Fault-Tolerant Routing Algorithm in Meshes with Solid Faults

Jong-Hoon Youn
Dept. of Computer Science
Oregon State University
Corvallis, OR 97330, U.S.A.
jhyun@cs.orst.edu

Bella Bose
Dept. of Computer Science
Oregon State University
Corvallis, OR 97330, U.S.A.
bose@cs.orst.edu

Seungjin Park
Dept. of Computer Science
Michigan Tech. University
Houghton, MI 49931, U.S.A.
spark@mtu.edu

Abstract

A fault-tolerant routing method that can tolerate solid faults using only two virtual channels is presented. The proposed routing algorithm not only uses a fewer number of virtual channels but also tolerates f-chains in the meshes. It is shown that the proposed algorithm is deadlock-free and livelock-free in meshes when it has nonoverlapping multiple f-regions.

Keywords: *fault-tolerant, solid faults, mesh networks, wormhole routing*

1 Introduction

Many recent multicomputers and multiprocessors are implemented with mesh topology. These computers usually use the e-cube routing algorithm with wormhole switching because of its simplicity. Deterministic routing algorithms, such as e-cube routing algorithm, establish the path as a function of the destination address and always supply the same path between every pair of nodes. When wormhole switching was invented, system designers chose the simple deterministic algorithms in order to keep routing hardware as compact and as fast as possible [5].

Since a deterministic routing algorithm always provides a fixed path for the same source and destination pair, it cannot tolerate even single node or link failure. Thus, fault tolerance is a dominant issue facing the design of interconnection networks for large-scale multiprocessor architectures. An extensive amount of work has been done on fault-tolerant routing in wormhole-switched networks [1, 2, 3, 4, 6, 7, 8, 9, 10, 11, 12]. The most common approach is to route messages around the faulty regions and so the complexity of the routing algorithm depends on the

shape of the faulty regions. One method has been proposed by Boppana and Chalasani in [1], where by making some appropriate fault-free nodes to faulty, rectangular regions are constructed.

An *f-ring* is a sequence of links and nodes that surround a faulty region and an *f-chain* is similar to f-ring but it touches one or more boundaries of the network. So, in an f-chain, there are two nodes marked as *end nodes* at which the f-chain touches the network boundaries. An f-ring or an f-chain is called as an *f-region*.

The algorithm proposed in [1] is a modification of the e-cube algorithm and uses two virtual channels per physical channel. The algorithm provides deadlock-free routing in mesh networks provided the f-rings are nonoverlapping. For the more complex faults with overlapping f-rings and f-chains, four virtual channels are used. Later Sui and Wang [10] have proposed an improved algorithm that tolerates overlapping f-rings and f-chains using only three virtual channels. By relaxing the restrictions on the shape of the faulty regions, in [3] the authors have devised a new routing algorithm that tolerates more general forms of f-rings, called *non-convex faults* using four virtual channels. This fault model is referred to as a *solid fault* model. Although they have devised a fault-tolerant routing algorithm for more relaxed faulty regions, the proposed algorithm does not consider f-chains or overlapping f-rings.

In this paper, we present a fault-tolerant routing method that can tolerate solid faults, shaped such as +, T, L and ⊥ using only two virtual channels. Since the e-cube routing with wormhole switching has been widely used in current multiprocessors and multicomputers, the algorithm presented in this paper is also based on the e-cube routing with wormhole switching. The proposed algorithm not only require a fewer number of virtual channels but also tolerates f-chains in the meshes. We assume that each node has only local knowledge of the faults. So, there is no need to maintain global fault information and also there is no restriction on the number of faults.

The rest of this paper is organized as follows. Section2 gives preliminaries that underlie this work. Section 3 describes the algorithms for acquiring position information on f-rings and f-chains. Section 4 presents the proposed fault-tolerant routing method and the proof of deadlock and livelock freedom. Finally in section 5, some concluding remarks are made.

2 Preliminaries

In this section, we briefly introduce the topology of *n*-dimensional mesh network and the well-known e-cube routing algorithm. Formally, an *n*-dimensional mesh has $k_0 \times k_1 \times k_2 \times \ldots \times k_{n-2} \times k_{n-1}$ nodes, k_i nodes along the dimension i, where $k_i \geq 2$ and $0 \leq i \leq n\text{-}1$. Each node X in an *n*-dimensional mesh is represented by $(x_{n-1}, x_{n-2}, \ldots, x_2, x_1, x_0)$, where $0 \leq x_i \leq k_i - 1$ and $0 \leq i \leq n\text{-}1$.

Two nodes X and Y are adjacent if and only if for some j, $0 \le i \le n\text{-}1$, $x_j = y_j \pm 1$ and $x_i = y_j$ all for $i \ne j$, $0 \le i \le n\text{-}1$. It is denoted by $X \leftrightarrow Y$.

