Routing and Broadcasting in Faulty Hypercube Computers

Tze Chiang Lee and John P. Hayes

Advanced Computer Architecture Laboratory
Electrical Engineering and Computer Science Department
University of Michigan
Ann Arbor, Michigan 48109

Abstract This paper examines routing and broadcasting algorithms for hypercube computers subject to node failures. First some simple message-passing algorithms are described which perform well with certain fault patterns, but poorly with others. The concept of an unsafe node is introduced to identify fault-free nodes that may cause communication difficulties in faulty hypercubes. It is then shown that by only using "feasible" paths that try to avoid unsafe nodes, routing and broadcasting can be substantially simplified. It is assumed that each active node is supplied with the fault status of all neighboring nodes within a specified radius k. A computationally efficient routing algorithm is presented which can route a message via a path of length no greater than $p+2$, where p is the minimum feasible distance from the source to the destination, provided that not all non-faulty nodes in the hypercube are unsafe, and $k = 1$. We further show that broadcasting can be achieved under the same fault conditions with only one more time unit than the fault-free case.

1. Introduction

Massively parallel distributed-memory machines are receiving increasing attention to meet the demand for high-speed reliable processing coupled with low hardware cost. Many interconnection structures have been proposed for these machines, and a few have been implemented. Among the latter, the n-dimensional hypercube or n-cube Q_n offers a number of advantages [6]. It has $N = 2^n$ nodes (processors) and a relatively small diameter which is achieved with $n2^{n-1}$ edges (links). Processors typically communicate by transmitting messages in store-and-forward fashion via intermediate processors. The hypercube architecture can handle a reasonable amount of interprocessor message traffic, and also provide some degree of fault tolerance. When one or more nodes fail, the relatively large number of links often enable the fault-free nodes to continue communicating with one another. In this paper, we examine the design of efficient routing and broadcasting algorithms for faulty hypercube computers.

The node-to-node routing problem for fault-free hypercubes has been studied by several researchers. Sullivan and Bashkow [1] describe what is now the standard algorithm for routing messages between arbitrary pairs of nodes of Q_n. This simple deterministic algorithm always finds a minimum-length path, whose length is at most n, with very little computation overhead. The message traffic congestion sometimes occurring with deterministic routing can be alleviated by using adaptive methods, such as routing messages via the least congested routes. Valiant [8] defines an alternative non-deterministic approach that adds randomness to the message traffic to reduce congestion. This method first sends each message to a randomly chosen node and then routes it to the final destination via a randomly chosen minimal path. The message delay is $O(\log N)$ with high probability. The standard algorithm for broadcasting a message to all nodes from a single source is also given in [1]; this falls in the class of Spanning Binomial Tree (SBT) algorithms which attempt to find an embedded binomial tree in the hypercube. Ho and Johnson [4] discuss the SBT algorithm in detail and also define a faster variant of it called the Multiple Spanning Binomial Trees (MSBT) algorithm. Kateseff [3] describes routing and broadcasting algorithms for "incomplete" hypercubes which are hypercubes with certain nodes removed. All the foregoing message-passing algorithms break down, however, when nodes of the hypercube fail.

This paper proposes several routing and broadcasting algorithms for hypercubes that are subject to node failures. We first show that if each node knows the locations of faulty nodes in its immediate vicinity, simple routing algorithms can be designed to tolerate certain "local" faults. These algorithms require little computation overhead to decide the next forwarding node for a message. However, under some complex fault patterns they cause unacceptable delay by routing messages through unnecessarily long paths. This problem can be solved by giving each node complete information about the fault locations in the hypercube and having it compute a minimal path for each message. However, the potentially large storage requirements for fault information as well as the associated computation overhead make this approach unattractive.

We next consider routing and broadcasting in faulty hypercubes when each node has limited information about the faults in the system. The basic idea is to identify certain nodes in Q_n that may cause excessive routing delay with the current fault pattern. By not forwarding messages to these "unsafe" nodes during message routing or broadcasting, the message-passing process can be simplified and its delay reduced. Some properties of unsafe nodes are presented, and a distributed algorithm with communication complexity $O(n^3)$ is devised to identify all the unsafe nodes. We show that after all unsafe nodes are identified, a computationally efficient routing algorithm can direct each message to its destination via a path of

* Permission to copy without fee all or part of this material is granted provided that the copies are not made or distributed for direct commercial advantage, the ACM copyright notice and the title of the publication and its date appear, and notice is given that the copying is by permission of the Association for Computing Machinery. To copy otherwise, or to republish, requires a fee and/or specific permission.

© ACM 1988 0-89791-273-X/88/0007/0346 $1.50

length no greater than the minimal source-destination path length plus two, provided that the number of faulty nodes is no more than $\lceil n/2 \rceil$. We also show that broadcasting can be achieved within $n+1$ steps. Extension of the results to systems with link failures is also discussed.

2. Preliminaries

Each node of an n-dimensional hypercube Q_n is given a distinct n-bit label $x_i = l_1 l_2 ... l_n$, such that two nodes connected by a link (edge) have labels that differ in exactly one bit. The number of bits in which labels x_i and x_j differ is denoted by $d(x_i,x_j)$; this is the Hamming distance between the nodes. A link connecting two nodes has *index i* if the labels of the nodes differ only in the i-th bit. A k-dimensional *subcube* (k-*subcube*) of Q_n, where $k \le n$, is a subgraph of Q_n which is a k-dimensional hypercube. A k-subcube of Q_n can be represented by a ternary vector $A = a_1 a_2 ... a_n$, where $a_i \in \{0,1,*\}$, and * denotes an element that can be both 0 and 1. For example, $A = 01**$ represents the 2-subcube of Q_4 with the node set {0100, 0101, 0110, 0111}. Given two subcubes $A = a_1...a_i...a_n$ and $B = b_1...b_i...b_n$, the *intercube distance* $D_i(A,B)$ between A and B along the i-th dimension is 1 if $\{a_i, b_i\} = \{1, 0\}$; otherwise, it is 0. The distance between two subcubes A, B is given by $D(A,B) = \sum_{i=1}^{n} D_i(A,B)$.

