Journal of Interconnection Networks, Vol. 2, No. 3 (2001) 365–378

LOCAL-SAFETY-INFORMATION-BASED BROADCASTING IN HYPERCUBE MULTICOMPUTERS WITH NODE AND LINK FAULTS

DONG XIANG and AI CHEN
Institute of Microelectronics, Tsinghua University, Beijing 100084, P. R. China
E-mail: xd@dns.ime.tsinghua.edu.cn

JIE WU
Dept. of Computer Sci. and Eng., Florida Atlantic University, Boca Raton, FL33431, USA
E-mail: jie@cse.fau.edu

Received 10 August 2000
Revised 1 March 2001

This paper presents a method to cope with fault-tolerant broadcasting in hypercube multicomputers with both node and link faults. The local safety concept is extended to faulty hypercubes with both node and link faults. The local-safety-based algorithm is used in a fully unsafe hypercube, where there is no safe node. A fully unsafe hypercube can be split into a set of maximal safe subcubes. We show that if these maximal safe subcubes meet certain requirements given in the paper, broadcasting can still be carried out successfully and in some case optimal broadcast is still possible. The method is extended to fault-tolerant routing and multicasting when the system contains both node and link faults.

Keywords: Broadcast subcube, fault-tolerant broadcasting, local safety, hypercube multicomputer, maximal safe subcube

1. Introduction

The desire for obtaining more powerful computers leads us to build multicomputers. Performance of a multicomputer system highly depends on the communication cost and the balance of computation. The hypercube architecture can handle a reasonable amount of message traffic, and also provide some degree of fault-tolerance. A couple of commercial or research hypercube systems have been constructed in the past two decades [1,11,17,18]. Especially, the recent SGI Origin 2000 employs hypercube architecture. When some nodes or links fail, communication between fault-free nodes should still continue. Fault-tolerant communication [2-5,7,10,12-15,19-25] has been studied extensively. The hypercube interconnection network is quite suitable for multicomputer and distributed shared memory systems according to Duato's experimental study [6].

Efficient broadcasting of data is one of the keys to the performance of a multicomputer. Broadcasting is the process of transmitting data from a node called the source to all other nodes once and only once. Data broadcasting in fault-free networks has been studied intensively in [1,8,9]. One-to-all fault-tolerant broadcasting passes a message from a source to all fault-free nodes in a faulty hypercube in the literature [2,7,13-15,19]. Park and Bose [12] presented an all-to-all broadcast algorithm for hypercube with up to $n/2$ link failures in a binary $n-$cube.

A couple of limited-global-fault-information-based methods were introduced to deal with fault-tolerant communication in hypercubes [4,5,10,19-25]. Lee and Hayes [10] proposed the concept of the safe node for the first time. Routing is to avoid unsafe nodes which could possibly lead to communication difficulties assume the hypercube is not fully unsafe. Lee and Hayes [10] also proposed a fault-tolerant broadcast algorithm based on the safe node concept. Priority order is determined based on status of neighbors of the node under process to send the broadcast label and the message in order to avoid communication difficulties. Wu and Fernandez [21], Chiu and Wu [5] refined the safe node concept. A fault-tolerant broadcast algorithm was presented based on the refined safe node concept in [21]. Just like [10], a message can be broadcast reliably only if the binary $n-$cube is safe although reliable message passing is still possible in a fully unsafe hypercube in many cases. Unsafe nodes were classified based on degree of safety status to facilitate the design of an efficient routing algorithm in [5]. A feasible path of length no more than the Hamming distance between the source and destination plus four can be established as long as the hypercube is not fully unsafe. A mechanism called the safety level was proposed to assist an efficient fault-tolerant broadcast in Wu [19]. Priority order to forward the broadcast data is determined by the safety level numbers.

However, safety level [19], safety vector [20], and routing capability [4] consider safety inside the $k-$distance neighborhood. It is found that a lot of further resilience of hypercube topology still have not been utilized by the above methods. Local safety [22-25] considers safety inside the minimum subcube that contains the source and destination. Local safety is proposed to cope with fault-tolerant broadcasting for hypercube multicomputers. A fully unsafe hypercube can be split into a unique set of maximal safe subcubes. Message-passing inside a maximal safe subcube can be completed reliably. An algorithm is proposed to calculate local safety information using which the required extra cost is only comparable to that to derive the global safety information. Local safety is utilized in the fault-tolerant broadcast algorithm by only considering safety of the broadcast subcube (which indicates the range of nodes the message should be sent).

Notation and definitions are introduced in Section 2. Local safety is proposed to assist fault-tolerant communication in Section 3. A fault-tolerant broadcast algorithm is presented according to the local safety information in Section 4. The local-safety-based broadcast algorithm is successfully extended to fault-tolerant routing and multicasting when the system contains both node and link faults in Section 5.