In the e-cube algorithm, the routing is done in dimensional order. Suppose $X = (x_{n-1}, x_{n-2}, \dots, x_2, x_1, x_0)$ is the source node and $Y = (y_{n-1}, y_{n-2}, \dots, y_2, y_1, y_0)$ is the destination node. The message travels along the i-th dimension till $x_i = y_i$, in the order i=0, 1, 2, … , n-1. It can be easily proved that this algorithm is deadlock free.

When there is a faulty node/link in the mesh network, a message can be blocked by the faulty node/link; further, this message can hold the channels which it has already acquired. This may cause the blockage of additional messages and as a result of this, there is a chance of occurrence of deadlock.

2.1 Fault model

The failure of a processing element and its associated routers is referred to as a node failure and the failure of any communication channel is referred to as a link failure. In our fault model, both node failures and link failures are considered. We assume that each node can test the fault status of the links incident to it and also it neighboring nodes. Thus, each node knows the fault status of its links and the neighboring nodes. Since the nodes do not know the global faulty information, messages routed through a portion of a faulty region may not use the shortest path.

Each node on an f-region is classified according to the number of faulty links incident to it and its functionality. There are four types: *convex node* which has no faulty link incident to it, *concave node* which has exactly two faulty links incident to it, *plain node* which has only one faulty link incident to it and *relay-only node* which performs only the routing operations.

A portion of an f-ring that consists of two convex nodes with only plain nodes, if any, between them is called a *convex section*. In solid fault model, there are exactly four convex sections. If a convex section of an f-ring is projecting towards the North (respectively South) side of the f-ring, it is called a *north (*respectively *south) convex section.*

The proposed method handles solid faults in meshes, shaped such as +, T, L and ⊥. More precisely, let l_1 and l_2 be any two links in the same row (respectively, column) of a given fault-region. In our fault model, no fault-free node exists between two faulty links, l_1 and l_2, in the same row (respectively, column).

Since the proposed routing method does not tolerate overlapping faulty regions, first we describe how the overlapping f-regions are merged and reformed during the formation of f-regions. The detection of overlapping f-regions is not difficult. Note that if there is no overlapping f-rings then all nodes in the f-ring will have exactly two links which are from the f-ring incident to it.

However, if there is an overlapping region then exactly two nodes will have three links, one each from the f-rings and the third common to these two f-rings, incident to them. So the overlapping f-ring can be easily detected. A similar method can be used to detect overlapping faulty regions in f-chains.

By sending shape-finding message in each f-regions as described in [3], all non-overlapping nodes get exactly one message, whereas the overlapping nodes get more than one messages. By making these overlapping nodes except the two extreme nodes in this portion faulty, two f-regions can be merged. After merging, the new f-region might contain a concave region and this needs to be changed to a convex region by making some appropriate nodes faulty. This can be done by sending another shape-finding message and making some non-faulty nodes to faulty.

A similar method can be used for detecting/merging overlapping regions in two f-chains or between an f-chain and an f-ring. In the case of f-chain, a shape finding message must be initiated from one of the end nodes.

3 The position on f-regions

In the proposed algorithm, each node has to know its *position* on the f-region, i.e., in which side it resides: North, South, East or West. In this section we describe an algorithm for this position information. The algorithm are described in terms of a 2-dimemsional mesh, but it is easily extended to higher dimensions.

Step 1: Every convex node in an f-region sends its position information, {North, South, East, West}, in both clockwise and counter-clockwise orientations. This message is propagated until it reaches a concave node, a convex node, or an end node. If a plain node or a concave node receives the position information message, then this node sets its position to the value in the received message. If a plain node on the f-region receives both ‘E’ and ‘W’, then it initially sets its position information to E; only the nodes contained in the north or south convex section receive both ‘E’ and ‘W’. If a plain node in the f-region receives both ‘N’ and ‘S’, then it sets the position information to N; only the nodes contained in the west or east convex section receive both ‘N’ and ‘S’ messages. So, the north, south, east and west convex sections are identified in this step. If a concave node receives ‘N’ (or ‘S’) and ‘E’ (or ‘W’) messages, then it sets its position information appropriately, as NW, SW, NE or SE.

If there is no end node in the f-region, then Step 2a is performed. Otherwise Step 2b is performed.

Step 2a (f-rings): The leftmost node in the north (respectively south) convex section injects a *position-update message* into the f-ring in counter-clockwise (respectively clockwise) direction. This message is propagated until it reaches the leftmost node in the south (respectively north) convex section. Similarly, the rightmost nodes in the north and south convex sections do the same thing. A position-update message contains the x coordinate of the starting node and p where $p \in \{W, E\}$. If a sender is a leftmost convex node, then p is 'W'. Otherwise p is 'E'.

