycle. Thus, no matter which algorithm is used, at most five messages can be successfully assigned in this cycle.

Although the random assignment is not optimal in terms of the number of messages transmitted in a cycle, the algorithm is very easy to implement. Also, since the routing decisions can be made simultaneously for all messages, the calculation is very fast. Thus, not only is the hardware cost low, but also the processing time required by a node to make the routing decisions can be



significantly reduced. In the next section, we compare the performance of this algorithm with an optimistic model. In the rest of this section, we calculate the mean message delay of this random assignment routing algorithm.

#### 2.4.2 Analysis of the Model

Let us proceed by determining the number of messages being transmitted by a node in a cycle. Let

$$f_{i,j} \triangleq \text{Prob} \left[ \begin{array}{l} \text{Given that there are } i \text{ messages in the node,} \\ j \text{ of them are transmitted in the cycle.} \end{array} \right]$$

Since one-bits in the header are assumed to be uniformly distributed and channel selection is made randomly, it follows that every outgoing channel of the node has an equal probability of being selected by a message. We assume that messages are stored in the buffer in the order of decreasing priority. Thus, the  $i$ th message is successfully assigned to an outgoing channel if and only if the channel is not chosen by any of the  $i - 1$  messages in front of it. It is not difficult to write down the following recursive equations:

$$f_{i,j} = \begin{cases} 1 & \text{if } i = j = 0 \\ 0 & \text{if } i < j \\ 0 & \text{if } j = 0 \text{ and } i = 1, 2, 3, \dots, \\ f_{i-1,j-1} \left(1 - \frac{i-1}{n}\right) + f_{i-1,j} \left(\frac{i}{n}\right) & \text{if } i = 1, 2, 3, \dots \text{ and } i \geq j \end{cases} \quad (2.33)$$

We further define the  $z$ -transform for  $f_{i,j}$  as follows:

$$F_i(z) \triangleq \sum_{j=0}^{\infty} f_{i,j} z^j \quad (2.34)$$

and

$$F_i^{(1)}(z) \triangleq \frac{dF_i(z)}{dz}. \quad (2.35)$$

Let  $\bar{f}_i$  be the mean number of messages successfully transmitted in a cycle, given that there are currently  $i$  messages in the node. Obviously,

$$\bar{f}_i = F_i^{(1)}(1). \quad (2.36)$$

From the definition of  $f_{i,j}$  in Equation (2.33), we immediately have

$$F_0(z) = 1. \quad (2.37)$$

Also, given that there is only one message in the node, the message must be successfully assigned to an outgoing channel. Thus,

$$F_1(z) = z. \quad (2.38)$$

In general, we have

$$\begin{aligned} F_i(z) &= \sum_{j=0}^{\infty} f_{i,j} z^j \\ &= \sum_{j=0}^{\infty} \left[ f_{i-1,j-1} \left(1 - \frac{j-1}{n}\right) + f_{i-1,j} \left(\frac{j}{n}\right) \right] z^j \\ &= z F_{i-1}(z) - \frac{z^2}{n} F_{i-1}^{(1)}(z) + \frac{z}{n} F_{i-1}^{(1)}(z) \\ &= z F_{i-1}(z) + \frac{z(1-z)}{n} F_{i-1}^{(1)}(z). \end{aligned} \quad (2.39)$$

Thus,

$$F_{i-1}^{(1)}(z) = n \frac{F_i(z) - z F_{i-1}(z)}{z(1-z)} \quad (2.40)$$

By letting  $z = 1$ , both the numerator and the denominator on the right-hand side of Equation (2.40) become zero. Using L'Hospital's rule, we obtain the following equation:

$$F_{i-1}^{(1)}(1) = n \frac{F_i^{(1)}(1) - F_{i-1}(1) - F_{i-1}^{(1)}(1)}{-1}. \quad (2.41)$$

This leads to the equation below:

$$\bar{f}_i = 1 + \left(1 - \frac{1}{n}\right) \bar{f}_{i-1}. \quad (2.42)$$

Solving the above recursive equation, we obtain

$$\bar{f}_i = n \left[1 - \left(1 - \frac{1}{n}\right)^i\right]. \quad (2.43)$$

Let us now define

$$\pi_i \triangleq \text{Prob} [\text{Node has } i \text{ messages at the beginning of a cycle}]. \quad (2.44)$$

Also, we define

$$\Pi(z) \triangleq \sum_{i=0}^{\infty} \pi_i z^i. \quad (2.45)$$

The mean number of messages successfully transmitted by a node per cycle is given by

$$\begin{aligned} & \sum_{i=0}^{\infty} \pi_i \bar{f}_i \\ &= n \sum_{i=0}^{\infty} \pi_i \left[1 - \left(1 - \frac{1}{n}\right)^i\right] \\ &= n - n \sum_{i=0}^{\infty} \pi_i \left(1 - \frac{1}{n}\right)^i. \end{aligned} \quad (2.46)$$

However, the channel utilization can be calculated as follows:

$$\frac{1}{n} \sum_{i=0}^{\infty} \pi_i \bar{f}_i = \rho. \quad (2.47)$$

From Equations (2.46) and (2.47), we have

$$\sum_{i=0}^{\infty} \pi_i \left(1 - \frac{1}{n}\right)^i = 1 - \rho. \quad (2.48)$$

Moreover, assuming that every outgoing channel is independent of all others and  $n$  is large enough, we approximate  $\pi_0$  as the probability that every outgoing channel of the node is idle in the cycle. That is,

$$\pi_0 = (1 - \rho)^n. \quad (2.49)$$

From Equations (2.48) and (2.49), and from the sum of probabilities, we have the following set of equations:

$$\Pi(0) = (1 - \rho)^n \quad (2.50)$$

$$\Pi\left(1 - \frac{1}{n}\right) = 1 - \rho \quad (2.51)$$

$$\Pi(1) = 1. \quad (2.52)$$

Many equations can be used to approximate  $\Pi(z)$ . Let us assume that

$$\Pi(z) = (1 - \rho)^{n-nz}. \quad (2.53)$$

Equation (2.53) is easily inverted to give

$$\pi_i = \frac{[-n \log(1 - \rho)]^i}{i!} (1 - \rho)^n \quad i=0, 1, 2, 3, \dots \quad (2.54)$$

Setting

$$x = -n \log(1 - \rho),$$

we rewrite Equation (2.54) as follows:

$$\pi_i = \frac{x^i}{i!} e^{-x} \quad i = 0, 1, 2, 3, \dots \quad (2.55)$$

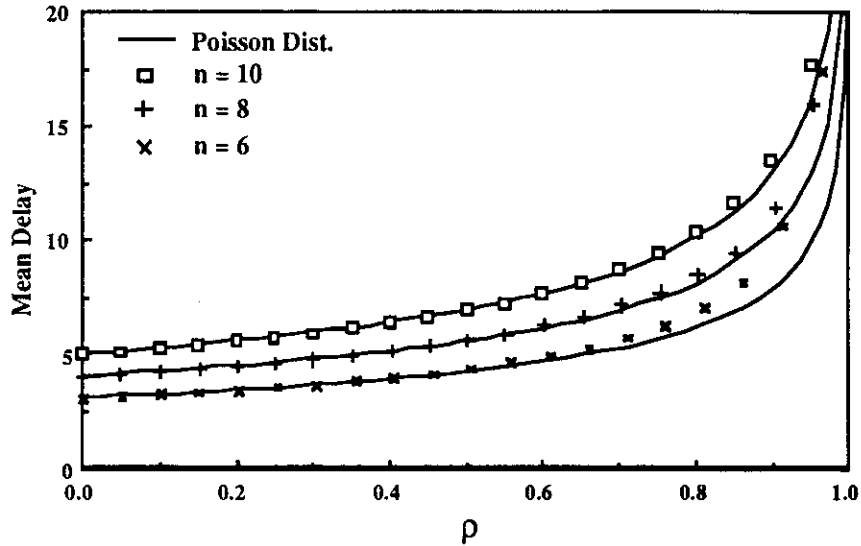


Figure 2.5: Mean delay of Boolean 6, 8, and 10-cube networks. Lines correspond to the Poisson approximation and points are taken from a simulation.

That is, the number of messages in a node is simply approximated by a Poisson distribution. Replacing  $\rho$  by  $\lambda\bar{d}/n$ , the mean queue length is given by

$$\bar{q} = -n \log \left( 1 - \frac{\lambda\bar{d}}{n} \right). \quad (2.56)$$

Applying Little's result [Lit61], we finally obtain an approximation for the mean message delay as follows:

$$T = -\frac{n}{\lambda} \log \left( 1 - \frac{\lambda\bar{d}}{n} \right). \quad (2.57)$$

### 2.4.3 Validation of the Approximation

This approximation model was also validated through simulations. Figure 2.5 shows the mean message delays of the Boolean  $n$ -cube network, where  $n =$

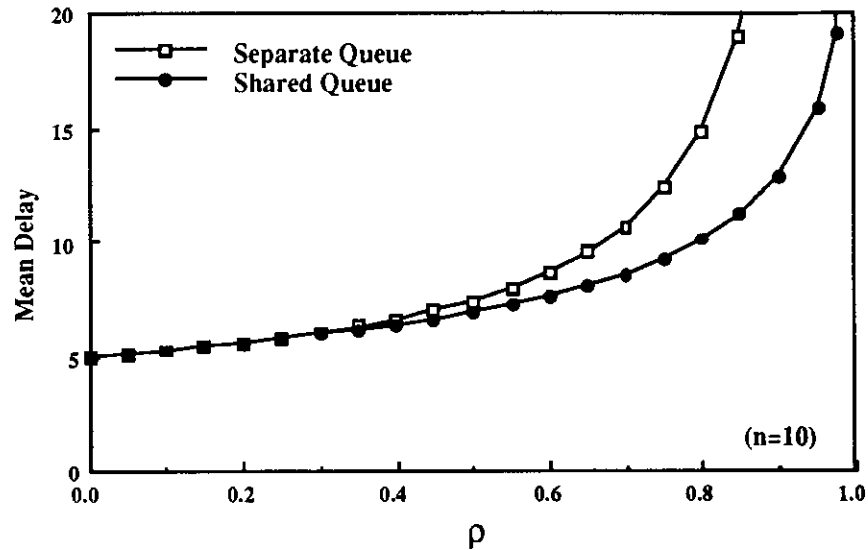


Figure 2.6: Comparing the mean delays obtained from models 1 and 2.

6, 8, 10. From this figure, we find that the results from approximation and simulation are very close to each other when  $n$  is large (e.g.,  $n$  is 6 or larger). In Figure 2.6, we compare the mean delays obtained from models 1 and 2 for a Boolean 10-cube network. We find that the shared-buffer model performs better than the separate-buffer model when the input rate is large.

## 2.5 A lower Bound on the Mean Delay

In this section, we find a lower bound on the mean message delay of any feasible routing algorithm in Boolean  $n$ -cube networks. Instead of considering any particular routing algorithm, we are here interested in the performance of an algorithm which is assumed to be able to transmit as many messages as possible

in each cycle. Obviously, the mean delay of such an algorithm serves as a lower bound.

### 2.5.1 The Optimistic Assumption

We make an optimistic assumption as follows. In each cycle, if the number of messages in a node is less than or equal to  $n$ , then all of these messages are transmitted to the node's neighbors. If the number of messages is larger than  $n$ , then exactly  $n$  of them are transmitted. We assume that all messages are forwarded to the directions leading closer to their destinations.

Note that even an exhaustive search cannot achieve this theoretic bound. As an example, let us consider the situation where there are only two messages in a node and each header has only one one-bit in the same dimension. In this case, only one of them can be transmitted in this cycle. Clearly, this lower bound is optimistic.

We rewrite  $f_{i,j}$  as follows:

$$f_{i,j} = \begin{cases} 1 & \text{if } i = j \leq n \\ 1 & \text{if } i > n \text{ and } j = n \\ 0 & \text{otherwise.} \end{cases}$$

Thus,

$$\bar{f}_i = \begin{cases} i & \text{if } i \leq n \\ n & \text{if } i \geq n. \end{cases}$$

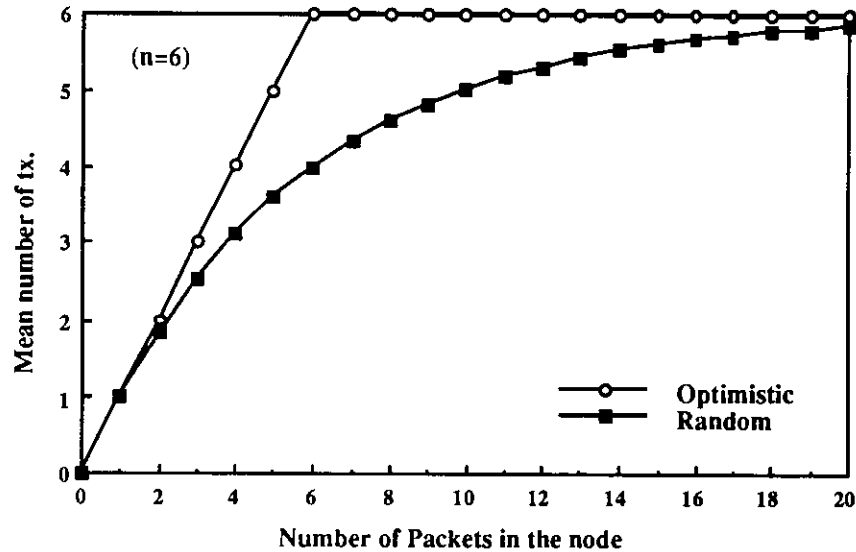


Figure 2.7: Mean number of messages successfully transmitted.

Figure 2.7 plots the mean number of messages successfully assigned in the optimistic bound model and the random assignment model. We note that, when the number of messages in a node is small, messages are unlikely to be blocked. In this case, the mean number of messages successfully transmitted in these two models is close. When the number of messages in a node is large, since the random assignment algorithm can successfully assign a large number (close to  $n$ ) of messages in a cycle, these two models are also comparable. The difference between these two models can be substantial when the mean number of messages in a node is moderate.



## 2.5.2 Analysis of the Model

We define the following random variables for a given node in a given cycle, say cycle  $k$ .

$x_k$  = number of messages at the beginning of cycle  $k$

$s_k$  = number of messages successfully transmitted in cycle  $k$

$t_k$  = number of transit messages received in cycle  $k$

$g_k$  = number of input messages received at the end of cycle  $k$

Clearly, we have the following equation:

$$x_{k+1} = x_k - s_k + t_k + g_k. \quad (2.58)$$

We now proceed to calculate the  $z$ -transform for the probability of finding  $i$  messages in a node for this optimistic model. We define the  $z$ -transform of the random variable  $x_k$  as

$$X_k(z) \triangleq \sum_{i=0}^{\infty} Prob[x_k = i] z^i. \quad (2.59)$$

By definition of expected value, the  $z$ -transform can be also written as

$$X_k(z) = E[z^{x_k}]. \quad (2.60)$$

Since the number of messages received from neighbors and the number of input messages accepted from local processors are independent of the number of messages the node currently has, we have

$$\begin{aligned} X_{k+1}(z) &= E[z^{x_{k+1}}] \\ &= E[z^{x_k - s_k + t_k + g_k}] \\ &= E[z^{x_k - s_k}] E[z^{t_k}] E[z^{g_k}]. \end{aligned} \quad (2.61)$$

We concern ourselves not with the time-dependent behavior, but rather with the limiting distribution for the random variable  $x_k$ , which we denote by  $\tilde{x}$ . Similarly, we denote  $\tilde{s}$ ,  $\tilde{t}$ , and  $\tilde{g}$  for  $s_k$ ,  $t_k$ , and  $g_k$ , respectively, as  $k \rightarrow \infty$ . Thus, we have

$$\begin{aligned} X(z) &= \lim_{k \rightarrow \infty} X_k(z) \\ &= E[z^{\tilde{x}-\tilde{s}+\tilde{t}+\tilde{g}}] \\ &= E[z^{\tilde{x}-\tilde{s}}] E[z^{\tilde{t}}] E[z^{\tilde{g}}]. \end{aligned} \quad (2.62)$$

In Section 2.3,  $P_t$  is defined as the probability of the channel receiving a transit message in a cycle and  $r_i$  is defined as the probability of a node receiving  $i$  transit messages in a cycle. In Equations (2.9), we have shown that

$$r_i = \begin{cases} \binom{n}{i} P_t^i (1 - P_t)^{n-i} & \text{for } i = 0, 1, 2, \dots, n \\ 0 & \text{otherwise} \end{cases}$$

where  $P_t = \lambda \bar{d}/n$ . Also, the arrival of input messages follows a geometric distribution. We thus have the following equation:

$$X(z) = E[z^{\tilde{x}-\tilde{s}}] (P_t z + 1 - P_t)^n \frac{1 - \alpha}{1 - \alpha z}. \quad (2.63)$$

For simplicity, we let

$$K(z) = (P_t z + 1 - P_t)^n \frac{1 - \alpha}{1 - \alpha z}. \quad (2.64)$$

We next examine the expectation on the right hand side of Equation (2.63) separately. We define

$$\pi_i \triangleq \text{Prob}[\tilde{x} = i].$$

Following the optimistic assumption and the definition of the  $z$ -transform, we have

$$\begin{aligned}
& E[z^{\tilde{x}-\tilde{s}}] \\
&= \sum_{i=0}^n \pi_i (z^{i-i}) + \sum_{i=n+1}^{\infty} \pi_i (z^{i-n}) \\
&= \sum_{i=0}^n \pi_i + \frac{1}{z^n} \sum_{i=n+1}^{\infty} \pi_i z^i \\
&= \sum_{i=0}^n \pi_i + \frac{1}{z^n} (X(z) - \sum_{i=0}^n \pi_i z^i) \\
&= \frac{1}{z^n} X(z) + \sum_{i=0}^n \pi_i (1 - \frac{z^i}{z^n}) \\
&= \frac{1}{z^n} X(z) + \frac{1}{z^n} \sum_{i=0}^{n-1} \pi_i (z^n - z^i). \tag{2.65}
\end{aligned}$$

After algebraic manipulation, we obtain

$$X(z) = \frac{\sum_{i=0}^{n-1} \pi_i (z^n - z^i)}{[z^n/K(z)] - 1}. \tag{2.66}$$

Applying Rouché's theorem [Tit39] to the expression  $(z^n/K(z) - 1)$  by taking the circle  $|z| = 1 + \delta$ , where  $\delta > 0$  is sufficiently small, we show that there are exactly  $n$  simple zeroes inside and on the closed contour  $|z| = 1$  for the denominator of Equation (2.66) and one zero in  $|z| > 1$ . (See Appendix A.)

Let  $\{z_j\}$ , where  $j = 0, 1, \dots, n-1$ , be those zeroes in and on  $|z| \leq 1$  and  $\{z_n\}$  be the one zero in  $|z| > 1$ . It is easy to see that one of the zeroes, say  $z_0$ , must be 1. From Equation (2.64), we rewrite

$$\begin{aligned}
z^n/K(z) - 1 &= \frac{z^n(1 - \alpha z) - (1 - \alpha)(P_t z + 1 - P_t)^n}{(1 - \alpha)(P_t z + 1 - P_t)^n} \\
&= \frac{-\alpha(z-1)(z-z_n) \prod_{j=1}^{n-1} (z-z_j)}{(1 - \alpha)(P_t z + 1 - P_t)^n}. \tag{2.67}
\end{aligned}$$

Since the  $z$ -transform of a probability distribution must be analytic in the region  $|z| \leq 1$ , the numerator of Equation (2.66) must vanish at  $z = z_j$ , for  $j = 0, 1, \dots, n-1$ . Thus, the numerator can be rewritten as

$$\sum_{i=0}^{n-1} \pi_i (z^n - z^i) = C (z-1) \prod_{j=1}^{n-1} (z - z_j). \quad (2.68)$$

Differentiating both sides and letting  $z = 1$ , we have

$$\sum_{i=0}^{n-1} \pi_i (n-i) = C \prod_{j=1}^{n-1} (1 - z_j) \quad (2.69)$$

where  $C$  is a constant to be evaluated below. We know that the channel utilization is

$$\begin{aligned} \rho &= \sum_{i=0}^{n-1} \pi_i \frac{i}{n} + \sum_{i=n}^{\infty} \pi_i \frac{n}{n} \\ &= 1 - \frac{1}{n} \sum_{i=0}^{n-1} \pi_i (n-i). \end{aligned}$$

From Equation (2.7), we replace  $\rho$  by  $\lambda \bar{d}/n$ . Thus

$$\sum_{i=0}^{n-1} \pi_i (n-i) = n - \lambda \bar{d}. \quad (2.70)$$

From Equations (2.69) and (2.70), we obtain

$$C = \frac{n - \lambda \bar{d}}{\prod_{j=1}^{n-1} (1 - z_j)}.$$

The numerator of Equation (2.66) can be rewritten as

$$\sum_{i=0}^{n-1} \pi_i (z^n - z^i) = (n - \lambda \bar{d}) (z-1) \prod_{j=1}^{n-1} \frac{z - z_j}{1 - z_j}. \quad (2.71)$$

From Equations (2.66), (2.67) and (2.71), we have

$$X(z) = \frac{(n - \lambda \bar{d}) (P_t z + 1 - P_t)^n}{-\lambda (z - z_n) \prod_{j=1}^{n-1} (1 - z_j)}. \quad (2.72)$$

Since  $X(z)$  is the  $z$ -transform of a probability distribution,  $X(1)$  must equal unity. Hence,

$$\frac{n - \lambda \bar{d}}{-\lambda(1 - z_n) \prod_{j=1}^{n-1} (1 - z_j)} = 1.$$

That is,

$$\prod_{j=1}^n (1 - z_j) = \frac{n - \lambda \bar{d}}{-\lambda(1 - z_n)}. \quad (2.73)$$

From Equations (2.72) and (2.73), we finally obtain the following equation:

$$X(z) = \frac{(1 - z_n)(P_t z + 1 - P_t)^n}{z - z_n} \quad (2.74)$$

where  $P_t = \lambda(\bar{d} - 1)/n$ . Recall that  $z_n$  is the only zero in  $|z| > 1$  for the expression  $z^n/K(z) - 1$ .