A *path P* of length l is an ordered sequence of nodes $(x_{i0}, x_{i1}, x_{i2}, ..., x_{il})$, where the x_{ij}'s are node labels with $0 \le j \le l$, and $x_{ik} \ne x_{ik+1}$, for $0 \le k \le l-1$. An equivalent representation of P, which we call a *link representation*, and which will be useful later, is $x_{i0}(j_1, j_2, ..., j_l)$, where j_k is the index of the link between x_{ik-1} and x_{ik}. This means that P starts from x_{i0}, first passes through the link with index j_1, then passes through the link j_2, and continues until it reaches the link with index j_l. A path between two nodes is *minimal* if there is no shorter path between the nodes. Two nodes separated by Hamming distance r have $r!$ different minimal paths connecting them. It is possible to find n disjoint paths between every pair of nodes, among which r paths have length r and $n - r$ have length $r + 2$.

In our discussion, the nodes of a hypercube represent processors which communicate with one another by passing messages over links in store-and-forward fashion. Each link between adjacent nodes represents a bidirectional communication channel for message passing. We assume that a processor can send out a message through only one channel at a time, and that each node-to-node message transfer requires one time unit. A routing algorithm directs the message from the source to the destination via some path in Q_n and is executed by each processor along the path. A node is *faulty* if the corresponding processor fails. It is assumed that a faulty node and all links connected to it are effectively removed from Q_n and so cannot be used for routing messages. An attempt to use a faulty node or one of its links may result in the loss of messages. A path containing no faulty nodes is called a *feasible* path. A feasible path is *minimal* if it is the shortest feasible path in Q_n between the end nodes. We say a routing algorithm for a faulty hypercube is *optimal* if it finds a minimal feasible path for each message, whenever such a path exists.

To facilitate our discussion, we define $f_i(x_k,x_j)$ to be the coordinate of the i-th bit from the left in which the labels x_i and x_j differ. A similar function $g_i(x_k,x_j)$ gives the position of the i-th common bit between x_k and x_j. If x_k and x_j have no (all) bits in common, g_i (f_i) assumes the value 0. For example,

$f_1(001101,001010) = 4$
$f_2(001101,001010) = 5$
$g_1(001101,001010) = 1$
$g_2(001101,001010) = 2$
$f_1(001101,001101) = 0$
$g_1(001101,110010) = 0$

We now briefly describe the standard deterministic routing and broadcasting algorithms used in fault-free hypercube computers. The basic routing algorithm ROUTE, as discussed in [1], is defined in Figure 1(a). Let *src* be the label of the source node and *dest* be the label of destination. The message traverses the path $src(f_1(src,dest), f_2(src,dest), ..., f_k(src,dest))$, where $k = d(src,dest)$. For example, if $src = 00000$ and $dest = 01101$, then the message path will be 00000/(2,3,5), i.e., from node 00000 to 01101 via nodes 01000 and 01100. Note that each time a node forwards a message, the distance to the destination is reduced by one. Hence this routing technique always selects a minimal path and so is optimal.

The basic broadcasting algorithm BROADCAST for fault-free hypercubes is summarized in Figure 1(b). An n-bit control word called *CONTROL* is sent with each message to tell a receiving node how it should continue broadcasting the message. Examples of such control words are the *travel* array in [3], and the *weight* variable used in [1]. If an intermediate node receives a message with the i-th bit of *CONTROL* set to 1, then it sends out a copy of this message on link i. Figure 2 shows how this algorithm works in Q_4. The 4-bit word *CONTROL* received by each node is shown in angle brackets, and an arrow labelled i indicates that a copy of the message is sent along the corresponding link in the i-th time unit. Note that the message paths of this broadcasting algorithm form a spanning tree of Q_n.

```
Algorithm ROUTE;
{In the following, curr is the current node label and dest is the
 destination node label.}
begin
    for every message
      if curr = dest
        then Retain message in node curr;
        else Send message to neighboring processor via link
             f1(curr,dest);
        {f1(xk,xj) is the first bit in which node labels xk and xj
         differ.}
end.
```

(a)

```
Algorithm BROADCAST;
{src is the source node label.}
begin
    if curr = src
      then for i = 1 to n   CONTROL[i] ← 1;
    for i = 1 to n
      if CONTROL [i] = 1
        then begin
             CONTROL [i] ← 0;
             Send a message and CONTROL to neighbor via link i;
        end;
end.
```

(b)

Figure 1. (*a*) Algorithm ROUTE for node-to-node routing; (*b*) algorithm BROADCAST for broadcasting in a fault-free hypercube.

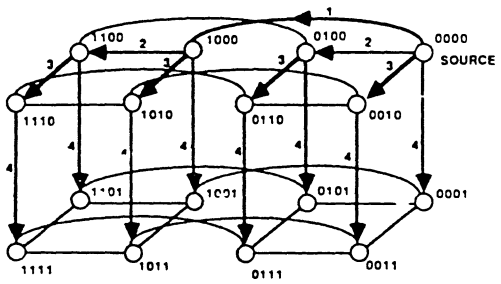

Figure 2. Broadcasting a message in a fault-free hypercube using BROADCAST.

3. Routing in Faulty Hypercubes

We begin by illustrating three common problems of routing messages in a faulty hypercube that our method addresses. Consider first the situation shown in Figure 3, which is Q_4 with three faulty (black) nodes 0110, 0101 and 0000. Suppose that a message is to be sent from node 1110 to node 0100. If node 1110 does not "know" that node 0110 has failed, and attempts to send the message through it, then the message is lost. But if node 1110 is aware that node 0110 is faulty, then it can send the message to the fault-free node 1100, which can forward the message to the destination node 0100.