When the leftmost node in the convex section receives the position-update message, it compares its x coordinate, x_d, to x_S. If $x_S > x_d$, then the node propagates the received message to its east neighbor and the neighbor performs the steps shown in Fig. 1. Similarly, when the rightmost node in the convex section receives the position-update message, it compares its x coordinate, x_d, to x_S. If $x_S < x_d$, then the node propagates the received message to its west neighbor and the neighbor performs the steps shown in Fig 1.

Let x_S be the x coordinate value contained in the message, x_C be the x coordinate value of the current node and p be the position information in the received message.

STEP:

```
IF (x_S > x_C AND p = W)  OR (x_S < x_C AND p = E)
THEN {
    propagate the message to its neighbor in the f-ring along the same direction
                    in which the message was received;
    IF the message traveled in the direction opposite to p,
    THEN {
        IF the current node is located in the north or south convex section
        THEN set the current node's position information to p.
        ELSE make the status of this node 'relay-only node'.
        make the neighbor node the current node.
        go to STEP
    }
}
```

Figure 1. Updating the position information of nodes in the f-ring

Step 2b (f-chains): The end node injects a *shape-change-request message* and it is propagated along the f-chains until it reaches a convex node. A shape-change-request message contains the location information of the end node. Let $N = (x_N, y_N)$ be an end node in the f-chain.

- If N is located in the east boundary, the shape-change-request message contains (y_N, W).
- If N is located in the west boundary, the shape-change-request message contains (y_N, E).
- If N is located in the north boundary, the shape-change-request message contains (x_N, S).
- If N is located in the south boundary, the shape-change-request message contains (x_N, N).

When a node receives a shape-change-request message, it performs the steps shown in Fig. 2.

```
Let M = (c, p) be the received message, where c is the x or y coordinate of the end node and
p ∈ {W, E, S, N} is the position information in the received message.

STEP:
    IF the current node is not a convex node in a convex section
    THEN {
        propagate the message to its neighbor in the f-chain along the same direction
        (i.e. clockwise or counter-clockwise) in which the message was received
        IF the message traveled in a direction different than the p,
        THEN {
            the node marks itself as faulty.
            initiate a node-disable message along the direction opposite to p.
            (This message is propagated until it reaches the network boundary and
             the nodes that receive a node-disable message also mark themselves as faulty.)
        }
        make the neighbor node the current node.
        go to STEP
    }
    ELSE    /* the current node is a convex node in the convex section */
        IF c ≠ the corresponding coordinate of the current node
        THEN {
            change its position information to p.
            initiate a position-change message along the directions of p and opposite to p.
            (This message is propagated until it reaches a convex node or a node located
             in the network boundary. The node that receives a position-change message
             changes its position to p)
        }
```

Figure 2. Updating the position information of nodes in the f-chains

Example 1: In Fig. 3, the dark nodes are faulty. Some convex nodes are (0,5), (3,7), (7,1) and (8,3), some concave nodes are (1,5), (3,6), (6,5) and (7,3) and some plain nodes are (0,4), (1,3), (2,3) (4,6) and (8,4). {(5,1), (6,1), (7,1), (5,1)↔(6,1), (6,1)↔(7,1)} is the south convex section. {(1,7), (2,7), (3,7), (1,7)↔(2,7), (2,7)↔(3,7)} is the north convex section.

Every convex node injects two position messages into the f-ring in Step 1 as shown in Fig 3. (a). For example, (7,1) sends 'S' to its northern neighbor and 'E' to its western neighbor. After Step 1, every node contains the position information as shown in Fig 3. (a). Furthermore, (1,7), (2,7) and (3,7) know that they are located in the north convex section and (5,1), (6,1) and (7,1) know that they are located in the south convex section. According to the rule given in Step 1, (2,7) and (6,1) set their position into 'E' and (0,4) and (8,4) set their position into 'N'.

In Step 2, the convex nodes in both north and south convex sections inject position-update messages with their *x* coordinate value and *p* where $p \in$ {W, E}; (1,7) sends the message <1, W>, (3,7) sends the message <3, E>, (5,1) sends the message <5, W> and (7,1) sends the message <7,

E>. After Step 2, (2,7) changes its position into 'W' because it propagates the message to its west neighbor. But (3,7) does not change the position information because it relays the message to the south neighbor. Furthermore, (3,6), (4,6), (4,3) and (5,3) become relay-only nodes; which implies that these nodes perform only the routing operations.

