Oblivious Deadlock-Free Routing in a Faulty Hypercube

Jin Suk Kim
Laboratory for Computer Science
Massachusetts Institute of Technology
545 Tech Square, Cambridge, MA 02139
jinskim@camars.kaist.ac.kr

Eric Lehman
Department of Computer Science
Massachusetts Institute of Technology
545 Tech Square, Cambridge, MA 02139
e_lehman@mit.edu

Tom Leighton
Department of Mathematics
Massachusetts Institute of Technology
545 Tech Square, Cambridge, MA 02139
ftl@math.mit.edu

Abstract

A central problem in massively parallel computing is efficiently routing data between processors. This problem is complicated by two considerations. First, in any massively parallel system, some processors are bound to fail, disrupting message routing. Second, one must avoid deadlock configurations in which messages permanently block one another. We present an efficient, oblivious, and deadlock-free routing algorithm for the hypercube. The algorithm tolerates a large number of faults in a worst-case configuration.

1. Introduction

A central problem in massively parallel computing is efficiently routing data between processors. A routing algorithm is employed to direct message traffic through the system. A good algorithm ensures that messages do not take inordinately long paths and that not too many messages are directed through a single communication channel.

Two considerations make routing more difficult. First, deadlock must be avoided. Deadlock occurs when there is a cycle of messages where each message in the cycle is unable to advance through the network until after the next message in the cycle advances, resulting in a permanent lockup. Second, some processors in a massively parallel system will inevitably fail. A routing algorithm should permit efficient, deadlock-free communication between the majority of healthy processors, even if some are faulty and consequently unable to communicate.

Routing algorithms are broadly classed into two categories. In an *oblivious* routing algorithm, the path of one message is unaffected by the presence of other messages in the network. The opposite is true in an *adaptive* routing algorithm; in this case, for example, messages may be directed away from congested parts of the network. The disadvantage of adaptive algorithms is that additional routing hardware is required.

We propose an oblivious routing algorithm for the hypercube architecture. What distinguishes our algorithm from previous work is that it remains efficient and deadlock-free even in the presence of a large number of faults in the worst possible configuration. Throughout the paper, n is the hypercube dimension, $N = 2^n$ is the number of processors, and F is the number of faulty processors. We assume that adjacent processors are connected by a pair of one-way communication channels running in opposite directions.

Deadlock became a major concern with the introduction of the wormhole routing model by Dally and Seitz [1]. These authors made the observation that a routing algorithm is deadlock-free if there is a total order of communication channels such that every message traverses channels in increasing order. Dally and Seitz proposed a hardware solution to the deadlock problem that effectively divided each channel into a number of "virtual channels", easing the construction of an appropriate total order. Subsequently, Peercy and Banerjee [5] extended the virtual channel approach to allow deadlock-free routing even in a hypercube with faulty processors. They proposed a distributed algorithm that identifies the shortest path between every pair of processors, and then sends messages along only these paths. Deadlock is ruled out by selective introduction of virtual channels. This scheme has the drawback that the number of virtual channels

required increases with the number of faults at an undetermined rate that appears to be dependent on the fault configuration. Kim and Shin [3] showed that virtual channels are unnecessary for oblivious, deadlock-free routing, even in the presence of several faults in a worst-case configuration. Their method is to locate a fault-free $(n-2)$-dimensional subcube, which is used as a communication backbone for the entire network. Kim and Shin's routing algorithm has the advantage of simplicity. However, the number of faults tolerated is asymptotically quite small, around $F = \log \log N$.

All of the preceding routing algorithms are deterministic. As a consequence, there exist computations that cause a single communication channel to become congested with messages [2]. In particular, there exist permutation routing problems resulting in $\Omega(\sqrt{N}/\log N)$ congestion on some channel. (In a permutation routing problem, every processors is the source and destination of a single message. The congestion on a communication channel is the number of messages routed across it.) In fact, for all of the above algorithms, there are natural permutation routing problems that exhibit this bad $\Omega(\sqrt{N}/\log N)$ congestion.