The mean queue length can be easily calculated as follows:

$$\begin{aligned} \bar{q} &= X^{(1)}(1) \\ &= \lambda(\bar{d} - 1) + \frac{1}{z_n - 1}. \end{aligned} \quad (2.75)$$

Applying Little's result [Lit61], we find the mean delay

$$\begin{aligned} T &= \frac{\bar{q}}{\lambda} \\ &= \bar{d} + \frac{1}{\lambda(z_n - 1)} - 1. \end{aligned} \quad (2.76)$$

### 2.5.3 Discussion: How good is the Random Assignment

Figure 2.8 presents the mean delay for the optimistic model and for the random assignment routing algorithm. We realize that, when the input rate is low, the

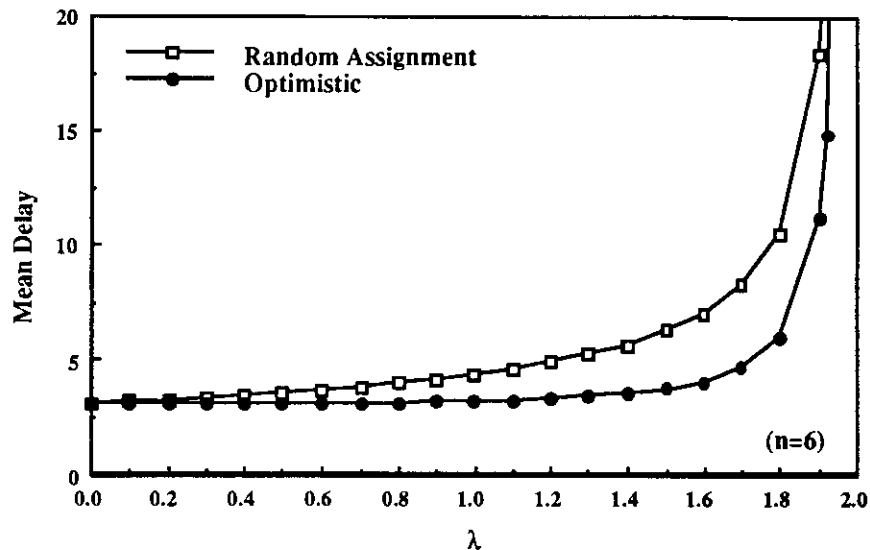


Figure 2.8: Comparing the mean delay of the random assignment algorithm with an optimistic lower bound in a Boolean 6-cube network.

queue length is very short. Messages are unlikely to be blocked. Thus, the mean delay of these two models is close. When the input rate is high, the number of messages in a node is large. As we have mentioned, the random assignment algorithm can successfully assign a large number (close to  $n$ ) of messages in a cycle. Thus, these two models are comparable. The difference between these two models can be substantial when the input rate is moderate. However, we note that, the mean delay is in terms of the number of cycles. If we consider the node processing time, due to its simplicity and fast calculation, the random assignment algorithm can perform even better in real time.

Moreover, since the random assignment algorithm makes its routing decision for all messages simultaneously without any preference, any “good” routing

algorithm should use a smaller number of cycles than does the random assignment algorithm to deliver messages. Thus, in terms of the number of cycles, the model for the random assignment algorithm can serve as an upper bound on the mean delay. We thus claim that the mean delay (in terms of cycles) of any “good” routing algorithm should be in between these two models.

## 2.6 Conclusions

In this chapter we have developed several models for evaluating the performance of Boolean  $n$ -cube networks. We showed that the shared-buffer model performs better than the separate-buffer model. We also showed that a very simple routing algorithm with random message assignment performs close to an optimistic bound. In this chapter, the buffer size is assumed to be unlimited. In the next chapter, we will present and analyze a deadlock-free routing algorithm for finite-buffered Boolean  $n$ -cube networks.

## CHAPTER 3

# Analysis of a Deadlock-free Routing Algorithm with the Deflection Technique

### 3.1 Introduction

Routing algorithms in a finite-buffered network must be carefully designed in order to avoid deadlocks. In this chapter, by exploiting the topological regularity of the network and applying a technique called deflection, we develop a very simple deadlock-free routing algorithm for Boolean  $n$ -cube networks with finite buffers. We show that with this algorithm all messages entering the network are delivered correctly to their destinations without discards or deadlocks. We also develop a mathematical model for evaluating the performance of this algorithm. We solve for the throughput of the network, the probability of acceptance of an input message, and the mean delay. We show that the throughput of the network does not degrade even when the network is full. We focus on the way the size of the buffer affects the performance. We show that the performance of a Boolean  $n$ -cube network is near optimal when each node is equipped with “ $2n$ ” buffers.



### 3.2 A Deadlock-free Routing Algorithm in Boolean $n$ -Cube Networks

Networks must be deadlock-free. One way deadlock can occur in a network is when each node in a chain is trying to send a message to its neighbor, but no node has any buffers available to accept messages. In this case, the blocked node also blocks its neighbors' outgoing channels and increases the chance of the neighbors becoming blocked. When the movement of messages comes to a halt, the system crashes. Several deadlock prevention techniques based on the concepts of removing or avoiding the cycles of channel dependency have been developed [MS80], [DS87]. However, without taking advantage of the underlying topological regularity, these algorithms can be too complicated to be used in high speed interconnection networks.

Deflection routing has been proposed for slotted networks where each node of the network has the same number of incoming and outgoing channels [Max85]. Let  $n_{in}$  be the number of incoming channels and  $n_{out}$  be the number of outgoing channels of a node. In a slotted network, a node can receive up to  $n_{in}$  messages from its neighbors in a cycle. To make sure all of the incoming messages are accepted, a node must make at least  $n_{in}$  buffers available to these messages. If  $n_{out} \geq n_{in}$ , then the node can always send enough number of messages to its neighbors such that the number of free buffers is larger than or equal to  $n_{in}$ .

We say that a message is deflected if it is sent along a direction leading farther away its destination. Clearly, the deflection technique can be applied to a Boolean  $n$ -cube network, as long as the network is synchronized (or slotted). A similar technique called "referral" is used in the Connection Machine [Hil85].

Our deadlock-free routing algorithm works as follows: We split the routing decision into two phases. The first phase is exactly the same as the random and parallel message assignment, in which every node randomly selects one of the one-bits from the header of every message currently in the node. If more than one message is selected with the same bit position, the one having the highest priority is successfully assigned to the channel. After the assignment, if a node finds that its free buffers will be less than  $n$ , it must assign some more messages in the second phase. Messages are transmitted only when the two-phase assignment has been completed. Input messages from local processors can only be admitted into a node at the end of a cycle when the node has finished exchanging messages with its neighbors. Obviously, without this restriction, messages coming from neighbors can be lost due to a full buffer.

To be more specific, we let  $M$  be the buffer size of each node. Clearly,  $M \geq n$ . Suppose a node currently has  $i$  messages and  $j$  of them are successfully assigned in the first phase. Then,  $(M - i + j)$  buffers will be free for the messages coming in from neighbors if there is no second phase. However, we wish to make available at least  $n$  buffers in case each of the node's  $n$  neighbors chooses to send it a message in the cycle. Thus, if  $(M - i + j) < n$ , the node assigns  $(n - M + i - j)$  messages in the second phase.

Since  $j$  channels have been assigned in the first phase, messages chosen in the second phase must be assigned only to the other  $(n - j)$  channels. We note that  $(n - j) \geq (n - M + i - j)$  in any case. That is, the node can always find free outgoing channels in the second phase for the assignment. We assume that in the second phase the node simply chooses  $(n - M + i - j)$  messages with the lowest priority which have failed in the first phase assignment, and assigns each

of them to a free outgoing channel. If a message is luckily assigned to a channel with the corresponding bit in its header being one, then this message will be forwarded in the direction toward its destination. Otherwise, the message will be deflected. In the Boolean  $n$ -cube network, a deflected message is sent one hop away from its destination and the corresponding bit in its header is switched from zero to one. The message then needs an extra hop to move itself back along this dimension before it arrives at its destination.

We note that the second phase assignment is essentially the hot potato algorithm proposed by Baran [Bar64] (in this algorithm, a node tries to get rid of its messages as fast as it can, by putting its messages on the shortest queue without knowing to where the messages should be forwarded).

An example of the two-phase message assignment for a node with  $M = 7$  buffers in a Boolean 6-cube network is given in Figure 3.1. The node is shown with 6 messages buffered. After the random assignment in the first phase, messages 1, 2, 3 and 5 are assigned to channels 3, 5, 0, and 1, respectively. The node must choose one more message in the second phase to provide a total of  $n = 6$  free buffers. We assume message 6 is assigned to channel 2. Thus, message 6 will be deflected in this cycle. If, instead, message 6 had been assigned to channel 4 in the second phase, then no message will be deflected.

Every message is timestamped in order to avoid livelocks [BBG87]. The timestamp contains the time when the message was created and the source node address. Messages are queued in an order based on their priority, where older messages have higher priority over younger ones. If two messages are created at the same time, then the message with lower source address has higher priority over the message with a higher source address. Since at least one of the stored

		Channel No.					
		5	4	3	2	1	0
msg. 1	→	0	1	①	0	1	0
msg. 2	→	①	0	1	0	0	0
msg. 3	→	1	0	0	0	0	①
msg. 4	→	①	1	0	0	1	0
msg. 5	→	0	0	0	0	①	1
msg. 6	→	1	1	①	①	1	0

- : Random Assignment in the first phase
- ⊙ : Successful assignment in the first phase
- : Assignment in the second phase

Figure 3.1: An example of two-phase message assignment.

messages is successfully assigned in the first phase and all messages successfully assigned in the first phase are forwarded, the message with highest priority always makes progress in every cycle. Whenever the oldest message has been delivered, another message, if there is any, becomes the oldest and proceeds without being blocked or deflected. Thus, the transmission time is bounded. A bound on evacuation time for deflection routing is discussed in [Haj91].

### 3.3 Analysis of the Deadlock-free Routing Algorithm

Boolean  $n$ -cube networks with finite buffers have been analyzed in [AP89]. However, the authors simply assumed that messages can be lost if they are

transmitted to a full node, which is not desirable. In this section we present an approximate model for evaluating the deadlock-free routing algorithm described above. We will focus on the effect of buffer size on performance.

### 3.3.1 Assumptions of the Model

We make the following assumptions as we did in Section 2.4. We assume that a message's destination is uniformly distributed over the network. The expected number of hops a message must travel in the network has been calculated as  $\bar{d} = n 2^{n-1} / (2^n - 1)$ , which approaches  $n/2$  when  $n$  is large. However, since a message can be deflected on the way to its destination, it may not be transmitted along an optimal path. (Recall that an optimal path from one node to another is the path whose length equals the Hamming distance between these two nodes.) Letting  $\bar{h}$  be the mean number of hops actually traversed by a message, we have  $\bar{h} \geq \bar{d}$ .

The arrival of input messages from local processors to a node are assumed to be based on a geometric distribution with a generation rate of  $\lambda$  messages per cycle. Let  $g_i$  be the probability that a node receives  $i$  input messages in a cycle. Then,

$$g_i = (1 - \alpha) \alpha^i, \text{ where } 0 < \alpha = \frac{\lambda}{1 + \lambda} < 1.$$

In Section 2.3, we have shown that  $\lambda < 2$  for the Boolean  $n$ -cube network with unlimited buffers. In a finite-buffered network, however, some of the input messages can be rejected when the node's buffers are full. We define  $P_A$  as the probability of an input message being accepted by the node.

To analyze the algorithm, we assume that the Boolean  $n$ -cube network is decomposed into a set of statistically identical nodes. A given node is modeled

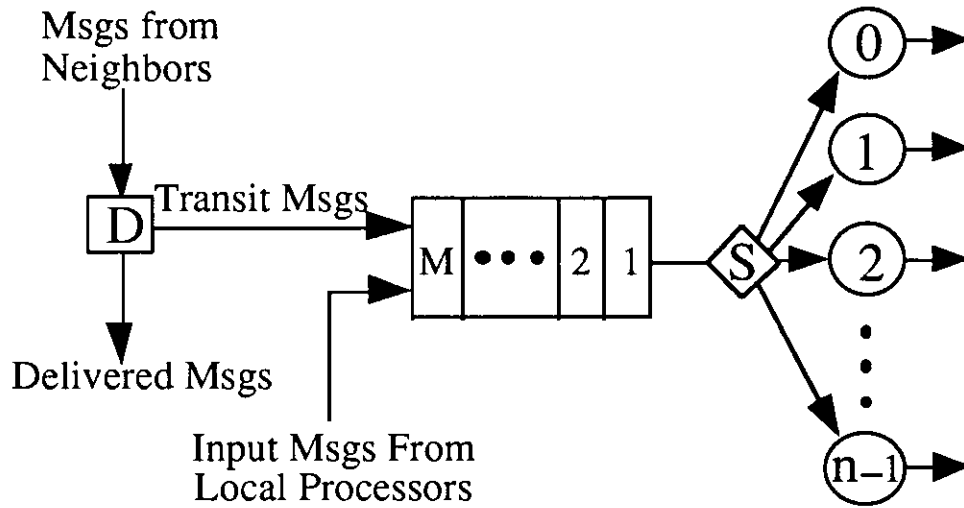


Figure 3.2: A node with finite buffers.

as a bulk-arrival and bulk-service system as shown in Figure 3.2, where  $M$  is the buffer size of the node. We assume there exists an equilibrium state for the node.

### 3.3.2 Imbedded Markov Chain Analysis

For a given node, let the sequence of random variables  $X_0, X_1, X_2, \dots$  form an imbedded Markov chain, where  $X_m$  is the number of messages the node has at the beginning of cycle  $m$ . An  $(M + 1) \times (M + 1)$  matrix  $P$ , which represents the one-cycle transition probability matrix of the node, is defined as

$$P = [p_{i,j}] ,$$

where

$$p_{i,j} = \lim_{m \rightarrow \infty} \text{Prob} [ X_m = j \mid X_{m-1} = i ] .$$

We further define the steady state probability vector  $\Pi$  as

$$\Pi = [\pi_0, \pi_1, \pi_2, \dots, \pi_M],$$

where

$$\pi_i = \lim_{m \rightarrow \infty} \text{Prob}[X_m = i] \text{ for } i = 0, 1, 2, \dots, M.$$

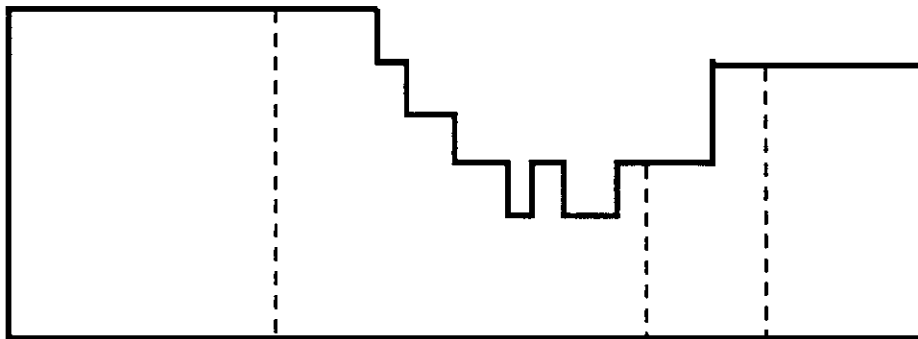
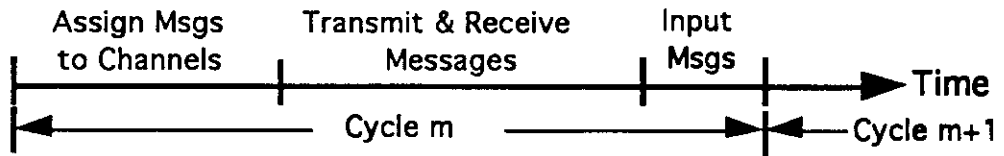
If the Markov chain is irreducible, aperiodic and recurrent nonnull, then  $\Pi$  can be uniquely determined through the following set of linear equations [Kle75]:

$$\begin{aligned} \Pi &= \Pi P \\ \sum_{i=0}^M \pi_i &= 1. \end{aligned} \tag{3.1}$$

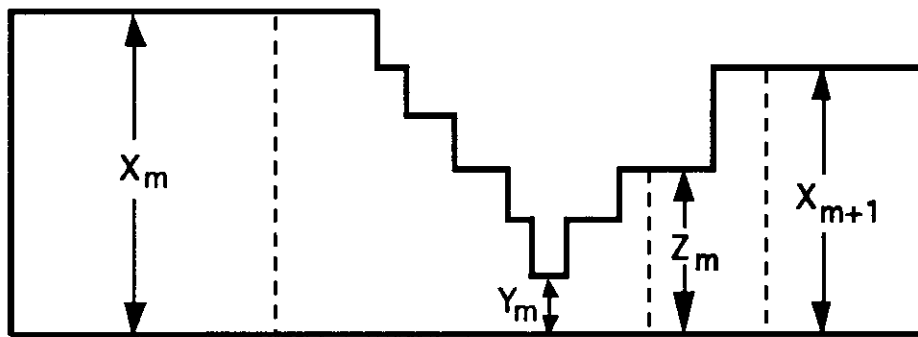
### 3.3.3 Calculating the Matrix P

The key problem of this model is to calculate the transition probabilities  $p_{i,j}$ . Based on the description of the routing algorithm, it follows that the node repeatedly makes its routing decisions, exchanges messages with its neighbors, and then admits some input messages from its local processors. Without loss of generality, we assume that in each cycle all of the transit messages are received after all of the selected messages have been transmitted. See Figure 3.3.

We let  $Y_0, Y_1, Y_2, \dots$  be a sequence of random variables, where  $Y_m$  is the number of messages the node has in cycle  $m$  after the node has transmitted all of its selected messages, but before it has received any transit messages. We also let  $Z_0, Z_1, Z_2, \dots$  be a sequence of random variables, where  $Z_m$  is the number of messages in buffers after the node has received all of its transit messages in cycle  $m$ , but has not accepted any input messages.



(a) Snapshot of queue fluctuations (real system)



(b) Snapshot of queue fluctuations (model)

Figure 3.3: Snapshot of queue fluctuations in cycle  $m$ .



Let us first determine the number of messages transmitted by the node in a cycle. Let  $f_{i,j}$  be the probability that  $j$  messages are successfully assigned to channels in the first phase, given that the node has  $i$  messages at beginning of the cycle. Since the first phase assignment is exactly same as the random and parallel assignment described in 2.4,  $f_{i,j}$  is given by Equation (2.33). In general,  $f_{i,j}$  may be written as

$$f_{i,j} = \begin{cases} \binom{n}{j} \sum_{k=0}^{j-1} (-1)^k \binom{j}{k} \left(\frac{j-k}{n}\right)^i & \text{if } \begin{cases} i \geq 1 \text{ and} \\ 1 \leq j \leq \min(i, n) \end{cases} \\ 1 & \text{if } i = j = 0 \\ 0 & \text{otherwise.} \end{cases} \quad (3.2)$$

Given that  $j$  messages are successfully assigned in the first phase, the second phase chooses  $n - M + i - j$  messages if  $i - j > M - n$ . We further let  $f'_{i,j}$  be the probability that  $j$  messages are successfully assigned to channels in both phases, given that the node has  $i$  messages at beginning of the cycle. Thus, we have

$$f'_{i,j} = \begin{cases} f_{i,j} & \text{if } i - j < M - n \\ \sum_{k=1}^{n-M+i} f_{i,k} & \text{if } i - j = M - n \\ 0 & \text{otherwise.} \end{cases} \quad (3.3)$$

We now proceed to determine the number of transit messages received in a cycle. Recall that  $P_t$  is the probability of receiving a transit message from an incoming channel in a cycle. Obviously, since input messages can be rejected

and transit messages can be deflected,  $P_t$  is not equal to  $\lambda(\bar{d} - 1)/n$ , which is given by Equation (2.8) for unlimited-buffered Boolean  $n$ -cube networks. We will solve for  $P_t$  later. However, given a  $P_t$ , the probability of a node receiving  $i$  transit messages in a cycle follows the binomial distribution; that is

$$t_i = \binom{n}{i} P_t^i (1 - P_t)^{n-i}, \quad \text{for } i = 0, 1, 2, \dots, n.$$

We further define the following three  $(M + 1) \times (M + 1)$  matrices:

$$D = [d_{i,j}],$$

where

$$d_{i,j} = \lim_{m \rightarrow \infty} \text{Prob} \{ Y_m = j \mid X_m = i \}. \quad (3.4)$$

$$R = [r_{j,k}],$$

where

$$r_{j,k} = \lim_{m \rightarrow \infty} \text{Prob} [ Z_m = k \mid Y_m = j ]. \quad (3.5)$$

$$A = [a_{k,l}],$$

where

$$a_{k,l} = \lim_{m \rightarrow \infty} \text{Prob} [ X_{m+1} = l \mid Z_m = k ]. \quad (3.6)$$

It is not difficult to obtain the following equations:

$$d_{i,j} = \begin{cases} f'_{i,i-j} & \text{if } 0 \leq j \leq i \leq M \\ 0 & \text{otherwise} \end{cases} \quad (3.7)$$

and

$$r_{j,k} = \begin{cases} t_{k-j} & \text{if } 0 \leq j \leq k \leq M \text{ and } k - j \leq n \\ 0 & \text{otherwise} \end{cases} \quad (3.8)$$

and

$$a_{k,l} = \begin{cases} (1 - \alpha) \alpha^{l-k} & \text{if } 0 \leq k \leq l \leq M - 1 \\ \alpha^{M-k} & \text{if } 0 \leq k \leq M \text{ and } l = M \\ 0 & \text{otherwise.} \end{cases} \quad (3.9)$$

Clearly, the one-cycle transition probability matrix  $P$  is given by

$$P = DRA. \quad (3.10)$$

That is, given a value for  $P_t$ ,  $P$  is determined. As a result, the probability distribution  $\pi_i$ , for  $i = 0, 1, 2, \dots, M$ , is found by solving for the set of linear equations in Equation (3.1). The remaining problem is to determine the value for  $P_t$ .

### 3.3.4 Determining the Value for $P_t$

The value for  $P_t$  must satisfy the condition that in equilibrium the network message input flow equals the message output flow. Recall that  $\rho$  is defined as the channel utilization, which is equal to the probability that a channel transmits a message in a cycle. We let  $\rho_1$  be the probability a channel is successfully assigned a message in the first phase, and  $\rho_2$  be the probability

that a channel is assigned a message in the second phase. Clearly,  $\rho = \rho_1 + \rho_2$ . It is also clear that, given the vector  $\Pi$ , we have

$$\rho = \sum_{i=1}^M \pi_i \sum_{j=1}^{\min(i,n)} f'_{i,j} \frac{j}{n}, \quad (3.11)$$

and

$$\rho_1 = \sum_{i=1}^M \pi_i \sum_{j=1}^{\min(i,n)} f_{i,j} \frac{j}{n}. \quad (3.12)$$

We further assume that every transmitted message has the same probability  $p$  of being assigned in the first phase and has the same probability  $q$  of being assigned in the second phase. Thus,

$$\begin{aligned} p &= \text{Prob} \left[ \begin{array}{l} \text{A message is assigned in the first phase,} \\ \text{given that it is transmitted in the cycle.} \end{array} \right] \\ &= \rho_1/\rho \end{aligned}$$

and

$$\begin{aligned} q &= \text{Prob} \left[ \begin{array}{l} \text{A message is assigned in the second phase,} \\ \text{given that it is transmitted in the cycle.} \end{array} \right] \\ &= \rho_2/\rho, \end{aligned}$$

where  $p + q = 1$ .

If a message is assigned in the second phase, it must be assigned to a free channel other than the one which the message has tried to assign in the first phase. If a message with  $i$  one-bits in its header is assigned to a channel in the second phase, it will be forwarded with probability  $(i - 1)/(n - 1)$  and be

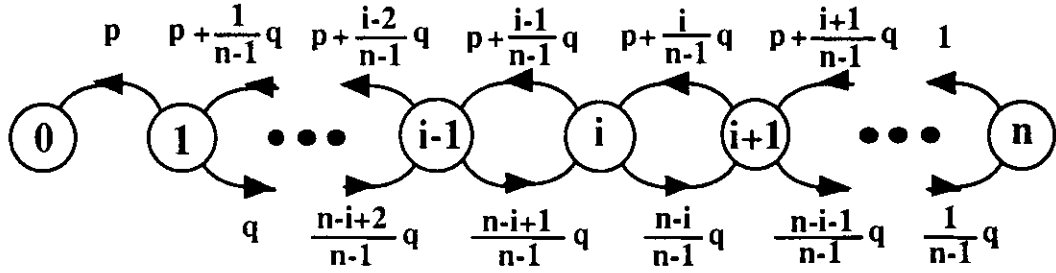


Figure 3.4: State transition diagram for the number of hops traversed by a message.

deflected with probability  $(n-i)/(n-1)$ . The state transition diagram for this is given in Figure 3.4, where the state numbers represent the number of one-bits in the header of a message and the transition labels are the probabilities of transition between states when a message is moved. State 0 is an absorbing state corresponding to the case when a message arrives at its destination. We are interested in the expected number of hops a message takes in traversing in the network.