2. Preliminaries

An $n-$dimensional hypercube has 2^n nodes (or processors). Each node v can be represented by a sequence of n binary bits $(v_n v_{n-1} \dots v_1)$, where $v_i \in \{0,1\}$. A subcube SC of a hypercube can be represented by a sequence of n bits $c_n c_{n-1} \dots c_2 c_1$, where $c_i \in \{0,1,*\}$, and "*" indicates don't care (can be assigned both 0 and 1). Two nodes are connected by a bidirectional link if and only if the binary representations of the two nodes differ in exactly one bit. The Hamming distance $H(x,y)$ between two nodes x and y is the number of bits in which the labels of x and y differ. The *spanning subcube* $SC(x,y)$ between two nodes x and y represents the smallest subcube that contains x and y. We only consider message-passing between fault-free nodes. A path is feasible if there is no faulty node in the path. A path is called a minimum path if length of the path is equal to the Hamming distance from the source to the destination. The node $s^{(i)}$ $(1 \leq i \leq n)$ represents the neighbor of s along dimension i in the hypercube.

A fault-free node in an $n-$dimensional hypercube is defined as *unsafe* [4,21] if it has at least two faulty neighbors, or three or more unsafe or faulty neighbors. A faulty hypercube is called *fully unsafe* if it contains no safe node; otherwise, it is a safe cube. An unsafe node is *ordinarily unsafe* if it has at least one safe neighbor, otherwise, it is *strongly unsafe.*

In the 4−cube given in Fig. 1, 0000 is safe, 0001 is ordinarily unsafe, and 1011 is strongly unsafe. Incomplete spanning binomial tree is utilized to implement broadcasting. An incomplete spanning binomial tree in an $n-$dimensional faulty hypercube is a connected subgraph of an $n-$level spanning binomial tree with the same root node that connects all fault-free nodes in the $n-$cube, whose root is called an $l-$node. Usually, the $l-$node set contains the safe node set. Fig. 1 presents broadcasting with the source 0100, which is a safe node. The incomplete binomial tree is presented in Fig. 1. Each fault-free node has a broadcast label, which shows the range that the message from the node should be distributed.

Definition 1 *The broadcast subcube of a node is defined based on the received broadcast label by replacing the bits of the node's address with don't care, where the received broadcast label of that node are assigned value 1.*

Let node 10100 in a 5−dimensional hypercube receive a broadcast label [11010]. The broadcast subcube of node 10100 is **1*0. Let the node 01011 receive a broadcast label [11010], the broadcast subcube of 01011 is **0*1. The broadcast subcube of the source of the broadcast message is the binary $n-$cube.

It is quite possible for an $n-$cube to be fully unsafe when it contains n or more than n faulty nodes [5]. Fault-tolerant communication inside a fully unsafe $n-$cube is impossible according to the safe node concept [5,10,21]. We are interested in reliable fault-tolerant broadcasting inside a fully unsafe hypercube.

Definition 2 *A node in an $n-$cube is locally unsafe inside a subcube if it has at least two faulty neighbors, or at least three locally unsafe or faulty neighbors inside*

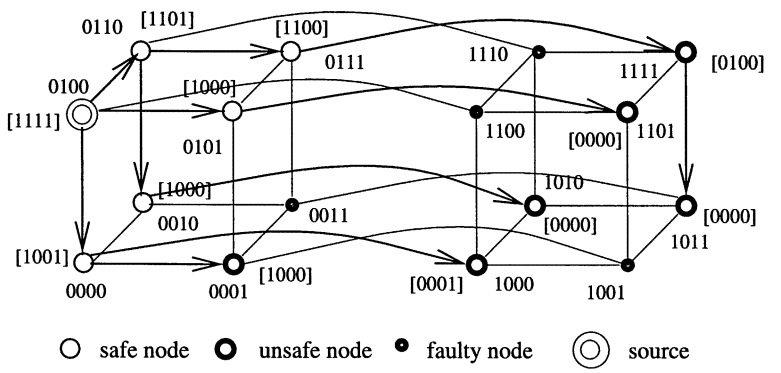

Figure 1: Binomial-tree-based broadcasting via limited-global safety.

the subcube; otherwise, it is locally safe in the subcube. The subcube is fully unsafe if it contains no locally safe node; otherwise, it is a safe subcube. Locally unsafe nodes inside a subcube SC is further classified: a locally unsafe node is locally ordinarily unsafe if it has at least one locally safe neighbor in SC; otherwise, it is a locally strongly unsafe node.

The following schemes like those in [5] are adopted to identify local safety information of a hypercube multicomputer system with both node and link faults:

- The end nodes of a faulty link inside the subcube are thought of as faulty nodes;
- the end nodes of a faulty link inside the subcube are considered as unsafe after safety information of the hypercube system has been determined.

A subcube can still be safe even though all nodes outside of it are faulty. Fig. 2 presents a fully unsafe 5−cube with 4 faulty nodes and 4 faulty links. The 5−cube is actually a fully unsafe cube, but node 00000 is locally safe in subcubes 0**** and ***0*, and it is locally ordinarily unsafe in *0*** and ****0. One node may have different local safety parameters in different subcubes. A definition for a maximal safe subcube is presented as follows.