Next suppose that node 0111 wants to communicate with node 0100; in this case, all the minimal paths between these nodes have been destroyed by the node failures. Depending on the routing algorithm used, the message may first be sent to node 1111 and then find its way to 0100 either through 1110 and 1100, or through 1101 and 1100. Although the length of each of these paths is greater than the Hamming distance (2) between the source and destination, the paths are minimal feasible paths. A different routing strategy that is "unaware" of the failure of node 0000 may send the message to node 0011 first. The message can subsequently find its way to node 0100 via node 0000. Since all the minimal paths from 0011 to 0100 are blocked by faulty nodes, another longer path has to be chosen. This increases the path length by at least two.

Finally, if node 1111 has to send a message to node 0100, the message can travel through nodes 1110 and 1100 to its destination. In this case the path length is three, which is the minimum. But if node 1111 first sends the message to node 0111, the extra routing delay illustrated in the previous example results. More complex routing problems can be constructed by varying the fault pattern.

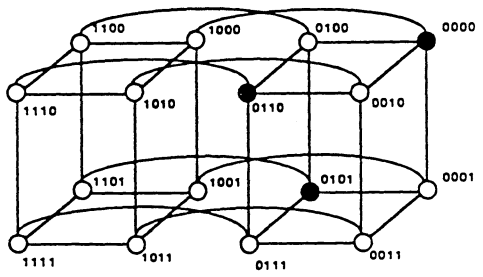

Figure 3. Hypercube Q_4 with three faulty (black) nodes.

The examples above show that knowledge of the locations of faults in Q_n is necessary to assure the the success and efficiency of message routing. We assume there exists a mechanism to detect the faulty/non-faulty status of a node and to provide this fault information to nodes that need it for routing decisions. Thus each node is assumed to store fault information, called its *fault list*, for some designated set of nodes. There is a simple and fairly natural way of defining the information that node x has about node y in terms of the distance between x and y. Node x is said to have *k-neighborhood* information if its fault list identifies all faulty nodes at distance k or less from x. If $k = n$, then each node knows the fault state of the entire system; when $k = 1$, each node knows only the state of its nearest neighbors.

We will assume that messages are only sent to or from non-faulty nodes. Later we will briefly discuss how cases where this condition is violated might be handled.

3.1 Two-State Nodes

First we assume that every node x is in one of the two states, faulty or non-faulty, and has a fault list identifying all faulty nodes in its k-neighborhood. (Later in Section 2.2, we will introduce a third condition called unsafeness to further refine the non-faulty state of a node.) When a message needs to be sent from or forwarded by x, node x attempts to compute a shortest feasible path based on its fault list. If $k = n$, i.e., x has a complete fault list, then an exhaustive search through all known source-destination paths always leads to an optimal routing decision by x. However, the size of the fault list that must be stored, and the high computation time to search through the fault list may make this approach unattractive. When $k < n$, the fault lists require less memory space and computation, but the routing algorithm may no longer be optimal. Thus there are obvious trade-offs between the quality of the routing decisions, the quantity of information stored in each node, and the computation time needed.

The routing computation time can be reduced by checking only the disjoint paths from source to destination to find a minimal feasible path. Figure 4 shows a routing algorithm, called ROUTE1(k), based on this approach. The parameter k indicates that each node's fault list specifies all faulty nodes within its k-neighborhood. ROUTE1(k) generates disjoint paths by using the fact that, if $a_i \neq a_j$ for $i \neq j$, then the paths $x(a_1,a_2,a_3,\ldots,a_m)$, $x(a_2,a_3,\ldots,a_m,a_1)$,..., $x(a_m,a_1,\ldots,a_{m-1})$, obtained by cyclically shifting the first path, have only their first and last nodes in common. Suppose that a message is to be sent between two nodes x and y separated by the Hamming distance r. Let $a_i = f_i(x,y)$, for $i = 1,\ldots,r$, and let *curr* be the current node label. ROUTE1(k) checks the first k nodes in each of the r paths $curr(a_1,a_2,a_3,\ldots,a_r)$, $curr(a_2,a_3,\ldots,a_r,a_1)$,..., $curr(a_r,a_1,\ldots,a_{r-1})$, against the current node's fault list to see if there exists a feasible path, i.e., one which is not blocked by faulty nodes. The first path found that has no faults in the first k nodes is chosen, and the message is sent to the node in this path that is adjacent to the current node. If all these minimal paths are blocked by faulty nodes, then ROUTE1(k) checks through the non-minimal path $P_i = curr(g_i(x,y),a_1,a_2,\ldots,a_r,g_i(x,y))$, for i from 1 to $n - r$. The first P_i which contains no faulty node among its first k nodes is then chosen.

Theorem 1: Suppose that a message is to be passed between two nodes separated by Hamming distance r. Algorithm ROUTE1(k) finds a minimal feasible path for the message if each non-faulty node in the system has no more than k faulty nodes in its k-neighborhood and $k < r$.

```
Algorithm ROUTE1(k);
{Each node keeps a fault list FAULT for its k-neighborhood.}
begin
    l ← d(curr,dest);
    {d(curr,dest) is the Hamming distance between curr and dest}
    for i = 1 to l   m(i) ← i;
    for i = 1 to l
       if no node in FAULT is in the first k nodes of the path
          curr \(f_{m(1)}(curr,dest), f_{m(2)}(curr,dest),...f_{m(l)}(curr,dest))
          then begin
             Send message via link f_{m(1)}(curr,dest),
             Exit;
          end;
          else  for i = 1 to l   m(i) ← (m(i) mod l) + 1;
                {Shift the indexes cyclically.}
    for i = 1 to n - l
       if no node in FAULT is in the first k nodes of the path
          curr \(g_i(curr,dest) f_1(curr,dest),...f_l(curr,dest),g_i(curr,dest))
          then Send message via link g_i(curr,dest);
end.
```

Figure 4. Algorithm ROUTE1(k) for routing messages in faulty hypercubes.