Let  $h_i$  be the expected number of steps to move from state  $i$  to state 0. We have the following recursive relations:

$$h_i = \begin{cases} 0 & \text{if } i = 0 \\ \left( p + \frac{i-1}{n-1}q \right) h_{i-1} + \left( \frac{n-i}{n-1}q \right) h_{i+1} + 1 & \text{if } i = 2, \dots, n-1 \\ h_{n-1} + 1 & \text{if } i = n. \end{cases}$$

It can be shown that

$$h_i = \sum_{j=n+1-i}^n \delta_j, \text{ for } i = 1, 2, \dots, n \quad (3.13)$$

where

$$\delta_i = \begin{cases} 1 & \text{if } i = 1 \\ \prod_{j=1}^{i-1} \frac{j}{1-\frac{j}{n-1}q} + \frac{n-1}{q} \sum_{m=1}^{i-1} \frac{1}{m} \prod_{j=m}^{i-1} \frac{j}{1-\frac{j}{n-1}q} & \text{for } i = 2, \dots, n. \end{cases}$$

Let  $\bar{h}$  be the expected number of hops actually traversed by a message. It follows that

$$\bar{h} = \sum_{i=1}^n d_i h_i. \quad (3.14)$$

We define  $\gamma_{out}$  as the output rate per node (i.e., the mean number of messages delivered per cycle per node). On average a message is expected to exit from the network after moving  $\bar{h}$  hops. Thus, we have

$$\gamma_{out} = n\rho/\bar{h}. \quad (3.15)$$

We now prove that, given the geometric arrival process of input messages,

$$P_A = 1 - \pi_M. \quad (3.16)$$

Let

$$\Pi' = [\pi'_0, \pi'_1, \dots, \pi'_M],$$

where

$$\pi'_i = \lim_{m \rightarrow \infty} \text{Prob}[Z_m = i] \text{ for } i = 0, 1, 2, \dots, M.$$

Note that an input message is dropped only if the buffers are full. Given that there are  $j$  messages in the node, if  $(M - j)$  or less messages are generated,

then all of these messages are accepted. If more than  $(M - j)$  messages are generated, exactly  $(M - j)$  of them are accepted. Thus,

$$\pi_i = \sum_{j=0}^i \pi'_j g_{i-j} \quad \text{for } i = 0, 1, \dots, M - 1. \quad (3.17)$$

and

$$\begin{aligned} \pi_M &= \sum_{j=0}^i \pi'_j \sum_{i=M-j}^{\infty} g_i \\ &= \sum_{j=0}^M \pi'_j \alpha^{M-j}. \end{aligned} \quad (3.18)$$

Let  $c_j$  be the expected number of input messages accepted into a network node per cycle, given that there are  $j$  messages in the node just before the arrival of the input messages. We have

$$c_j = \sum_{i=0}^{M-j-1} g_i i + \sum_{i=M-j}^{\infty} g_i (M - j). \quad (3.19)$$

Recall that  $g_i$  is the probability that  $i$  input messages are generated from local processors in a cycle. However,

$$\begin{aligned} &\sum_{i=0}^{M-j-1} g_i i + \sum_{i=M-j}^{\infty} g_i (M - j) \\ &= \sum_{i=0}^{M-j-1} [(1 - \alpha)\alpha^i i] + (M - j) \sum_{i=M-j}^{\infty} [(1 - \alpha)\alpha^i] \\ &= \left[ \frac{\alpha - \alpha^{M-j}}{1 - \alpha} - (M - j - 1)\alpha^{M-j} \right] + (M - j)\alpha^{M-j} \\ &= \frac{\alpha}{1 - \alpha} (1 - \alpha^{M-j}) \end{aligned} \quad (3.20)$$

We immediately have the input rate per node,  $\gamma_{in}$ , given by

$$\begin{aligned} \gamma_{in} &= \sum_{j=0}^M \pi'_j c_j \\ &= \sum_{j=0}^M \pi'_j \left[ \frac{\alpha}{1 - \alpha} (1 - \alpha^{M-j}) \right] \end{aligned}$$

$$\begin{aligned}
&= \frac{\alpha}{1-\alpha} \left[ \sum_{j=0}^M \pi_j - \sum_{j=0}^M \pi_j \alpha^{M-j} \right] \\
&= \frac{\alpha}{1-\alpha} \left[ \sum_{j=0}^M \pi_j - \sum_{j=0}^M \pi_j \sum_{i=M-j}^{\infty} g_i \right] \\
&= \frac{\alpha}{1-\alpha} [1 - \pi_M] \\
&= \lambda [1 - \pi_M] \tag{3.21}
\end{aligned}$$

In equilibrium, the input rate  $\gamma_{in}$  must be equal to the output rate  $\gamma_{out}$ . From Equations (3.15) and (3.21), we have that

$$\lambda(1 - \pi_M) = n\rho/\bar{h}. \tag{3.22}$$

This must be satisfied by the given  $P_t$  and the calculated probability distribution  $\pi_i$ . In all the examples we have studied, we have found that the input rate is a decreasing function of  $P_t$  while the output rate is an increasing function. Moreover, the input rate is larger than the output rate when  $P_t$  approaches 0 and the input rate is smaller than the output rate when  $P_t$  approaches 1. Thus, in these examples,  $P_t$  exists and is unique. See Figure 3.5.

### 3.3.5 Delay and Throughput

Let  $\gamma$  be the throughput of a node, which is defined as the number of messages delivered to the local processors of a node per cycle (or equivalently the number of messages accepted from its local processors by the node per cycle). Thus,  $\gamma = \gamma_{in} = \gamma_{out}$ . Given that  $\Pi$  has been found, we have

$$\gamma = \lambda(1 - \pi_M). \tag{3.23}$$



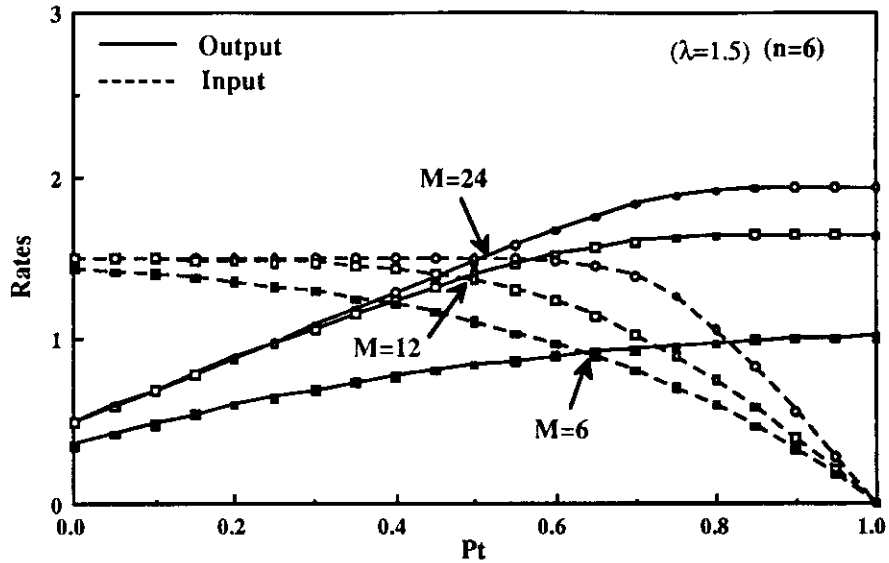


Figure 3.5: Input rate and output rate vs a given  $P_t$

The throughput of the network is then equal to  $\gamma N$ . The mean queue length of a node is given by

$$\bar{q} = \sum_{i=1}^M i \pi_i. \quad (3.24)$$

Applying Little's result [Lit61], the average message delay is given by

$$T = \frac{\bar{q}}{\gamma} \quad (3.25)$$

### 3.3.6 Effects of Deflection

We now calculate the effect of backtracking. Let us define

$$P_f = \text{Prob}[A \text{ channel forwards a message in a cycle}]$$

$$P_r = \text{Prob}[A \text{ channel backtracks a message in a cycle}],$$

where  $P_f + P_r = \rho$ . The net progress made by a channel in a cycle is equal to  $P_f - P_r$ . We realize that whenever a message is deflected, it is sent one hop farther away from its destination node. The message then needs an extra step of forwarding to compensate for this loss. We further let

$L$  = Number of channels in the network,

$K$  = Number of cycles in a (long) time period, and

$S$  = Number of messages served by the network in the period.

The expected number of one-bits in headers which are changed to zeroes in the network in  $K$  cycles is simply  $LKP_f$ . The expected number of zero-bits in headers changed to ones is  $LKP_r$ . As a result, the expected number of one-bits "decreased" due to the network's transmission is  $LK(P_f - P_r)$ , which must be equal to the number of one-bits in the headers of input messages from local processors in  $K$  cycles when the network is in a steady state. We have

$$LK(P_f - P_r) = S\bar{d} \quad (3.26)$$

Moreover, the expected number of moves of messages (forward or deflection) in the network in  $K$  cycles is

$$LK(P_f + P_r) = S\bar{h} \quad (3.27)$$

From Equations (3.26) and (3.27), we have

$$\frac{\bar{h}}{\bar{d}} = \frac{P_f + P_r}{P_f - P_r}. \quad (3.28)$$

Solving for  $P_r$  and  $P_f$ , we have

$$P_r = \frac{\rho}{2} \left(1 - \frac{\bar{d}}{\bar{h}}\right), \quad (3.29)$$

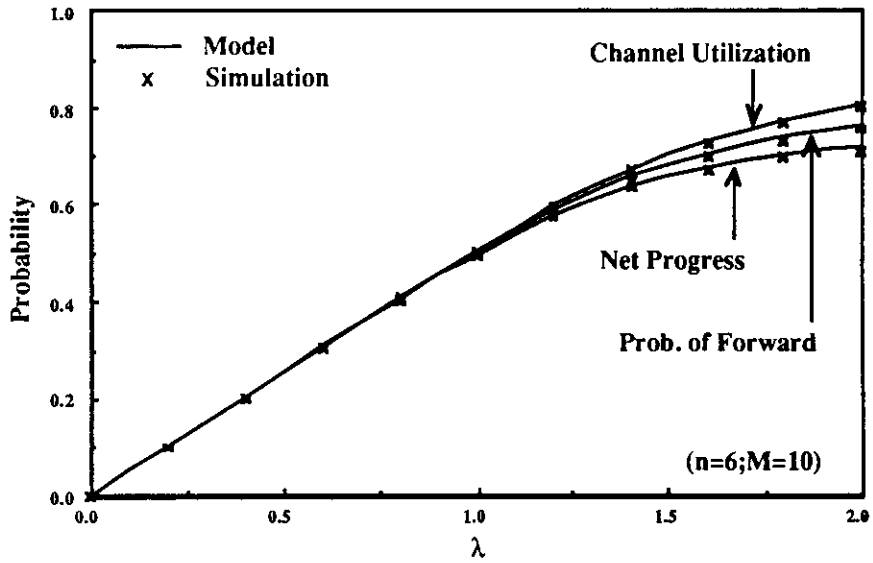


Figure 3.6: Channel utilization and probability of forwarding.

$$P_f = \frac{\rho}{2} \left( 1 + \frac{\bar{d}}{h} \right). \quad (3.30)$$

The net progress of a channel per cycle is then given by

$$P_f - P_r = \rho \frac{\bar{d}}{h}. \quad (3.31)$$

### 3.3.7 Validation of the Model and Discussion

The accuracy of this mathematical model was validated by comparing it with simulation. For each set of system parameters (i.e., the input rate, the buffer size, and the network size), the simulation ran for a time period long enough to have the network arrive at a steady state. In this section, we present several simulation results for a Boolean 6-cube network with various buffer sizes in a

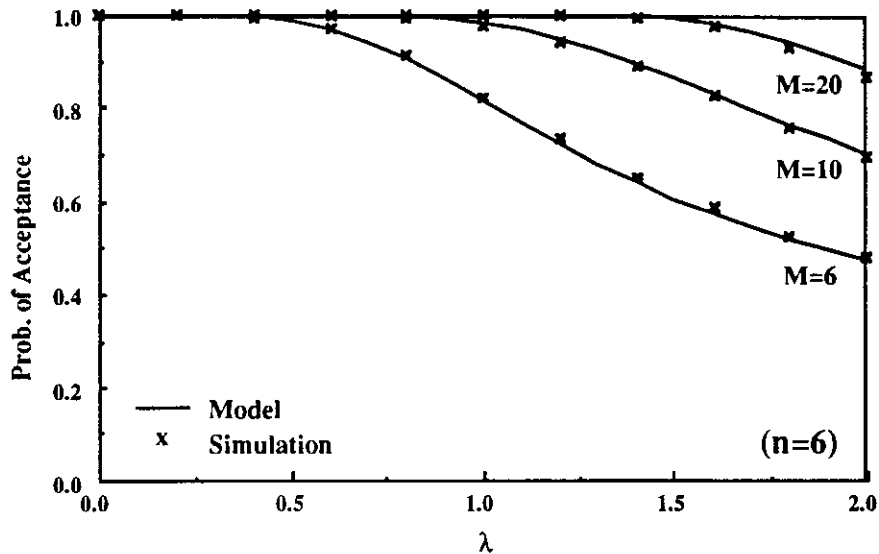


Figure 3.7: Probability of acceptance of an input message.

node. We find that the match between our model and the simulation results is very encouraging.

Figure 3.6 shows the utilization, the probability of forwarding, and the net progress of a channel when  $M = 10$ . We note that the difference between the channel utilization and the probability of forwarding is the probability of deflection. This figure shows that the effect of deflection is minor even if the buffer size is relatively small. (Recall that the buffer size must be larger than or equal to  $n$ .) Figure 3.7 presents the probability of acceptance of an input message for various buffer sizes. Again, we see that with a small number of buffers in a node, the network accepts most of the input messages unless the input rates are very large. (Recall that when  $\lambda$  approaches 2, the channel utilization approaches 1 in the infinite-buffered network.) From Figure 3.8 we find that the throughput

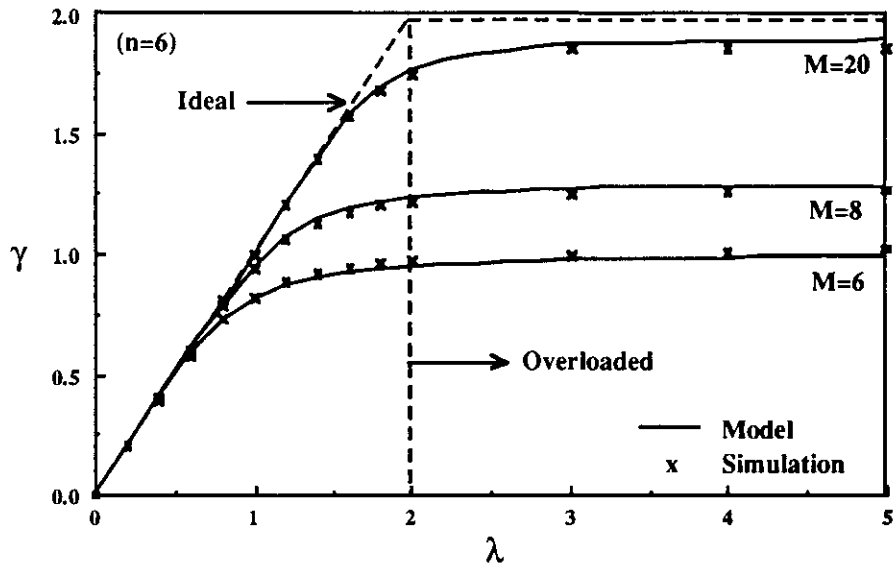


Figure 3.8: Throughput per node vs new message generation rate.

of the network does not degrade even if the input message generation rate is much larger than what can be accepted by the network. The dashed line in this figure shows the ideal throughput of the network, given that there are an infinite number of buffers at each node.

Figure 3.9 shows the mean delay as a function of the throughput per node. The “last” point of each curve is for a very large input rate. (Compare with the results in Figure 3.8 for very large input rates.) We realize that when the message generation rate is large, although a node with a larger buffer space can accept more messages, the queue grows as the buffer size increases. As a result, the mean message delay is also large. In the next section, we will combine these two performance measures (i.e. throughput and delay) together and find the “optimal” operating point.

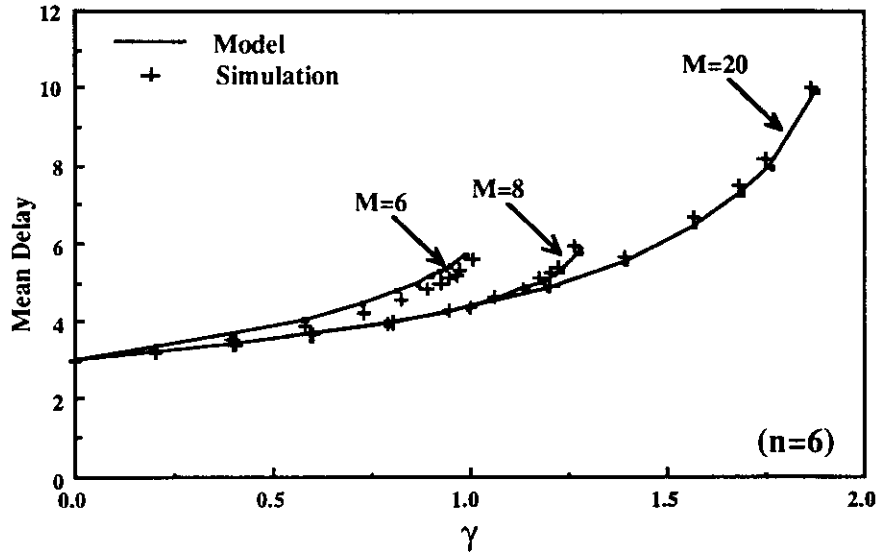


Figure 3.9: Mean message delay vs the throughput of each node.

### 3.4 Optimization Issues

In most queueing systems, two performance measures, response time and throughput, compete with each other. Typically, by raising the throughput of the system, which is desirable, the mean response time is also raised, which is not desirable. These two performance measures were combined into a single measure known as *power*, defined as follows [Kle78]

$$Power = \frac{\text{Throughput of the Network}}{\text{Mean Response Time}}.$$

For our system we have

$$Power = \frac{\gamma N}{T}.$$

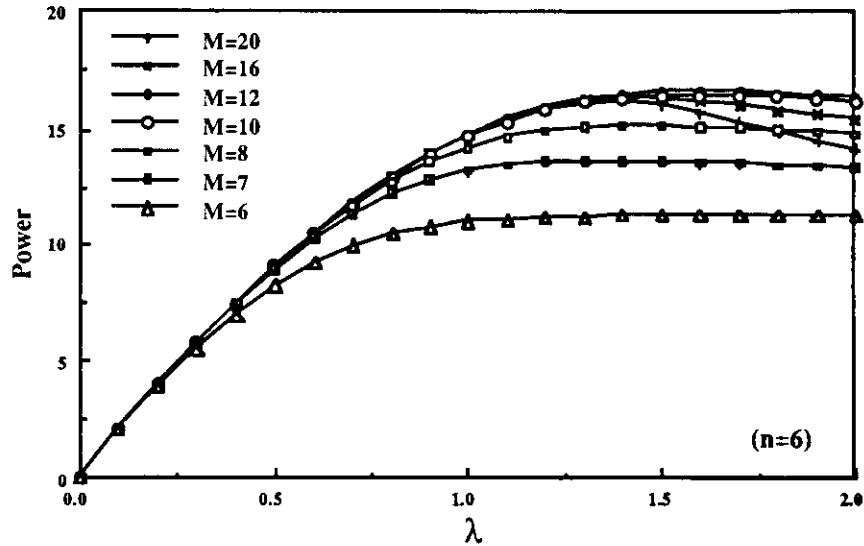


Figure 3.10: Power vs applied input rates.

A system is said to be operating at an optimal point if the power is maximized. Figure 3.10 shows the power as a function of  $\lambda$  for various buffer sizes. The peak of each curve identifies the optimal generation rate for each buffer size. This result can serve as a guide to network flow control. We find that, when input rates are small, an increase in buffer size increases the power. However, assigning a large number of buffers to a node cannot further improve the performance. When input rates are large, an increase in buffer size will significantly increase the throughput of the network. However, if the buffer size is large enough, adding more buffers just increases the queue length in each node, but does not affect the throughput. Note that, in our model, the mean delay was obtained only for those messages accepted to the network. Thus, since the message delay is significantly increased, the power is decreased. This leads to

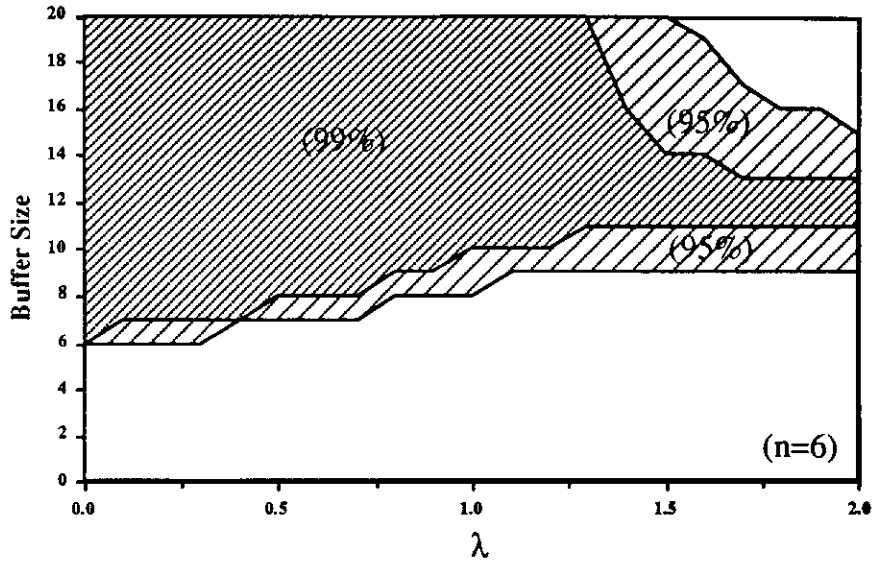


Figure 3.11: The buffer sizes which can deliver the percentage of the maximal achievable power.

a conclusion that installing many buffers in each node is just not necessary.

Figure 3.11 plots, for a Boolean 6-cube network with different input rates, the area of buffer sizes which can deliver a percentage (e.g., 95% or 99%) or more of the maximal achievable power. We find that when input rate is small, a small number of buffers in a node is enough to approach to the maximum achievable power. With this small input rate, a network with a large number of buffers just behaves as if it has an infinite number of buffers. When the input rate is large, a small number of buffer results in too much deflection and wastes the communication capacity, while a large number of buffers increases delay significantly. For a wide range of input rates, we find that “ $2n$ ” is the optimal buffer size assignment for a Boolean  $n$ -cube network. We performed this procedure for many networks with different sizes and all showed similar



behavior.