Definition 3 *An $m-$dimensional subcube is defined as a maximal safe subcube if it is safe, and any $k-$dimensional ($k \geq m+1$) subcube contains it is fully unsafe.*

In the rest of this section, a couple of properties of faulty hypercubes based local safety are presented. Proofs of them are similar to those in [22-25].

Property 1 *If a node is locally unsafe in a $k-$dimensional subcube SC_1, it is still locally unsafe in an $m-$dimensional ($m > k$) subcube SC_2 if SC_2 contains SC_1.*

Property 2 *Suppose a node s is locally safe in an $m-$dimensional subcube SC_2, it is still locally safe in a $k-$dimensional subcube SC_1 ($k < m$) if SC_2 contains SC_1.*

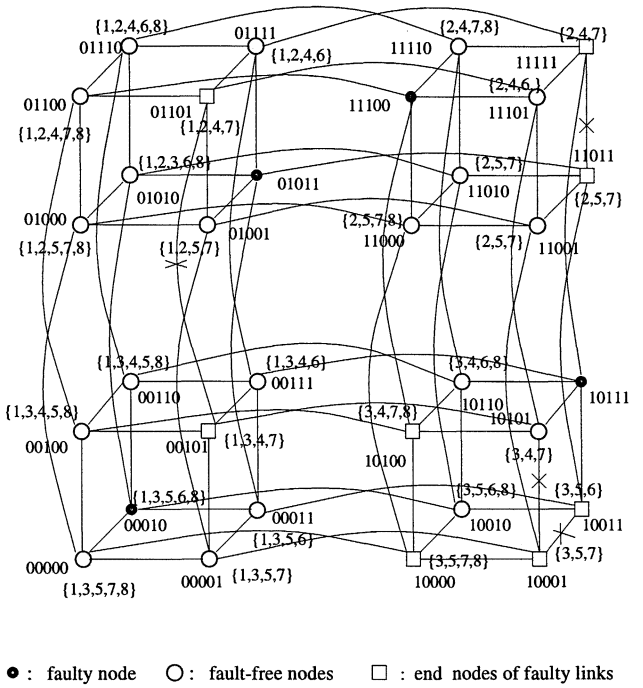

Figure 2: Local safety for fault-tolerant broadcasting.

Property 3 *Let x be locally strongly unsafe in a subcube SC, there exists at least one locally ordinarily unsafe neighbor of x in SC if the subcube SC is safe.*

The following property shows an important feature of a safe $n-$cube or subcube.

Property 4 *A $k-$dimensional ($k \leq n$) subcube is safe if and only if the $n-$cube contains a fault-free $k-2$ subcube, each node of which has at most one neighboring faulty node or end node of a faulty link.*

Property 5 *There always exists a minimum feasible path between two fault-free nodes x and y if the spanning subcube $SC(x,y)$ is safe even though the hypercube is fully unsafe.*

Property 6 *A minimum feasible path between the source s and destination d is available using local safety of the maximal safe subcube msc that contains the spanning subcube $SC(s,d)$ if one of s and d is locally safe in msc even though the hypercube is fully unsafe.*

There may still exist a feasible path of length no more than $H(s,d)+4$ even if the spanning subcube is fully unsafe, which can be illustrated as the following properties.

Property 7 *A feasible path of length no more than $H(s,d)+2$ between the source s and destination d is available using local safety of the maximal safe subcube msc containing the spanning subcube $SC(s,d)$ if one of s and d is locally ordinarily unsafe*

(none of s and d is locally safe) inside msc even though the hypercube and $SC(s,d)$ are fully unsafe.

Property 8 *A feasible path of length no more than $H(s,d)+4$ between the source s and destination d using local safety of the maximal safe subcube containing the spanning subcube $SC(s,d)$ if both of s and d are locally strongly unsafe in the maximal safe subcube even if the hypercube and $SC(s,d)$ are fully unsafe.*

3. Local Safety Information

We use the following scheme to obtain local safety for all nodes concurrently if the $n-$cube is fully unsafe. For each node v (binary representation is $v_n v_{n-1} \dots v_2 v_1$), check local safety of the node in $v_n * \dots **$, $*v_{n-1} * \dots **$, ..., $** \dots *v_2 *$, $*** \dots *v_1$ concurrently. If the node has at least two faulty neighbors (end nodes of a link fault inside the subcube should be thought of as faulty when calculating local safety information, which are considered as unsafe after local safety information has been obtained) or at least three locally unsafe or faulty neighbors inside a subcube, then the node is locally unsafe in the subcube. The node stores the local safety of its neighbors and itself if the subcube has been found to be safe. When we find at least two $(n-1)-$dimensional subcubes that contains the node are unsafe, local safety of all $2(n-1)$ $(n-2)-$dimensional subcubes that contains the node should be checked. The above process should be continued until local safety of all maximal safe subcubes with sizes greater than the given limit has been got.