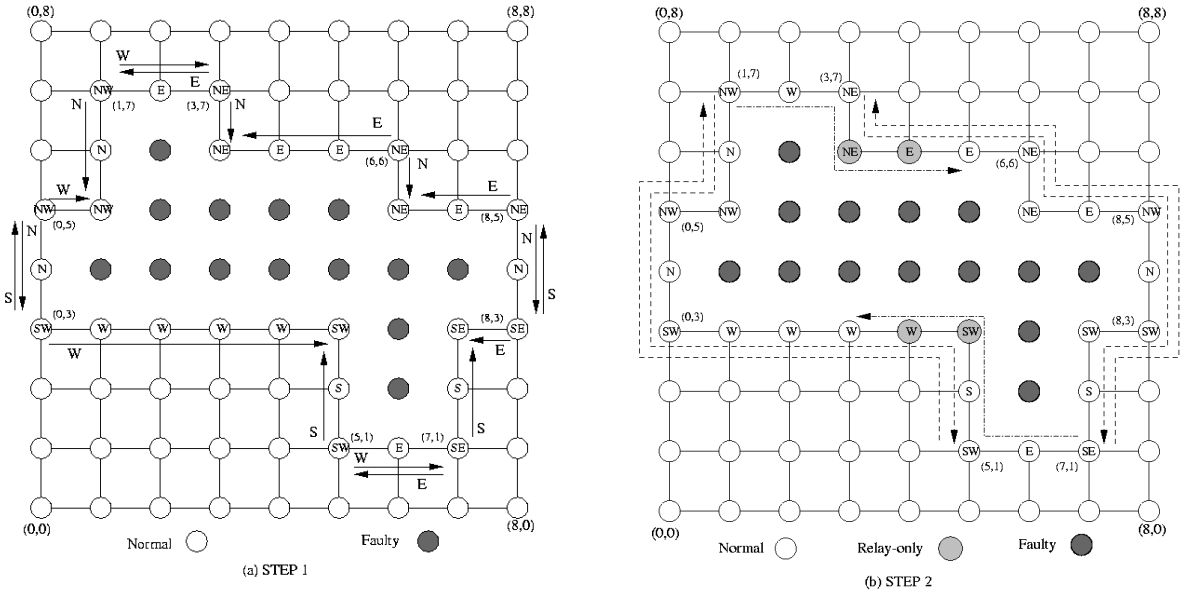

Figure 3. Example of position information in the f-ring

Example 2: An example of Step 2b is shown in Fig. 4. There are two end nodes in Fig. 4 (a), (8,1) and (8,5). Since they are located in the east boundary of the 2D mesh, the shape-change-request message contains <y coordinate, *W*>. The shape-change-request message injected from (8,1) (i.e. the message <1, *W*>) is propagated until it reaches (6,1). When (7,1) receives the message, it propagates the message along the f-chain in the same orientation. Because it propagates the message to its west neighbor, (6,1), it does not inject a node-disable message. When the message arrives at a convex node, (6,1), the receiver compares its *y* coordinate to the *y* coordinate in the message. Since they are same, (6,1) does not do anything.

Now, let us consider the shape-change-request message injected from (8,5) (i.e. the message <5, *W*>). This message travels along the f-chain in the counter-clockwise orientation from (8,5) to (6,7). First, the message is sent to (7,5). (7,5) sends it to (7,6), injects a node-disable message to its east neighbor, (8,5), and marks itself as faulty. (8,5) becomes a faulty when it receives the node-disable message. Although (7,6) is a convex node, it is not located in the convex section. Thus, when it receives the shape-change-request message, it just sends the message to the west

neighbor, (6,6). (6,6) relays the shape-change-request message to (6,7), injects a node-disable message to its east neighbor, (7,6), and marks itself as faulty. (7,6) and (8,6) become faulty when they receive the node-disable message. Since (6,7) is the convex node in the closest convex section, it does not propagate the message any more. Instead, it changes its position information to 'West' and propagates the new position, 'West' to its west and east neighbors. The new position is propagated until it reaches the convex node, (5,7) and the node (8,7), which is located in the network boundary.