### 3.5 Conclusions

In this chapter we developed an approximation for evaluating the performance of a deflection routing algorithm in Boolean  $n$ -cube networks with finite buffers. With this algorithm, when a node is nearly full, it needs to force out some of its messages to its neighbors; some of these messages can be deflected. The deflected messages are sent one hop away from their destinations. We showed that all messages entering the network are guaranteed to arrive at their destination without deadlocks or discards. The performance of this algorithm has been analyzed by mathematical models and simulations. We showed that the effect of the deflection is minor even if every node has only a small number of buffers. We showed that the throughput of the network does not degrade even when the network is full. Given the "power" function, we showed that only a few buffers (i.e., " $2n$ " ) are essential to optimize the performance.

## CHAPTER 4

# Analysis of Deflection Routing in $k$ -Ary $n$ -Cube Networks

### 4.1 Introduction

In this chapter we develop a mathematical model to evaluate a deflection routing algorithm for a general class of networks called  $k$ -ary  $n$ -cubes [SB77]. A number of network topologies including rings, tori, Omega networks, and hypercubes (Boolean  $n$ -cubes) all belong to this family. Examples of multiprocessor systems based on such a network include the CMU-Intel iWarp [Bor88], the Ametek 2010 [Sei88], and the MIT J-machine [Dal89], in addition to the hypercube-based systems.

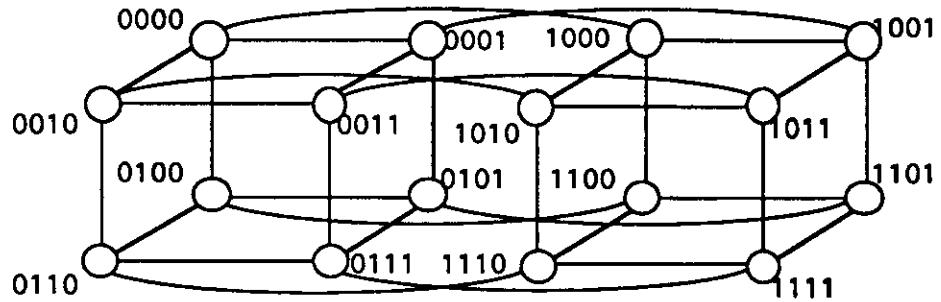
A  $k$ -ary  $n$ -cube network consists of  $k^n$  nodes, each addressed by a  $n$ -digit address  $(a_{n-1}, a_{n-2}, \dots, a_0)$ , where  $0 \leq a_i \leq (k - 1)$ . We refer to  $n$  as the dimension and  $k$  as the radix of the network. The  $i$ th digit of the address,  $a_i$ , represents the node's position in the  $i$ th dimension. We assume that links are unidirectional. Nodes are interconnected in such a way that there is a link from node  $(a_{n-1}, a_{n-2}, \dots, a_0)$  to node  $(b_{n-1}, b_{n-2}, \dots, b_0)$  if and only if there exists an  $i$  such that  $(a_i - b_i)_{\text{mod } k} = 1$  and  $a_j = b_j$  for  $j = 0, 1, \dots, i - 1, i + 1, \dots, n - 1$ . Every node can only send messages to its lower neighbor of each dimension. That is, node  $(a_{n-1}, a_{n-2}, \dots, a_0)$  can only send messages to nodes

$(a_{n-1}, \dots, a_{i+1}, (a_i - 1)_{\text{mod } k}, a_{i-1}, \dots, a_0)$  for  $i = 0, 1, \dots, n - 1$ . One may also view a node as an intersection of  $n$  ring structures; each ring consists of  $k$  nodes.

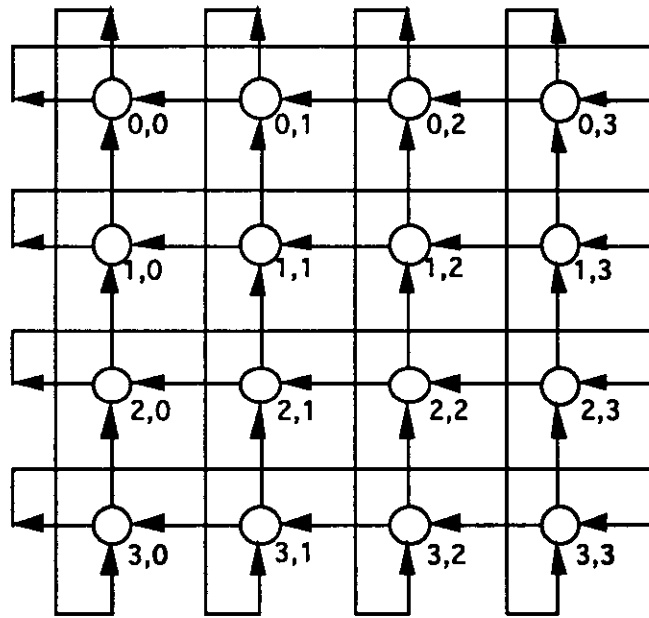
In Figure 4.1, we show two 16-node networks, one constructed as a binary (Boolean) 4-cube and the other as a 4-ary 2-cube. Note that, in a Boolean  $n$ -cube network, there are only two nodes in a dimension (or ring); each node has a link to the other. In Figure 4.1(a), for simplicity, we show only a link between two nodes in a dimension.

Links can also be bidirectional. It is worth mentioning here that bidirectional links allow applications to exploit communication locality. For example, if an object  $\alpha$  sends a message to an object  $\beta$ , then it is likely that  $\beta$  will also send a (response) message back to  $\alpha$ . If  $\alpha$  and  $\beta$  are assigned to two neighboring nodes, then the round-trip route takes only two hops. In a unidirectional network, such a round-trip route will take  $k$  hops. However, with bidirectional links, each node handles  $2n$  incoming and  $2n$  outgoing channels, twice as many and therefore more costly than that of the unidirectional network. Due to the design and manufacturing cost, we may not be able to construct a large-dimensional network with bidirectional links. In this chapter, we assume that links are unidirectional and the connections are end-around.

Given that the destination address of a message is  $(b_{n-1}, b_{n-2}, \dots, b_0)$  and the message is currently at node  $(a_{n-1}, a_{n-2}, \dots, a_0)$ , we say that the  $i$ th dimension is a “valid” dimension if and only if  $a_i \neq b_i$ . We say that a message is *forwarded* if it is sent along a valid dimension, and therefore ends up one hop closer to its destination. A message is *deflected* if it is sent along a nonvalid dimension. A deflected message will be  $k - 1$  hops further away from its destination than it was before being deflected.



(a) A binary (Boolean) 4-cube network.



(b) A 4-ary 2-cube network

Figure 4.1: The examples of  $k$ -ary  $n$ -cube networks with 16 nodes.

The choice of a good network topology is highly sensitive to the assumptions about the network. Dally [Dal90] analyzed the performance of  $k$ -ary  $n$ -cube networks with various dimensions under the assumption that the bisection width is held constant, where the bisection width is defined as the minimum number of wires which must be cut in order to divide the network into two equal halves. The analysis suggests that lower-dimensional networks yield lower latency. However, node delays and queueing effects were ignored in his analysis. Agarwal [Agr91] also analyzed the performance of  $k$ -ary  $n$ -cube networks by taking node and link delays into account. He showed that the best network has a moderately high dimension. However, neither Dally nor Agarwal considered the effects of buffer size on performance.

In this chapter, we analyze a deflection routing algorithm for  $k$ -ary  $n$ -cube networks with finite buffers. We calculate the buffer size for each node which allows the network to deliver the near-optimal performance for a wide range of input rates. We examine the performance of the  $k$ -ary  $n$ -cubes with various dimensions under the constraint of fixed bisection width. Both the effects of node processing time and queueing are taken into account in the analysis.

## 4.2 Deflection Routing in $k$ -Ary $n$ -Cube Networks

Deflection routing can be used in a  $k$ -ary  $n$ -cube network, as long as the network is synchronized into cycles, each of which consists of two phases. We apply the concept of the two-phase message assignment developed earlier for Boolean  $n$ -cube networks to  $k$ -ary  $n$ -cube networks as follows: In the first phase, each node selects a valid dimension for each message currently in the node. We assume that the selection of a valid channel for each message is done simultaneously

and independently. If more than one message selects the same dimension, the one with the highest priority (i.e., the oldest one) is successfully assigned to the corresponding outgoing channel; other messages are saved in the buffers.

The way of selecting a valid channel in the first phase is different from what we have for the Boolean  $n$ -cube network. In the Boolean  $n$ -cube network, every one-bit in the header indicates a valid channel. Selecting any one of these valid dimensions will decrease the number of valid channels and the distance to the destination. In the  $k$ -ary  $n$ -cube network, however, assigning a message to a valid channel does not necessarily decrease the number of valid dimensions by one. In order to exploit the multiplicity of paths provided by the topology, the routing algorithm should reduce a message's valid dimensions only when necessary. For example, in the 4-ary 2-cube network (Figure 4.1 (b)), if node (3,1) wants to transmit a message to node (0,0), it should first send the message to node (2,1), as long as the corresponding outgoing channel is available. Otherwise, if the message is first sent to node (3,0), then the message has only one valid dimension left. In the latter case, the message must be forwarded along a unique dimension for the rest of its journey. Since a message can be forced out in the second phase, a message with fewer valid dimensions is more likely to be deflected.

Badr and Podar [BP89] presented an optimal routing policy for mesh-connected networks. In their work, they defined  $v(i, j)$  as the success-probability of delivering a message from node  $(i, j)$  to node  $(0, 0)$ , and  $S(i, j)$  as a set of neighboring nodes which maximize  $v(i, j)$ . Assuming uniform traffic, they showed that

$$(a) S(i, i) = \{(i - 1, i), (i, i - 1)\},$$

$$(b) S(i, j) = \{(i - 1, j)\} \text{ for } i > j,$$

$$(c) S(i, j) = \{(i, j - 1)\} \text{ for } i < j.$$

That is, a message should be sent in a direction such that it minimizes the difference in the number of hops left among the valid dimensions. We apply this technique to the first phase message assignment, although we do not provide a formal proof of optimality here.

Suppose a message in node  $A = (a_{n-1}, a_{n-2}, \dots, a_0)$  is to be sent to node  $B = (b_{n-1}, b_{n-2}, \dots, b_0)$ . Let  $e_i$  be the number of hops the message needs to travel in dimension  $i$ . Thus,  $e_i = (a_i - b_i)_{\text{mod } k}$ . The vector  $(e_{n-1}, e_{n-2}, \dots, e_0)$  of a message is referred as the header of the message. Suppose the message is moved along the  $m$ th dimension. The corresponding digit,  $e_m$ , is changed as follows:

$$e_m \longrightarrow (e_m - 1)_{\text{mod } k}$$

Clearly, if  $e_m > 0$ , the message is forwarded and  $e_m$  decreases by one. If  $e_m = 0$ , the message is deflected; in this case,  $e_m$  is changed to  $k - 1$ .

A valid dimension  $i$  is called a *preferred dimension* if and only if  $e_i \geq e_j$  for all  $j = 0, 1, \dots, n - 1$ . Our first phase message assignment is to select a preferred dimension for each message. If there is more than one preferred dimension in the header, the node randomly chooses one of them. To further simplify our presentation, we call a valid dimension  $j$  a *terminating dimension* if  $e_j = 1$ . That is, after the message is forwarded along a terminating dimension, the number of valid dimensions will be reduced by one. If a node cannot make at least  $n$  buffers available in the first phase, the node must assign more messages in its second phase as described in Section 3.2.

### 4.3 Analysis of the Deflection Routing Algorithm

In this section, we develop an approximation model for evaluating the deflection routing algorithm in  $k$ -ary  $n$ -cube networks with finite buffers. We are interested in the throughput of the network, the probability of acceptance of an input message, the mean message delay, and the effect of the buffer size on performance.

#### 4.3.1 Assumptions of the Model

We assume that a message's destination is uniformly distributed over the network. The mean number of hops a message needs to travel in each dimension is  $(k - 1)/2$ , given that a node is allowed to transmit a message to itself. The mean number of hops a message must travel in the network is then calculated as  $\bar{d} = n(k - 1)/2$ . We assume that a node cannot inject a message into the network if the message is for the node itself. In this case, the mean number of hops becomes  $\bar{d} = n(k - 1)k^n/[2(k^n - 1)]$ . However, due to deflection in finite-buffered networks, a message may not always travel along a shortest path.

At the end of a cycle, the arrival of input messages to a node is assumed to follow a geometric distribution with a generation rate of  $\lambda$  messages per cycle. Let  $g_i$  be the probability of a node receiving  $i$  input messages in a cycle. Then,

$$g_i = (1 - \alpha) \alpha^i, \text{ where } 0 < \alpha = \frac{\lambda}{1 + \lambda} < 1.$$

Given that all input messages are accepted and are routed via a shortest path (in an infinite-buffered case), the channel utilization is given by

$$\rho = \lambda \frac{\bar{d}}{n}$$



$$= \lambda \frac{k-1}{2} \frac{k^n}{(k^n-1)},$$

which must be less than 1. Thus,

$$\lambda < [2(k^n - 1)]/[(k - 1)k^n]. \quad (4.1)$$

When  $n$  is large, the inequality can be approximated by  $\lambda < 2/(k - 1)$ , which is referred to as the ideal communication capacity of a node. However, in a finite buffered network, input messages are blocked when the source node's buffers are full. We define  $P_A$  as the probability of an input message being accepted by the node.

### 4.3.2 Analysis of the Model

The approach we use to solve the model is similar to what we have developed for the Boolean  $n$ -cube network in the previous chapter. Here we briefly describe the method and the related variables. We assume that the  $k$ -ary  $n$ -cube network is decomposed into a set of statistically identical nodes. We model each node individually. We further assume there exists an equilibrium state for the node. Clearly, this is an approximation model.

Let  $P$  be the one-cycle transition probability matrix of the node, where an element  $p_{i,j}$  of the matrix is defined as the probability that, given the node has  $i$  messages at the beginning of the last cycle, the node will have  $j$  messages at the beginning of this cycle. Let  $\Pi$  be the steady state probability vector, where an element  $\pi_i$  is the probability of finding  $i$  messages at the beginning of a given cycle. Define  $M \geq n$  to be the buffer size of a node. If the matrix  $P$  is given, then  $\Pi$  can be determined by solving the following equations [Kle75]:

$$\Pi = \Pi P$$

$$\sum_{i=0}^M \pi_i = 1.$$

Without loss of generality, we divide a cycle into three sub-cycles: departure of messages, reception of transit messages, and reception of input messages. The departure process is determined by the algorithm a node uses to assign its messages to the outgoing channels. The number of input messages accepted in a cycle is determined by the message generation distribution and the number of free buffers at the node. However, since the mean number of hops a message actually traverses is unknown due to deflection, it is very difficult to give an explicit expression for the number of transit messages received in a cycle.

Let  $P_t$  be the probability of receiving a transit message from an incoming channel in a cycle. Given  $P_t$ , the probability of receiving  $i$  transit messages in a cycle is obtained as follows:

$$t_i = \binom{n}{i} P_t^i (1 - P_t)^{n-i}, \quad i = 0, 1, 2, \dots, n.$$

Thus, the one-cycle transition probability matrix is constructed and  $\Pi$  can be found. Therefore,  $\Pi$  is uniquely determined by the value of  $P_t$ . We need one more condition to solve for  $P_t$ , which is the following: *For the system to be stable, the input flow to the network must equal the output flow out of the network.*

We now calculate  $\bar{h}$ , the mean number of hops actually traversed by a message in the network. Following our routing algorithm, a node randomly selects a preferred dimension in the first phase. Every outgoing channel has the same probability of being selected under uniform traffic. Given  $i$  messages in the node, the probability ( $f_{i,j}$ ) of successfully assigning  $j$  of them to channels in the

first phase is given by Equation (2.33), which is repeated below:

$$f_{i,j} = \begin{cases} \binom{n}{j} \sum_{k=0}^{j-1} (-1)^k \binom{j}{k} \left(\frac{i-k}{n}\right)^i & \text{if } i \geq 1 \text{ and } 1 \leq j \leq \min(i, n) \\ 1 & \text{if } i = j = 0 \\ 0 & \text{otherwise} \end{cases}$$

Given that  $j$  messages are successfully assigned in the first phase and  $(i - j) > (M - n)$ , then  $(n - M + i - j)$  messages must be assigned to free outgoing channels in the second phase. Let  $f'_{i,j}$  be the probability that  $j$  messages are successfully assigned in both phases, given that the node initially has  $i$  messages.

We have

$$f'_{i,j} = \begin{cases} f_{i,j} & \text{if } i - j < M - n \\ \sum_{k=1}^{n-M+i} f_{i,k} & \text{if } i - j = M - n \\ 0 & \text{otherwise.} \end{cases} \quad (4.2)$$

We further have the following equation:

$$\rho = \sum_{i=1}^M \pi_i \sum_{j=1}^{\min(i,n)} f'_{i,j} \frac{j}{n} \quad (4.3)$$

$$\rho_1 = \sum_{i=1}^M \pi_i \sum_{j=1}^{\min(i,n)} f_{i,j} \frac{j}{n} \quad (4.4)$$

$$\rho_2 = \rho - \rho_1, \quad (4.5)$$

where  $\rho$  is the probability of a channel transmitting a message in a cycle,  $\rho_1$  is the probability of a channel transmitting a message which is assigned in the first phase, and  $\rho_2$  is the probability of a channel transmitting a message which

is assigned in the second phase. Clearly,  $\rho = \rho_1 + \rho_2$ . We further define

$$p = \rho_1/\rho$$

and

$$q = \rho_2/\rho.$$

Given that a message is successfully assigned and transmitted in a cycle, let  $P_m$  be the probability that the message is assigned to the  $m$ th dimensional outgoing channel. We further let  $n_p$  be the number of preferred dimensions of the message. Based on our routing algorithm, it follows that given the message is successfully assigned in the first phase, it must be assigned to one of its preferred dimensions. Each of the  $n_p$  preferred dimensions of the message has the same probability of being selected. Thus, given that a message is transmitted, the probability that it is successfully assigned to a particular preferred dimension in the first phase is  $p/n_p$ . A preferred dimension can also be assigned in the second phase. Clearly, if a preferred dimension has been selected but not successfully assigned in the first phase, it cannot be chosen again in the second phase because the corresponding outgoing channel must have been assigned by some other message in the first phase. That is, a preferred dimension is assigned in the second phase only if it is not selected in the first phase. Moreover, every dimension of a message except those selected in the first phase have the same probability of being assigned in the second phase. Thus, given that the message is transmitted, the probability that it is assigned to a preferred dimension in the second phase is  $(1 - \frac{1}{n_p})\frac{q}{n-1}$ . We then obtain the following equations:

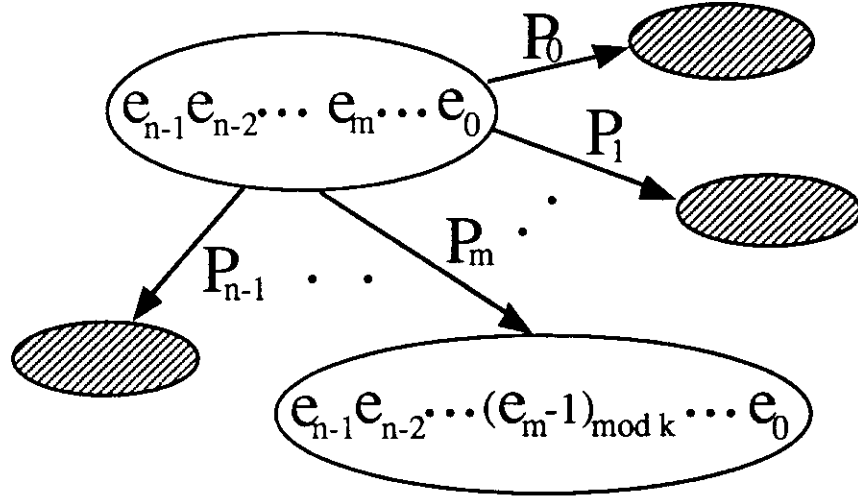


Figure 4.2: Transition diagram for calculating the mean path length.

$$P_m = \begin{cases} \frac{p}{n_p} + (1 - \frac{1}{n_p}) \frac{q}{n-1} & \text{if the } m\text{th dimension is preferred} \\ \frac{q}{n-1} & \text{otherwise.} \end{cases} \quad (4.6)$$

We now describe a transition diagram as follows: Each state in the diagram is identified by  $n$  digits with radix  $k$  which corresponds to  $(e_{n-1}, e_{n-2}, \dots, e_0)$ , the header of a message. When the message is moved to a neighboring node, the header changes. The state of the message also changes accordingly. In Figure 4.2, we show the (probabilistic) transitions between neighboring states.

Let  $h(e_{n-1}, e_{n-2}, \dots, e_0)$  be the mean number of hops a message actually travels from state  $(e_{n-1}, e_{n-2}, \dots, e_0)$  to state  $(0, 0, \dots, 0)$ . Thus,  $h(e_{n-1}, e_{n-2}, \dots, e_0)$

can be written as follows:

$$\begin{aligned} & h(e_{n-1}, \dots, e_m, \dots, e_0) \\ = & 1 + \sum_{m=0}^{n-1} P_m h(e_{n-1}, \dots, e_{m+1}, (e_m - 1)_{\text{mod } k}, e_{m-1}, \dots, e_0) \end{aligned}$$

Solving this set of linear equations, we find the value of  $h$  function for each state. Moreover, given the probability of an input message initially being in a particular state, then  $\bar{h}$ , the mean number of hops traversed by that message can be found. Moreover, the output rate of a node is given by

$$\gamma_{out} = n\rho/\bar{h}. \quad (4.7)$$

The input rate per node is given by

$$\gamma_{in} = \lambda(1 - \pi_M). \quad (4.8)$$

$P_i$  must therefore satisfy the following equation:

$$\lambda(1 - \pi_M) = n\rho/\bar{h}. \quad (4.9)$$

The major drawback of this approach is that the number of states in the transition diagram is equal to the number of nodes in the network. When the network size grows to hundreds (or more) nodes, this approach becomes infeasible. In the rest of this section, we present another approach to find  $\bar{h}$ .

We let each state of the diagram be identified by  $(n_v, n_d)$ , where  $n_v$  is the number of valid dimensions of the message, and  $n_d$  is the number of hops from the current node to its destination. Given that a message has  $n_v$  valid dimensions, the following condition must hold:  $n_v \leq n_d \leq (k - 1)n_v$ . That is, we need  $(k - 1)n_v - n_v + 1$  states to represent all the cases in which a message has

k	n	number of states
16	2	45
4	4	25
2	8	9

Table 4.1: Number of states required for a network with 256 nodes.

$n_v$  valid dimensions. A message has up to  $n$  valid dimensions. Thus, the total number of states required for the transition diagram is given by

$$\begin{aligned}
& \sum_{i=0}^n [i(k-1) - i + 1] \\
&= \sum_{i=0}^n [i(k-2) + 1] \\
&= (k-2) \frac{n(n+1)}{2} + n + 1.
\end{aligned}$$

In Table 4.1 we show, for a network with 256 nodes, the number of states required in the state transition diagram. Compared with our previous approach, the number of states required for the transition diagram is dramatically reduced.