Algorithm local-safety()
for each fault-free node v_1 in the $n-$cube,
parallel do
local-safety-information(v_1)
parallel end

Procedure local-safety-information(v)

1. Let the binary expression of v be $v_n v_{n-1} \dots v_2 v_1$, place subcubes $v_n * \dots **$, $*v_{n-1} \dots * *$, ..., and $** \dots *v_1$ into the current subcube set.
2. Concurrently check the state of v inside each subcube SC in the current subcube set by steps 3 and 4.
3. Set all fault-free nodes in SC as locally safe.
4. While the state v in SC is unstable, do
   let SC contain dimensions $i_1, i_2, \dots, i_k \in \{0, 1, \dots, n\}$ $(k \geq 3)$, set state of v in SC as locally unsafe if at least two of $v's$ neighbors along dimensions $i_1, i_2, \dots, i_k$ are faulty or at least three of the neighbors are locally unsafe or faulty.
5. If v is locally unsafe in SC, and v has at least one locally safe neighbor in SC, set state of v in SC as locally ordinarily unsafe; otherwise, set state of v in SC as locally strongly unsafe.
6. For every unsafe subcube SC_1 containing v with size no less than a given limit,

(a) get the $(k-1)-$dimensional subcubes SC_2 of SC_1;

(b) if the size of SC_2 is greater than the given limit, put SC_2 into the current subcube set.

7. If there exists at least one subcube in the current subcube set, go to step 2.

In step 4, "unstable" indicates the states of the node in the last two consecutive steps are different. The overall communication steps to obtain local safety is $O(p{\cdot}m)$, where p is usually only 2 or 3, or a little greater in most cases, and m is the maximal number of steps to get local safety each round which is similar to that to derive the global safety information like [5,21]. The computing complexity for each node between two communication steps should be $O(n \cdot K)$, where n is the number of dimensions of the hypercube and K is the maximum number of subcubes checked in one step. This is only the worst case. The actual computing steps for each fault-free node between two communication steps while calculating local safety information should be less in most cases. The above algorithm also works well for nodes in a disconnected cube. Broadcast according to local safety can handle cases when the system contains more faults because other metrics consider safety corresponding to the whole system or the $k-$distance neighborhood. Local safety in this paper considers safety inside the broadcast subcube.

The amount of local safety information that each node needs to keep is not so much. Consider a fault-free node v in a $10-$cube. Let v keep local safety information of all maximal safe subcubes of size no less than 6 that contain v. There are at most 210 $6-$dimensional subcubes that contain the node v. Usually, the number of maximal safe subcubes with sizes less than 6 that contain v is much less than 210. Local safety information of a fault-free node and its fault-free neighbors should be kept by each fault-free node. The node v needs to keep local safety information of at most 2310 subcubes, which is acceptable. A node in a current multicomputer can have several Mbytes of local memory. The main cost to calculate local safety information is the time to communicate local safety information to its fault-free neighbors each step, which is mainly determined by the set-up time. The time for a node to calculate local safety information each step should be trivial because it is easy for a current multicomputer to have hundreds of MHZ processor inside each node. Therefore, we can say that the cost to get local safety information should be 2 or 3 times (or a little more) of that to calculate the global safety information [5,21].

It is clear that there exists a unique set of maximal safe subcubes for a faulty hypercube. It is unnecessary to find all maximal safe subcubes. We present a threshold in the above algorithm to limit the size of the maximal safe subcubes utilized. Usually, 2 or 3 rounds is enough. The faulty hypercube as shown in Fig. 2 contains 4 faulty nodes and 4 faulty links. There are 8 $4-$dimensional maximal safe subcubes in the fully unsafe $5-$cube as shown in Fig. 2. The 8 maximal safe subcubes are 0****, *1***, *0***, **1**, **0**, ***1*, ***0*, and ****0, which are labelled as maximal safe subcubes from 1 to 8, respectively. Fig. 2 also indicates the

maximal safe subcubes that contain the fault-free nodes. For example, node 00000 has a label $\{1, 3, 5, 7, 8\}$, which represents the maximal safe subcubes 0****, *0***, **0**, ***0*, and ****0 contain node 00000. Each node s keeps local safety of the node and its neighbors, which can be stored in $(n+1)$ lists as follows. The last list records local safety information of the node, and the first n lists record local safet y of its neighbors. Each element $a(b)$ of the $i-$th list indicates local safety of $s^{(i)}$ inside the $b-$th maximal safe subcube is a, where a can be assigned 5, 3, 2, and 0 which indicates locally safe, locally ordinarily unsafe, and locally strongly unsafe, and faulty, respectively. Let s be node 00001. It needs to keep the following lists:

$s^{(1)}(00000) : 5(1), 3(3), 5(5), 5(7), 3(8)$
$s^{(2)}(00011) : 3(1), 3(3), 3(5), 3(6)$
$s^{(3)}(00101) : 3(1), 3(3), 3(4), 3(7)$
$s^{(4)}(01001) : 3(1), 5(2), 5(5), 5(7)$
$s^{(5)}(10001) : 1(3), 3(5), 5(7)$
$s : 5(1), 3(3), 5(5), 5(7)$

4. Fault-Tolerant Broadcasting via Local Safety Information

Assume each fault-free node keeps the local safety of itself and its fault-free neighbors. We show optimal broadcasting is still possible in many cases even though the hypercube is fully unsafe. The sufficient condition for the existence of an optimal broadcasting, a fault-tolerant broadcast algorithm are presented for a fully unsafe hypercube. A case study is also presented according to the algorithm.