Proof: There are r disjoint minimal paths between the source and destination nodes. Since $k < r$, the source always selects a path which it considers feasible according to its limited (k-neighborhood) fault information, and sends the message to the next node in that path. The same procedure is repeated until the message is forwarded to a node x at distance $k + 1$ from the destination. Node x then chooses a path P which it thinks is feasible among the $k + 1$ disjoint minimal paths from x to the destination, and forwards the message to the next node in P. Note that there always exists such a P since: (1) $k + 1 > k$, the maximum number of faults; (2) x knows the fault status of the first k nodes in P; and (3) the (k+1)-th node is the destination node, which is assumed fault-free. We also observe that if node x_1 finds that $P = (x_1, x_2, ..., x_l)$ is feasible for $l \le k$, then node x_i, where $1 \le i \le l$, can determine that $P^* = (x_i, x_{i+1}, ..., x_l)$ is feasible, since each node follows the same checking procedure. Hence this routing algorithm forwards the message through the minimal feasible path.

Corollary 1: If $k \le 2$ and no non-faulty node has more than k faulty nodes in its 1-neighborhood, then ROUTE1(k) is optimal.

When $k \le 2$, ROUTE1(k) searches the disjoint minimal paths for one which is feasible according to the current node's fault list. If there exists a minimal path which is feasible, the message will be sent through it. If all minimal paths are blocked, ROUTE1(k) finds a feasible path of length two greater than the minimal path length, and this must be a minimal feasible path. Hence ROUTE1(k) is optimal, for $k \le 2$. When $k \ge 3$, the disjoint paths no longer cover all minimal paths, hence the routing may not be optimal.

To improve the quality of the routing decisions for $k \ge 3$, ROUTE1(k) can be modified to form a new algorithm ROUTE2(k) that searches the set of all minimal paths (non-disjoint as well as disjoint) for a feasible one according to the current fault list. If no such path exists, the method of ROUTE1(k) is used to search for the shortest non-minimal path that is feasible.

Theorem 2: ROUTE2(k) is optimal for $k \le n$-1, provided that there are no more than k faulty nodes in the k-neighborhood of each non-faulty node.

Proof: Suppose that a message is to be passed between two nodes separated by Hamming distance r, where $k < r$, ROUTE2(k) reduces to ROUTE1(k), which can always find a minimal feasible path by Theorem 1. When $k = r$, ROUTE2(k) also finds a minimal feasible path, if one exists, by checking all possible minimal paths. If all such paths are blocked (which needs at least k faulty nodes), then ROUTE2(k) will direct the message to a node at distance $k + 1$ from the destination; this reduces to the case $r = k + 1$ considered above. Finally, when $k > r$, ROUTE2(k) checks all the minimal paths between the source and destination to find a feasible one. If the minimal paths are all blocked, ROUTE2(k) checks the other $n - r$ disjoint paths with length $r + 2$. Since $n - r > k - r$, at least one such path is a minimal feasible path. Therefore ROUTE2(k) finds a minimal feasible path for a message where $k > r$. Since in all cases ROUTE2(k) finds the minimal feasible path, the theorem follows.

We also note that ROUTE2(n) is optimal provided that there are fewer than n faults in the system. One can easily see that ROUTE2(1) = ROUTE1(1) and ROUTE2(2) = ROUTE1(2). ROUTE2(1) is optimal only when no non-faulty node has more than one faulty nearest neighbor, while ROUTE2(n) places no such restriction on the fault pattern. These results show that the straightforward ROUTE1/ROUTE2 approach can tolerate many faults if the faults are dispersed. However, optimality is lost if the conditions stated in Corollary 1 and Theorem 2 are violated.

3.2 Three-State Nodes

We now introduce the notion of an unsafe node based on the observation made in the previous section that certain nodes surrounded by faulty nodes can trap or delay messages forwarded to them by ROUTE1(1), the weakest version of ROUTE1(k). A non-faulty node is in the *unsafe* state if it has at least two faulty or unsafe nearest neighbors. All unsafe nodes can be identified recursively from this definition. For example, node 0010 in Figure 3 is unsafe because it is adjacent to two faulty (black) nodes. Similarly, node 0001 is unsafe. Since 0011 is adjacent to the non-faulty but unsafe nodes 0010 and 0001, it too is unsafe, and so on. A non-faulty node which is not unsafe is *active*. Thus a node may now be in one of the three states: active, unsafe or faulty. A subcube of dimension 1 or more is an *unsafe subcube* if it contains only unsafe or faulty nodes. An unsafe subcube is maximal if it is contained in no other unsafe subcube. For example, the 3-subcube 0*** of Figure 3 is a maximal unsafe subcube. We will show that if there are no more than $\lceil n/2 \rceil$ faulty nodes in an n-dimensional hypercube, a ROUTE1(1)-like algorithm guarantees near-minimal routing delay.

Before proceeding, we consider how all unsafe nodes might be identified by an actual hypercube system. Figure 5 is a distributed algorithm called FIND_UNSAFE to identify unsafe nodes which is designed for parallel execution on a hypercube. We assume that each node keeps lists of the faulty and unsafe nodes of its 1-neighborhood. Initially, each node has only a list of faults in its 1-neighborhood, and an empty unsafe list. Each non-faulty node then checks its faulty and unsafe lists to decide if it itself is unsafe, and it exchanges this decision with all its non-faulty nearest neighbors. The unsafe list of each node is then updated if necessary. The communi-

```
Algorithm FIND_UNSAFE;
{Each node keeps a fault list FAULT of length l_f, and an
 unsafe list UNSAFE with length l_s for its 1-neighborhood.
 Initially UNSAFE is empty.}
begin
    for i = 1 to B_n begin
    {B_n is the the number of times each node exchanges
     its status information with its non-faulty nearest
     neighbors; see the Appendix }
       if l_f + l_s > 1
          then Mark the current node as unsafe;
       for j = 1 to n begin
          Receive status s from neighbor x connected by
          link j;
          if s = unsafe and x is not in UNSAFE begin
             Add x to UNSAFE;
             l_s ← l_s + 1;
          end;
       end;
    end;
end.
```

Figure 5. Algorithm FIND_UNSAFE for identifying unsafe nodes in Q_n.

cation complexity of this algorithm is $O(n^3)$; see the Appendix. If each node has n-neighborhood information, then an algorithm with computation complexity $O(f^3)$ can be devised to identify all unsafe nodes, where f is the number of faults.