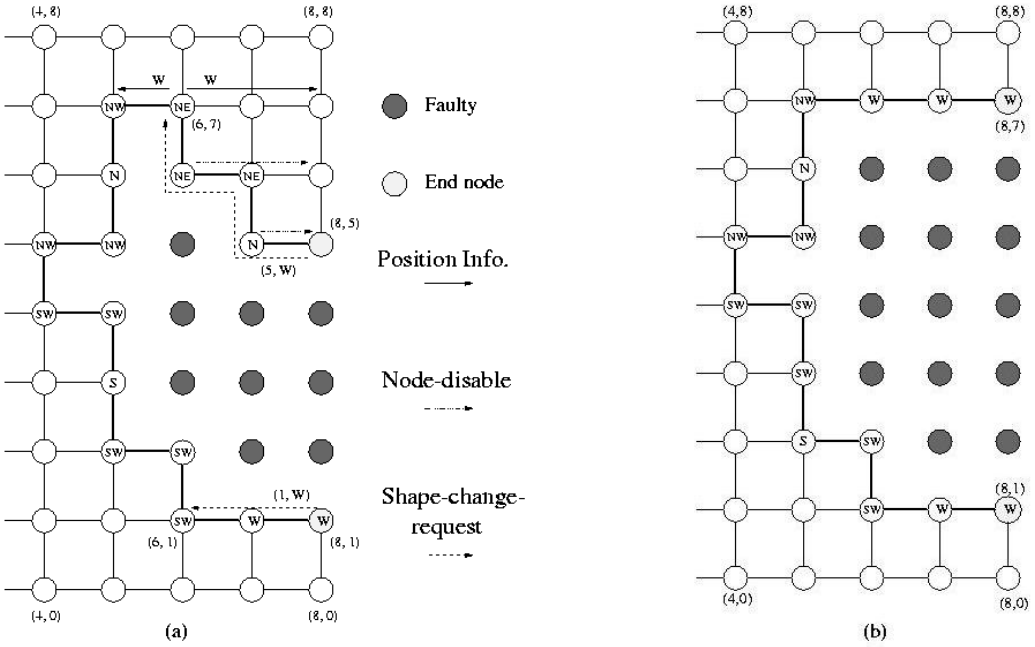

Figure 4. Example of position information in the f-chain

4 Routing algorithms

Let *M* denote a message to be routed from the current node, (x_c, y_c), to the destination node (x_d, y_d). Our algorithms are based on the e-cube routing, in which all messages are routed in two phases: in the first phase the message is routed along d_0 dimension (row dimension) until $(x_c = x_d)$ and in the second phase it is routed along d_1 dimension (column dimension) until it reaches the destination. M is said to be *row message* if it is in its first phase and *column message*, otherwise. Further, row messages traveling from West to East (respectively, East to West) are *WE* (respectively, *EW*) messages. Likewise, column messages traveling from North to South

(respectively, South to North) are *NS* (respectively, *SN*) messages. The proposed fault-tolerant routing algorithm requires only two virtual channels, C_0 and C_1, and tolerates multiple nonoverlapping f-regions.

4.1 A Fault-Tolerant Routing Algorithm (FT-Ecube)

Let (x_c,y_c) be the current host node of the given message M. If $(x_c, y_c) = (x_d, y_d)$, then M has reached its destination and so it is consumed. At the source node, M is labeled as WE or EW message depending on the *x* dimension values of the source and destination addresses; M is labeled as WE if $x_c < x_d$ and as EW otherwise. Once it reaches a node where $(x_c = x_d)$, M is changed as a column message (i.e. NS or SN message) and then it continues to travel towards its destination. M travels in the network based on the e-cube algorithm until it reaches an f-region. Once it reaches the f-region, two things can happen:

1. If its e-cube hop is on the f-region, then it continues to travel in the f-region, or
2. If its e-cube hop is blocked, then it is misrouted.

In both cases, the message traveling on an f-region can use the following virtual channels depending on the message type: Row (i.e. WE and EW) messages use C_0 and column (i.e. NS and SN) messages use both C_0 and C_1. Column messages usually use C_1 virtual channels. But, if a NS (respectively SN) message is misrouted from south to north (respectively from north to south) along the f-rings, it uses C_0 virtual channels until it leaves the f-ring.

If the next e-cube hop for M is not blocked by a fault, then M is routed using the e-cube algorithm. Otherwise it is misrouted according to the rules given in Table 1. Once its direction (i.e. clockwise or counter-clockwise) on the current f-region is set, it will never be changed until *M* leaves the f-region or it changes its message type. If a message takes an e-cube hop on a link that is not on an f-region, it can use any virtual channel without causing deadlock.

```
Procedure FT-Ecube (M)
/*      The address of the current node of M is (xc, yc) and the destination is (xd, yd).
        When a message is generated,
            it is set to normal and labeled as WE if xc < xd and as EW otherwise.
        IF the message type is EW, or WE use C0 virtual channel
        ELSE IF the message type is NS or SN, use C1 virtual channel          */

1 IF (xc = xd) and (yc = yd), THEN M is consumed by the current node and return.
2 IF M is a row message and (xc = xd),
        THEN set M's status to normal and change M's type to SN if (yd > yc) or NS if (yc > yd).
3 IF M's status is normal {
   IF the next e-cube hop is blocked by a fault,
        THEN set M's status misrouted and route M according to Table 1.
          ELSE M is routed using the e-cube hop.      /* If the next e-cube hop is available */
  }
```

```
4 IF M's status is misrouted {
        IF the next e-cube hop is available {
            IF (M is a row message) OR (M is a column message AND x_c = x_d)
                THEN set M's status to normal and M is routed using the e-cube hop.
        }
        ELSE M is routed along the orientation that is being used by the message.
  }
```