Given that a message is moved over the  $m$ th dimension, its state  $(n_v, n_d)$  is changed in one of the following three ways:

$$(I) \quad (n_v, n_d) \longrightarrow (n_v, n_d - 1) \quad \text{if } e_m > 1$$

$$(II) \quad (n_v, n_d) \longrightarrow (n_v - 1, n_d - 1) \quad \text{if } e_m = 1$$

$$(III) \quad (n_v, n_d) \longrightarrow (n_v + 1, n_d + k - 1) \quad \text{if } e_m = 0$$

The transition probabilities are calculated as follows: Consider the state where  $n_v = i$  and  $n_d = j$ . We look at the three possible cases:

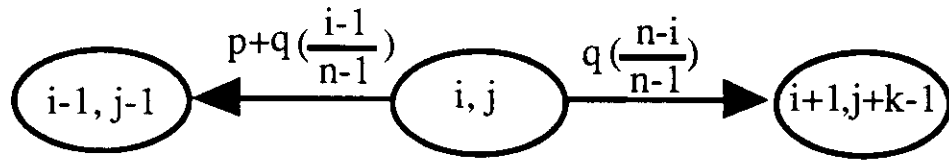
**Case I:**  $i = j$

Each valid dimension has only one hop to traverse. This is also known as an extreme case of Boolean  $n$ -cube networks where each dimension has at most one hop to traverse. If the message is selected in the first phase, it is forwarded and the number of valid dimensions is decreased by one. Given the message is selected in the second phase, a valid dimension is selected with probability  $(i - 1)/(n - 1)$ , and a nonvalid dimension is selected with probability  $(n - i)/(n - 1)$ . The transition probabilities are shown in Figure 4.3(a).

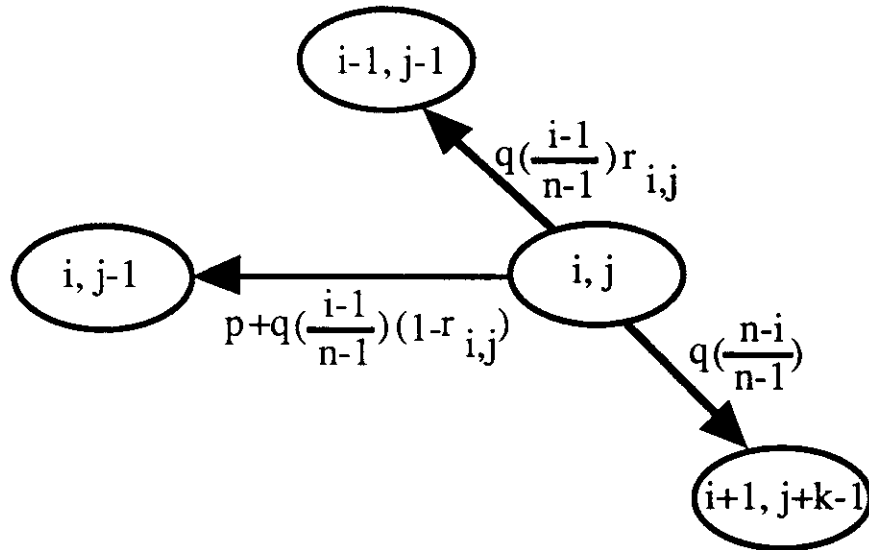
**Case II:**  $i < j \leq (i - 1)(k - 1) + 1$

If the message is successfully assigned in the first phase, it must be assigned to a preferred dimension. Since  $i < j$ , the preferred dimension cannot be a terminating dimension. Thus, the number of valid dimensions does not change. Given that the message is selected in the second phase, it can be assigned to a valid dimension with probability  $(i - 1)/(n - 1)$ , and assigned to a nonvalid dimension with probability  $(n - i)/(n - 1)$ . We define  $r_{i,j}$  as the probability that the message is assigned to a terminating dimension, given that the message

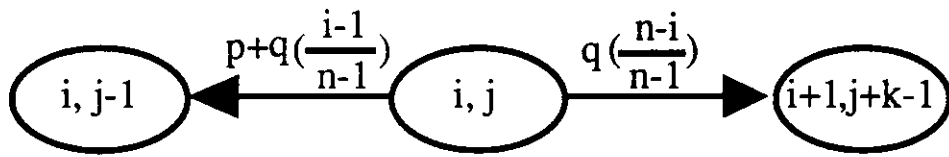




(a) Case I :  $i = j$



(b) Case II :  $i < j \leq (i-1)(k-1)+1$



(c) Case III :  $(i-1)(k-1)+1 < j \leq i(k-1)$

Figure 4.3: State transition diagram for calculating mean path length.

is assigned to a valid dimension in the second phase. We approximate  $r_{i,j}$  as follows: The number of ways to distribute the  $j$  hops into the  $i$  valid dimensions such that every dimension has at least one hop is

$$\binom{(j-i) + (i-1)}{i-1} = \binom{j-1}{i-1},$$

given that there is no limit on the number of hops to be put in a dimension. (Note that the maximum number of hops in a dimension is  $k-1$  for real systems.) The number of ways to distribute the  $j$  hops into the  $i$  valid dimensions such that there are exactly  $m$  terminating dimensions is given by

$$\binom{i}{m} \binom{(j-i) - (i-m) + (i-m) - 1}{i-m-1} = \binom{i}{m} \binom{j-i-1}{i-m-1}.$$

Moreover, the message must have chosen one of the non-terminating dimensions in the first phase. Thus, given that the message is assigned to one of the other  $i-1$  valid dimensions in the second phase, the probability that it is assigned to a terminating dimension is  $m/(i-1)$ , given that there are  $m$  terminating dimensions. Thus,  $r_{i,j}$  is approximated by

$$r_{i,j} = \frac{\sum_{m=1}^{i-1} \binom{i}{m} \binom{j-i-1}{i-m-1} \frac{m}{i-1}}{\binom{j-1}{i-1}}.$$

The transition probabilities are shown in Figure 4.3(b).

**Case III:**  $(i-1)(k-1) + 1 < j \leq i(k-1)$

In this case, none of the valid dimensions is a terminating dimension. The transition probabilities are the same as those in Case I, except that when the message is forwarded, its  $i$  remains constant while its  $j$  decreases by one. Figure 4.3 (c) shows the corresponding transition probabilities.

We note that the transition might not exist for some boundary states; in this case the transition probability is always 0. Given these transition probabilities, the set of linear equations for finding the mean number of hops to move from one state to state  $(0,0, \dots, 0)$  is determined. As a result, the mean number of hops traversed by a message,  $\bar{h}$ , is also determined. For all the cases we have studied, a unique  $P_i$  has been found. Given the  $P_i$ , we then obtain the distribution of messages in a node. Finally, we solve for the mean queue length, the throughput of the network, and the mean delay. Other performance measures such as channel utilization, probability of acceptance of an input message, and the probability of a channel deflecting a message can also be found.

### 4.3.3 Validation of the Model

We validated the model against simulation with different network structures, sizes, and workloads. The simulator routes messages through the network entirely based on our two-phase message assignment. After the message assignment is completed, every node simultaneously transmits messages to its neighbors. At the end of a cycle, every node generates new messages based on a geometric distribution with an input rate of  $\lambda$  messages per cycle. Some input messages can be dropped if the buffers are full. The simulator generates statistics such as the channel utilization, the probability of acceptance of input messages, the mean number of hops traversed by a message, the mean number of messages delivered in a cycle, the mean queue length, and the mean delay.

We now show some numerical results for a network with 64 nodes. Larger networks have also been simulated and similar behavior was observed. Noting that the Boolean  $n$ -cube network is a special case which we have evaluated in

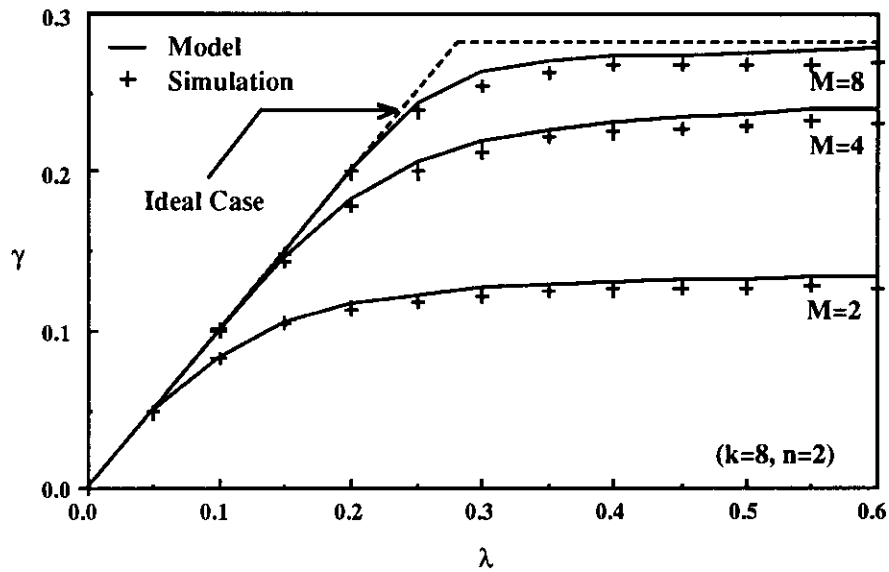


Figure 4.4: Throughput per node for a 8-ary 2-cube network.

the previous chapter, we are now more interested in the results from another extreme case, the 8-ary 2-cube network. Figure 4.4 illustrates the throughput of a node (mean number of messages delivered per node per cycle) in the 8-ary 2-cube network obtained from our model and the simulator. The dashed line corresponds to an infinite buffered network, in which every message is accepted if the network is not overloaded and every accepted message is routed to its destination without deflection. We find that the throughput does not degrade even if the network is overloaded. The probability of acceptance of input messages can simply be obtained by dividing the throughput by the input rate.

Figure 4.5 shows the mean delay as a function of the throughput per node, where the curves are obtained from our model and the “dots” are from the sim-

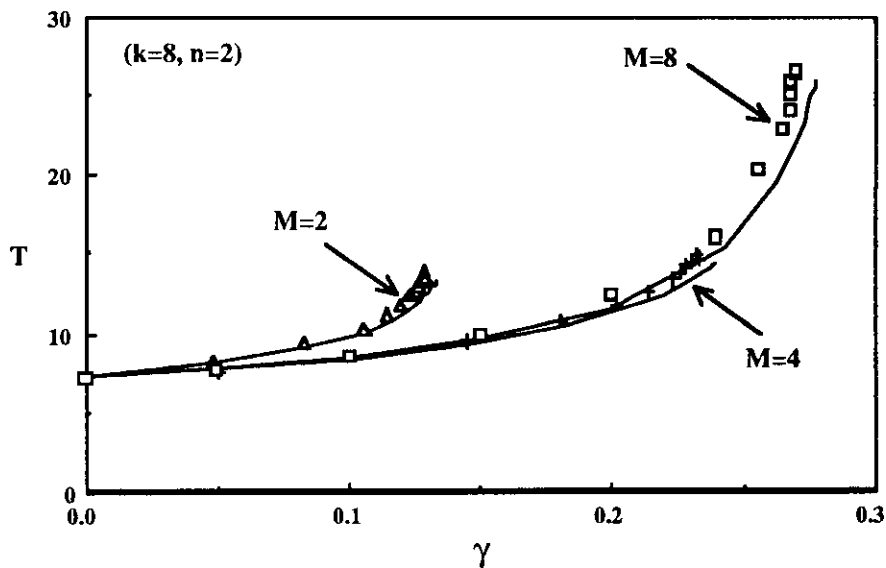


Figure 4.5: Mean delay vs throughput per node for a 8-ary 2-cube network.

ulation. The "last" point of each curve corresponds to the limiting throughput in Figure 4.4. Note that, when the input rate is very large, the throughput of the network with  $M = 8$  is only about 15% larger than that of the network with  $M = 4$ , while the mean delay of the network with  $M = 8$  is almost twice as large as that of the network with  $M = 4$ . From these two figures, we find that the match between the model and the simulation results is very encouraging.

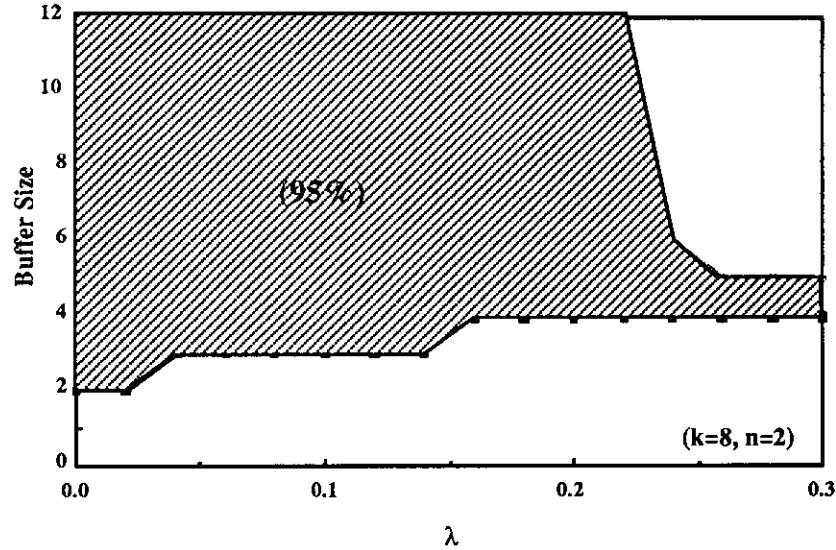


Figure 4.6: The buffer size in a node which delivers 95% of the maximum achievable power in a 8-ary 2-cube network.

## 4.4 Discussion

### 4.4.1 Optimal Buffer Assignment

In this section, we calculate the buffer size which maximizes the power of the network system. Recall that the power of a queueing system is defined as its throughput over its mean response time [Kle78].

Figure 4.6 plots, for a 8-ary 2-cube network with different input rates, the area of buffer sizes which deliver a percentage (e.g., 95%) of the maximal achievable power. We find that when the input rate is small, a small number of buffers in a node is enough to approach the maximum achievable power. When the input rate is large, a small number of buffers simply results in too much deflection which wastes the communication capacity. As buffer size increases, the

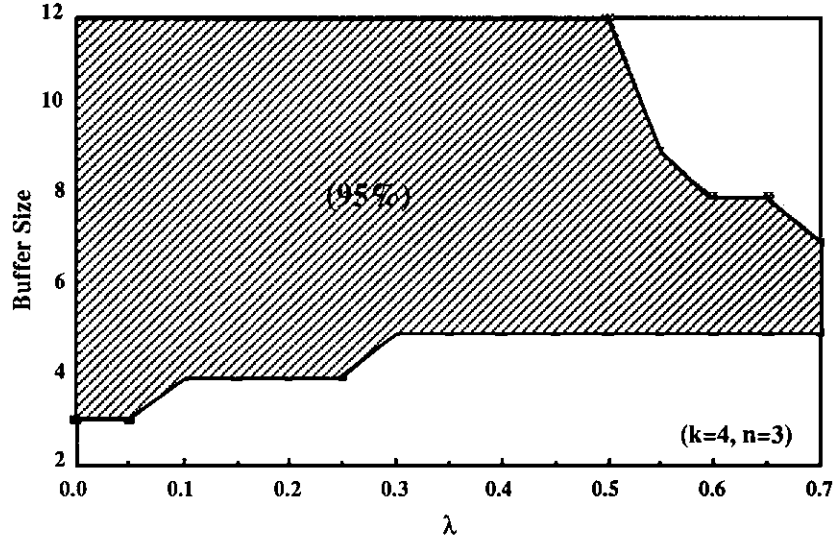


Figure 4.7: The buffer size in a node which delivers 95% of the maximum achievable power in a 4-ary 3-cube network.

probability of deflection decreases and the throughput grows, but it approaches its maximal achievable power quickly. Further assignment of buffers cannot significantly increase the throughput but does increase the queue length and the delay. Thus, the power is also decreased. We find that assigning “ $2n$ ” buffers to a node is a very good choice for a wide range of input rates. Figure 4.7 shows a similar behavior for a 4-ary 3-cube network. Note that in these figures we are only interested in the cases in which the input rate ranges in  $0 < \lambda < 2/(k - 1)$ , where  $2/(k - 1)$  is the ideal maximal throughput of a node.

#### 4.4.2 Choice of Dimensions and Radixes

Networks with higher dimensions consist of more communication channels (or links) than do lower-dimensional networks. Thus, given the constant channel

width (number of bits to be transmitted over a channel in parallel), the higher-dimensional networks provide larger communication bandwidth than do the lower-dimensional networks. However, the bandwidth of a VLSI computing system cannot be increased arbitrarily. Dally [Dal90] defined bisection width as the minimal total number of wires which must be cut to separate the network into two equal halves. In general, the limitation on the bisection width imposes bounds on the minimum layout area, allowable system size, and cost of the network [Agr91]. To add more channels in the system, every channel must be equipped with less wires. That is, the channel width of a higher-dimensional network is less than that of a lower-dimensional network. Thus, the higher-dimensional network will spend more time in transmitting a message than the lower-dimensional network. A realistic comparison of networks with different dimensions is to hold the bisection width constant [Dal90].

Given that the network is embedded in a plane, the bisection width of a  $k$ -ary  $n$ -cube network with  $W$ -bit channels can be calculated as  $2Wk^{n-1}$ . To hold the bisection width constant, the channel width of a  $k$ -ary  $n$ -cube network,  $W(k, n)$ , is given by  $\frac{k}{2}c$ , where  $c$  is a constant. In the following calculation, we normalize to a Boolean  $n$ -cube network with one-bit channels. That is,  $c$  is set to 1. Moreover, we assume the length of a message to be  $L$  bits. Thus, the transmission time of a message between two neighboring nodes in a  $k$ -ary  $n$ -cube network is

$$\begin{aligned} T_x &= L/W(k, n) \\ &= \frac{2L}{k}. \end{aligned}$$

We further assume that in each cycle, the time spent by a node to process



messages is  $T_P$ . Thus, the cycle time in a  $k$ -ary  $n$ -cube network is given by

$$T_P + T_X = T_P + \frac{2L}{k}.$$

Clearly, given that the bisection width is held constant, the cycle time of a higher-dimensional network is longer than that of a lower-dimensional network. Since multiprocessor systems are usually physically close, we assume that the propagation delay is negligible.

Define  $\beta$  as  $T_P/L$ , where  $L$  is the transmission time of an  $L$ -bit message in a Boolean  $n$ -cube network with one-bit channels. A larger  $\beta$  corresponds to longer node processing time and/or shorter messages. Thus, the cycle time of a  $k$ -ary  $n$ -cube network is given by

$$T_c(k, n) = \left( \beta + \frac{2}{k} \right) L.$$

Let  $\gamma$  be the throughput (number of messages delivered per cycle time) per node and  $T$  be the mean delay (in cycles), respectively. By taking the cycle time of a Boolean  $n$ -cube network as a unit of (real) time, the cycle time of a  $k$ -ary  $n$ -cube network is normalized as  $\frac{\beta+2/k}{\beta+1}$  units of time. Thus, we normalize the throughput per node of a  $k$ -ary  $n$ -cube network as

$$\gamma_N = \gamma \frac{\beta + 1}{\beta + 2/k}.$$

The mean delay is also normalized as

$$T_N = T \frac{\beta + 2/k}{\beta + 1}.$$

In the following discussion, we assume that every node in a  $k$ -ary  $n$ -cube network is equipped with  $2n$  buffers. Figures 4.8-4.10 show, for a network with 64 nodes, the normalized delay as a function of the normalized throughput.

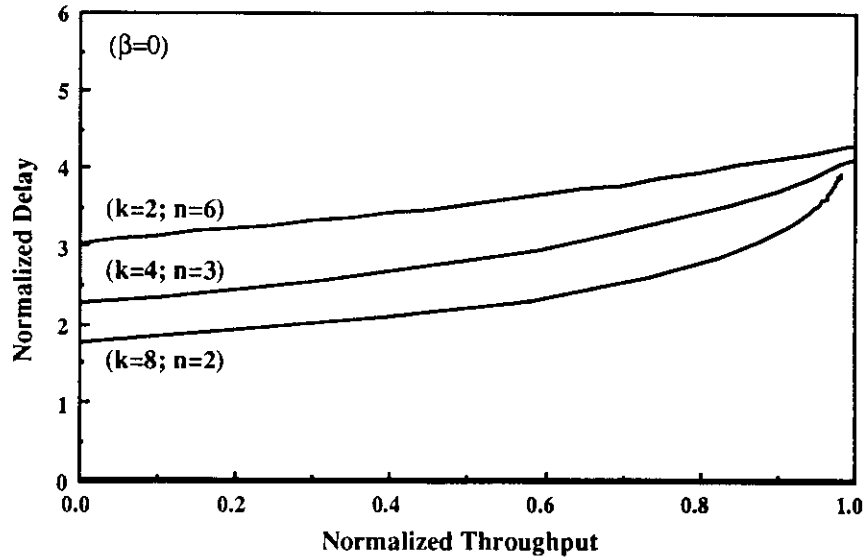


Figure 4.8: Normalized delay as a function of the normalized throughput for a network with 64 nodes when  $\beta = 0$ .

given that  $\beta = 0, 0.25$ , and  $1$ , respectively. We find that, given  $\beta = 0$  (although it is impossible), the lower-dimensional networks perform better than the higher-dimensional networks over a wide range of traffic loads. However, as  $\beta$  grows, then the higher-dimensional networks perform better than the lower-dimensional networks when the traffic loads carried by the network are moderate to high. However, we realize that the lower-dimensional network is easier and cheaper to design and manufacture than the higher-dimensional network.

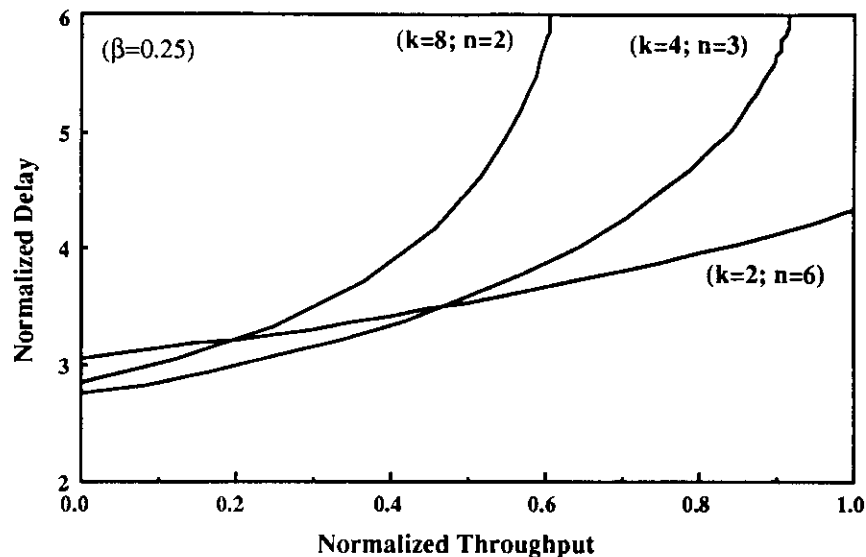


Figure 4.9: Normalized delay as a function of the normalized throughput for a network with 64 nodes when  $\beta = 0.25$ .