The following theorem defines the structure of the unsafe nodes of Q_n.

Theorem 3: The unsafe and faulty nodes of a hypercube form a set of maximal unsafe subcubes such that the distance between any two of them is at least three.

To see why this is true, we first observe the following property of algorithm FIND_UNSAFE. Two nodes (or subcubes) which are either faulty or unsafe, and at distance two from each other, will make the node between them unsafe if that node is non-faulty. This is because the intermediate node has at least two unsafe or faulty nearest neighbors. Therefore unsafe subcubes at distance two or less from one another will merge into a bigger unsafe subcube. These merge operations will not cease until all unsafe subcubes are at least at distance three from one another.

As discussed in the Appendix, it takes at least $\lceil k/2 + 1 \rceil$ faulty nodes to make a k-subcube unsafe. Hence if the number of faulty nodes in Q_n is no more than $\lceil n/2 \rceil$, Q_n can not be unsafe. In the following discussion, we always assume that Q_n is not unsafe.

Next we consider using the unsafe node information to simplify the routing problem. A routing algorithm ROUTE3(1), which is an extension of ROUTE1(1) to handle unsafe nodes, is described in Figure 6. The underlying principle of ROUTE3(1) is to avoid forwarding messages to unsafe nodes, if possible. However, ROUTE3(1) will send a message to an unsafe neighbor if all the active nearest neighbors lead to infeasible paths, or paths that are of non-minimal length.

ROUTE3(1) may be summarized as follows. For a message at node *curr* destined for node *dest*, ROUTE3(1) checks sequentially if the node connected by the link indexed $f_i(curr,dest)$ is active, for i from 1 to $d(curr,dest)$. The first such node is chosen as the next node for forwarding the message. If there exists no such active neighbor, then ROUTE3(1) chooses an unsafe neighbor connected by the link indexed $f_i(curr,dest)$ with the smallest i. If all neighbors connected by the links indexed with $f_i(curr,dest)$, for i from 1 to $d(curr,dest)$, are faulty, ROUTE3(1) forwards the message to an active neighbor connected by the link with the smallest index. Note that since Q_n is not unsafe, each unsafe node in Q_n must have at least one active nearest neighbor.

We claim that if only the active nodes communicate with one another, algorithm ROUTE3(1) always finds a minimal feasible path for every message. Thus the algorithm is optimal in this particular case. This can be as follows. After identifying all the unsafe nodes, the remaining active nodes will each have at most one unsafe or faulty nearest neighbor. If we use ROUTE1(1) to route messages and allow no messages to be passed via unsafe nodes (as if they were faulty), then by Corollary 1 in the previous section, algorithm ROUTE1(1) is optimal. Since ROUTE3(1) employs the same strategy as ROUTE1(1) in this case, ROUTE3(1) is also optimal. Figure 7 illustrates an example of this case. Active (white) node a wants to communicate with active node b. ROUTE3(1) routes the message sent from a via a minimal feasible path that avoids unsafe (gray) nodes.

Now consider the case when the source node is active and the destination is unsafe. ROUTE3(1) forwards the message to the destination via a minimal feasible path in which all intermediate nodes are active. An example of this is shown in Figure 7 for nodes c and d. Similarly, in the reverse case when the source node is unsafe and the destination is active, ROUTE3(1) also forwards the message via a minimal path. Hence in both cases ROUTE3(1) is optimal.

Finally, suppose that both the source and destination nodes are unsafe. If they are in different maximal unsafe subcubes, then there always exist minimal paths between them with all intermediate nodes active. ROUTE3(1) forwards the message through one of these minimal paths. An example is also given in Figure 7 for nodes e and f. If the source and destination nodes are in the same unsafe subcube, the message is forwarded to the destination inside this unsafe subcube along

```
Algorithm ROUTE3(1);
begin
    l ← d(curr,dest);
    for i = 1 to l
      if neighbor connected by link f_i(curr,dest) is not in either
        UNSAFE or FAULT
        then begin
        {Select an active neighbor in a minimal path to dest.}
            Send message via link f_i(curr,dest);
            Exit;
        end;
    for i = 1 to l
      if neighbor connected by link f_i(curr,dest) is not in FAULT
        then begin
        {Select an unsafe neighbor in a minimal path to dest.}
            Send message via link f_i(curr,dest);
            Exit;
        end;
    for i = 1 to n-l
      if neighbor connected by link g_i(curr,dest) is not in either
        FAULT or UNSAFE
        then begin
        {Select other active neighbor.}
            Send message via link g_i(curr,dest);
            Exit;
        end;
end.
```

Figure 6. Algorithm ROUTE3(1) for routing messages in faulty hypercubes with unsafe nodes.

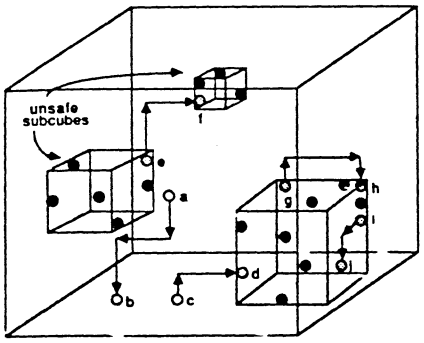

Figure 7. Message paths used by ROUTE3(1) in a faulty hypercube with unsafe nodes.

a minimal feasible path, if ROUTE3(1) can find one, e.g., the path between nodes i and j in Figure 7. However, if at certain intermediate nodes all such paths are blocked by faulty nodes, the message is sent out from an unsafe subcube to an active node. Then it is forwarded via a minimal path from this active node to the destination with all intermediate nodes active as described in the previous case. This makes the total path length two more than the length of a minimal path. An example can be found in Figure 7 for nodes g and h. Based on the above discussion we now arrive at the following theorem.

