(5)

Deadlock-Free Routing in a Faulty Hypercube

by

Eric Lehman

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Master of Science in Computer Science

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

May 1998

Author .....Eric Lehman.....................................
Department of Electrical Engineering and Computer Science
May 18, 1998

Certified by..[illegible]..................
F. T. Leighton
Professor of Applied Mathematics
Thesis Supervisor

Accepted by ....
Arthur Smith
Chairman, Departmental Committee on Graduate Students

Deadlock-Free Routing in a Faulty Hypercube

by

Eric Lehman

Submitted to the Department of Electrical Engineering and Computer Science
on May 18, 1998, in partial fulfillment of the
requirements for the degree of
Master of Science in Computer Science

Abstract

In massively parallel computers, processors are often connected in a hypercube configuration. Each vertex in the hypercube represents a processor, and each edge represents a communication link. One problem in such a system is avoiding *deadlock*, a state where there is a cycle of processors, each waiting on the next indefinitely. A second problem is that in a system with many processors, some are bound to fail, leaving a *faulty hypercube*. This thesis investigates the problem of efficiently moving data between processors in a faulty hypercube while avoiding deadlock.

This work was supported by the US Army through grant DAAH04-95-1-0607 and by DARPA under contract N00014-95-1-1246.

Thesis Supervisor: F. T. Leighton

Title: Professor of Applied Mathematics

Acknowledgments

The author would like to thank Hristo Begin, JinSuk Kim, and Tom Leighton for helpful discussions.

Contents

List of Figures

Chapter 1

Introduction

A central problem in massively parallel computing is moving data efficiently between processors. One simple approach to this problem is to run a wire between every pair of processors to permit direct communication. However, as the number of processors grows large, the number of wires becomes enormous, and this approach becomes wholly impractical. An alternative is to permit each processor to communicate directly with only a few others. A message originating at one processor may then pass through a number of other processors before reaching its destination.

This alternate approach to moving data efficiently between processors raises a number of tricky issues. What if a message takes an inordinately long and circuitous path? What if many messages are all directed across one wire, causing a traffic jam? And what if message traffic becomes so badly locked up that some messages become permanently stuck? Furthermore, suppose that we do devise an intricate scheme that tiptoes around all these issues. In a massively parallel computer, some components are bound to fail occasionally. When this happens, will our scheme fall apart completely?

This thesis addresses the above questions for the special case of a parallel computer with processors arranged in a hypercube as in [6, 15], for example. In particular, we give an algorithm for routing messages in a hypercube that tolerates many faulty processors, uses only short paths, rarely causes message traffic to pile up, and never causes a complete deadlock.

The remainder of this chapter formalizes the problem that we address and the result that we claim. Section 1.1 introduces the hypercube architecture and our assumptions about faults. Section 1.2 describes the form of the data moving between processors, and the general features of an algorithm guiding the movement of data. Section 1.3 discusses various ways to evaluate the quality of such an algorithm. Section 1.4 states our result.

1.1 The Faulty Hypercube

We assume that processors are arranged in an n-dimensional hypercube configuration. That is, there is one processor for each n-bit binary string. Two processors can communicate directly if their binary strings differ in a single bit position. In this case, we assume that the two processors are connected by a pair of one-way communication channels running in opposite directions. We will often use graph terminology, referring to processors as nodes and to communication channels as arcs or edges. We will always use n to denote the hypercube dimension and $N = 2^n$ to denote the number nodes.

Some components are bound to fail from time to time in a massively parallel system. For example, there might be a fault in a processor or in a communication link between two processors. In this thesis, we consider only processor faults. Furthermore, we assume that a faulty processor is not only incapable of computation, but also is unable to convey data. In graph terms, a processor fault deletes a node and all incident arcs.

There are two common, but different assumptions about the distribution of faults. In one case, each processor is assumed to fail with probability p independently of all others. In the other case, an adversary places faults in the worst possible configuration. We analyze the performance of our routing algorithm in the presence of worst-case faults.

1.2 Routing

There are several common ways to model the data moving between processors. We describe only two here. In the *store-and-forward* model, each message is represented by an indivisible *packet*. Each processor maintains one or more queues which store packets until they can be forwarded to another processor. At any moment, a packet is either stored in a queue at a particular node or is in flight along a communication link. In the *wormhole* model, a message is divided into small, fixed-size pieces called *flits*. The originator of the message sends flits to a neighboring node over some communication link. After the first flit is sent, the link is reserved until after the last flit has been transmitted. The neighboring node begins sending out flits as soon as the first one is received, thus reserving another link. The message resembles a worm in that it moves through the network as a strung-out sequence of flits. The edges occupied by the worm are reserved until the worm moves on. In this thesis, we are primarily concerned with the wormhole model.

A node that sends a message is called the *source*, and the node that receives the message is called the *destination*. The path taken by the message is called a *route*. A scheme defining the route taken by a message as it travels between a given source and destination is called a *routing algorithm*. There are two broad classes of routing algorithms. In an *oblivious* routing algorithm, the route of one message is unaffected by the presence of other messages in the network. By contrast, in an *adaptive* routing algorithm, the route of one message may be affected by the presence of others; for example, if a traffic jam develops, subsequent messages might be directed toward quieter areas of the system. This example suggests that adaptive routing algorithms may have a substantial advantage. The disadvantage is that adaptivity typically requires additional hardware support. The routing algorithm proposed in this thesis is oblivious.

1.3 Evaluating a Routing Algorithm

The quality of a routing algorithm can be evaluated in many ways. We focus on four measures of quality.

First, a good routing algorithm should have *short routes.* That is, a message should not take a winding, circuitous path between its source and destination. Such long paths would imply slow delivery of messages and thus poor performance of the parallel computer. The length of a route is defined to be the number of edges that it contains. In an n-dimensional hypercube, there is a route of length at most n between every pair of nodes. Thus, a good routing algorithm for a hypercube should not use routes with length significantly exceeding n.

Second, a good routing algorithm should have *low congestion.* This means that not too many messages should be directed across a single communication link. Since messages can only cross at a certain rate, high congestion again would imply slow delivery of messages and poor system performance. Naturally, the congestion on an edge depends on the particular routing problem; that is, congestion depends on the number, sources, and destinations of messages sent. In this thesis, we analyze the congestion arising from two types of routing problem. In a *permutation routing problem*, each node is the source and destination of one message. More generally, in a *h-relation* each node is the source and destination of at most h messages. Since some nodes may be faulty and others may be isolated by faults, we will actually consider permutations and h-relations on only a subset of "usable" nodes.

A third desirable property of a routing algorithm is that it be *deadlock-free.* That is, there should be no way to form a cycle of messages such that each is unable to advance until after the next one advances. This is primarily a concern in wormhole routing, because under that model an edge is reserved until the entire message passes through. This protracted reservation of edges heightens the risk of deadlock. Nevertheless, deadlock can even occur under the store-and-forward model if queues become full.

Here is an example of a deadlock situation in the wormhole model. Suppose that processors are arranged in a 2-cube or square. Every node tries to send a message to the diagonally opposite node by a route circling clockwise around the square. At some point, every message has reserved the first edge of its route. Now no message can advance because all edges are reserved, but no message can unreserve an edge until it advances. This is deadlock.

A simple way to prove that an oblivious routing algorithm is deadlock-free was described in [5]. The method is to give an ordering of all edges so that every route crosses edges in increasing order. We will apply this method to a simple routing algorithm in the next chapter, and later use it to prove that our own routing algorithm is deadlock-free.

A fourth desirable property of a routing algorithm is *fault tolerance.* If some processors fail, then the algorithm should prescribe fault-free routes while still giving good performance. For example, a modest number of faults should not force long paths, high congestion, or deadlock. We do not consider dynamic fault tolerance in this thesis; that is, if a fault occurs, then the current computation may be disrupted.