Figure 5. Fault-Tolerant e-cube routing algorithm

Node Position	Message Type	Used Virtual Channels	Misrouted Direction
N	WE	C_0	Misroute M along the *clockwise orientation* until the e-cube hop exists
	EW	C_0	Misroute M along the *counter-clockwise orientation* until the e-cube hop exists
S	WE	C_0	Misroute M along the *counter-clockwise orientation* until the e-cube hop exists
	EW	C_0	Misroute M along the *clockwise orientation* until the e-cube hop exists
W	NS	C_0, C_1	Misroute M along the *counter-clockwise orientation* until the e-cube hop exists and $x_c = x_d$
	SN	C_0, C_1	Misroute M along the *clockwise orientation* Until the e-cube hop exists and $x_c = x_d$
E	NS	C_0, C_1	Misroute M along the *clockwise orientation* Until the e-cube hop exists and $x_c = x_d$
	SN	C_0, C_1	Misroute M along the *counter-clockwise orientation* until the e-cube hop exists and $x_c = x_d$

Table 1. Direction to be set for the misrouted messages on f-regions.

Example 3: Let us consider the message M_1 from (8,1) to (1,7) in Fig. 6. M_1 is routed as a normal EW message from (8,1) to (7,1). At (7,1) its next e-cube hop is blocked and its direction is set to counter-clockwise, since the position of (7,1) is north. M_1 is misrouted from (7,1) to (7,2). At (7,2) M_1's next e-cube hop is available, it becomes a normal EW message and is routed from (7,2) to (3,2). Note that a C_0 channel is used from (7,2) to (6,2) and both C_0 and C_1 channels can be used from (6,2) to (3,2). At (3,2) its next e-cube hop is blocked and so it is misrouted in the clockwise orientation to (1,1). At (1,1), M_1 is set to a normal SN message since its x dimension offset is equal to zero. Then, it is routed by e-cube from (1,1) to (1,5). Since M_1 is blocked by a fault in (1,5), it is misrouted in the counter-clockwise orientation using C_1 virtual channels from (1,5) to (2,7).

At (2,7), the SN message is misrouted from north to south. Thus, M_1 uses C_0 virtual channels until it leaves the f-ring. It is misrouted from (2,7) to (1,6) in the same (i.e. counter-clockwise) orientation using C_0. At (1,6), M_1 becomes a normal SN message again because its next e-cube

hop is available and also because both the current node and the destination node are in the same column. M_1 is routed from (1,6) to (1,7) using the e-cube algorithm.

We now consider the message, M_2 from (3,5) to (6,2) in Fig. 6. M_2 is routed as a normal WE message from (3,5) to (5,5). At (5,5), its next e-cube hop is faulty and its direction is set to clockwise, since the position of (5,5) is north. M_2 is misrouted from (5,5) to (6,6) using C_0 channels. When M_2 arrives (6,6), its x dimensional offset is zero and it becomes a normal NS message. But the next e-cube hop is blocked at this node and it is changed as a misrouted NS message. M_2 is misrouted from (6,6) to (6,3) using C_1 channels in the counter-clockwise orientation. At (6,3), M_2 becomes a normal NS message again. This is because its next e-cube hop is available and also the current node and the destination node are in the same column. Then, it is routed to (6,2) by the e-cube algorithm using either C_0 or C_1 channel.