## 4.5 Conclusions

In this chapter we developed an approximation for evaluating the performance of a deflection routing algorithm for a very general class of networks called  $k$ -ary  $n$ -cubes with finite buffers. We showed that the throughput of the network does not degrade even when the network is full. We generalized our previous observation that the near-optimal assignment of buffers in each node is “ $2n$ ” for the  $n$ -dimensional network. We realized that, in general, the choice of the optimal network is difficult since the performance of the network is highly dependent on the operation of the network and the traffic loads carried by the network. Building a very high-dimensional network is usually not economical

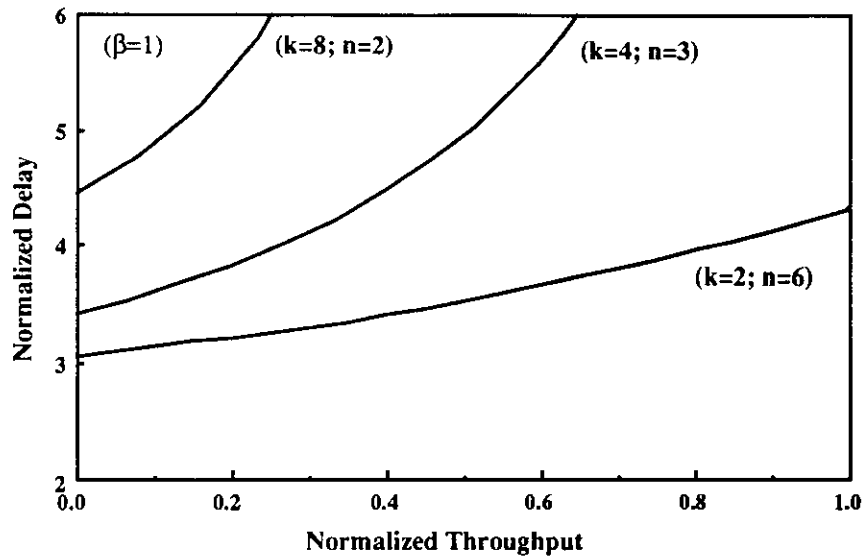


Figure 4.10: Normalized delay as a function of the normalized throughput for a network with 64 nodes when  $\beta = 1$ .

and may not even be possible. However, a network with a very low dimension may not be feasible due to its limitation on communication bandwidth.

## CHAPTER 5

# Fault-tolerant Routing in Boolean $n$ -Cube Networks

### 5.1 Introduction

Although an interconnection network is usually operated in a well-protected environment, faults may occur. When nodes or communication links fail, the regularity of a Boolean  $n$ -cube network is destroyed and the original routing algorithm presented in Figure 1.3 may no longer be applicable. For example, in Figure 5.1, where faulty nodes are drawn as black dots, nodes 0110 and 0101 cannot communicate with each other via an optimal path since all the optimal paths between these two nodes are blocked. (Recall that, based on the original routing algorithm, processors can only communicate by transmitting messages through an optimal path.) However, nodes 0110 and 0101 should be able to communicate with each other via a non-optimal path through nodes 1110, 1100 and 1101. To build a reliable multiprocessor system, the presence of fault-tolerant routing to ensure successful communications between any pair of non-faulty nodes is essential. It is also important to keep the routing algorithm as simple as possible.

Fault-tolerant routing can be achieved by employing the network with spare nodes and links so that when some nodes or links are down, the network can

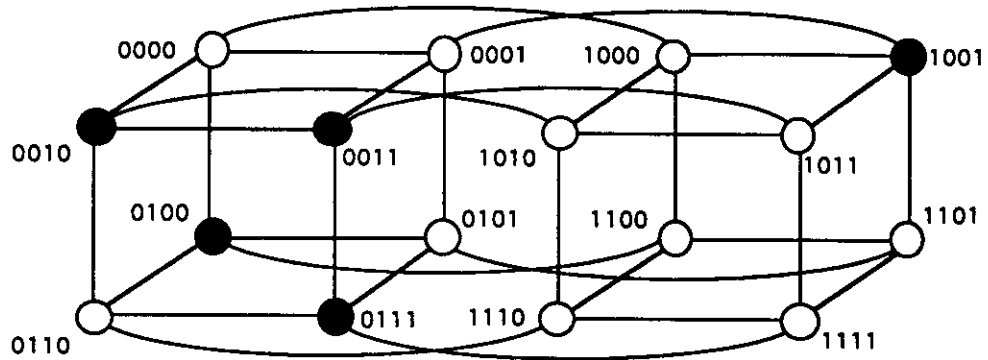


Figure 5.1: A Boolean 4-cube network with node faults.

be reconfigured such that the reconfigured network contains the same structure as the original one [Ren86] [HLN87] [DH91]. Hence, the reconfigured network is able to use the original routing algorithm. However, the drawbacks of these approaches are that the number of faults to be tolerated is usually very limited and the cost of tolerating multiple faults is high.

Fault-tolerant routing can also be achieved by equipping each node or each message with information about the status of network (i.e., the locations of the faults). Algorithms which require each node to know the global status of the network have been reported in [CS89]. One can even assume that the surviving part of the network has an irregular topology, in which case each node maintains a routing table as used in networks such as the ARPANET [MW77] [Kle76]. However, as the network grows in size, the amount of storage space and time needed to maintain the routing tables become prohibitive.

Since a number of faults in a richly-connected Boolean  $n$ -cube network may not destroy its entire regularity, the fault-tolerant routing algorithms should take advantage of the remaining topological regularity. Several algorithms re-

quiring each node to know only the status of its local components have also been presented in [GS88] [LH88] [CS90a]. These algorithms are based on the depth-first search approach. With this approach, one first attempts to route messages in the forward directions. When a message reaches a dead end, i.e. no non-faulty neighbors in the forward directions, the current node sends the message to a non-faulty node in the backward direction in hopes of routing the message through another path. The limitation of this approach is that either the number of hops traversed by a message may grow without bound [GS88] or the total number of faulty components to be tolerated is restricted (e.g., less than  $n$ ) [LH88] [CS90a].

In particular, Chen and Shin [CS90a] developed a set of fault-tolerant routing algorithms which equip each message with a tag to keep track of the path traveled so far, and, hence, avoid visiting a node more than once. To tolerate more than  $n - 1$  faults, a more complicated procedure is required to guide backtracking when a message reaches a dead end. The drawbacks of this approach are that the length of the messages is variable and the computation overhead is not trivial. With their algorithms, to further guarantee that every message is routed to its destination via a shortest path, every node must be equipped with nonlocal status [CS90b].

In this chapter, we first evaluate the effect of faulty nodes on the performance of a Boolean  $n$ -cube network. We then develop a set of fault-tolerant routing algorithms for the Boolean  $n$ -cube network. Our approach to fault-tolerance is to restore the regularity of a damaged network in order to keep the routing algorithms as simple as possible. Basically, our algorithms work for any number of faults as long as the network remains connected.

One way to restore the regularity of a Boolean  $n$ -cube network in the presence of node failures is to simply disable the nodes with more than one bad neighbor. The remaining network is called a "1-degraded subnet." We develop a very simple optimal-path routing algorithm for such a subnet. Although the 1-degraded subnet can easily be constructed, many non-faulty nodes may have to be disabled. We further construct a "convex subnet" in which every pair of surviving nodes is connected with at least one optimal path. We show that the optimal-path routing algorithm also works for the convex subnet, and that only a small number of non-faulty nodes need to be disabled.

To fully preserve the processing power of the network, we also developed a two-level hierarchical fault-tolerant routing scheme without disabling a single node. Here, a non-convex network is decomposed into a set of clusters; each cluster is essentially a convex subcube. Each node maintains a small routing table where each entry of the table corresponds to a destination cluster. Messages must be first routed to their destination clusters using the routing tables and then routed to their destination nodes using the optimal-path routing algorithm. Since no nodes or links are disabled, the rich connection of the network is fully maintained. We show that the increase in the mean path length caused by hierarchical routing is very small.

## 5.2 Degradation of Networks with Node Faults

Let us first present a simple queueing model to evaluate how the performance of a network is degraded by node failures. In the development of the following model, we assume that the failure rate is small and every pair of surviving nodes is connected via an optimal path. Let each node fail independently with



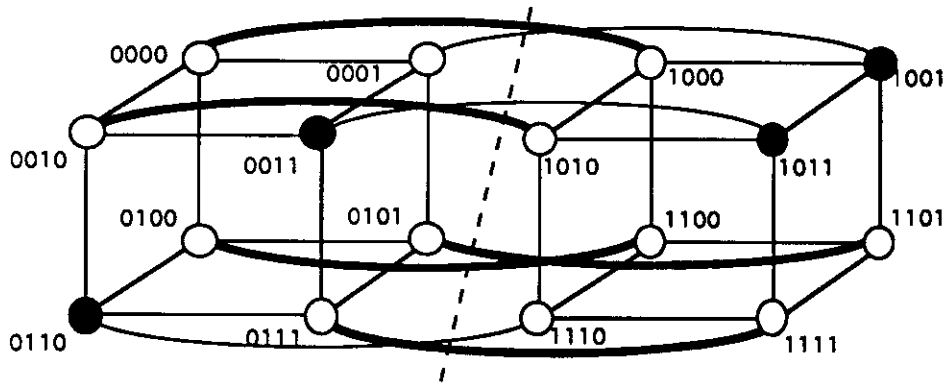


Figure 5.2: A cut in dimension 3. (Surviving links crossing the cut are shown as heavy lines).

probability  $p$ . We also assume that the arrival of input messages to each node follows a Poisson process with a rate of  $\lambda$  messages per unit time; message lengths are random and drawn independently from an exponential distribution. Since a link survives if and only if both nodes at its ends survive, we have

$$\text{Prob}[A \text{ link survives.}] = (1 - p)^2.$$

By “cutting” a Boolean  $n$ -cube network in a particular dimension as shown in Figure 5.2, where the network is cut in dimension 3, we show that the number of links crossing a cut is  $2^{n-1}$ . Thus, the expected number of surviving links crossing a cut is given by

$$2^{n-1}(1 - p)^2. \tag{5.1}$$

For example, in Figure 5.2, there are 5 surviving links, which are shown as heavy lines, crossing the cut.

Note that the expected number of surviving nodes in the network is  $2^n(1 - p)$ .

We further assume that messages are uniformly destined to all other surviving nodes in the network. Thus,

$$\begin{aligned}
 \sigma &\triangleq \text{traffic intensity from a given source} \\
 &\quad \text{to a particular destination} \\
 &= \frac{\lambda}{2^n(1-p) - 1}.
 \end{aligned} \tag{5.2}$$

From Figure 5.2 we note that every message which is generated from a node in the left subcube which is destined to a node in the right subcube must travel over one of the surviving links in order to cross the cut. If the message travels along an optimal path between these two nodes, the message must travel across the cut exactly once. Also, we assume that the traffic crossing this cut is perfectly balanced. Since the average number of surviving nodes in each subcube is  $2^{n-1}(1-p)$  and each will send  $\sigma$  units of traffic to every other surviving node in the network, each node will send  $\sigma[2^{n-1}(1-p)]$  units of traffic across each cut per unit of time. There are  $2^{n-1}(1-p)$  nodes in each subcube doing this, and traffic is balanced on each link. Thus, we have

$$\begin{aligned}
 \rho &= \text{Traffic load per channel} \\
 &= \frac{[2^{n-1}(1-p)]^2 \sigma}{2^{n-1}(1-p)^2} \\
 &= \frac{\lambda}{2(1-p) - 2^{1-n}}.
 \end{aligned} \tag{5.3}$$

Here, we assume that a link is split into two unidirectional channels in opposite directions. Obviously, this model yields an optimistic bound.

We apply Kleinrock's *Independence Assumption* [Kle76] which is often used in the delay analysis of communication networks. This assumption states that each time a message is received at a node within the network, its transmission

time is chosen independently from an exponential distribution. We assume the mean transmission time of a message equals one unit of time. Thus, each channel is modeled as an M/M/1 system with Poisson arrivals at a rate  $\lambda/[2(1-p) - 2^{1-n}]$  and with an exponential service time whose mean is one unit of time. In order for this system to be stable, we require that  $\rho < 1$ , that is,

$$\lambda < 2(1-p) - 2^{1-n}. \quad (5.4)$$

Let  $\Gamma$  be the throughput of the network. Thus,

$$\Gamma = \lambda 2^n (1-p) \quad (5.5)$$

$$< 2[2^n(1-p) - 1](1-p) \quad (5.6)$$

where  $2[2^n(1-p) - 1](1-p)$  is clearly the mean network communication capacity.

The mean message delay is then given by [Kle76]

$$\begin{aligned} T &= \sum_{i=1}^L \frac{1}{\Gamma} \frac{\rho}{1-\rho} \\ &= L \left\{ \frac{1}{\lambda 2^n (1-p)} \right\} \left\{ \frac{\frac{\lambda}{2(1-p) - 2^{1-n}}}{1 - \frac{\lambda}{2(1-p) - 2^{1-n}}} \right\} \end{aligned} \quad (5.7)$$

where  $L$  is the number of surviving channels in the network ( $L = 2^{n-1}(1-p)^2 2n$ ).

Thus,

$$T = \frac{n(1-p)}{2(1-p) - 2^{1-n} - \lambda}. \quad (5.8)$$

For  $n \gg 1$ , we obtain the following approximations:

$$\rho \approx \frac{\lambda}{2(1-p)} \quad (5.9)$$

$$\Gamma < 2^{n+1}(1-p)^2 \quad (5.10)$$

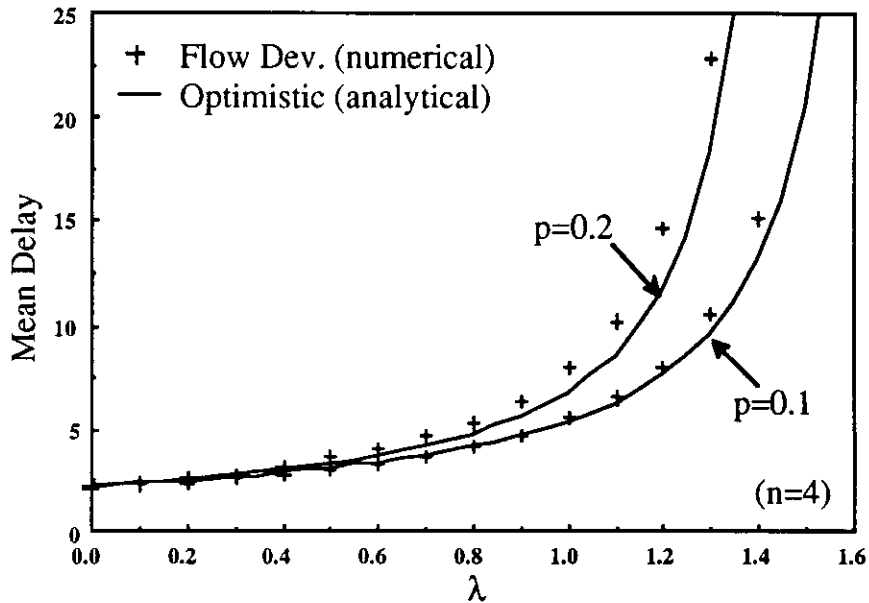


Figure 5.3: Minimal achievable delay of a Boolean 4-cube network with two different failure rates.

$$T \approx \frac{n(1-p)}{2(1-p) - \lambda}. \quad (5.11)$$

In Figure 5.3 we show the mean message delay obtained from our optimistic model for a Boolean 4-cube network with two different failure rates. We also ran a flow deviation program [FGK73] to find the minimal achievable delay. We find that our assumptions are appropriate if failure rates are small.

Moreover, as we have used the technique in previous chapters, we define *power* as the throughput of the network over the mean message delay [Kle79]. A system is said to be operating at an optimal point if the power at that point is maximized. For  $n \gg 1$ , we find that the power is maximized when  $\lambda = 1 - p$

which is equal to half the maximum allowed throughput per node, as found in [Kle79].

### 5.3 Routing in 1-Degraded Subnets

In this section we describe an algorithm to restore the regularity of a damaged Boolean  $n$ -cube network which contains only node failures. In fact, in an interconnection network, since nodes (or processors) are more complex than links, nodes have higher failure rates than links. It is assumed that a faulty node and all links connected to it are effectively removed from the network. Thus, to consider a link failure between two non-faulty nodes, one may simply disable one of these two nodes so that the faulty link is also removed. We will release this assumption in the next section. We further make the following assumptions:

- The remaining network is connected.
- Each node knows the status of its neighboring nodes. The detection of the fault status of a neighboring node can be achieved by requiring each non-faulty node to send a “live” message to all its neighboring nodes periodically. When a node does not receive such a message for a time period, the node knows that this neighboring node has been down.
- A node cannot transmit a message to a faulty neighboring node.
- If a message is found to be destined to a faulty neighboring node, the current node discards the message and then informs the source node of the fact. In the following discussions, we simply assume that a message cannot be destined to a faulty node.

### 5.3.1 The $k$ -Degraded Subnet

A network is said to be  $k$ -degraded if every surviving node in the network has at most  $k$  “bad” (to be discussed) neighbors. A damaged network can easily be made  $k$ -degraded in a distributed manner as follows: Every surviving node (or non-faulty node initially) has a list which gives the status of its neighbors. Every surviving node keeps checking its list and disables itself if it has more than  $k$  “bad” neighbors; in this case, it must inform all its surviving neighbors of the change in its status. The disabling process stops when no nodes further change their status. Here, during each step of the iteration, the “bad” nodes include all faulty nodes and the nodes which have been disabled in previous iterations. This disabling process can be executed by all nodes in parallel. We call the remaining network a “ $k$ -degraded subnet.” In the following discussion, we will focus on 1-degraded subnets.

A similar algorithm is found in [LH88], in which the communication complexity of the algorithm was shown to be  $O(n^3)$ . Given that a node has the capability of simultaneously transmitting and receiving multiple messages to and from its neighbors, then the time required by the algorithm to converge is  $O(n^2)$ . It has also been shown that the least number of faulty nodes needed to disable the entire Boolean  $n$ -cube network is  $\lceil n/2 + 1 \rceil$  [LH88].

### 5.3.2 The Optimal-path Routing Algorithm

We now present a very simple adaptive routing algorithm for 1-degraded subnets. We let *neighbor\_status* be an  $n$ -bit binary number in which a bit is set to one if its corresponding neighbor is surviving. Otherwise, the bit is reset to zero. This routing algorithm is shown in Figure 5.4, where “&” is a bit-wise

```
When a message is received,  
if ( header = 0 )  
    Send the message to the local processor.  
else  
    valid_channels <- header & neighbor_status.  
    Randomly select a 1-bit from valid_channels.  
    Change the corresponding 1-bit in the header to 0.  
    Send the message over the selected channel.
```

Figure 5.4: The optimal-path routing algorithm for 1-degraded subnets

AND function. Note that this algorithm is very simple and requires each node to know only its neighboring status. This algorithm is called the optimal-path routing algorithm since every message of the network is routed to its destination along an optimal path.

The proof that this routing algorithm works for 1-degraded Boolean  $n$ -cube subnets is as follows: If a node receives a message with more than one one-bit in its header, since every surviving node has at most one bad neighbor, the node surely can find a valid dimension (or channel) to transmit the message. If the message has only a single one-bit in its header, then the corresponding neighbor must be surviving. Otherwise, the message is destined to a faulty node; in this case, the message must be discarded.

### 5.3.3 Discussion

This approach works well in the situation where a whole cluster of nodes has been “bombed out.” As an example of such spatially correlated faults, one may consider a power-supply failure which disables the entire cluster of nodes

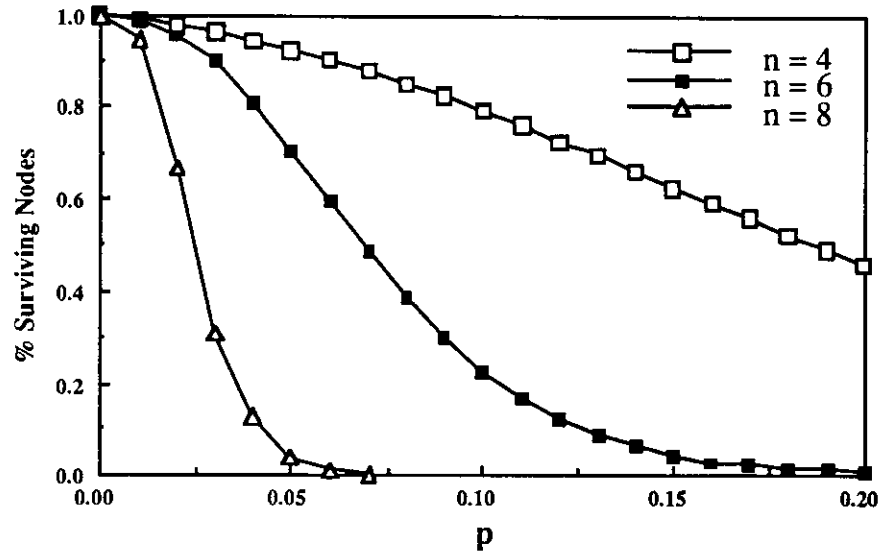


Figure 5.5: Percentage of surviving nodes in the 1-degraded subnets.

supported by it. Another example is that of an enemy projectile scoring a hit on such a network interconnecting processors on a ship, aircraft, tank, etc.

The disadvantage of this approach for constructing 1-degraded subnets is that, in a large Boolean  $n$ -cube network with a large failure rate, since every node has a large number of neighbors, the probability that a node has more than one bad neighbor can be large. A disabled node also increases the probability of its neighbors being disabled. As a result, many nodes may have to be disabled; hence, the computational power of the system is significantly reduced. Figure 5.5 shows, for networks of different sizes, the percentage of nodes that remain after the disabling iteration in the simulation settles down, given that node faults occur independently and randomly with a given probability.



## 5.4 Routing in Convex Subnets

In this section we develop a heuristic algorithm to construct a “convex” subnet in an attempt to save more non-faulty nodes than in the 1-degraded subnet. A Boolean  $n$ -cube network (or subnet) is said to be convex if and only if every pair of surviving nodes is connected with at least one optimal path. (Here, we borrow the word “convex” from geometry, where a contour is convex if and only if every point inside this contour can be “seen directly” from any other points of the contour.) We show that the optimal-path routing algorithm for 1-degraded subnets also works for convex subnets.

### 5.4.1 The Convex Subnet

Boolean  $n$ -cube networks have the following topological properties: A node and its  $k$  links can uniquely address a  $k$ -subcube which contains the node and these links, where  $0 \leq k \leq n$ . Note that such a  $k$ -subcube is the minimal subcube containing the node and the links. For example, in a Boolean 4-cube network, the node 0110 and the links in dimensions 0 and 1 address the subcube 01XX. Moreover, any two nodes which are  $k$  hops away in distance can uniquely identify a Boolean  $k$ -subcube.

We assume that there is a central control unit which collects information from every surviving node of the network and makes decisions about how to disable a node. Here is a heuristic algorithm for constructing a convex subnet: We let  $List[i]$  be a check-list which contains all surviving nodes with  $i$  bad links. Here, the “bad” links include the faulty links and the links connecting to the faulty and disabled nodes. Nodes on  $List[i]$  have higher priority for disablement

than any other nodes on  $List[j]$ , if  $i > j$ . That is, the node with the most bad links (i.e., the worst connection) has the highest priority of being disabled.

We choose a node, say node  $\alpha$ , from the highest priority non-empty check-list, say in  $list[k]$ , and identify the  $k$ -subcube containing node  $\alpha$  and all its  $k$  bad links. For example, in Figure 5.1, node 0110 and its bad links in dimensions 0, 1, and 2 identify the 3-subcube 0XXX. It is clear that without routing through the links outside the subcube, node  $\alpha$  cannot communicate with any other surviving nodes of the subcube. We choose to disable the smaller side of the subcube. Thus, if there are more than two surviving nodes in the subcube, node  $\alpha$  is disabled. If there are exactly two surviving nodes in the subcube, we choose to disable either one of them which has less bad nodes in its neighboring area. If node  $\alpha$  is the only surviving node in the subcube, it is safe and removed from the check-list. A safe node may be brought into the check-lists again if any of its neighbors is disabled later. The algorithm stops when every surviving node is safe. Our simulations show that the number of surviving nodes remaining in the convex subnet constructed by our heuristic algorithm is very close to that obtained by an exhaustive search.