1.4 Our Result

As noted above, we assume that an adversary places faults in a worst-case configuration. For example, the adversary might try to impose long paths by putting faults on all short paths. He might try to force congestion by arranging faults so as to funnel many messages through a single link. Or he might try to coerce a routing algorithm into allowing deadlock.

The main result of this thesis is a routing algorithm that defeats such an adversary on all fronts. We present an oblivious, deadlock-free routing algorithm that tolerates $O(N/\log^3 N)$ worst-case faults while maintaining short paths and low congestion. Specifically, any permutation can be routed on a prescribed set of $(1 - o(1))N$ nodes such that every path has length at most $(1 + o(1))n$, the expected congestion on every edge is constant, and with high probability all edges have congestion $O(\log N)$.

The remaining chapters are organized as follows. Chapter 2 reviews previous work.

Our new routing algorithm is described and analyzed in Chapter 3. Conclusions and open problems are discussed in Chapter 4.

Chapter 2

Previous Work

Routing in the hypercube and related networks has been studied extensively. In this chapter, we review only two most relevant branches of this previous work. Section 2.1 describes prior research into similar problems, and Section 2.2 summarizes prior work using similar techniques.

We will frequently refer to terminology and techniques introduced in Section 2.2 when we describe our own routing algorithm in the next chapter.

2.1 Previous Work on Similar Problems

Wormhole routing and the associated deadlock problem were first considered by Dally and Seitz [5]. The authors proved the key result that an oblivious routing algorithm is deadlock-free if and only if there is some ordering of the edges such that every route traverses edges in increasing order.[1] Dally and Seitz addressed the deadlock problem in part with a hardware fix. In effect, they made several copies of hypercube edges by introducing "virtual channels". This considerably simplifies the problem of constructing the ordering of the edges needed to prove that a routing algorithm is deadlock-free.

[1]Schwiebert [14] subsequently showed that under different assumptions no ordering of the edges is necessary to ensure deadlock-freedom for an oblivious algorithm in the wormhole model. Although Schwiebert's assumptions are possibly better, his counterexample is sufficiently contrived that it has no obvious implications for the design of routing algorithms.

Kim and Shin [9] studied the problem of constructing an oblivious, deadlock-free routing algorithm for a faulty hypercube. Specifically, they proposed a routing algorithm for a hypercube with up to $\log n + \log\log n + O(1)$ faults in a worst-case configuration. Their technique requires an $(n-2)$-dimensional subcube with no faults. A route consists of three stages. First, the message moves from the source to the nearest node in the healthy $(n-2)$-cube. From there, the message moves to the node in the healthy $(n-2)$-cube closest to the destination. Finally, the message travels to the destination. There are additional optimizations if, for example, there is a healthy $(n-1)$-cube. Kim and Shin's procedure has the advantage of simplicity. Though the number of tolerable faults is asymptotically quite small, it may be sufficient in practice. For example, 5 faults can be tolerated in a 10-cube.

If the deadlock-free condition is put aside, then not only routing, but also general computation is fairly well understood for the faulty hypercube. Bruck, Cypher, and Soroker [3] considered a hypercube with $O(n^c)$ faults in a worst-case configuration, for any constant c. They showed that such a faulty hypercube could run any normal algorithm only a constant factor slower than a healthy hypercube. (For a definition of "normal algorithm", see [10].) The techniques in this paper can also be adapted to give an oblivious, deadlock-free routing algorithm for a hypercube with $O(n^c)$ worst-case faults. Aiello and Leighton [1] subsequently improved this result, using a completely different algorithm. They showed that a hypercube with $O(n^c)$ worst-case faults can run any algorithm with constant slowdown.

Results are even stronger for the random fault model. Hastad, Leighton, and Newman [7] considered a hypercube where nodes are faulty independently with some probability $p < 1$. They showed that with high probability this faulty hypercube could emulate a healthy hypercube with constant slowdown. In particular, any routing algorithm on a healthy hypercube translates to a routing algorithm on the faulty hypercube with congestion and path lengths increased by only a constant.

Several recent publications describe adaptive algorithms for deadlock-free routing in faulty networks. For adaptive algorithms, the requirement that routes must traverse edges in increasing order is considered too restrictive. That is, there exist attractive deadlock-free routing algorithms that violate this requirement. Much effort has been devoted to finding minimal conditions necessary to ensure that an adaptive routing algorithm is deadlock-free. Two recent papers along these lines are due to Park and Agrawal [13] and Anjan, Pinkston, and Duato [2].

2.2 Previous Work Using Similar Techniques

In this section, we describe three previous routing algorithms that use methods related to our own. Each builds on the one before, and the last algorithm is the prior result most similar to our own.

Probably the most natural routing scheme on the hypercube is the *bit-fixing* algorithm. In this algorithm, the dimensions are numbered from 1 to n in an arbitrary way. The bit-fixing algorithm always directs a message across the lowest-numbered dimension in which its present position differs from its destination. In this way, the bit-fixing algorithm defines a unique path between a source and a destination; we call this a *bit-fixing path.* This term will be used extensively in the next chapter. For example, suppose that a message has source 1010 and destination 0111. If we number the dimensions from left to right, then the message would take the route $1010 \rightarrow 0010 \rightarrow 0110 \rightarrow 0111$. Bit-fixing is also sometimes called xyz routing, because in the three-dimensional case a message moves first in the x direction, then in the y direction, and finally in the z direction.

The bit-fixing algorithm has many nice properties. It is oblivious, deterministic, and always uses shortest paths. For a random-destination routing problem, the edge congestion is $\frac{1}{2}$ in expectation and $O(\log N)$ with high probability. (Here and after "high probability" is defined as probability at least $1 - \frac{1}{N}$.) Furthermore, bit-fixing is deadlock-free. We can prove this by giving an ordering of the edges such that every route traverses edges in increasing order. Suppose that we put all edges crossing dimension one first in the ordering, then we put all edges crossing dimension two, and so forth. The edges crossing a particular dimension can be ordered arbitrarily relative to one another. Since bit-fixing sends messages across dimensions in ascending order, every route crosses edges in increasing order, and so the algorithm is deadlock-free.

The principal drawback of bit-fixing is that there exist natural permutation routing problems for which the congestion on an edge is $\Omega(\sqrt{N})$. The bad worst-case congestion of bit-fixing can not be overcome without the use of randomization or adaptive routing; Kaklamanis, Krizanc, and Tsantilas [8] have proven that every deterministic, oblivious routing algorithm on the hypercube has edge congestion $\Omega(\sqrt{N}/\log N)$ for some permutation.

Valiant and Brebner [16] found an elegant way to introduce randomization into routing; in fact, the general term *randomized routing* typically refers to their specific approach. Instead of sending a message directly from source to destination using bit-fixing, Valiant and Brebner proposed sending the message from the source to a random intermediate node and then on to the destination, using bit-fixing in both stages. Intuitively, this two-stage approach breaks a single, potentially "hard" routing problem into two random routing problems, both of which are "easy" with high probability.

Valiant and Brebner's use of two stages does have disadvantages. First, the expected edge congestion is 1, double that of simple bit-fixing. More importantly, the length of the path traveled by any message is expected to be $\log N$, regardless of whether the source and destination are close or far apart. However, randomized routing has one big advantage: for every permutation routing problem, the worst congestion on any edge is $O(\log N)$ with high probability. This is a substantial improvement on the $\Omega(\sqrt{N})$ edge congestion that can arise from simple bit-fixing.

Both of the preceding routing algorithms were designed for a fault-free hypercube. However, Leighton, Maggs, and Sitaraman [11] showed how to extend randomized routing to handle faults. Their techniques are quite similar to our own. (In fact, our `Algorithm A`, which we describe in Section 3.2, is largely a translation of [11] from a butterfly to a hypercube.) Their modification of randomized routing has two main elements. First, if a node is faulty or even close to too many faults, then it can not serve as the source or destination of a route. Second, the random intermediate node is chosen from a restricted set of nodes so that the source-intermediate and intermediate-destination paths are guaranteed to be fault-free.