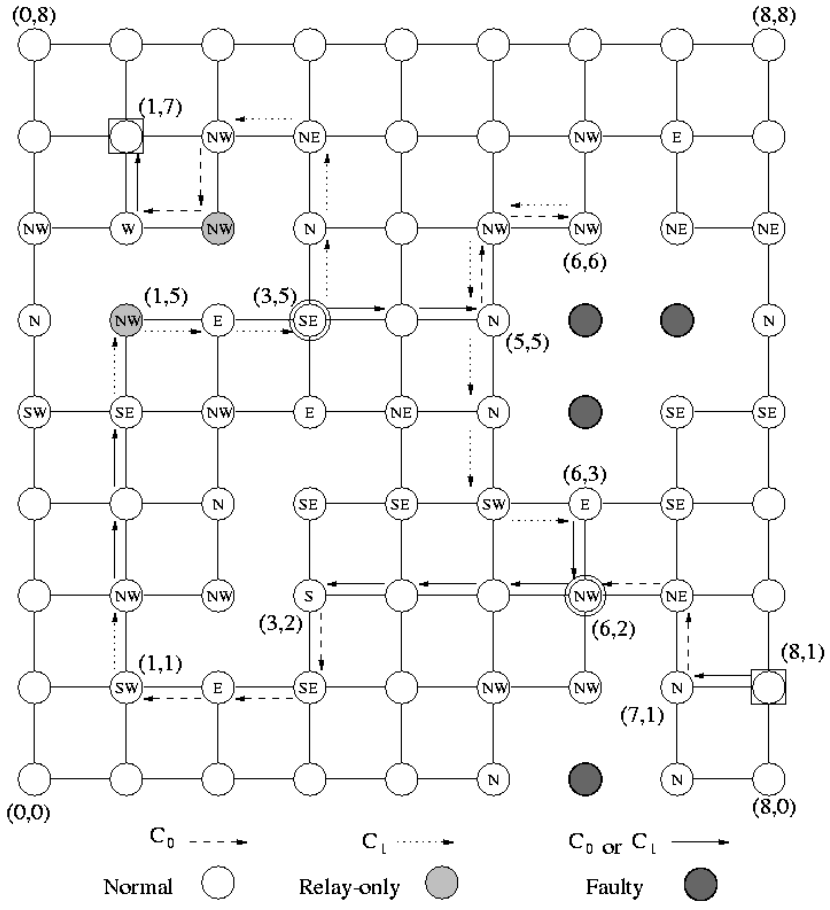

Figure 6. Examples of FT-Ecube routing.

4.2 The proof of deadlock freedom

Misrouted WE messages never travel the eastern part of an f-ring and misrouted EW messages never travel the western part of an f-ring. Furthermore, note that nodes that located in the column between the north and south convex sections in f-rings have been made as relay-only nodes. Thus, there are some virtual channels in C_0 that are never used by EW or WE messages. Those channels are located in the northern or southern part of f-rings. An example of unused virtual channels is shown in Fig. 7. The dotted arrows indicate unused virtual channels in the f-ring.

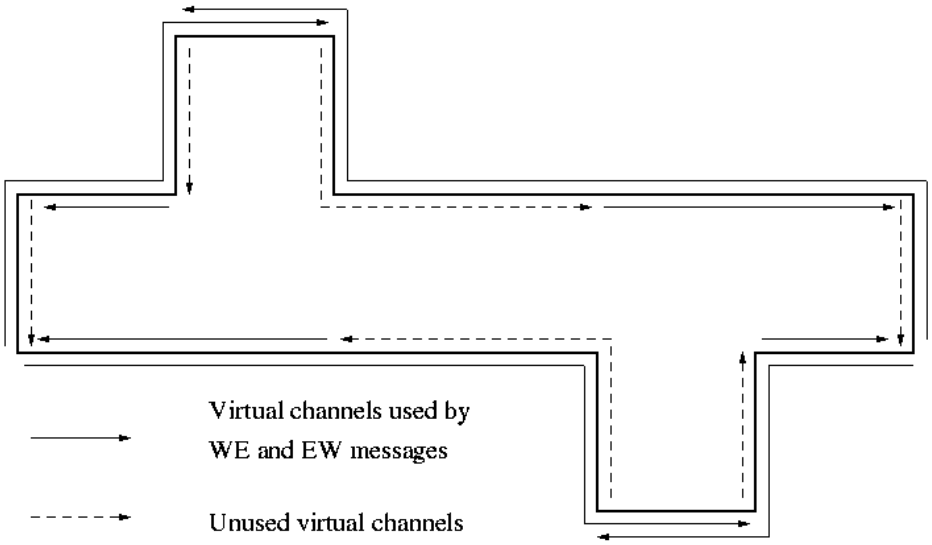

Figure 7. Unused virtual channels in C_0

Theorem 1. *FT-Ecube Routing is deadlock-free and livelock-free in meshes when it has nonoverlapping multiple f-regions.*

Proof: Since FT-Ecube routing is based on the e-cube routing, row messages may turn into column messages, but column messages do not become row messages. Thus, to prove the deadlock freedom, it is sufficient to show that there is no deadlock among row messages and column messages.

The deadlock freedom among WE and EW messages is straightforward. WE messages travel from west to east but not from east to west and they never touch the eastern part of an f-ring. Similarly, EW messages travel only from west to east and they never touch the western part of an f-ring. Thus, EW messages and WE use exclusive set of virtual channels and so, WE and EW messages do not cause any deadlock.

To prove the deadlock freedom among NS and SN messages, first we consider the deadlock freedom in f-rings. Since the deadlock freedom in multiple f-rings is shown in [3], we show only the deadlock freedom in single f-ring.

There are two cases:

1) There are at least two nodes one from north convex section and the other from the south convex section located in the same column (for example, refer Fig. 8. (a) and Fig. 8. (b)):

Let $W_N = (x_N, y_N)$ be the leftmost node in the north convex section, $W_S = (x_S, y_S)$ be the leftmost node in the south convex section and x_{MAX} be *max* (x_N, x_S). When a node $V=(x_v, y_v)$ located in the north or south convex section receives a column message *M*, it routes *M* along the east side of the f-ring if $x_v > x_{MAX}$ and along the west side otherwise. Because the same *x* coordinate, x_{MAX}, is used for deciding the misrouted orientation in north and south convex sections, the NS and SN messages are always misrouted in the opposite orientations. Also, the misrouted NS (respectively SN) messages may go in the south (respectively north) convex section up to a node whose *x* coordinate is the same as that of the *x* coordinate of the destination. Thus, the virtual channels used by NS and SN messages are disjoint and NS and SN messages do not cause any deadlock in f-rings.