Theorem 4: If Q_n is not unsafe, then ROUTE3(1) routes messages from source to destination via a path of length no more than the minimal path length plus two.

As mentioned above, when the source and destination nodes of a given message are in the same unsafe subcube, the message may be forwarded via a path of length two more than the minimal path length by ROUTE3(1). However, not all such messages will experience this routing delay. To analyze this issue, we have conducted a series of simulation experiments that estimate the average number of unsafe nodes generated by different numbers of faulty nodes. Figure 8 summarizes the results. It can be seen that as long as the number of faulty nodes is no more than $\lceil n/2 \rceil$, the percentage of the unsafe nodes present in the system is less than 15% for $n \geq 5$. In other words, if each node has equal probability of communicating with any other node, the percentage of messages which can have the extra delay of two is less than 15%. This low percentage affirms the usefulness of the unsafe node concept.

The computation involved in ROUTE3(1) is fairly modest. Since there is at most one unsafe or faulty neighbor in its 1-neighborhood, each active node does no more than two label comparisons before it forwards a message. For an unsafe node, the number of comparisons is bounded by $\Theta(n)$. Whenever a new faulty node is reported, the FIND_UNSAFE algorithm may be used to update each node's lists of faulty and unsafe nodes. After this has been done, ROUTE3(1) can be used for efficient routing, as indicated by Theorem 4. It should be noted that as long as Q_n is not unsafe, there is no need to store more than 1-neighborhood information in each node, or to use more complicated routing strategies. A sufficient condition guaranteeing this efficient routing is that the number of faulty nodes be no more than $\lceil n/2 \rceil$.

In all the foregoing discussion, we assumed that no messages are sent to or received from faulty nodes. When this assumption cannot be made, the following actions can be taken. If a faulty node x tries to send a message to other nodes, one of its nearest neighbors will receive the message first. If this neighbor is non-faulty, it knows the fault state of x and can be programmed to refuse to forward any message coming from x. If some non-faulty node sends a message to a faulty node, the message must be forwarded through some non-faulty nearest neighbor of the destination. Since this non-faulty node knows the state of the destination node, it can return the message to the sender. The returned message may refer the sender to the system supervisor for the next step, which can either be to terminate the communication process or to specify a new destination node.

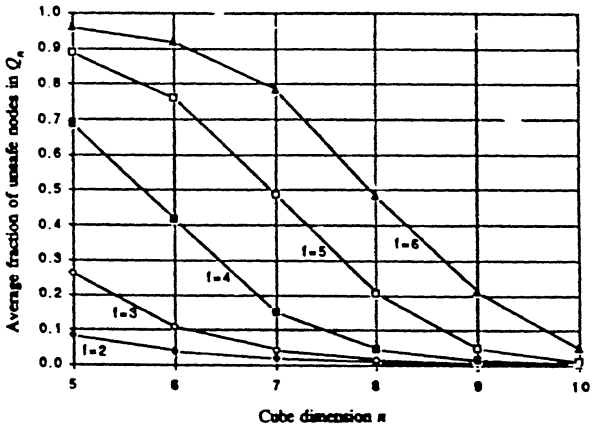

Figure 8. Average fraction of unsafe nodes in Q_n; f is the number of faults.

4. Broadcasting in Faulty Hypercubes

Suppose that each non-faulty node has unsafe and faulty node lists which cover all nodes in its 1-neighborhood. We now describe an algorithm BROADCAST1(1) to broadcast messages from an active node to all non-faulty nodes in n time units; see Figure 9. We then develop a more general algorithm called BROADCAST2(1) to broadcast messages among non-faulty nodes in $n + 1$ time units.

BROADCAST1(1) is obtained by slightly modifying the basic BROADCAST algorithm given in Figure 1. As before, we assume that an n-bit word *CONTROL* is sent along with each message. This is used to record each link not traversed because the nodes connected by it are unsafe or faulty, and to provide information telling the receiving node how to continue broadcasting. Initially all bits in *CONTROL* are set to 1. An active node x sends a message to an active neighbor via link i if the corresponding bit i of *CONTROL* is 1; it sends a message to an unsafe neighbor only if bit i of *CONTROL* is 1 and all other bits are 0's. This bit i is set to 0 when the message is sent out by x. Note that if x has to broadcast a message to several nearest neighbors one of which is unsafe, the unsafe neighbor is the last to receive the message. In this arrangement, no unsafe nodes have to transmit messages since they receive the all-0 control word. When there are no faulty or unsafe nodes, BROADCAST1(1) reduces to the original BROADCAST algorithm.

Figure 10 shows how BROADCAST1(1) works in Q_4 when there are some faulty (black) and unsafe (gray) nodes in the system. Suppose that nodes 1100 and 0101 are faulty, and

```
Algorithm BROADCAST1(1);
begin
    for k = 1 to n
      if CONTROL [k] = 1
        then if neighbor connected by link k is not in FAULT
                or UNSAFE
                then begin
                  CONTROL [k] ← 0;
                  Send message and CONTROL via link k;
                end;
    for k = 1 to n
      if CONTROL [k] = 1
        then if neighbor connected by link k is in UNSAFE
                then begin
                  CONTROL [k] ← 0;
                  Send message and CONTROL via link k;
                end;
end.
```

Figure 9. Algorithm BROADCAST1(1) for broadcasting messages from an active source in a faulty hypercube with unsafe nodes.