Leighton, et al. describe their algorithm in the context of a butterfly network, rather than a hypercube. Roughly, they show that their algorithm retains the advantages of randomized routing despite the presence of faults. Specifically, they show that in an N-node butterfly network with at most $F = \frac{N}{12 \log N}$ faults, an arbitrary permutation can be routed on a majority of the nodes. The congestion on every edge is $O(\log N)$ with high probability.

2.3 Comparison with Our Work

Our new routing algorithm compares most directly with the result of Kim and Shin [9]. Both are oblivious, deadlock-free, and fault tolerant. Our algorithm tolerates many more faults, $O(N/\log^3 N)$ compared to $O(\log\log N)$. We also achieve lower congestion through use of randomization. There is a permutation giving congestion $\Omega(\sqrt{N})$ under Kim and Shin's scheme. Our algorithm, however, has expected congestion at most $4 + o(1)$ on every edge, and with high probability every edge has congestion $O(\log N)$. Nevertheless, Kim and Shin's result has one key advantage. While their algorithm is simple and practical, our scheme requires that inordinately large amounts of data be stored at every processor.

As noted above, the methods of Leighton, Maggs, and Sitaraman [11] are closest to our own. They also present an oblivious routing algorithm for a hypercubic network based on randomized routing. There are three main differences. First, our scheme is deadlock-free, while the one proposed in [11] is not. Second, our scheme uses paths of length at most $(1+o(1))n$, but their approach gives paths of length $2n$ when translated from the butterfly to a hypercube. On the other hand, even though our routes are shorter, we do not prove that messages are actually delivered faster. Leighton et al. prove that an arbitrary permutation is routed in time $O(\log N)$ with high probability under the store-and-forward model. We prove no upper bound on routing time.

Chapter 3

A New Routing Algorithm

In this chapter we describe our new algorithm for oblivious, deadlock-free routing in a hypercube with worst-case faults. The description is divided into three parts. The first section introduces a basic building block, a small generalization of randomized routing. In the second section, we construct an initial algorithm for routing in a hypercube containing faulty nodes. This scheme gives low congestion and short routes, but can create deadlock. We amend the scheme in the third section to make it deadlock-free.

For reference, the basic building block described in the first section is called *general randomized routing*. The initial routing algorithm for a faulty hypercube introduced in the second section is called `Algorithm A`. Our complete, deadlock-free routing scheme, which we describe in the third section, is called `Algorithm B`.

3.1 General Randomized Routing

In this section we describe the basic building block of our new algorithm, a small variation on randomized routing that we call *general randomized routing*. The technique is summarized in Figure 3-1 and discussed further below.

General randomized routing is obtained by adjusting Valiant and Brebner's [16] original randomized routing scheme in two ways. First, we can restrict the set of nodes that send and receive messages. In particular, we define a set $A \subseteq \{0,1\}^n$ of

- There is a set $A \subseteq \{0,1\}^n$ of *active nodes.*
- For all $s, d \in A$, there is a set $I_{sd} \subseteq \{0,1\}^n$ of *valid intermediate nodes.*
- A message is routed from source $s \in A$ to destination $d \in A$ as follows:

```
Select a valid intermediate node $i$ uniformly at random from $I_{sd}$.
phase 1: Route the message from $s$ to $i$ using bit-fixing.
phase 2: Route the message from $i$ to $d$ using bit-fixing.
```

Figure 3-1: *This is a summary of general randomized routing.*

active nodes and require the source and destination of every message to be an element of A. Second, we can force the random intermediate node visited by a message to be drawn from a restricted set of nodes. More precisely, for each pair of nodes $s, d \in A$, we define a non-empty set $I_{sd} \subseteq \{0,1\}^n$ of *valid intermediate nodes.* If a message is sent from source s to destination d, then the intermediate node must be an element of I_{sd}. Throughout this chapter, we will frequently refer to these sets A and I_{sd}.

A message is routed much as in the original randomized routing scheme. Let $s \in A$ be the source of the message, and let $d \in A$ be the destination. First, we pick an intermediate node i uniformly at random from I_{sd}. The actual routing is then broken into two phases. In the first phase, the message is routed from the source s to the intermediate node i using bit-fixing. In the second phase, the message is routed from the intermediate node i to the destination d, again using bit-fixing. More generally, any number of messages can be routed by applying this algorithm for each message. However, if multiple messages are routed, then the corresponding intermediate nodes should be picked mutually independently.

We will prove two properties of general randomized routing. First, it inherits the low congestion of ordinary randomized routing, provided that the sets I_{sd} of valid intermediate nodes are all sufficiently large. Second, no node is used too often as an intermediate node. More precisely, we define the *load* on a node i to be the number of messages with i as the intermediate node. That is, each message contributes a unit of load to exactly one node, the randomly-selected intermediate node; a message does not contribute a unit of load to every node that it visits. We will prove that every node has low load, again provided that all the sets I_{sd} are sufficiently large. Load is relevant because the load on a node in this algorithm will correspond to the congestion on an edge in a later algorithm.

We capture the notion that all sets I_{sd} of valid intermediate nodes are "sufficiently large" as follows. Let I be the size of the smallest set I_{sd}; that is, let $I = \min_{s,d \in A} |I_{sd}|$. We will we express congestion and load bounds in terms of this parameter I.

The following theorem characterizes congestion and load in general randomized routing. In particular, we analyze the routing of an h-relation. This is not more enlightening than studying a permutation routing problem, but the generality will be needed later. All of the probabilities and events discussed below are over the sample space defined by randomly picking an intermediate node for each message.

Theorem 1 *Suppose that general randomized routing is used to route an h-relation on the set A of active nodes. The following bounds on congestion and load hold.*

1. *The expected congestion is at most Nh/I on every edge.*
2. *With probability at least $1 - 1/8N$, every edge has congestion $O(\log N + Nh/I)$.*
3. *The expected load is at most Nh/I on every node.*
4. *With probability at least $1 - 1/8N$, every node has load $O(\log N + Nh/I)$.*

The proof of this theorem is long and dry, but encapsulates almost all of the messy calculations in this thesis. The rest of this section is devoted to the proof. There is one subsection for each of the four bounds claimed.

3.1.1 Bound on Expected Congestion

We must show that the expected congestion is at most Nh/I on an arbitrary edge e. Initially, we consider only congestion "during the first phase"; that is, we count only the number of messages that cross edge e while en route to an intermediate node. We will show that the expected congestion on edge e is at most $Nh/2I$ during the first phase. Then we will indicate how to adapt this argument to give the same bound for the second phase. The overall bound of Nh/I on edge congestion follows by linearity of expectation.

We still must show that the expected congestion on edge e during the first phase is at most $Nh/2I$. Each active node is the source of h messages. However, we can partition all the messages into h "batches" so that every active node is the source of a single message in each batch. (Note, however, that several messages in a batch can have the same destination.) We will show below that during the first phase the expected congestion on an edge e contributed by a single batch of messages is at most $N/2I$. Linearity of expectation then implies that the total congestion on the edge e during the first phase is at most $Nh/2I$ as desired.

All that remains is to show that the contribution of a single batch of messages to the congestion on edge e during the first phase is at most $N/2I$. Thus, hereafter, we are concerned with just one batch of messages, and we assume that each active node is the source of exactly one message.