4.1. *Broadcast Algorithm*

A message can always be passed optimally from a node s if s is locally safe in its broadcast subcube, which implies that a message can be broadcast optimally even though the $n-$cube is fully unsafe [23]. We would like to construct a couple of broadcast subcubes starting from the source. Consider the source has at most one faulty neighbor in the $n-$cube. Let Q_{n-1}, Q_{n-2}, Q_{n-3}, ..., Q_{n-m} be broadcast subcubes of the fault-free neighbors $s^{(i_1)}$, $s^{(i_2)}$, ..., $s^{(i_m)}$ $(n-1 \leq m \leq n)$ of the source, where $i_1, i_2, \ldots, i_m \in \{1, 2, \ldots, n\}$ and the subscripts indicate the sizes of the corresponding broadcast subcubes. The following techniques are utilized to guide fault-tolerant broadcasting.

- Try to avoid sending the message and the broadcast label to fault-free neighbors which has at least two faulty neighbors inside the broadcast subcube or is connected with a faulty link in the broadcast subcube;

- consider the source has one connected faulty link or at least two faulty neighbors inside the broadcast subcube, send the broadcast label to the last fault-free neighbor along dimension i inside the broadcast subcube without resetting the $i-$th bit.

The node receives the unmodified label do not send the message back to its predecessor. Algorithm *broadcast()* broadcasts a message using local safety information. The algorithm sets all bits of the broadcast source's label to 1. When the $n-$cube is fully unsafe, *broadcast()* tries to find a fault-free neighbor $s^{(i)}$ of s along dimension i, whose broadcast subcube is contained in a maximal safe subcube. The message can be broadcast reliably if the broadcast subcube is contained in a maximal safe subcube. Assume $size(k)$ is the size of the maximal safe subcube k that contains the node v, in which the local safety of the node is $safety(v,k)$ (5 for locally safe, 3 for locally ordinarily unsafe, 2 for locally strongly unsafe, and 0 for faulty), the safety summation measure is calculated using the following expression,

$$safe(v) = size(k) \cdot safety(v,k)$$

Algorithm broadcast() /* Let s be the broadcast source */

1. If the node s is the broadcast source, for $i = 1$ to n, $label[i] \leftarrow 1$; do 2, 3, 4;
2. If the broadcast subcube of s is contained in a maximal safe subcube msc, then *broadcast* the message inside msc based on the local safety information of msc just like that in [21] and send the message and broadcast label without resetting the corresponding bit for fault-free neighbor of the source when the source has at least two faulty neighbors; otherwise $f_{br} \leftarrow 0$; if s has at least two faulty neighbors in its broadcast subcube, then 5; otherwise 3, 4.
3. While $f_{br} = 0$, do
   (a) $f_{br} \leftarrow 1$, for $i = 1$ to n
      i. if $label[i] = 1$ and the broadcast subcube of $s^{(i)}$ is contained in a maximal safe subcube, in which $s^{(i)}$ is locally safe, then (ii)
      ii. $label[i] \leftarrow 0$; send the message and $label$ to $s^{(i)}$; $f_{br} \leftarrow 0$.
   (b) for $i = 1$ to n
      i. if $label[i] = 1$ and the broadcast subcube of $s^{(i)}$ is contained in a maximal safe subcube, and $s^{(i)}$ has at most one faulty neighbor and no neighboring faulty link inside the broadcast subcube, then (ii),
      ii. $label[i] \leftarrow 0$, send the message and the $label$ to $s^{(i)}$; $f_{br} \leftarrow 0$.
   (c) for $i = 1$ to n
      i. if $label[i] = 1$ and the broadcast subcube of $s^{(i)}$ is contained in a maximal safe subcube, then (ii),
      ii. $label[i] \leftarrow 0$, send the message and $label$ to $s^{(i)}$; $f_{br} \leftarrow 0$.
4. If there still exists a fault-free neighbor of s, which has not received the message and broadcast label, for each $label[i] = 1$ $(1 \leq i \leq n)$

(a) if $s^{(i)}$ is fault-free and has the most safety summation measure, then (b);

(b) $label[i] \leftarrow 0$; send the message, *label* to $s^{(i)}$ via dimension i.

5. Do the same process as steps 3, 4, each time only check whether the node $s^{(i)}$ is the last unprocessed fault-free neighbor of s, if it is not, send the message and *label* by resetting $label[i]$; if $s^{(i)}$ is the last unprocessed fault-free neighbor of s, send the message and *label* to $s^{(i)}$ without resetting $label[i]$.