Leighton, Maggs, and Sitaraman [4] recently devised a highly fault-tolerant routing algorithm for a hypercubic network that avoids this congestion problem through the use of randomization as in [6]. They proved that in an N-node butterfly network with at most $F = N/12 \log N$ faults, an arbitrary permutation can be routed on a majority of the nodes with congestion $O(\log N)$ on every edge with high probability.

We present a oblivious routing algorithm with fault tolerance comparable to Leighton, Maggs, and Sitaraman and with the freedom from deadlock guaranteed by Kim and Shin. In particular, in a hypercube with $O(N/\log^3 N)$ faults, our algorithm routes an arbitrary permutation on a set of $(1-o(1))N$ nodes with paths of length $(1+o(1))n$ and with congestion $O(\log N)$ on every edge with high probability. We make no use of virtual channels, and the number of faults tolerated is a double exponential improvement on Kim and Shin. We use shorter paths than Leighton, Maggs, and Sitaraman, and our algorithm is deadlock-free.

We build up our algorithm over the next three sections. Section 2 introduces a key component that we call general randomized routing. Section 3 describes an initial algorithm for routing in a hypercube containing faulty nodes. This scheme, which we call `Algorithm A`, gives low congestion and uses short routes, but can create deadlock. We extend this result in Section 4 to obtain `Algorithm B`, our complete fault-tolerant, deadlock-free routing scheme.

2. General Randomized Routing

In this section we describe a basic building block, a routing algorithm that we call *general randomized routing*. The technique is summarized in Figure 1 and discussed further below.

- 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 by bit-fixing.
  phase 2: Route the message from i to d by bit-fixing.

Figure 1. General randomized routing.

General randomized routing is a slight modification of the randomized routing scheme proposed by Valiant and Brebner [6]. In ordinary randomized routing, a message is sent from a source to a destination in two stages. The message is first sent from the source to a random intermediate node, and then is sent from the intermediate node to the destination. In each stage, the message is routed using the *bit-fixing* algorithm, which is also known as *dimension-order routing* and the *e-cube* algorithm. In bit-fixing, dimensions are ordered in an arbitrary way, and a message is always directed along the lowest dimension in which its current position and its destination differ. We will make frequent reference to *bit-fixing paths*, by which we mean paths that messages using the bit-fixing algorithm could take.

General randomized routing is obtained by adjusting Valiant and Brebner's [6] 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 *active nodes* and require the source and destination of every message to be elements 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 the paper, we will frequently refer to these sets A and I_{sd}. As before, messages are routed in two stages, using bit-fixing for each stage.

We state 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. The theorem below states 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, where every processor is the source and destination of h messages. This may not be more enlightening than studying a permutation routing problem, but the generality will be needed later.

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 uses standard linearity of expectation and Chernoff arguments.

3. Tolerating Faults

We now describe 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 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 bit-fixing paths in 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*. Note that if a processor is faulty, then every path with that source or destination is a faulty path.

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.

- 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 dest 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 such that:
  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 paths have total length at most $n + \sqrt{2n \log 6n}$.
- To route a message from source $s \in A$ to dest $d \in A$:

  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 2. Algorithm A.

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

3. *Every node has expected load* $(1 + o(1))h$.
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 $I_{s\,d}$ 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 $I_{s\,d}$ with two lemmas. Intuitively, a hypercube must contain many fault-free bit-fixing paths in order for the sets A and $I_{s\,d}$ to be large. The reason is that almost all of the membership requirements for sets A and $I_{s\,d}$, outlined in Figure 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 2.1 *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 $I_{s\,d}$ should be large. This intuition is confirmed by a second lemma.