Let $P_{s,i}$ be an indicator variable for the event that a message in the batch has source s and intermediate node i. The congestion on a particular edge e during the first phase is equal to the sum of all indicators $P_{s,i}$ such that the edge e lies on the bit-fixing path from the source s to the intermediate node i. If we let $s \rightsquigarrow i$ denote the set of edges on the bit-fixing path from s to i, then the preceding sentence can be rewritten in symbols as follows:

$$\text{congestion on edge } e = \sum_{s,i:\ e \in s \rightsquigarrow i} P_{s,i}$$

Taking expectations on both sides of the above equation gives:

$$\begin{aligned}
\text{Ex(congestion on edge } e) &= \text{Ex}\left(\sum_{s,i:\ e\in s\rightsquigarrow i} P_{s,i}\right)\\
&= \sum_{s,i:\ e\in s\rightsquigarrow i} \text{Ex}(P_{s,i})\\
&= \sum_{s,i:\ e\in s\rightsquigarrow i} \Pr(P_{s,i}=1)
\end{aligned}$$

We will upper bound the expected congestion on edge e by upper bounding both the magnitude and number of the terms in the last summation above.

First, we bound the magnitude of each term in the summation. In other words, we upper bound $\Pr(P_{s,i} = 1)$, the probability that there is a message with source s and intermediate node i. If s is an active node, then there is a message with source s. Let d be the destination of that message. An intermediate node is chosen uniformly at random from the set I_{sd} of valid intermediate nodes. If i is a valid intermediate node, then it is chosen with probability $1/|I_{sd}| \leq 1/I$. Thus, if s is an active node and i is a valid intermediate node, then $\Pr(P_{s,i} = 1) \leq 1/I$. On the other hand, if s is not an active node or i is not a valid intermediate node, then there can not be a message with source s and intermediate node i; that is, $\Pr(P_{s,i} = 1) = 0$. In both cases, we have the bound $\Pr(P_{s,i} = 1) \leq 1/I$.

Next, we upper bound the number of terms in the summation. Each term corresponds to a bit-fixing path from a source s to an intermediate node i that contains the edge e. All of these paths must cross the dimension along which the edge e is aligned. Furthermore, note that every such path is uniquely identified by specifying whether or not it crosses each of the remaining $n-1$ dimensions. Hence, each edge e is contained in exactly $2^{n-1} = N/2$ paths, and so the summation contains exactly $N/2$ terms.

We can now upper bound the expected congestion on an edge by substituting the results from the two preceding paragraphs into the congestion expression given above.

$$\text{Ex(congestion on edge } e) = \sum_{s,i:\ e\in s\rightsquigarrow i} \Pr(P_{s,i}=1) \leq \frac{N}{2}\cdot\frac{1}{I}$$

Thus, the expected congestion on edge e during the first phase arising from a single batch of messages is at most $N/2I$ as claimed.

All that remains is to indicate how to modify the above argument to show that the expected congestion on edge e during the second phase is also at most $Nh/2I$. For the second phase, a "batch" is redefined so that every active node is the *destination* of one message in each batch. Then each variable $P_{s,i}$ is replaced by a variable $P_{i,d}$, which indicates that there is a message with intermediate node i and destination d. Otherwise, the argument is the same.

3.1.2 High-Probability Bound on Congestion

We must show that with probability at least $1 - 1/8N$, every edge has congestion $O(\log N + Nh/I)$. In all, there are $m = |A| \cdot h \leq Nh$ messages to be routed. Index these message from 1 to m in an arbitrary way. Let E_k be an indicator for the event that message k crosses edge e. The congestion on edge e is thus the sum of the indicators E_k.

$$\text{congestion on edge } e = \sum_{k=1}^{m} E_k$$

There are two key facts about the indicator variables E_k. First, the expected value of the sum is at most Nh/I. This follows from the upper bound on expected congestion computed in Part 1 of the proof. Second, the indicators E_k are mutually independent, since intermediate nodes are chosen mutually independently.

Given these two facts, we can get a high-probability bound on the congestion on edge e with a Chernoff bound [4, 10]. For all $c \geq 1$, we have:

$$\Pr\left(\text{congestion on edge } e > c \cdot \frac{Nh}{I}\right) \quad < \quad e^{-c(\log c + \frac{1}{c} - 1) \cdot \frac{Nh}{2I}}$$

To get the right result, we choose $c = \frac{I}{Nh} \cdot \log(4N^2 \log N) + e^2$. Substituting this definition of c into the Chernoff bound gives:

$$\begin{aligned}
\Pr\left(\text{congestion on } e > c \cdot \frac{Nh}{I}\right) &< e^{-\left(\frac{I}{Nh}\cdot\log\left(4N^2\log N\right)\right)\left(\log e^2+\frac{1}{e^2}-1\right)\cdot\frac{Nh}{I}} \\
&< e^{-\log(4N^2\log N)} \\
&= \frac{1}{4N^2\log N}
\end{aligned}$$

We have shown that a particular edge has congestion exceeding $c \cdot Nh/I$ with probability at most $1/(4N^2 \log N)$. An n-cube contains a total of $\frac{1}{2}N\log N$ edges. Therefore, by the union bound, the probability that some edge has congestion exceeding $c \cdot Nh/I$ is at most $\frac{1}{2}N \log N \cdot 1/(4N^2\log N) = \frac{1}{8N}$. Thus every edge has congestion at most

$$\begin{aligned}
c \cdot \frac{Nh}{I} &= \left(\frac{I}{Nh}\cdot\log(4N^2\log N) + e^2\right)\cdot\frac{Nh}{I} \\
&= \log(4N^2\log N) + e^2\frac{Nh}{I} \\
&= O\left(\log N + \frac{Nh}{I}\right)
\end{aligned}$$

with probability at least $1 - 1/8N$, as claimed.

3.1.3 Bound on Expected Load

We must prove that the expected load on an arbitrary node i is at most Nh/I. Index messages from 1 to m as before. Let M_j be an indicator for the event that node i is the intermediate node for message j. The load on node i is the sum of the indicators M_j. Taking expectations gives:

$$\mathrm{Ex}(\text{load } i) = \sum_{j=1}^{m} \Pr(M_j = 1)$$

We upper bound the above summation by bounding each term. The intermediate node for message j is selected from a set of at least I valid intermediate nodes. If node

i is not a valid intermediate node for message j, then $\Pr(M_j = 1) = 0$. Otherwise, if node i is a valid intermediate node for message j, then $\Pr(M_j = 1) \leq 1/I$. In both cases, $\Pr(M_j = 1) \leq 1/I$, and we can upper bound the sum as follows:

$$\mathrm{Ex}(\text{load } i) = \sum_{j=1}^{m} \Pr(M_j = 1) \leq \frac{Nh}{I}$$

3.1.4 High-Probability Bound on Load

We must show that with probability at least $1 - 1/8N$, every node has load $O(\log N + Nh/I)$. As before, let M_j be an indicator for the event that node i is the intermediate node for message j. The indicator variables M_j are mutually independent, since intermediate nodes are chosen mutually independently. Therefore, we can apply a Chernoff bound to get the high-probability result.

$$\begin{aligned}
\Pr\left(\text{load } i > \log 8N^2 + e^2\frac{Nh}{I}\right) &= \Pr\left(\text{load } i > \left(\frac{I}{Nh}\cdot \log 8N^2 + e^2\right)\cdot\frac{Nh}{I}\right) \\
&< e^{\left(\frac{I}{Nh}\log 8N^2\right)\cdot\left(\log e^2 + \frac{1}{e^2} - 1\right)\frac{Nh}{I}} \\
&< e^{-\log 8N^2} \\
&= \frac{1}{8N^2}
\end{aligned}$$

Thus, a particular node i has load exceeding $\log 8N^2 + e^2(Nh/I)$ with probability at most $\frac{1}{8N^2}$. Since there are N nodes in total, the union bound implies that with probability at least $1 - 1/8N$, every node has load at most $\log 8N^2 + e^2(Nh/I) = O(\log N + Nh/I)$.

3.2 An Initial Routing Algorithm

This section describes an initial algorithm for routing in a faulty hypercube, which we call `Algorithm A`. This scheme has good congestion and short routes, but is not deadlock-free. The algorithm is summarized in Figure 3-2 and discussed below.