In step 2, there exists at least one fault-free node cannot get the broadcast message using the previous label sending scheme [9,10,19,21]. The technique to send the broadcast label without resetting the corresponding bit can make the unreachable nodes reachable. A flag f_{br} is adopted to guide whether the broadcast should be continued or not, which is set as 0 initially. Let us show how to generate the broadcast subcube of $s^{(i)}$ in step 3. For example, the node 10101(v) has a broadcast label [11011], the broadcast subcube of 00101 ($v^{(5)}$) should be 0*1**, while the broadcast subcube of 10100 ($v^{(1)}$) is **1*0.

The algorithm *broadcast()* can optimally broadcast the message of a node s inside its broadcast subcube BSC if s is locally safe in a maximal safe subcube which contains BSC. The sufficient condition for optimal broadcasting inside a fully unsafe $n-$cube can be stated as follows: the algorithm *broadcast()* can optimally broadcast the message of the source s in n steps if s has at most one faulty neighbor and no neighboring faulty link and a sequence of fault-free neighbors $s^{(i_1)}$, $s^{(i_2)}$, ..., $s^{(i_m)}$ $(n-1 \leq m \leq n)$ are locally safe in a sequence of maximal safe subcubes which contain the broadcast subcubes of the nodes $s^{(i_1)}$, $s^{(i_2)}$, ..., $s^{(i_m)}$, respectively, where $i_1, i_2, \ldots, i_m \in \{1, 2, \ldots, n\}$. It should be noted that a time-optimal broadcasting is always impossible if the source is an end node of a faulty link.

4.2. *A Case Study*

Let node 10000 in the same faulty 5−cube as shown in Fig. 2 be the broadcast source. Node 00000 is locally safe in the maximal safe subcube 0****, therefore, the broadcast message with a broadcast label [01111] is forwarded to 00000. An optimal broadcast is available inside 0**** at node 00000. The broadcast message with a label [01011] is sent to 10100 (it can also be passed to 11000 with a broadcast label [00011] at this point) because 10100 is locally safe in the maximal safe subcube **1**. The broadcast message at node 10100 can be broadcast optimally in 1*1** using local safety information of the maximal safe subcube **1**. The broadcast message is then sent to 11000 with a broadcast label [00011] because 11000 is locally safe inside the maximal safe subcube **0** that contains the broadcast subcube 110**. Therefore, the broadcast message of 11000 can be broadcast optimally inside the broadcast subcube 110**. Finally, the broadcast message can be broadcast optimally inside 100** because 10000 is locally safe in *0***. Fig. 3 presents the time-optimal broadcast of the broadcast message and broadcast labels of the fault-free nodes.

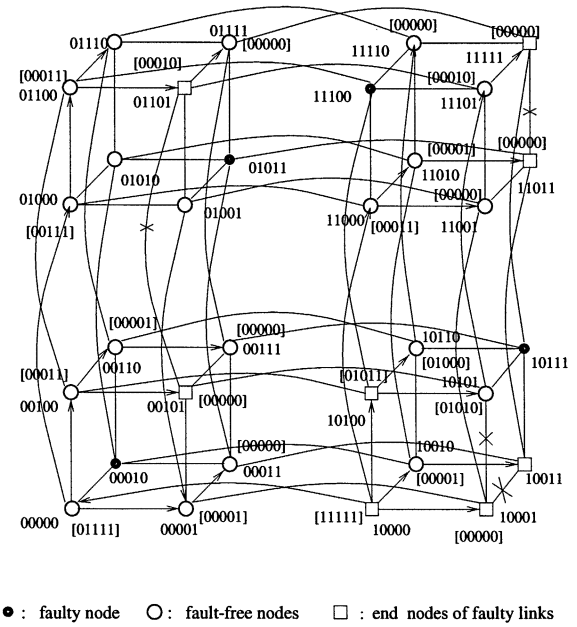

Figure 3: Optimal fault-tolerant broadcasting with local safety.

5. Extensions to Fault-Tolerant Routing and Multicasting

The extensions to fault-tolerant routing and multicasting when the system contains both node and link faults are introduced in this section. The faulty 4−cube given in Fig. 4 contains 4 faulty nodes 0011, 1100, 1110, 1001, and two link failures 000− and 01 − 0 (000− indicates the link connecting nodes 0000 and 0001). End nodes of a faulty link are thought of as faulty when local safety information of the subcube that contain the faulty link is checked. Certainly, the 4−cube is fully unsafe. There exist the following maximal safe subcubes in the fully unsafe 4−cube given in Fig. 4: 1***, *1**, **1*, ***0, ***1, and 0*0*. A message should never be routed to a node whose shortest paths leading to the destination are blocked by faulty links when $H(s,d) = 2$, where s and d are the source and destination, respectively.

The message can be routed to a locally safe node v in the maximal safe subcube *msc* if the source s is locally safe in *msc* and $H(s,d) \geq 3$ (d is the destination). The message should be routed to a fault-free neighbor whose link leading to the destination d is not blocked by a link failure in a minimum path from the source s to d if $H(s,d) = 2$. In order to implement the above scheme, each node keeps fault information (including faulty node and link failure) of its fault-free neighbors.