Lemma 2.2 *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* $I_{s\,d}$ *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* $I_{s\,d}$ *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 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 $I_{s\,d}$ for $s, d \in A$ consists of all nodes satisfying the three conditions defined in Figure 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 $n + \sqrt{2n\log 6n}$. A standard argument about the binomial distribution shows that at most $N/6n$ nodes are distance greater than $\frac{1}{2}(n+\sqrt{2n\log 6n})$ from the source node s. (Intuitively, the average distance from s taken over all nodes is $n/2$, and a vanishingly small fraction of nodes are more than about $\sqrt{\log n}$ standard deviations above this mean. The constants are chosen to make subsequent calculations work out.) 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. □

We can now prove Theorem 2 by plugging the sizes of sets A and $I_{s\,d}$ 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 2.2 states that every set $I_{s\,d}$ of valid intermediate nodes has size at least $(1-1/n)N$. This implies that $I = \min_{s,\,d\in A}|I_{s\,d}|$ 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 2.2, the set A of active nodes has size at least $(1-1/n)N = (1-o(1))N$. □

4. Eliminating Deadlock

This section presents our complete 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 our main theorem, which gives performance guarantees for `Algorithm B`.

4.1. 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.

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.

General randomized routing can produce a deadlock involving just two routes. The problem is that the second leg of one route can overlap the first leg of another and vice-versa, creating a cycle that can lead to deadlock.

We eliminate this problem `Algorithm B`. The scheme is still approximately general randomized routing; each route consists largely of two stages, and bit-fixing is used in each stage. The trick is to define two disjoint classes of hypercube edges. The first stage of a route uses only "class one" edges, and the second stage 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 stage of one route can not overlap the first stage of another route, because the two stages consist of edges in different classes.

4.2. 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. The 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} define a 2-cube. 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 4. `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: 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.

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, 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 counterclockwise 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. The intuitive plan was to do the first stage of general randomized routing on one class of edges and the second stage on a second class of edges. 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 stage of general randomized routing are the edges internal to the

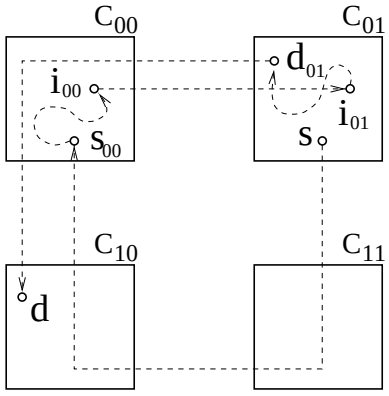


Figure 3. This figure shows a route defined by `Algorithm B`. In this case, the source s is in subcube C_{01} and the destination d is in subcube C_{10}. The route is indicated with a dotted line. Each straight segment represents a single hypercube edge, and the two curved segments represent bit-fixing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

subcube C_{00}. Similarly, the edges internal to subcube C_{01} are the "class two" edges used for the second stage of general randomized routing. The second stage of one route can not overlap the first stage of another route, because all first stages are contained in subcube C_{00} and all second stages 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. 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.

- 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} such that:
  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 4. Algorithm B

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.

4.3. Analysis of `Algorithm B`

We can now state the main result. 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 $N/3n^2(n+2)$

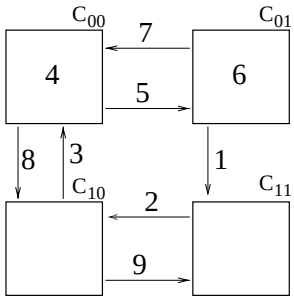
faults, but that the theorem below assumes only $N/12n(n-2)^2$ faults. This is again a "scaling down" by two dimensions.

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 5.



(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 5. 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 summed 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 3.1 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 3.1 *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 3.2 `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 $O(\log N)$.*

5. Acknowledgments

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

References

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

[2] 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.

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

[4] 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.

[5] M. Peercy and P. Banerjee. Distributed algorithms for shortest-path and deadlock-free routing and broadcasting in arbitrarily faulty hypercubes. In *FTCS-20*, pages 218–225, 1990.

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