`Algorithm A` is general randomized routing with particular definitions for the set A of active nodes and the sets I_{sd} of valid intermediate nodes. The definitions of these sets are given below. The main concern in these definitions is the presence or absence of faults along certain paths through the hypercube. We define a *faulty path* to be a path containing one or more faulty nodes. If a path contains no faults, then we call it a *fault-free path.* If a node s is faulty, then, for example, every path originating at s is faulty.

The set A of active nodes is defined according to two criteria. For a node a to be active, we require first that at most $N/3n$ of the bit-fixing paths with source a be faulty. Second, we also require that at most $N/3n$ of the bit-fixing paths with destination a be faulty. Taken together, these two conditions require that node a be well-connected to the rest of the hypercube; a node that is nearly isolated by faults will not satisfy these conditions.

The sets I_{sd} of valid intermediate nodes are defined according to three criteria. For a node i to be a valid intermediate node between a source s and destination d, we require that both the bit-fixing path from s to i and the bit-fixing path from i to d be fault-free. These two conditions are necessary to avoid routing a message to a faulty node. The third condition is that these two paths have total length at most $n + \sqrt{2n \log 6n}$. Naively, randomized routing could send a message across nearly $2n$ edges, if the source were close to the destination and the intermediate node were far from both. However, such long routes have little value, so we disallow them.

The facts that we need about `Algorithm A` are stated below as a theorem.

Theorem 2 *Suppose that an n-cube contains at most $N/3n^2(n+2)$ faults. Then* `Algorithm A` *routes any h-relation on the set of active nodes with the following guarantees:*

1. *Every edge has expected congestion at most $(1 + o(1))h$.*

2. *With probability at least $1 - 1/8N$, every edge has congestion $O(\log N + h)$.*

3. *Every node has expected load $(1 + o(1))h$.*

- There is a set $A \subseteq \{0,1\}^n$ of active nodes consisting of all nodes a that satisfy two conditions:

  1. At most $N/3n$ bit-fixing paths with source a are faulty.
  2. At most $N/3n$ bit-fixing paths with destination a are faulty.

- For all $s, d \in A$, there is a set $I_{sd} \subseteq \{0,1\}^n$ of valid intermediate nodes consisting of all nodes i that satisfy three conditions:

  1. The bit-fixing path from s to i is fault-free.
  2. The bit-fixing path from i to d is fault-free.
  3. These two paths have total length at most $n + \sqrt{2n \log 6n}$.

- A message is routed from source $s \in A$ to destination $d \in A$ as follows:

  Select a valid intermediate node i uniformly at random from I_{sd}.
  phase 1: Route the message from s to i using bit-fixing.
  phase 2: Route the message from i to d using bit-fixing.

Figure 3-2: `Algorithm A`.

4. *With probability at least* $1 - 1/8N$, *every node has load* $O(\log N + h)$.

5. *The length of every route is at most* $(1 + o(1))n$.

Furthermore, the set of active nodes has size $(1 - o(1))N$.

The only difficulty in proving this theorem is computing sizes for the set A of active nodes and the sets I_{sd} of valid intermediate nodes. The congestion and load claims will then follow from Theorem 1, which characterizes general randomized routing.

We establish lower bounds on the size of the set A and the sets I_{sd} with two lemmas. Intuitively, a hypercube must contain many fault-free bit-fixing paths in order for the sets A and I_{sd} to be large. The reason is that almost all of the membership requirements for sets A and I_{sd}, outlined in Figure 3-2, are that various bit-fixing paths be fault-free. This motivates the first lemma, which states that if a hypercube does not contain too many faults, then most bit-fixing paths are fault-free.

Lemma 1 *If an n-cube has F faults, then at most $\frac{n+2}{2}NF$ bit-fixing paths contain a fault.*

Proof: We first show that a single fault is contained in exactly $\frac{n+2}{2}N$ bit-fixing paths. Any bit-fixing path containing this single fault can be uniquely identified by specifying the set of dimensions crossed by the path, and the number of these dimensions that are crossed before the fault is reached. For a path of length l, the set of dimensions crossed by the path can be selected in $\binom{n}{l}$ ways, and the number of these dimensions crossed before the fault is reached can be chosen in $(l+1)$ ways. Summing over all possible path lengths gives the total number of bit-fixing paths containing a single fault:

$$\sum_{l=0}^{n}(l+1)\binom{n}{l} = \frac{n+2}{2}N$$

Consequently, a hypercube with F faults can have at most $\frac{n+2}{2}NF$ faulty bit-fixing paths. ■

If a hypercube contains few enough faults, then the preceding lemma implies that almost all bit-fixing paths are fault-free. In such a case, the earlier intuitive argument would say that the sets A and I_{sd} should be large. This intuition is confirmed by a second lemma.

Lemma 2 *Suppose that a hypercube contains at most $N/3n^2(n+2)$ faults. Then the following bounds hold on the sizes of set A and sets I_{sd} as defined in* `Algorithm A`.

1. *The set A of active nodes has size at least $(1-1/n)N$.*

2. *For all $s, d \in A$, the set I_{sd} of intermediate nodes has size at least $(1-1/n)N$.*

Proof: (Part 1.) A node is active if it satisfies the two conditions defined in Figure 3-2. Assume for the purpose of contradiction that more than $N/2n$ nodes violate the first condition. That is, there are more then $N/2n$ nodes, each of which is the source of more than $N/3n$ faulty bit-fixing paths. This implies that the hypercube contains more than $N/2n \cdot N/3n = N^2/6n^2$ faulty bit-fixing paths overall. But this contradicts the preceding lemma, which states that the total number of faulty bit-fixing paths is at most $\frac{n+2}{2}NF \leq \frac{n+2}{2}N \cdot N/3n^2(n+2) = N^2/6n^2$. A similar argument shows that at most $N/2n$ nodes violate the second condition. Putting these two facts together, the number of nodes violating at least one of the conditions is at most $N/2n + N/2n = N/n$. The number of nodes satisfying both conditions is thus at least $(1 - 1/n)N$ as claimed.

(Part 2.) The set I_{sd} for $s, d \in A$ consists of all nodes satisfying the three conditions defined in Figure 3-2. Since s is an active node, only $N/3n$ nodes i violate the first condition, which is that the bit-fixing path from s to i be fault-free. Similarly, since d is an active node, only $N/3n$ nodes i violate the second condition, which is that the bit-fixing path from i to d be fault-free. We will show below that at most $N/3n$ nodes violate the third condition as well. Altogether, the number of nodes violating at least one of the three conditions is at most $N/3n + N/3n + N/3n = N/n$. Therefore, the number of nodes satisfying all three conditions is at least $(1 - 1/n)N$ as claimed.

All that remains is to show that at most $N/3n$ nodes violate the third condition, which is that the total length of the bit-fixing paths s to i and i to d can be at most $n + \sqrt{2n \log 6n}$. We will show that at most $N/6n$ nodes are distance greater than $\frac{1}{2}(n + \sqrt{2n \log 6n})$ from the source node s. By the same argument, at most $N/6n$ nodes are similarly distant from the destination d. Thus, there are at most $N/3n$ intermediate nodes i such that the distance from s to i plus the distance from i to d exceeds $n + \sqrt{2n \log 6n}$. Since bit-fixing always uses shortest paths, at most $N/3n$ nodes violate the third condition.