Figure 5.6 illustrates a convex subnet constructed for the damaged Boolean 4-cube network as shown in Figure 5.1. Note that all links connected to a faulty node or a disabled node are also disabled. To consider link failures, in Figure 5.7, we show a convex subnet constructed for a network which contains a faulty link between nodes 1001 and 1101. Clearly, our algorithm works for link failures.

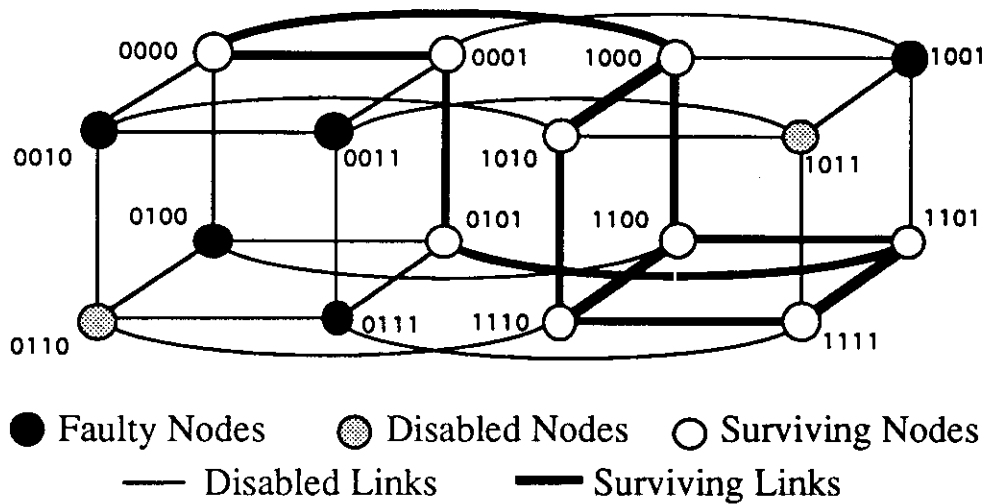


Figure 5.6: A convex subnet constructed for the damaged Boolean 4-cube network as shown in Figure 5.1.

#### 5.4.2 The Routing Algorithm

We now prove that the optimal-path routing algorithm developed in the previous section for 1-degraded subnets also works for convex subnets. Suppose that the destination node of a message is  $k$  hops away from the node where the message is currently residing. A  $k$ -subcube can be constructed to contain the current node and the destination node. Since the current node and the destination node are connected via at least one optimal path, the current node has at least one surviving link leading to the destination node. Such a surviving link corresponds to a valid dimension of the message. Thus, the current node can always find a valid dimension to transmit the message. Applying the same argument to every intermediate node, we show that every message in a convex subnet is forwarded to its destination by our optimal-path routing algorithm.

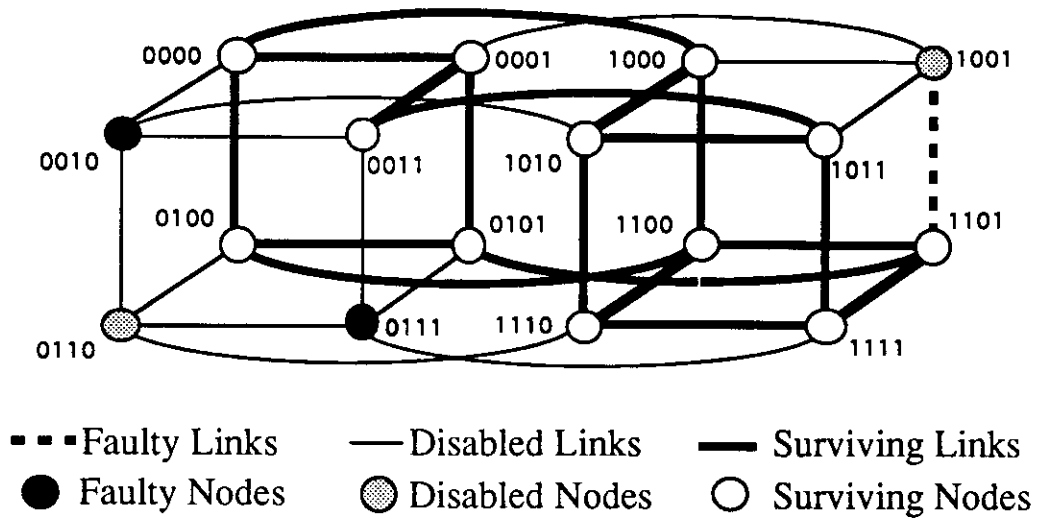


Figure 5.7: A convex subnet constructed for a network containing a faulty link.

### 5.4.3 Discussion

Figure 5.8 shows the percentage of nodes which remain in the convex subnet, given that nodes initially fail independently and randomly with a given probability. Comparing this with the percentage of nodes which remain in the 1-degraded subnet, we find that the number of surviving nodes is dramatically increased. In Figure 5.9, we compare these two disabling schemes by showing the percentage of surviving nodes in a Boolean 8-cube network.

It is very difficult to analytically evaluate the performance of arbitrary networks in a dynamic traffic environment. To verify the effectiveness of our routing algorithm, we extensively simulated the optimal-path routing algorithm in the convex subnet. We also ran a flow deviation program to find the minimal achievable delay. These results are also compared with the optimistic bound obtained in Section 5.2. Figure 5.10 shows, for different input rates, the mean

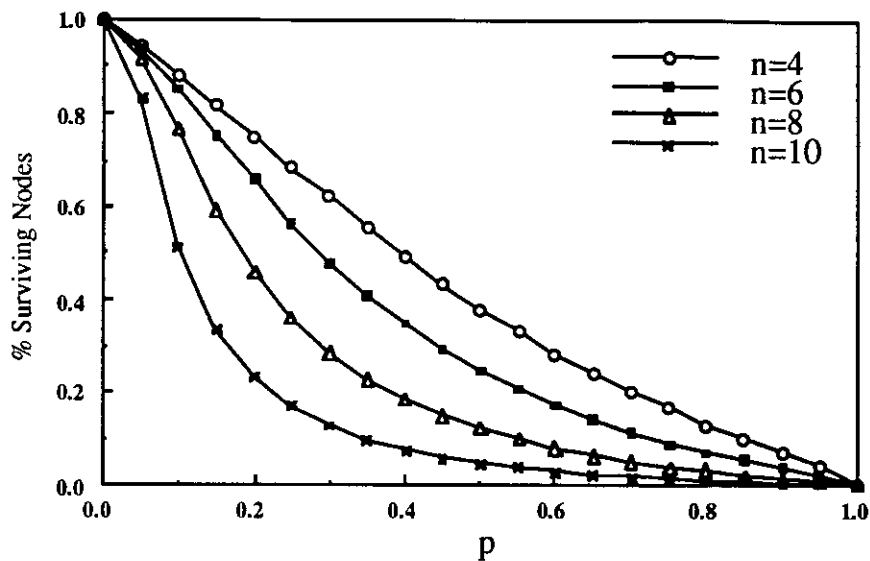


Figure 5.8: Percentage of nodes remaining in the convex subnets.

message delay in the convex subnet of a Boolean 6-cube network. The results with 95% confidence shown here were from 100 randomly generated patterns, each containing 6 faulty nodes. We find that the mean message delay is very close to the minimal achievable bound, which is also very close to the optimistic bound.

## 5.5 Routing in Two-level Hierarchical Networks

In this section, without disabling any non-faulty node, we restore the regularity of a damaged Boolean  $n$ -cube network by decomposing the network into a set of clusters; each cluster is essentially a subcube with the convexity property. i.e., every pair of the surviving nodes of a cluster is connected with at least one

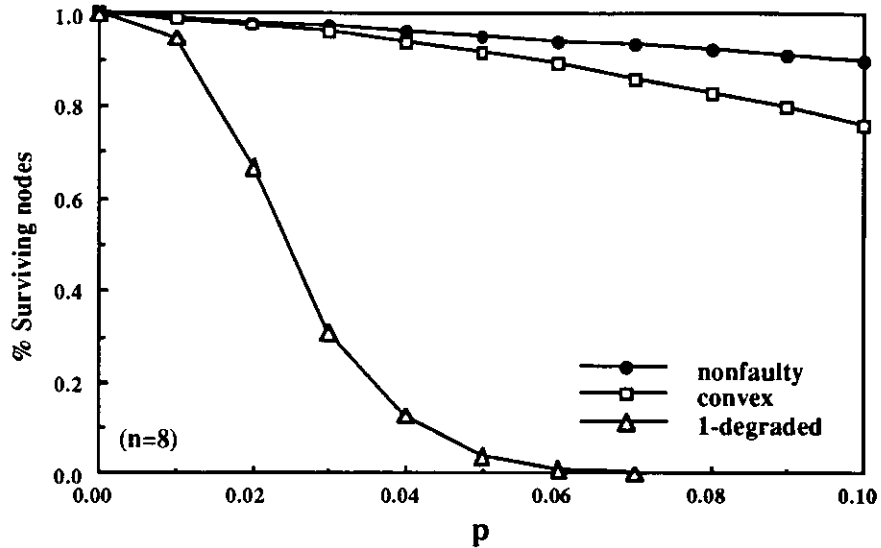


Figure 5.9: Comparison of percentage of surviving nodes in a Boolean 8-cube network.

optimal path. A *two-level hierarchical* routing algorithm, which requires every node to maintain a small routing table, is then developed.

### 5.5.1 Network Decomposition

In this section, a surviving node is said to be *unsafe* if the minimal subcube which contains the node and its all bad links has more than one surviving node. Recall that the bad links include the faulty links and the links connecting to the faulty nodes. Clearly, a node without any bad link is safe. A cube is not convex if it contains at least one unsafe surviving node.

A non-convex Boolean  $n$ -cube network is decomposed into a set of clusters

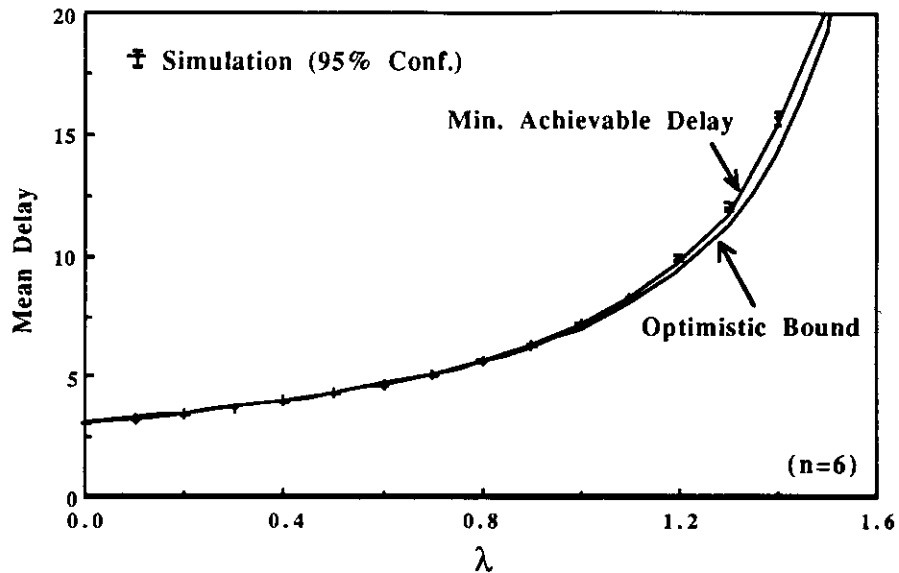


Figure 5.10: Mean delay for the convex subnets of the Boolean 6-cube networks with 6 faulty nodes.

as follows: For all unsafe surviving nodes, the central control unit counts the number of bad links in each dimension. The central control unit then chooses the dimension having the most bad links and cuts the network into two clusters along this dimension; each cluster is an  $(n - 1)$ -subcube. The algorithm for decomposing a Boolean  $n$ -cube network is shown in Figure 5.11. A subcube needs to be further decomposed if it is not convex.

As an example, let us examine the Boolean 4-cube network with 5 faulty nodes as shown in Figure 5.1. Clearly, the network is not convex. Table 5.1 shows the bad link dimensions for each unsafe surviving node. In this example, dimension 1 has the the maximum number of bad links (i.e., 5). We decompose

```

for  $i=0$  to  $n-1$ ,  $bad\_links[i] <- 0$ .
for each unsafe surviving node of the cube,
  for each bad link of the node, say in dimension  $i$ ,
     $bad\_links[i] = bad\_links[i] + 1$ .
if (  $bad\_links[i] = 0$ , for all  $i$  ) stop.
else
  Choose a dimension  $k$  such that, for all  $i$ ,
     $bad\_links[k] \geq bad\_links[i]$ .
  Decompose the cube into two subcubes in dimension  $k$ .

```

Figure 5.11: The algorithm for decomposing a Boolean  $n$ -cubenetwork.

the network along dimension 1. As a result, the network is separated into two subcubes  $XX0X$  and  $XX1X$ . In this case, fortunately, both of these two subcubes are convex. Thus, the decomposition process stops. The two-level hierarchical network is shown in Figure 5.12.

### 5.5.2 The Two-level Hierarchical Routing Algorithm

Every surviving node maintains a cluster routing table with one entry for each destination cluster. Each entry gives the address of a destination cluster, the best outgoing channel for that cluster, and a relative weight (usually delay is used as the weight). Any algorithm (e.g., the ARPANET-like algorithm) can be used to maintain the cluster routing table.

Messages must first be routed to their destination clusters. When a node receives a message, if the destination node of the message does not belong to the cluster in which the node resides, the message is sent to a neighbor based on the node's cluster routing table. Otherwise, the message is further



node	bad dimensions
0000	1 2
0001	1 3
0101	0 1
0110	0 1 2
1011	1 3

Table 5.1: Bad dimensions for the unsafe surviving nodes of the Boolean 4-cube network as shown in Figure 5.1.

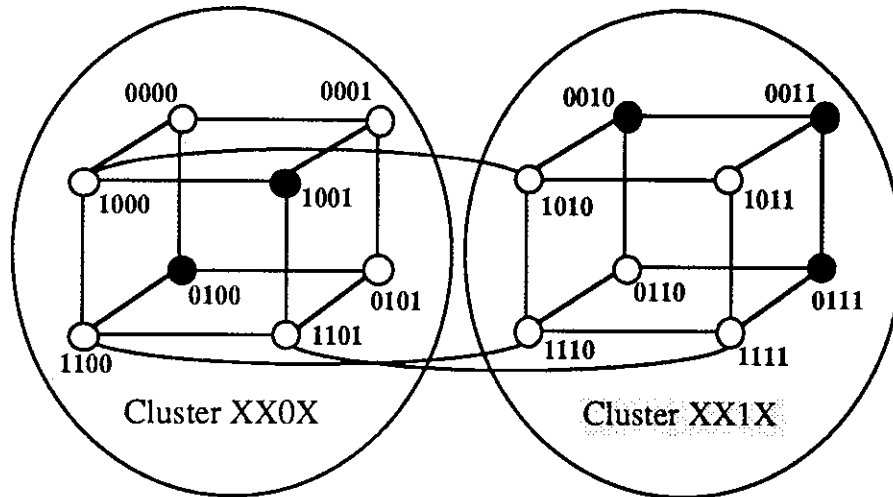


Figure 5.12: A two-level hierarchical structure of the Boolean 4-cube network as shown in Figure 5.1.

routed to its destination node by using the optimal-path routing algorithm. To exploit the possible multiple paths from one node to another and balance the network's traffic, the cluster routing tables should be updated periodically. We may further improve the performance by providing multipath routing, where each entry of the routing table gives multiple choices of outgoing channels. Also, after a message has arrived at its destination cluster, if there exist multiple valid dimensions, the message should be transmitted along the most lightly loaded outgoing channel.

### 5.5.3 Discussion

The two-level hierarchical routing approach has the following advantages:

- No good links or nodes are eliminated. The processing power of the non-faulty part of the network is fully maintained.
- The number of clusters generated by our decomposition algorithm is typically small. The size of the routing table is significantly reduced from the ARPANET-like routing table. Given that there are only node failures, the number of clusters cannot exceed the number of faulty nodes in the network. The reason is that every cluster must contain at least one faulty node. Otherwise, this cluster must be combined with another cluster to form a larger cluster. For example, the mean number of clusters of a Boolean 6-cube network with a given number of faulty nodes is shown in Figure 5.13.
- The optimal-path routing algorithm developed for 1-degraded subnets works for each cluster. Only the message to be sent to another cluster needs to compute its next node by using the routing table.
- By carefully maintaining the routing table, the number of hops traversed by a message is bounded.
- The increase in the mean path length caused by hierarchical routing is very small. In Figure 5.14, for a Boolean 6-cube network, we show the mean path length of the two-level hierarchical networks and that of the original networks.

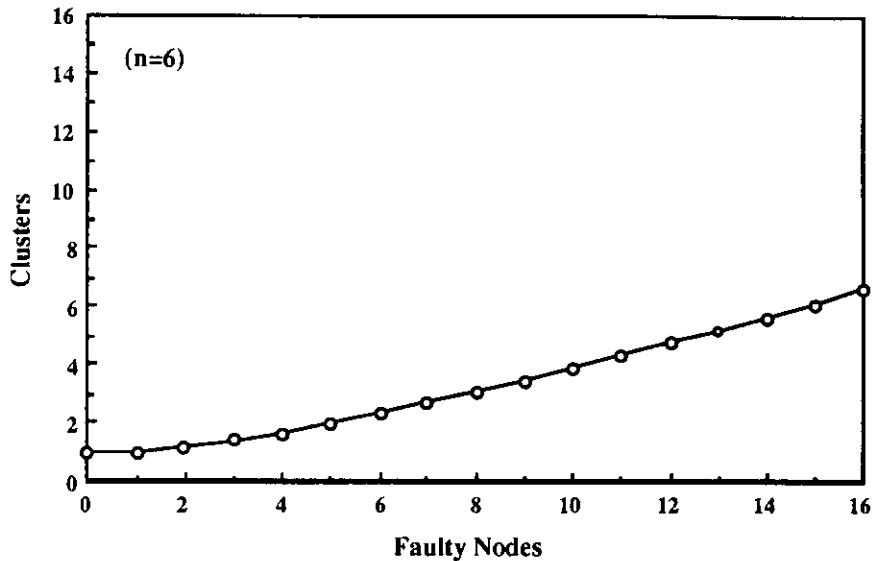


Figure 5.13: Mean number of clusters generated by our decomposition algorithm for a Boolean 6-cube network.

## 5.6 Conclusions

In this chapter, we first developed a queueing model to evaluate the degradation of a damaged Boolean  $n$ -cube network with node faults. We next developed an adaptive fault-tolerant routing algorithm for 1-degraded subnets. This algorithm is very simple; it makes routing decisions based only on the node's local status. This algorithm routes every message to its destination via an optimal path.

We further exploited the remaining regularity of a damaged Boolean  $n$ -cube network and developed a heuristic algorithm to construct a convex subnet in which every pair of surviving nodes is connected with at least one optimal path.

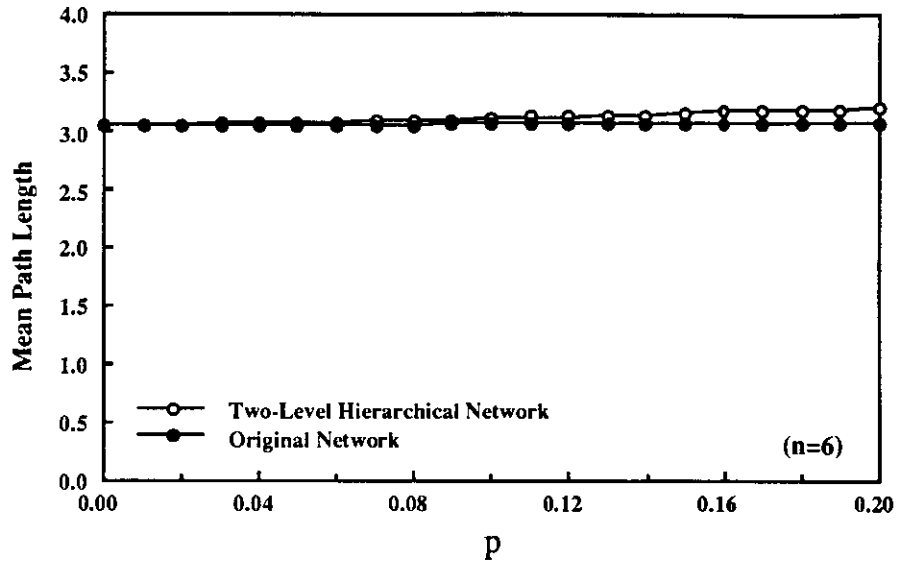


Figure 5.14: Comparison of the mean path length of the two-level hierarchical network and the original network.

We showed the algorithm used in a 1-degraded subnet also works for a convex subnet. Only a small number of non-faulty nodes must be disabled. The performance of the optimal-path routing algorithm in convex subnets was studied. We found the mean message delay is very close to the minimal achievable bound.

To preserve all the non-faulty nodes in the network, we also developed a two-level hierarchical routing scheme. A damaged Boolean  $n$ -cube network is decomposed into a set of clusters; each is essentially a convex subcube. Every surviving node in the network is required to maintain a small routing table. A two-level hierarchical routing algorithm was then developed. Without disabling a single non-faulty node, this approach maintains the network's rich

connections. We showed that the increase in the mean path length caused by hierarchical routing is very small.

## CHAPTER 6

### Conclusions and Future Research

This dissertation proposed a deadlock-free routing algorithm for the  $k$ -ary  $n$ -cube network with finite buffers and several fault-tolerant routing algorithms for the Boolean  $n$ -cube network, and analyzed their performance via mathematical models, approximations, and simulation.

Chapter 2 presented several basic models for evaluating the performance of the Boolean  $n$ -cube network. Performance bounds were also examined. We showed that the mean delay of our routing algorithm with random message assignment is close to an optimistic lower bound.

In Chapters 3 and 4, we developed and analyzed a deadlock-free routing algorithm with deflection for  $k$ -ary  $n$ -cube networks with finite buffers. We intensively studied the Boolean  $n$ -cube network in Chapter 3. With our algorithm, all messages entering the network are delivered to their destination with loss or deadlocks. We showed that the throughput of the network never degrades, and that a small number of buffers in a node is sufficient to deliver good performance. We generalized our model in Chapter 4 and found that the assignment of  $2n$  buffers in a node is a good approximation to the “optimal” choice for a  $k$ -ary  $n$ -cube network. We also analyzed the networks with various combinations of dimension  $n$  and radix  $k$ . We found that the choice of the optimal network is difficult since the performance of the network is highly

dependent on the operation of the network and the traffic loads carried by the network. Building a very high-dimensional network is usually not economical or may not be possible. However, a network with a very low dimension is limited by its communication bandwidth.