If the source s is locally ordinarily unsafe in the maximal safe subcube *msc*, send the message to a locally safe neighbor v in a minimum feasible path from s to d if possible. Otherwise, send the message to a locally ordinarily unsafe neighbor of s in a minimum feasible path from s to d if possible. Otherwise, send the message to

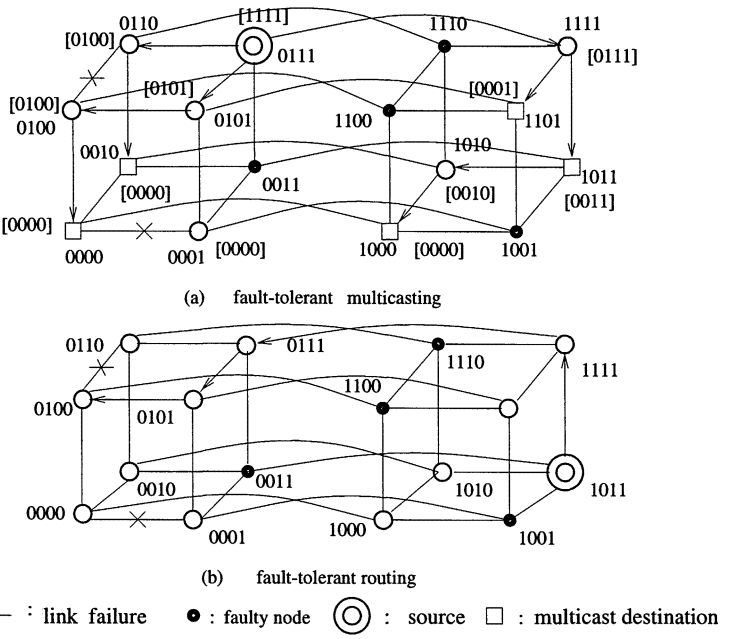

Figure 4: Local-safety-information-based fault-tolerant communication in hypercubes with node and link failures.

a locally safe neighbor in a non-minimum feasible path from s to d (which is always available) when the above two cases are not met.

If the source s is locally strongly unsafe in msc, send the message to a locally ordinarily unsafe neighbor in a minimum feasible path from s to d if possible. Otherwise, send the message to a locally strongly unsafe neighbor of s along a minimum feasible path from s to d if possible. Select a locally ordinarily unsafe neighbor of s in a non-minimum feasible path from s to d when both cases as stated are not met.

Consider a message is routed from 1011 to 0100 in the faulty 4−cube as presented in Fig. 4(b). The next node 1011 can be selected in a minimum path from 1011 to 0100, which is contained in a maximal safe subcube with the destination 0100. Let 1111 be selected as the next node, which is locally safe in the maximal safe subcube ***0 that contains both 0100 and 1111. Node 1111 routes the message to a locally safe neighbor 0111 in a minimum path from 1111 to 0100. Node 0111 cannot route the message to 0110 which is connected with the destination 0100 by a faulty link. Node 0010 should route the message to 0101 in this case. The selected feasible path from 1011 to 0100 is presented in Fig. 4(b).

Fault-tolerant multicasting is also similar when the system contains both node and link faults. Our method just handle fault-tolerant routing between each pair of source and destination as stated above while makes all destinations share links as many as possible in order to minimize traffic. The detailed fault-tolerant multicasting algorithm is similar to that in [24]. Consider a multicast problem with a

source 0111 and a destination set $\{0000, 0010, 1000, 1101, 1011\}$. As for the destination 0010, 0110 is the only next node from the source, which is connected with destination 0010 via a fault-free link. The best next node of the destination 0000 is 0101, which is locally safe in a maximal safe subcube 0*0* that contains both 0101 and 0000. Certainly, the next node of the destination 0000 can also be 0110, but 0110 is locally ordinarily unsafe in the maximal safe subcube ***0 that contains both 0110 and 0000. The next node for destinations 1000, 1010 and 1011 is 1111 because 1111 is locally safe in the maximal safe subcube 1***. The whole multicast tree of the multicast problem can be found from Fig. 4(a).

6. Conclusions

Local safety information was extended to handle fault-tolerant broadcasting in hypercube multicomputers with node and link faults. Some further resilience of the hypercube topology was utilized by the broadcast algorithm compared with the previous methods. An algorithm was presented to calculate local safety information of a faulty hypercube. We also found that the extra cost to calculate local safety information is comparable to that to get the global safety information. Local safety information was well utilized in the fault-tolerant broadcast algorithm by only considering safety of the broadcast subcube. The sufficient condition for optimal broadcast of a message in a fully unsafe hypercube was also presented. The proposed broadcast algorithm can still work well even though the system contains more faults than previous methods. Fault-tolerant routing and multicasting schemes based on node and link faults were also introduced.

Acknowledgments

This paper is supported in part by the national science foundation under grant 69773030, fundamental research grant of Tsinghua University, and in part by national science foundation of US under grant CCR9900646.