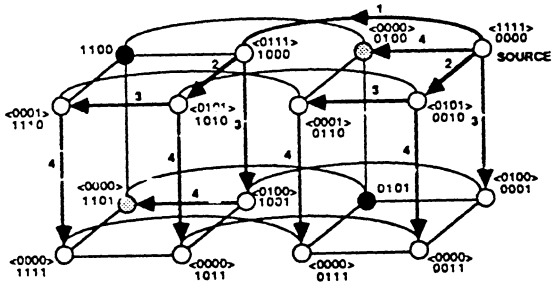

Figure 10. Broadcasting a message from an active source node using BROADCAST1(1); unsafe node are gray.

nodes 0100 and 1101 are unsafe. A message is to be broadcast to all non-faulty nodes from an active source node 0000. Node 0000 first sends the message to node 1000 with *CONTROL* = <0111>. This control word instructs node 1000 to broadcast to all nodes in 1***. Node 1000 finds that node 1100 is faulty, and all the control words sent to node 1010 and 1001 from 1000 have accordingly a 1 in the second bit position. Node 1010 then broadcasts the message to 1*1*, while node 1001 broadcasts to 1*01, which contains the unsafe node 1101. Node 0000, after sending a message to node 1000, continues to broadcast to 0***. Following a similar procedure, node 0000 subsequently sends the message to 0010 and 0001, and asks them to broadcast to the designated subcubes. In the last time unit, node 0000 sends a message to its unsafe neighbor 0100 to complete the broadcasting. Assume that each message transmission takes one time unit. The total broadcasting time required in this example is 4, which is the dimension of the cube.

Theorem 5: BROADCAST1(1) broadcasts a message from an active node to all non-faulty nodes in n time units, provided that Q_n is not an unsafe cube.

Proof: As noted above, BROADCAST1(1) assigns each active node to broadcast to a designated subcube S; the subcube size is indicated by the number of 1's in *CONTROL*. We prove the theorem by induction on the dimension of S. Since Q_n is not unsafe, it contains at least one active node x. Suppose that x is required by BROADCAST1(1) to broadcast to a 1-subcube consisting of the node set $\{x,y\}$. There are only three cases consider: (1) If y is active x sends a message to y; (2) If y faulty, then no message is sent out; (3) If y is unsafe, then sends a message to this unsafe node. All these cases require most one time unit to complete the broadcasting.

Suppose that the active node x assigned to broadcast to k-subcube S can complete its broadcasting in k time unit Now consider the case where S is a $(k+1)$-subcube. Node x fir broadcasts to an active neighbor y in S. Since $k + 1 \geq k$, . least one active neighbor must exist, otherwise x would b unsafe. Then x and y effectively partition S into two subcube S_1 and S_2; x continues to broadcast to one of the subcubes, sa S_1, while y broadcasts to S_2. Since S_1 and S_2 are each o dimension k, by the induction hypothesis the message can b broadcast to all nodes of S_1 and S_2 in k time units. Hence th broadcast to S is completed in $k + 1$ time units, and therefor the theorem follows by induction.

Execution of BROADCAST1(1) is quite simple sinc each active node only needs to check its unsafe and fault list as many times as 1 appears in the received control word. The length of the unsafe or fault list is no greater than one sinc each active node can have at most one unsafe or faulty neigh bor.

We now describe a second broadcasting algorithm, called BROADCAST2(1), derived from BROADCAST1(1). This algorithm can broadcast messages from any non-faulty node, and so is not restricted to active source nodes. When the source is active, BROADCAST2(1) reduces to BROAD-CAST1(1). When the source is unsafe, BROADCAST2(1) first sends a message to an active neighbor connected by the link with the smallest index. The active neighbor node then starts to broadcast using the same strategy as BROADCAST1(1) except that no message is sent back to the original unsafe source. Figure 11 illustrates how BROADCAST2(1) works in a faulty Q_4 with an unsafe source node 0100.

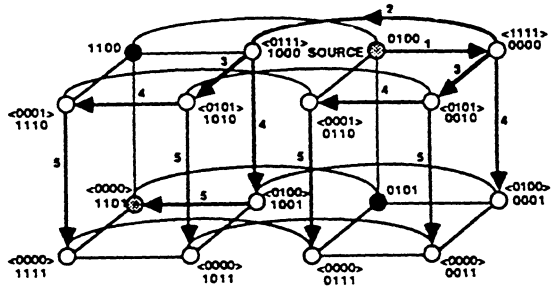

Figure 11. Broadcasting a message from an unsafe source node using BROADCAST2(1).

Theorem 6: BROADCAST2(1) broadcasts a message from any non-faulty node to all non-faulty nodes within $n + 1$ time units, provided that Q_n is not an unsafe cube.

5. Conclusions

We have presented some novel routing and broadcasting algorithms for faulty hypercube computers. They rely on giving each node limited information on the status of nodes in some defined neighborhood around it. The unsafe node concept was introduced and shown to simplify routing decisions in many instances. When the number of faulty nodes in Q_n is no

greater than $\lceil n/2 \rceil$, the ROUTE3(1) algorithm can transmit messages via a path of length no greater than the minimal path length plus two. Broadcasting can be achieved within $n + 1$ time units. The main advantages of the routing and broadcasting techniques presented in this paper are their bounded delay and their simplicity.

The foregoing results can be readily extended to accommodate link failures. For this case, we can modify the definition of unsafe node as follows. A non-faulty node is unsafe if it either has at least two faulty or unsafe nearest neighbors or is an end-node of a faulty link. This modification guarantees that the end-nodes of a faulty link are unsafe. The newly defined unsafe nodes still form a set of disjoint unsafe subcubes. If ROUTE3(1) is modified such that no message is passed through faulty links, then Theorem 4 holds for hypercubes with faulty links. BROADCAST2(1) does not permit unsafe nodes to send out messages to its unsafe neighbors. Hence no messages can pass through faulty links, so Theorem 6 can also be directly extended to hypercubes with link failures.

Acknowledgement

This work was supported by the Office of Naval Research under Contract No. N00014 85 K 0531.

References

[1] H. Sullivan, T. Bashkow and D. Klappholz, "A large scale, homogeneous, fully distributed parallel machine," *Proc. Fourth Symp. on Computer Architecture*, pp. 105-124, March 1977.