To complete the proof, we must show that at most $N/6n$ nodes are distance greater than $\frac{1}{2}(n + \sqrt{2n \log 6n})$ from the source node s. Orient the hypercube so that s is the node with all coordinates 0. Let $i_1, i_2, \ldots, i_n$ be the coordinates of a node i chosen uniformly at random from $\{0,1\}^n$. Then $i_1, i_2, \ldots, i_n$ are uniform, mutually independent Bernoulli variables. We can write the distance between the source s and the random node i as $\sum i_k$ and then bound this sum with a Chernoff bound [4, 12].

$$\begin{aligned}
\Pr\left(\sum_{k=1}^{n} i_k > \frac{1}{2}(n + \sqrt{2n\log 6n})\right) &= \Pr\left(\sum_{k=1}^{n} i_k > \left(1 + \sqrt{\frac{8\log 6n}{n}}\right)\cdot\frac{n}{2}\right) \\
&< e^{-\frac{1}{4}\frac{8\log 6n}{n}\cdot\frac{n}{2}} \\
&= e^{-\log 6n} \\
&= \frac{1}{6n}
\end{aligned}$$

Thus, the distance between the source s and a random node i exceeds $\frac{1}{2}(n + \sqrt{2n\log 6n})$ with probability at most $1/6n$. Combinatorially, this implies that at most $N/6n$ nodes i are distance greater than $\frac{1}{2}(n + \sqrt{2n\log 6n})$ from the source s. ■

We can now prove Theorem 2 by plugging the sizes of sets A and I_{sd} into Theorem 1, which describes general randomized routing.

Proof: *(of Theorem 2)*

(Part 1.) By Theorem 1, the expected congestion on every edge is at most Nh/I. Lemma 2 states that every set I_{sd} of valid intermediate nodes has size at least $(1 - 1/n)N$. This implies that $I = \min_{s,d\in A} |I_{sd}|$ is at least $(1 - 1/n)N$. Substituting in this expression for I into Nh/I, we get that the expected congestion on every edge is at most $(1 + \frac{1}{n-1})h = (1 + o(1))h$.

(Parts 2-4.) These results follow immediately by substituting the bound on I into the corresponding parts of Theorem 1, just as in Part 1 above.

(Part 5.) Every route has length at most $n+\sqrt{2n\log 6n} = (1+o(1))n$ by condition 3 in the definition of the sets I_{sd} of valid intermediate nodes.

Finally, by Lemma 2, the set A of active nodes has size at least $(1 - 1/n)N = (1 - o(1))N$. ■

3.3 Eliminating Deadlock

This section presents our oblivious, deadlock-free routing algorithm for a faulty hypercube. We call this scheme `Algorithm B`. This algorithm retains the good congestion and path length features of `Algorithm A`, but incorporates an additional trick to eliminate deadlock.

The section is broken into three subsections. The first describes the intuition behind the trick to eliminate deadlock. The second section gives the details of `Algorithm B`. The final section contains the main theorem of this thesis, which gives performance guarantees for `Algorithm B`.

3.3.1 Intuition Behind Eliminating Deadlock

One main consideration underlies the trick used in `Algorithm B` to eliminate deadlock: bit-fixing is deadlock-free, but general randomized routing is not.

Recall that bit-fixing is deadlock-free because there exists a total ordering of hypercube edges so that every route traverses edges in increasing order. In particular, we put all the dimension 1 edges first (in an arbitrary order), followed by all the dimension 2 edges, and so forth.

On the other hand, general randomized routing can produce a deadlock involving just two messages. Recall that each route produced by general randomized routing is a concatenation of two bit-fixing routes. The problem that arises in general randomized routing is that the second leg of one route can overlap the first leg of another and vice-versa. This can create deadlock. Such a situation is shown in Figure 3-3.

We preclude such a situation in `Algorithm B`. The scheme is still approximately general randomized routing. The trick is to define two disjoint classes of hypercube edges. The first leg of a route uses only "class one" edges, and the second leg uses only "class two" edges. No deadlock configuration can exist in a single class of edges, because within one class we will use only bit-fixing, which is deadlock-free. Furthermore, no deadlock configuration spanning both classes can exist either; the second leg of one route can not overlap the first leg of another route as in Figure 3-3, because the two legs consist of edges in different classes.

We will come back and connect this intuitive description to the actual routing algorithm after giving the details of `Algorithm B`.

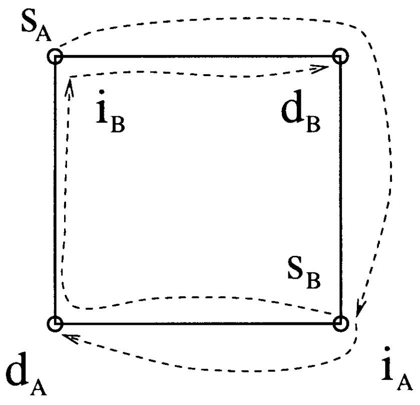


Figure 3-3: *This figure shows how deadlock can arise if even two messages are sent using general randomized routing. The network shown is a 2-cube. Message A follows the path from source s_A to intermediate node i_A to destination d_A, indicated by dashed arrows outside the square. Message B follows the path from source s_B to intermediate node i_B to destination d_B, indicated by dashed arrows inside the square. Deadlock can occur because the second leg of route A overlaps the first leg of route B (on the bottom edge), and the second leg of route B overlaps the first leg of route A (on the top edge).*

3.3.2 Description of `Algorithm B`

This section describes `Algorithm B` in detail.

We need two pieces of notation. First, throughout the description of `Algorithm B`, we regard the n-cube as a set of four $(n-2)$-cubes obtained by cutting along the first two dimensions. We call the four subcubes C_{00}, C_{01}, C_{11}, and C_{10}. Following the natural convention, subcube C_{jk} contains all nodes with first coordinate j and second coordinate k. Our second piece of notation is that if x is an arbitrary node, then x_{jk} denotes the node with first coordinate j, second coordinate k, and all remaining coordinates in agreement with x. Thus, node x_{jk} is always contained in subcube C_{jk}. The nodes x_{00}, x_{01}, x_{11}, and x_{10} form a square, as shown in Figure 3-4. Note that the node x itself is necessarily one of these four nodes.

We can now proceed with the description of the algorithm, which is summarized in Figure 3-6. `Algorithm B` shares the major features of `Algorithm A`. Specifically, there is a set A of active nodes, and the source and destination of every message are in this set. Also, a message visits a random intermediate node picked uniformly from a set I_{sd} of valid intermediate nodes. These sets will be fully defined after we describe the route traveled by a message. Only one fact about these sets is needed right away:

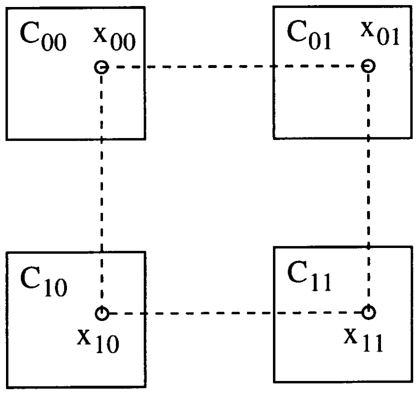


Figure 3-4: *This figure illustrates the two new notations introduced in this section. The entire picture represents an n-cube. Each of the four squares represents an $(n-2)$-cube obtained by cutting along the first two dimensions. We call these subcubes C_{00}, C_{01}, C_{11}, and C_{10}. If x is an arbitrary node, then x_{ij} denotes the node with first coordinate i, second coordinate j, and with all remaining coordinates in agreement with x. The nodes x_{00}, x_{01}, x_{10}, and x_{11} are connected in a square as shown. (The dotted lines represent single edges.) The node x itself must be one of these four nodes.*

all sets I_{sd} of valid intermediate nodes are completely contained in the single subcube C_{00}.

A message is routed from a source s to a destination d as follows. As a preliminary step, an intermediate node $i = i_{00}$ is selected uniformly at random from the set of valid intermediate nodes I_{sd}. The actual route consists of five phases, which are described below and illustrated in Figure 3-5.