References

1. L. N. Bhuyan and D. P. Agrawal, "Generalized hypercubes and hyperbus structure for a computer network," *IEEE Trans. Computers*, vol. 33, no. 4, 1984.
2. J. Bruck, "On optimal broadcasting in faulty hypercubes," *Discrete Applied Mathematics*, vol. 53, pp. 3-13, 1994.
3. M. S. Chen and K. G. Shin, "Depth-first search approach for fault-tolerant routing in hypercube multicomputers," *IEEE Trans. Parallel and Distributed Systems*, vol. 1, no. 2, pp. 152-159, 1990.
4. G. M. Chiu and K. S. Chen, "Use of routing capability for fault-tolerant routing in hypercube multicomputers," *IEEE Trans. Computers*, vol. 46, no. 8, pp. 953-958, 1997.
5. G. M. Chiu and P. S. Wu, "A fault-tolerant routing strategy in hypercube systems," *IEEE Trans. Computers*, vol. 45, no. 2, pp. 143-155, 1996.
6. J. Duato and M. P. Malumbres, "Optimal topology for distributed shared-memory mul-

tiprocessors: hypercubes again ?" Proc. 2-th *Int. Euro-Par Conf.*, pp. 205-212, 1996.
7. P. Fraigniaud, "Asymptotically optimal broadcasting and gossiping in faulty hypercube multicomputers," *IEEE Trans. Computers*, vol. 41, no. 11, pp. 1410-1419, 1992.
8. S. L. Johnson and C. T. Ho, "Optimum broadcasting and personalized communication in hypercubes," *IEEE Trans. Computers*, vol. 38, no. 9, pp. 1249-1268, 1989.
9. H. P. Katseff, "Incomplete hypercubes," *IEEE Trans. Computers*, vol. 37, no. 5, pp. 604-608, 1988.
10. T. C. Lee and J. P. Hayes, "A fault-tolerant communication scheme for hypercube computers," *IEEE Trans. Computers*, vol. 41, no. 10, pp. 1242-1256, 1992.
11. J. Laudon and D. Lenoski, "The SGI Origin: A ccNUMA highly scalable server," Proc. of *Int. Symp. Comput. Architecture*, pp. 241-251, 1997.
12. S. Park and B. Bose, "All-to-all broadcasting in faulty hypercubes," *IEEE Trans. Computers*, vol. 46, no. 7, pp. 749-755, 1997.
13. M. Peercy and P. Banerjee, "Distributed algorithms for shortest path, deadlock-free routing and broadcasting in arbitrarily faulty hypercubes," Proc. of *IEEE Int. Symp. on Fault-Tolerant Computing*, pp. 218-225, 1990.
14. C. S. Raghavendra, P. J. Yang, and S. B. Tien, "Free dimensions-An effective approach to achieving fault tolerance in hypercubes," *IEEE Trans. Computers*, vol. 44, no. 9, pp. 1152-1157, 1995.
15. P. Ramanathan and K. G. Shin, "Reliable broadcast in hypercube multicomputers," *IEEE Trans. Computers*, vol. 37, no. 12, pp. 1654-1657, 1988.
16. Y. Saad and M. H. Schultz, "Topological properties of hypercubes," *IEEE Trans. Computers*, vol. 37, no. 7, pp. 867-872, 1988.
17. C. L. Seitz, "The cosmic cube," *Communications of the ACM*, vol. 28, no. 1, pp. 22-33, 1985.
18. H. Sullivan, T. R. Bashkow, and D. Klappholz, "A large scale, homegeneous, fully distributed parallel machine," Proc. of 4-th *Annual Symp. on Computer Architecture*, pp. 105-124, 1977.
19. J. Wu, "Safety levels–An efficient mechanism for achieving reliable broadcasting in hypercubes," *IEEE Trans. Computers*, vol. 44, no. 5, pp. 702-706, 1995.
20. J. Wu, "Adaptive fault-tolerant routing in cube-based multicomputers using safety vectors," *IEEE Trans. Parallel and Distributed Systems*, vol. 9, no. 4, pp. 321-334, 1998.
21. J. Wu and E. B. Fernandez, "Reliable broadcasting in faulty hypercube computers," *Microprocessing and Microprogramming*, vol. 46, pp. 241-247, 1993.
22. D. Xiang and J. Wu, "Reliable unicasting in faulty hypercube multicomputers using local safety information," Proc. of 4-th *IEEE Int. Conf. on Algorithms and Architectures for Parallel Processing*, pp. 617-628, Dec., 2000.
23. D. Xiang and J. Wu, "Fault-tolerant broadcasting in hypercube multicomputers using local safety information," Technical Report, TR-CSE-99-36, Florida Atlantic University, 1999.
24. D. Xiang and J. Wu, "Reliable multicasting in hypercube multicomputers using local safety information," Proc. of 13-th *Int. Conf. on Parallel and Distributed Computing Systems*, pp. 529-534, Aug., 2000.
25. D. Xiang, "Fault-tolerant routing in hypercube multicomputers using local safety information," Accepted to appear in *IEEE Trans. Parallel and Distributed Sys.*, 2001.
26. X. D. Zhang, "System effects of interprocessor communication latency in multicomputers," *IEEE Micro*, vol. 11, no. 2, pp. 12-15 and 52-55, 1991.