[2] J. G. Kuhl and S. M. Reddy, "Distributed fault tolerance for large multiprocessor systems," *Proc. Seventh Symp. on Computer Architecture*, pp. 23-30, May 1980.

[3] K. P. Kateseff, "Incomplete hypercubes," *Hypercube Multiprocessors 1987* (Proc. 2nd Conf. on Hypercube Multiprocessors), SIAM, Philadelphia, pp. 258-264, 1987.

[4] C. T. Ho and S. L. Johnson, "Distributed routing algorithms for broadcasting and personalized communication in hypercubes," *Proc. Int'l Conf. on Parallel Processing*, pp. 640-648, Aug. 1986.

[5] M. C. Pease, "The indirect binary n-cube microprocessor array," *IEEE Trans. Computers*, vol. C-26, pp. 458-473, May 1977.

[6] Y. Saad and M. H. Schultz, "Data communication in hypercubes," Tech. Report YALEU/DCS/RR-389, Dept. of Computer Science, Yale Univ., June 1985.

[7] C.L. Seitz, "The Cosmic Cube," *Commun. of the ACM*, vol. 28, pp. 22-33, Jan. 1985.

[8] L. G. Valiant, "A scheme for parallel communication," *SIAM J. Computing*, vol. 11, pp. 350-361, May 1982.

Appendix: Complexity of FIND_UNSAFE

In the following we consider the complexity of algorithm FIND_UNSAFE (Figure 5). Each node learns the states (active, faulty, or unsafe) of all of its nearest neighbors by checking with the non-faulty ones. A complete check of its neighbors by one node is assumed to take nc communication time units, where n is the cube dimension and c is a constant depending on the (average) number of message exchanged with each neighbor during the checking process. FIND_UNSAFE marks all unsafe nodes in Q_n by a sequence of check steps $t = 1, 2, \ldots$, each of duration nc. Figure 12 illustrates the execution of FIND_UNSAFE by Q_4 with the three indicated faulty nodes present. The number inside each unsafe node indicates the time t at which the node is marked unsafe because of the neighboring faulty or unsafe nodes. Nodes 0100 and 0010 become unsafe at $t = 1$, since each has two faulty neighbors. Now nodes 1100 and 0101 have both faulty and unsafe neighbors causing them to be marked unsafe at $t = 2$, and so on.

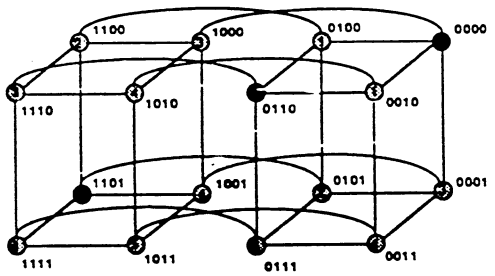

Figure 12. Sequence in which nodes in a faulty Q_4 are marked unsafe by FIND_UNSAFE.

To determine an upper bound on the execution time of FIND_UNSAFE, we consider the following worst-case situation. Let S_{00}, S_{01}, S_{10} and S_{11} be subcubes of dimension m that are connected to form an $(m+2)$-cube. Assume that at some time t, subcube S_{00} becomes unsafe, S_{01} and S_{10} have no unsafe or faulty nodes, and S_{11} has one faulty node x but no unsafe nodes. Since the distance between S_{00} and x is two, then at $t + 1$, one node is marked unsafe by FIND_UNSAFE in each of S_{01} and S_{10}, say x_1 and x_2, respectively. Note that since each node in S_{01} and S_{10} has an unsafe neighbor in S_{00}, the nodes connected to x_1 or x_2 in S_{01} and S_{10} should be marked unsafe at $t + 2$. Thus x_1 and x_2 act as "seed" nodes in their respective m-subcubes, so that m time units after they are marked unsafe, all non-faulty nodes in S_{01} and S_{10} have been marked unsafe by FIND_UNSAFE. At most one more time unit is needed to make S_{11} unsafe after S_{01} and S_{10} become unsafe, since each node in S_{11} has an unsafe neighbor in S_{01} and S_{10}.

The foregoing argument leads to the following recursive formula for the worst-case time B_m required to mark an unsafe m-subcube:

$$B_{m+2} = B_m + m + 2$$

Rewriting this with $n = m + 2$, we obtain $B_n = B_{n-2} + n$, which holds for even n, where $n \geq 4$, and $B_2 = 1$. B_n can be further rewritten in the following closed form.

$$B_n = n^2/4 + n/2 - 1$$

When n is odd, we can construct an upper bound B_n as follows. First partition the odd-dimensional cube Q_n into two even $(n-1)$-dimensional subcubes S_1 and S_2. It takes at most $(n-1)^2/4 + (n-1)/2 - 1$ time units to mark one of these subcubes, say S_1, unsafe in the worst case. Then to make S_2 unsafe, there must be at least one faulty node residing in S_2. Assume this is the case and that the non-faulty nodes in S_2 are to be marked unsafe. It takes at most n - 1 additional time units to make S_2, and therefore Q_n, unsafe. Hence

$$B_n = (n-1)^2/4 + (n-1)/2 - 1 + (n-1) = n^2/4 + n - 9/4$$

Thus we conclude that $B_n = n^2/4 + n/2 - 1$ when n is even, and $B_n = n^2/4 + n - 9/4$ when n is odd. Note that B_n is a tight bound when n is even, but not when n is odd. Since we assumed that each time unit is of duration nc, the execution time of FIND_UNSAFE is bounded above by ncB_n, hence the communication complexity of FIND_UNSAFE is $O(n^3)$.

We also observe that the least number of faults needed to make an Q_n unsafe is $\lceil n/2 + 1 \rceil$. Figure 12 shows such an example where $n = 4$ and $\lceil n/2 + 1 \rceil = 3$ faults cause Q_4 to become unsafe.