Several fault-tolerant routing schemes based on the idea of restoring the regularity of a damaged Boolean  $n$ -cube network were developed in Chapter 5. One way to restore the regularity of a network in the presence of only node failures is to simply disable the nodes with more than one bad neighbor. The remaining network is called a "1-degraded subnet." A very simple optimal-path routing algorithm was developed for such a subnet. This approach works well in the situation where a whole cluster of nodes have been "bombed out." However, many nonfaulty nodes may have to be disabled. We further developed a heuristic algorithm to construct a "convex subnet" in which every pair of surviving nodes is connected with at least one optimal path. This approach considers both node and link failures. We show that the optimal-path routing algorithm also works for the convex subnet, and that only a small number of nonfaulty nodes need to be disabled. We also developed a two-level hierarchical fault-tolerant routing scheme without disabling any nodes. With this approach, the rich connectivity of the network is fully maintained. We showed that the increase in the mean path length caused by hierarchical routing is typically very small.

## 6.1 Future Work

In this section, we describe a few possible future research areas.



### 6.1.1 Virtual Cut-through with Deflection

The virtual cut-through switching scheme was first proposed by Kermani and Kleinrock in 1979 [KK79]. In this switching scheme, as soon as the header of a message is received by an intermediate node, the node can determine on which channel to transmit the message. If the selected channel is free, then the message can be sent out of the node before it is completely received. Hence, the delay due to unnecessary buffering in front of an idle channel can be avoided. A message is buffered in an intermediate node only when the selected channel is busy.

It is interesting to analyze the virtual cut-through switching scheme for the Boolean  $n$ -cube network with the deflection technique. In the following discussion, messages are assumed to be of fixed length. We further assume that a message is divided into  $m + 1$  segments; the first segment is the header of the message. Moreover, a cycle of time is divided into  $n + m$  mini-cycles. A mini-cycle consists of the time to process and transmit a message segment to its neighbor.

The routing algorithm works as follows: The major role of the first  $n$  mini-cycles is to establish the connection. In each of the first  $n$  mini-cycles, say  $k$ , every node examines all the headers it has to see if there is any message with the  $k$ th bit set, and, if there is any, to move one of them along the  $k$ th outgoing channel of the node. Whenever a header is successfully assigned to a channel, the rest of the message follows the header and moves to the neighbor in the following  $m$  mini-cycles.

The deflection technique is used when all buffers are full. Let  $M$  be the buffer size in each node. In the  $k$ th mini-cycle,  $0 \leq k \leq n$ , if there are  $M$  headers

inside a node and the node cannot find any bit set in the  $k$ -th dimension, the node has to force out one message across the  $k$ -th channel. In this case, the message is deflected.

We note that every channel can only be requested by a node once in a cycle. The  $k$ -th channel of a node must be free before the  $k$ -th mini-cycle. Thus, in each of the first  $n$  mini-cycles, each node has the potential to send a header to its neighbor along the corresponding channel. However, the node cannot send headers to its neighbors after the first  $n$  mini-cycles even if the node finds some free output channels. Otherwise, a node might be full and use up all its channels after the first  $n$  mini-cycles. In this case, a message from a neighbor would be dropped which is not desirable. Thus, the last  $m$  mini-cycles are solely for transmission.

When the network is heavily loaded, messages are likely to be blocked in intermediate nodes and deflected on a longer path. Since the virtual cut-through switching scheme consists of  $m + n$  mini-cycles in a cycle, its cycle time is longer than that of the store-and-forward scheme which consists of only  $m + 1$  mini-cycles. Thus, when traffic loads are high and messages are short (i.e.  $m$  is small), the virtual cut-through scheme can perform worse than store-and-forward. The development of a model to analyze the performance of the virtual cut-through switching scheme with deflection in  $k$ -ary  $n$ -cube networks would be of interest.

### 6.1.2 Communication Locality

Our performance models assumed that the traffic is uniformly distributed over the network. However, communication locality may exist and, if so, could be

exploited so that the performance of the multiprocessor systems could be improved. In general, communication locality depends on several factors such as the characteristics of the application algorithm, the interconnection network, the compiler, and the operating system. Both the throughput and latency of the networks can be significantly improved if communication locality can be exploited. One of the reasons why the lower-dimensional networks perform worse than the higher-dimensional networks is that the mean distance between two nodes in the lower-dimensional networks is larger than that in the higher-dimensional networks. By exploiting communication locality, we may significantly improve the performance of a low-dimensional network. We would like to see how the choice of the optimal interconnection network is affected by locality.

### **6.1.3 Dynamic Restoration of Regularity**

It is likely that a nonfaulty node goes down or a faulty node is repaired and return to service after the convex subnet has been created. Clearly, reconstruction of a new convex subnet based on the original one can be easier and faster than reconstruction without this information. We expect that the run-time reconstructed one keeps as many nonfaulty nodes as possible. The reconstruction of a two-level hierarchical network is also interesting. A faulty node may further separate a cluster into two clusters, and a repaired node may result in the join of the two clusters. In both case, the two-level hierarchy and the routing table in each node must be updated.

#### 6.1.4 Fault-tolerant Routing for $k$ -Ary $n$ -Cube Networks

Our fault-tolerant routing algorithms for Boolean  $n$ -cube networks exploit the network's topological properties. Although the concept of restoring the regularity of a damaged network works very well for the Boolean  $n$ -cube networks, these algorithms cannot be directly applied to  $k$ -ary  $n$ -cube networks. For example, in Figure 4.1(b), node (1,2) has no obvious way to send messages to node (1,0) if node (1,1) is faulty. We need to develop a general rule for restoring the regularity of a damaged  $k$ -ary  $n$ -cube network.

### 6.2 Final Remarks

The choice of the optimal interconnection network for a multiprocessor is highly sensitive to the assumptions about the operation of the network, the constraints that apply to the design, and the applications. We have provided several models and algorithms to understand and improve the performance of interconnection networks, but there remains much work to be done.

# APPENDIX A

## Search for Zeroes in the Optimistic Model

In this Appendix, we prove that there are exactly  $n$  simple zeroes inside and on the closed contour  $|z| = 1$  and one zero in  $|z| > 1$  for the denominator of Equation (2.66) in Section 2.5. We assume that  $\lambda > 0$ . Let

$$f(z) = (1 - \alpha)(pz + q)^n, \quad (\text{A.1})$$

and

$$g(z) = z^n(1 - \alpha z), \quad (\text{A.2})$$

where

$$p = \frac{\lambda}{n}(d - 1),$$

and

$$q = 1 - p.$$

Let

$$z = (1 + \delta)\cos\theta + i(1 + \delta)\sin\theta,$$

where  $\delta > 0$  is sufficiently small. We then have the following equations:

$$\begin{aligned} |z^n| &= (1 + \delta)^n & (\text{A.3}) \\ |1 - \alpha z| &= |1 - \alpha(1 + \delta)\cos\theta - i\alpha(1 + \delta)\sin\theta| \\ &= \left[ (1 - \alpha(1 + \delta)\cos\theta)^2 + (\alpha(1 + \delta)\sin\theta)^2 \right]^{1/2} \\ &= \left[ 1 + \alpha^2(1 + \delta)^2 - 2\alpha(1 + \delta)\cos\theta \right]^{1/2} \end{aligned}$$

$$> 1 - \alpha(1 + \delta) \quad (\text{A.4})$$

and

$$\begin{aligned} |(pz + q)^n| &= |pz + q|^n \\ &= |p(1 + \delta) \cos\theta + q + ip(1 + \delta) \sin\theta|^n \\ &= \left[ (p(1 + \delta) \cos\theta + q)^2 + (p(1 + \delta) \sin\theta)^2 \right]^{n/2} \\ &= \left[ p^2(1 + \delta)^2 + q^2 + 2pq(1 + \delta) \cos\theta \right]^{n/2} \\ &\leq [p(1 + \delta) + q]^n \\ &= (1 + p\delta)^n \end{aligned} \quad (\text{A.5})$$

We want to show, when  $\delta > 0$  is sufficiently small, that

$$(1 + \delta)^n [1 - \alpha(1 + \delta)] > (1 - \alpha)(1 + p\delta)^n$$

Let

$$h(x) = (1 + x)^n [1 - \alpha(1 + x)] - (1 - \alpha)(1 + px)^n. \quad (\text{A.6})$$

Obviously,

$$h(0) = 0. \quad (\text{A.7})$$

Differentiating Equation (A.6), we have

$$h^{(1)}(x) = n(1 + x)^{n-1} - \alpha(n + 1)(1 + x)^n - (1 - \alpha)np(1 + px)^{n-1}.$$

Replacing  $\alpha$  and  $p$  by  $1/(1 + \lambda)$  and  $\lambda(d - 1)/n$ , respectively, and letting  $x = 1$ , we have

$$\begin{aligned} h^{(1)}(0) &= n - (n + 1)\alpha - np(1 - \alpha) \\ &= n - (n + 1)\frac{\lambda}{1 + \lambda} - n\frac{\lambda}{n}(d - 1)\frac{1}{1 + \lambda} \\ &= \frac{1}{1 + \lambda} [n - \lambda d] \\ &> 0. \end{aligned} \quad (\text{A.8})$$

From Equations (A.7) and (A.8), we have

$$h(\delta) > 0, \quad (\text{A.9})$$

where  $\delta$  is sufficient small and  $\delta > 0$ . That is,

$$(1 + \delta)^n [1 - \alpha(1 + \delta)] > (1 - \alpha)(1 + p\delta)^n$$

Therefore, given that  $z = (1 + \delta)\cos\theta + i(1 + \delta)\sin\theta$ , we have the following relations:

$$\begin{aligned} |g(z)| &= |z^n(1 - \alpha z)| \\ &= |z^n| |(1 - \alpha z)| \\ &> (1 + \delta)^n [1 - \alpha(1 + \delta)] \\ &> (1 - \alpha)(1 + p\delta)^n \\ &> (1 - \alpha) |pz + q|^n \\ &= |(1 - \alpha)(pz + q)^n| \\ &= |f(z)| \end{aligned} \quad (\text{A.10})$$

We may now apply Rouché's theorem [Tit39], which states that if  $f(z)$  and  $g(z)$  are functions of  $z$ , analytic inside and on a closed contour  $C$ , and if  $|f(z)| < |g(z)|$  on  $C$ , then  $g(z)$  and  $g(z) + f(z)$  have the same number of zeroes inside  $C$ . In our case, the contour  $C$  is  $|z| = 1 + \delta$ , for a sufficiently small  $\delta > 0$ . But,  $g(z)$  has exactly  $n$  zeroes at  $z = 0$  and a zero at  $z = 1/\alpha$ . We note that since  $\alpha = \lambda/(1 + \lambda)$ , thus  $1/\alpha = 1 + 1/\lambda$ . Also, since  $\lambda < 2$  for the system to be stable, we can always find a sufficiently small  $\delta > 0$  such that  $(1 + \delta) < 1/\alpha$ . That is,  $z = 1/\alpha$  is a zero outside the circle  $|z| = 1$ . Therefore,  $f(z) + g(z)$  has exactly  $n$  zeroes inside and on the closed contour  $|z| = 1 + \delta$  and one zero outside

the contour. We let  $z = 0$ , identified by  $z_0$ , be a zero for  $f(z) + g(z)$ , and let other zeroes inside and on the contour  $|z| = 1$  be  $z_j$ , where  $j = 1, 2, \dots, n - 1$ .

We next prove that all the zeroes  $z_0, z_1, \dots, z_{n-1}$  are simple. For if it were not so, we should have, for some  $z$ ,

$$z^n(1 - \alpha z) = (1 - \alpha)(pz + q)^n$$

and

$$nz^{n-1} - \alpha(n+1)z^n = (1 - \alpha)np(pz + q)^{n-1}.$$

After some calculations, we find that such a  $z$  must satisfy the equation

$$p\alpha z^2 + (n+1)q\alpha z - nq = 0. \quad (\text{A.11})$$

Let

$$a = p\alpha$$

$$b = (n+1)q\alpha$$

$$c = -nq.$$

One of the zeroes for Equation (A.11) is

$$z' = \frac{-b + \sqrt{b^2 - 4ac}}{2a}.$$

If  $z' \leq 1$ , then

$$b^2 - 4ac \leq (2a + b)^2$$

Thus,

$$a + b + c \geq 0$$

Replacing  $a$ ,  $b$ , and  $c$  by  $p\alpha$ ,  $(n+1)q\alpha$ , and  $-nq$ , respectively, we need

$$p\alpha + (n+1)q\alpha + nq \geq 0$$



That is,

$$\alpha \geq nq(1 - \alpha) \tag{A.12}$$

We further replace  $\alpha$  and  $q$  by  $\lambda/(1 + \lambda)$  and  $1 - \lambda(d - 1)/n$ , respectively. To satisfy Equation A.12, we need  $\lambda d \geq n$ . This is in contradiction to our requirement of the system being stable. Thus, we have that every  $0 \leq z_i \leq 1$  is simple.

Now, let

$$z'' = \frac{-b - \sqrt{b^2 - 4ac}}{2a}.$$

Obviously,  $z'' < 0$ . If  $z'' \geq -1$ , then we need

$$\begin{aligned} b^2 - 4ac &\leq 4a^2 - 4ab + b^2 \\ &< 4a^2 + 4ab + b^2. \end{aligned}$$

But we have shown that  $b^2 - 4ac > 4a^2 + 4ab + b^2$ . Thus, each  $-1 \leq z_i \leq 0$  is also simple.

## REFERENCES

- [Agr91] A. Agrawal, "Limits on Interconnection Network Performance," *IEEE Transactions on Parallel and Distributed Systems*, vol. 2, pp. 398-412, October 1991.
- [AP89] S. Abraham and K. Padmanabham, "Performance of Direct Binary  $n$ -Cube Networks for Multiprocessors," *IEEE Transactions on Computers*, vol. C-38, pp. 1000-1011, July 1989.
- [AS88] W. C. Athas and C. L. Seitz, "Multicomputers: Message-Passing Concurrent Computers," *IEEE Computers*, pp. 9-24, August 1988.
- [BA84] L. N. Bhuyan and D. P. Agrawal, "Generalized Hypercube and Hyperbus Structures for a Computer Network," *IEEE Transactions on Computers*, vol. C-33, pp. 323-333, April 1984.
- [Bar64] P. Baran, "On Distributed Communication Networks," *IEEE Transactions on Communication Systems*, Vol. CS-12, pp. 1-9, March 1964.
- [BBG87] J. Balzewicz, J. Brzezinski and G. Gambosi, "Time-Stamp Approach to Store-and-Forward Deadlock Prevention," *IEEE Transaction on Communications*, vol. COM-35, pp. 490-495, May 1987.
- [Bor88] S. Borkar *et al.*, "iWarp: An Integrated Solution to High-speed Parallel Computing," in *Proceedings of Supercomputers 88*, November 1988.
- [BP89] H. G. Badr and S. Podar, "An Optimal Shortest-Path Routing Policy for Network Computers with Regular Mesh-Connected Topologies," *IEEE Transactions on Computers*, vol. 38, pp. 1362-1371, October 1989.
- [CBN81] W. Chou, A. W. Bragg and A. A. Nilsson, "The Need for Adaptive Routing in the Chaotic and Unbalanced Traffic Environment," *IEEE Transactions on Communications*, vol. COM-29, pp. 481-490, April 1981.
- [CS89] M. S. Chen and K. G. Shin, "On Hypercube Fault-Tolerant Routing Using Global Information," in *Proceedings of the Fourth Conference on Hypercube Concurrent Computers and Applications*, pp. 83-86, March 1989.

- [CS90a] M. S. Chen and K. G. Shin, "Depth-First Search Approach for Fault-Tolerant Routing in Hypercube Multicomputers," *IEEE Transactions on Parallel and Distributed Systems*, vol. 1, pp.152-159, April 1990.
- [CS90b] M.-S. Chen and K. G. Shin, "Adaptive Fault-Tolerant Routing in Hypercube Multicomputers," *IEEE Transactions on Computers*, vol. 39, pp. 1406-1416, December 1990.
- [Dal89] W. J. Dally *et al.*, "The J-Machine: A Fine-Grain Concurrent Computer," In *Proceedings of IFIP Congress*, 1989.
- [Dal90] W. J. Dally, "Performance Analysis of  $k$ -ary  $n$ -cube Interconnection Networks," *IEEE Transactions on Computers*, vol. 39, pp. 775-785, June 1990.
- [DH91] S. Dutt and J. P. Hayes, "Designing Fault-tolerant Systems Using Automorphisms," *Journal of Parallel and Distributed Computing*, vol. 12, pp. 249-268, 1991.
- [DS87] W. J. Dally and C. L. Seitz, "Deadlock-Free Message Routing in Multiprocessor Interconnection Networks," *IEEE Transactions on Computers*, vol. C-36, pp. 547-553, May 1987.
- [Fen81] T. Feng, "A Survey of Interconnection Networks," *IEEE Transactions on Computers*, vol. C-30, pp. 12-27, December 1981.
- [FGK73] L. Fratta, M. Gerla, and L. Kleinrock, "The Flow Deviation Method - An Approach to Store-and-forward Communication Network Design," *Networks*, vol. 3, pp. 97-133, 1973.
- [FK71] G. L. Fultz and L. Kleinrock, "Adaptive Routing Techniques for Store-and-forward Computer Communication Networks," in *Proceedings of the International Conference on Communications*, Montreal, Canada, pp. (39-1)-(39-8), June 1971.
- [GS88] J. M. Gordon and Q. F. Stout, "Hypercube Message Routing in the Presence of Faults," in *Proceedings of the Third Conference on Hypercube Concurrent Computers and Applications*, pp. 318-327, January 1988.
- [HB84] K. Hwang and F. Briggs, *Computer Architecture and Parallel Processing*, McGraw-Hill, New York, 1984.
- [Hil85] W. D. Hillis, *The Connection Machine*, MIT Press, 1985.

- [HM89] J. P. Hayes and T. Mudge, "Hypercube Supercomputers," *Proceedings of the IEEE*, Vol. 77, pp. 1829-1841, December 1989.
- [Int85] Intel Scientific Computers, *iPSC User's Guide*, No. 175455-001, Santa Clara, August 1985.
- [Haj91] B. Hajek, "Bounds on Evacuation for Deflection Routing," *Distributed Computing* (5), pp. 1-6, 1991.
- [HLN87] J. Hastad, T. Leighton, and M. Newman, "Reconfiguring a Hypercube in the Presence of Faults," in *Proceedings of 19th Annual ACM Symposium Theory of Computing*, pp. 274-284, May 1987.
- [JH89] S. L. Johnsson and C. T. Ho, "Optimal Broadcasting and Personalized Communication in Hypercube," *IEEE Transactions on Computers*, Vol. 38, pp. 1249-1268, September 1989.
- [KK77] L. Kleinrock and F. Kamoun, "Hierarchical Routing for Large Networks, Performance Evaluation and Optimization," *Computer Networks*, vol. 1, pp. 155-174, January 1977.
- [KK79] P. Kermani and L. Kleinrock, "Virtual Cut-Through: A New Computer Communication Switching Technique," *Computer Networks*, Vol. 3, pp. 267-286, September 1979.
- [Kle75] L. Kleinrock, *Queueing Systems, Volume I: Theory*, John Wiley and Sons, New York, 1975.
- [Kle76] L. Kleinrock, *Queueing Systems, Volume II: Computer Applications*, John Wiley and Sons, New York, 1976.
- [Kle78] L. Kleinrock, "On Flow Control in Computer Networks," *Proceedings of the International Conference on Communications*, Vol. 2, pp. 27.2.1-27.2.5, June 1978.
- [Kle79] L. Kleinrock, "Power and Deterministic Rules of Thumb for Probabilistic Problems in Computer Communications," *International Conference on Communications*, pp. 43.1.1-43.1.10, June 1979.
- [KR88] C. Kim and D. A. Reed, "Adaptive Packet Routing in a Hypercube," in *Proceedings of the Third Conference on Hypercube Concurrent Computers and Applications*, pp. 625-630, January 1988.

- [Lan82] C. R. Lang Jr., *The Extension of Object-Oriented Languages to a Homogeneous, Concurrent Architecture*, Technical Report 5014, Department of Computer Science, California Institute of Technology, May 1982.
- [LH88] T. C. Lee and J. P. Hayes, "Routing and Broadcasting in Faulty Hypercube Computers," in *Proceedings of the Third Conference on Hypercube Concurrent Computers and Applications*, pp. 346-354, January 1988.
- [Lit61] J. D. C. Little, "A Proof of the Queueing Formula  $L = \lambda W$ ," *Operations Research*, vol. 9, pp. 383-387, May 1961.
- [Max85] N. F. Maxemchuk, "Regular and Mesh Topologies in Local and Metropolitan Area Networks," *AT&T Technical Journal*, Vol. 64, No. 7, pp. 1659-1686, September 1985.
- [Max88] N. F. Maxemchuk, "Distributed Clocks in Slotted Networks," in *Proceedings of Infocom '88*, pp. 119-125, March 1988.
- [MS80] P. M. Merlin and P. J. Schweitzer, "Deadlock Avoidance - Store-and-Forward Deadlock," *IEEE Transactions on Communications*, Vol. COM-28, pp. 345-354, March 1980.
- [MW77] J. M. McQuillan and D. C. Walden, "The ARPA Network Design Decisions," *Computer Networks*, vol. 1, pp. 243-289, August 1977.
- [NC85] NCUBE Corporation, *NCUBE/ten: An Overview*, Beaverton, OR, November 1985.
- [PTLP85] J. Peterson, J. Tuazon, D. Liderman, and M. Pniel, "The Mark III Hypercube-Ensemble Concurrent Computer," in *Proceedings of 1985 International Conference on Parallel Processing* pp. 71-73, August 1985.
- [Ren86] D. A. Rennels, "On Implementing Fault-tolerance in Binary Hypercubes" in *Proceedings of 16th Fault Tolerant Computing Symposium*, pp. 344-349, June 1986.
- [SB77] H. Sullivan and T. R. Bashkow, "A Large Scale, Homogeneous, Fully Distributed Parallel Machine I," in *Proceedings of the Fourth Symposium Computer Architecture*, pp. 105-117, March 1977.

- [Sei85a] C. L. Seitz, "The Cosmic Cube," *Communications of the ACM*, vol. 28, pp. 22-33, January 1985.
- [Sei85b] C. L. Seitz *et al.* *The Hypercube Communication Chip*, Display File 5182:DF:85, Department of Computer Science, California Institute of Technology, March 1985.
- [Sei88] C. L. Seitz *et al.* "The Architecture and Programming of the Ametek Series 2010 Multicomputer," In *Proceedings of the Third Conference on Hypercube Concurrent Computers and Applications*, pp. 33-36, January 1988.
- [Sie85] H. J. Siegel, *Interconnection Networks for Large-Scale Parallel Processing*, Lexington Books, 1985.
- [SS88] Y. Saad and M. H. Schultz, "Topological Properties of Hypercubes," *IEEE Transactions on Computers*, vol. C-37, pp. 867-872, July 1988.
- [SS89] Y. Saad and M. H. Schultz, "Data Communication in Hypercubes," *Journal of Parallel and Distributed Computing*, vol. 6, pp. 115-135, 1989.
- [SW90] Q. F. Stout and B. Wagar, "Intensive Hypercube Communication - Prearranged Communication in Link-Bound Machines," *Journal of Parallel and Distributed Computing*, vol. 10, pp. 167-181, 1990.
- [Tan88] A. S. Tanenbaum, *Computer Networks*, Second Edition, Prentice-Hall, 1988.
- [Tit39] E. C. Titchmarsh, *Theory of Functions*, Second Edition, Oxford University Press, London, 1939.
- [TMC87] Thinking Machines Corporation, *Connection Machine Model CM-2 Technical Summary*, Technical Report HA87-4, April 1987.