In the first phase, the message moves to subcube C_{00}. In particular, the message is routed from the source s to node s_{00} by traversing some suffix of the path $s_{01} \rightarrow s_{11} \rightarrow s_{10} \rightarrow s_{00}$. (Recall that the source s must be one of the four nodes on this path.) If the subcubes C_{ij} are drawn as in Figure 3-5, then the message visits the subcubes in clockwise order; thus, we refer to this as clockwise routing. In the second phase, the message moves from s_{00} to the random intermediate node i_{00} using bit-fixing. In the third phase, the message crosses the single edge from i_{00} to i_{01}. In the fourth phase, the message moves from i_{01} to d_{01}, again using bit-fixing. In the final phase, the message is routed counter-clockwise to the destination d; that is, the route is some prefix of the path $d_{01} \rightarrow d_{00} \rightarrow d_{10} \rightarrow d_{11}$.

We can now connect `Algorithm B` to the intuition given in the preceding section.

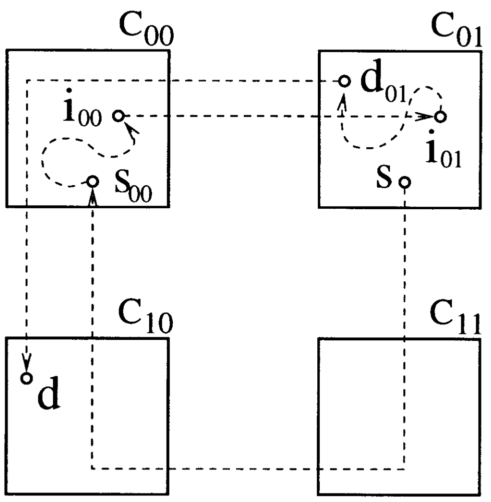


Figure 3-5: *This figure shows a route defined by* `Algorithm B`. *In this case, the source s is in subcube C_{01} and the destination d is in subcube C_{10}. The route is indicated with a dotted line. Each straight segment represents a single hypercube edge, and the two curved segments represent bit-fixing paths. Roughly the routing scheme is: go clockwise to subcube C_{00}, do the first leg of general randomized routing, hop over to subcube C_{01}, do the second leg of general randomized routing, and then go counter-clockwise to the destination.*

The intuitive plan was to do the first leg of general randomized routing on one class of edges and the second leg on a second class of edges. This would guarantee that a deadlock situation like the one shown in Figure 3-3 could not occur. The second and fourth phases of `Algorithm B` correspond to the two phases of general randomized routing. From this perspective, the "class one" edges used for the first leg of general randomized routing are the edges internal to the subcube C_{00}. Similarly, the edges internal to subcube C_{01} are the "class two" edges used for the second leg of general randomized routing. The second leg of one route can not overlap the first leg of another route, because all first legs are contained in subcube C_{00} and all second legs are contained in subcube C_{01}.

To finish describing `Algorithm B`, all that remains is to define the sets A and I_{sd}. For the purpose of these two definitions, we consider a node x faulty if *any* of the nodes x_{00}, x_{01}, x_{11}, or x_{10} is faulty.

The set A of active nodes consists of all nodes a satisfying two conditions. These are analogous to the two conditions on active nodes in `Algorithm A`. First, at most $N/12(n-2)$ bit-fixing paths with source a_{00} and destination in C_{00} can be faulty.

- A node x is marked faulty if any of the nodes x_{00}, x_{01}, x_{11}, or x_{10} are faulty.

- There is a set $A \subseteq \{0,1\}^n$ of active nodes consisting of all nodes a that satisfy two conditions:

  1. At most $N/12(n-2)$ bit-fixing paths with source a_{00} and destination in cube C_{00} are faulty.
  2. At most $N/12(n-2)$ bit-fixing paths with source in cube C_{01} and destination a_{01} are faulty.

- For all $s, d \in A$, there is a set $I_{sd} \subseteq \{0,1\}^n$ of valid intermediate nodes consisting of all nodes i in C_{00} that satisfy three conditions:

  1. The bit-fixing path from s_{00} to i_{00} is fault-free.
  2. The bit-fixing path from i_{01} to d_{01} is fault-free.
  3. These two paths have total length at most $n - 2 + \sqrt{2(n-2)\log 6(n-2)}$.

- A message is routed from source $s \in A$ to destination $d \in A$ as follows:

  Select a node $i = i_{00}$ uniformly at random from I_{sd}.
  phase 1: Route clockwise from s to s_{00}.
  phase 2: Route from s_{00} to i_{00} using bit-fixing.
  phase 3: Route from i_{00} to i_{01} by the direct edge.
  phase 4: Route from i_{01} to d_{01} using bit-fixing.
  phase 5: Route counter-clockwise from d_{01} to d.

Figure 3-6: `Algorithm B`.

Second, at most $N/12(n-2)$ bit-fixing paths with source in C_{01} and destination a_{01} can be faulty. Note that all routes with source a pass through a_{00}, and all routes with destination a pass through a_{01}. Thus, intuitively, these conditions ensure that every active node a is well-connected to the rest of the hypercube. The limit of $N/12(n-2)$ faulty bit-fixing paths is obtained by taking the corresponding bound of $N/3n$ in `Algorithm A` and "scaling down" to dimension $n-2$. That is, n is replaced by $n-2$, and N is replaced by $N/4$. The rationale for this "scaling down" will be given in the next section.

Finally, we must define the sets I_{sd} of valid intermediate nodes. The set I_{sd} consists of all nodes i in subcube C_{00} satisfying three conditions. Again, these conditions are analogous to those in `Algorithm A`. First, the bit-fixing path from s_{00} to i_{00} must be fault-free. This ensures that a message does not encounter a fault during phase two of the algorithm. Second, the bit-fixing path from i_{01} to d_{01} must be fault-free. This ensures that no fault is encountered during phase four. Finally, we restrict the total length of these two bit-fixing paths to $(n-2)+\sqrt{2(n-2)\log 6(n-2)}$. (This complicated expression is also formed by taking the analogous expression in `Algorithm A`, $n+\sqrt{2n\log 6n}$, and "scaling down" by two dimensions.) This ensures that `Algorithm B` generates short routes.

3.3.3 Analysis of `Algorithm B`

We can now state the main result of this thesis. This is a theorem saying that `Algorithm B` is highly fault-tolerant, produces short routes, rarely creates congestion, and never creates deadlock. Note that Theorem 2 assumed $N/3n^2(n+2)$ faults, but that the theorem below assumes only $N/12n(n-2)^2$ faults. This is again a "scaling down" by two dimensions.

Theorem 3 *Suppose that an n-cube contains at most $N/12n(n-2)^2$ faults. Then* `Algorithm B` *routes any h-relation on the set A of active nodes with the following guarantees:*

1. *Deadlock can not occur.*

2. *The length of every route is at most* $(1 + o(1))n$.

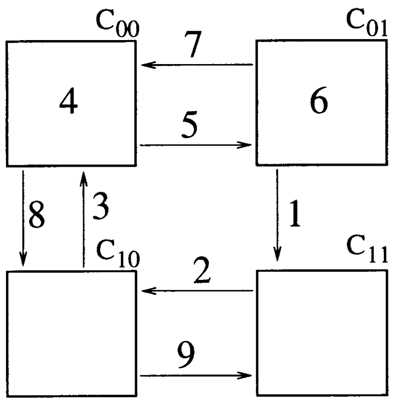
3. *Every edge has expected congestion at most* $(4 + o(1))h$.

4. *Every edge has congestion* $O(\log N + h)$ *with probability at least* $1 - 1/N$.

Furthermore, the set of active nodes has size $(1 - o(1))N$.

Proof:

(Part 1.) We prove that `Algorithm B` is deadlock-free by giving an ordering of the edges such that every route crosses edges in increasing order. Such an ordering is defined in Figure 3-7.



(1)	phase 1:	edges from C_{01} to C_{11}
(2)		edges from C_{11} to C_{10}
(3)		edges from C_{10} to C_{00}
(4)	phase 2:	edges within C_{00}, ordered by increasing dimension
(5)	phase 3:	edges from C_{00} to C_{01}
(6)	phase 4:	edges within C_{01}, ordered by increasing dimension
(7)	phase 5:	edges from C_{01} to C_{00}
(8)		edges from C_{00} to C_{10}
(9)		edges from C_{10} to C_{11}

Figure 3-7: *This figure shows an ordering of the edges proving that* `Algorithm B` *is deadlock-free. In the diagram, the numbers indicate where sets of edges appear in the ordering. The table below describes each set of edges in text. The routing phase that makes use of each set of edges is also noted.*

(Part 2.) We can upper bound the length of a route by summing over the five phases. Phases 1, 3, and 5 contribute at most $3 + 1 + 3 = 7$ edges. Phases 2 and 4 contribute at most $(n-2) + \sqrt{2(n-2)\log 6(n-2)}$ edges by property 3 of the sets I_{sd}. Adding these quantities shows that every route has length at most $(1+o(1))n$ as claimed.

The remainder of the proof follows from one key observation. After phase 1 of `Algorithm B` is completed, there are $4h$ messages at every active node in C_{00}. Before phase 4, there are $4h$ messages at every active node in C_{01}. The middle three phases of `Algorithm B` correspond exactly to routing a $4h$-relation on an $(n-2)$ cube with `Algorithm A`, except that between the two routing phases of `Algorithm A` messages jump from C_{00} to C_{01}. Establishing this correspondence was the rationale for "scaling down" the number of faults and the definitions of sets A and I_{sd} by two dimensions. Using this correspondence, the remainder of the proof follows from Theorem 2 as described below.

(Part 3.) We must show the expected congestion on every edge is at most $(4+o(1))h$. For edges internal to cubes C_{00} and C_{01}, a bound of $(1+o(1))\cdot 4h = (4+o(1))h$ follows from Part 1 of Theorem 2. Note that the congestion on an edge from C_{00} to C_{01} is equal to the load on the endpoint of that edge in C_{00}. Thus, the expected congestion on such edges is at most $(4+o(1))h$ by Part 3 of Theorem 2. All remaining edges have congestion at most $3h$.

(Part 4.) We must show that with probability at least $1-1/N$, every edge has congestion $O(\log N)$. Part 2 of Theorem 2 implies that every edge internal to C_{00} and C_{01} has congestion $O(\log N + h)$ with probability at least $1-1/2N$. Part 4 of Theorem 2 implies that every node in C_{00} has load at most $O(\log N + h)$ with probability at least $1-1/2n$. This implies that the edges from C_{00} to C_{01} have congestion $O(\log N + h)$ with probability at least $1-1/2n$. All other edges have congestion at most $3h$. By the union bound, every edge has congestion $O(\log N + h)$ with probability at least $1-1/N$ as claimed.

Finally, Theorem 2 implies that C_{00} contains $(1-o(1))(N/4)$ active nodes. Since a node a is active if and only if a_{00} is active, the total number of active nodes is four times greater, $(1-o(1))N$. ∎

The theorem above gives interesting corollaries for two special values of h, the number of messages starting and finishing at each node. First, suppose that $h = \log N$. In particular, suppose that each of the N nodes sends $\log N$ messages to the opposite node, distance $\log N$ away. Then the total congestion over all edges must be at least $N \log^2 N$. Since there are only $N \log N$ edges, some edge must have congestion at least $(N \log^2 N)/(N \log N) = \log N$. Corollary 3 states that with high probability the congestion on every edge is $O(\log N)$, which is within a constant factor of optimal, for every $(\log N)$-relation.

Corollary 3 *If a* $(\log N)$*-relation on the set of active nodes is routed with* `Algorithm B`*, then with high probability every edge has congestion* $O(\log N)$*. Deadlock-freedom and bounds on path length and the number of active nodes hold as before.*

Finally, the main result claimed in the introduction follows when we set $h = 1$.

Corollary 4 `Algorithm B` *is an oblivious, deadlock-free algorithm that routes any permutation on a prescribed set of at least* $(1 - o(1))N$ *nodes in a hypercube with* $\Omega(N/n^3)$ *faults using paths of length at most* $(1 + o(1))n$ *such that every edge has expected congestion at most* $4 + o(1)$*, and with probability at least* $1 - 1/N$*, every edge has congestion* $O(\log N)$*.*

Chapter 4

Conclusion

We have shown an oblivious, deadlock-free routing algorithm for a hypercube with $O(N/\log^3 N)$ faults. The algorithm routes any permutation on a set of $(1-o(1))N$ nodes with constant expected congestion, congestion $O(\log N)$ with high probability, and with paths of length at most $(1+o(1))n$.

Several related problems remain open. While our result is a double-exponential improvement on the best previous result, we completely sacrifice practicality; a better balance between fault-tolerance and feasibility would be nice.

Putting aside the deadlock-free condition, there remains the problem of embedding a healthy hypercube into a hypercube with more than $O(n^c)$ faults with constant congestion, load, and dilation. On the other hand, putting aside the issue of fault-tolerance, necessary and sufficient conditions for the existence of a low-congestion, deadlock-free routing algorithm are also unknown.

Bibliography

[1] W. A. Aiello and T. Leighton. Coding theory, hypercube embeddings, and fault tolerance. *Symposium on Parallel Algorithms and Architectures*, pages 125–136, July 1991.

[2] K. V. Anjan, T. M. Pinkston, and J. Duato. Generalized theory for deadlock-free adaptive wormhole routing and its application to disha concurrent. In *International Parallel Processing Symposium*, pages 815–821, 1996.

[3] J. Bruck, R. Cypher, and D. Soroker. Tolerating faults in hypercubes using subcube partitioning. *IEEE Transactions on Computers*, 41:599–605, 1992.

[4] H. Chernoff. A measure of asymptotic efficiency for tests of a hypothesis based on the sum of observations. *Annals of Mathematical Statistics*, 23:493–507, 1952.

[5] W. Dally and C. Seitz. Deadlock free message routing in multiprocessor interconnection networks. *IEEE Transactions on Computers*, pages 547–553, May 1987.

[6] B. Duzett and R. Buck. An overview of the ncube3 supercomputer. In *Proceedings of the Fourth Symposium on the Frontiers of Massively Parallel Computation*, pages 458–464, 1992.

[7] J. Hastad, T. Leighton, and M. Newman. Fast computation using faulty hypercubes. In *Symposium on the Theory of Computation*, pages 251–263, 1989.

[8] C. Kaklamanis, D. Krizanc, and T. Tsantilas. Tight bounds for oblivious routing in the hypercube. In *Symposium on Parallel Algorithms and Architectures*, pages 31–36, 1990.

[9] J. Kim and K. G. Shin. Deadlock-free fault-tolerant routing in injured hypercubes. *IEEE Transactions on Computers*, pages 1078–1088, September 1993.

[10] F. T. Leighton. *Introduction to Parallel Algorithms and Architectures.* Morgan Kauffman, 1992.

[11] T. Leighton, B. M. Maggs, and R. K. Sitaraman. On the fault tolerance of some popular bounded-degree networks. *SIAM Journal on Computing*, to appear.

[12] R. Motwani and P. Raghavan. *Randomized Algorithms.* Cambridge Press, 1995.

[13] H. Park and D. P. Agrawal. Generic methodologies for deadlock-free routing. In *International Parallel Processing Symposium*, pages 638–643, 1996.

[14] L. Schwiebert. Deadlock-free oblivious wormhole routing with cyclic dependencies. In *Symposium on Parallel Algorithms and Architectures*, pages 149–158, 1997.

[15] C. Seitz. The cosmic cube. *Communications of the ACM*, pages 22–33, January 1985.

[16] L. G. Valiant and G. J. Brebner. Universal schemes for parallel communication. In *Symposium on the Theory of Computation*, pages 263–277, 1981.