2) The intersection of the x coordinates of the south and north convex sections is disjoint (for example, see Fig. 8. (c) and Fig 8. (d)):

In this case, NS and SN messages are misrouted in the opposite directions each other except in the region 'R'. Here NS and SN messages are misrouted in the same orientation. But they use different set of virtual channels in this region. Further, the C_0 virtual channels used by NS and SN messages are never used by row messages. Thus, the virtual channels used by NS and SN messages are disjoint.

The NS messages misrouted in the clockwise (respectively counter-clockwise) orientation never travel the west (respectively east) convex section and the NS messages misrouted in the counter-clockwise orientation never touch the east convex section. Thus, NS messages do not cause any deadlock in f-rings. Similarly, SN messages do not cause any deadlock in f-rings either.

In the case of f-chains, the virtual channels used by NS and SN are also disjoint. NS messages are never routed from south to north and SN messages are never routed from north to south, in f-chains. Also, NS and SN messages are always misrouted in the opposite orientations in f-chains. Thus, NS and SN messages use exclusive sets of virtual channels in f-chains and do not cause deadlock in FT-Ecube routing.

In FT-Ecube routing, messages are misrouted by a finite number of hops only when they are blocked by f-regions. Otherwise, they proceed toward their destinations using the e-cube algorithm. Since there is a finite number of f-rings in the network and the messages never visit the same f-ring more than twice, the amount of misrouted steps is also finite. Therefore, every message will eventually reach its destination without any livelock.

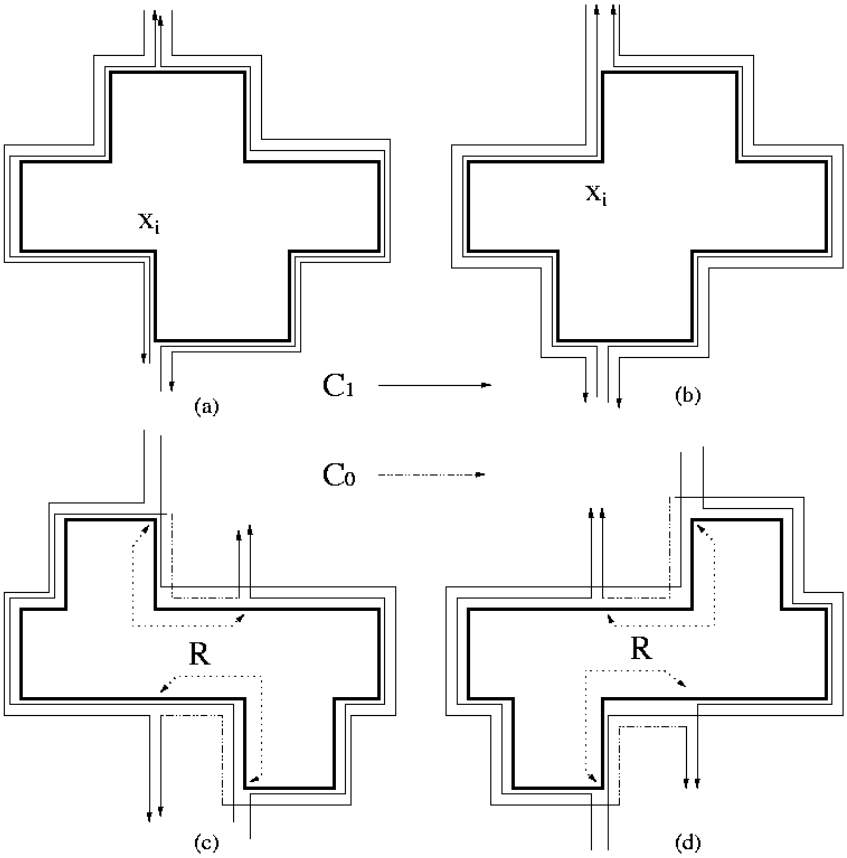

Figure 8. Misrouted orientations for column messages

5 Conclusions

In this paper, we have proposed a fault-tolerant routing algorithm for meshes. The proposed algorithm, FT-Ecube, can tolerate solid faults using only two virtual channels. In addition to the knowledge about the neighbor nodes on the f-region, each node has to know its position on the f-region. The position information can be attained after the formation of the f-region using the

proposed method. FT-Ecube not only reduces the number of virtual channels but also tolerates f-chains in the meshes. We assume that each node has only local knowledge of faults. So, there is no need to maintain global fault information and there is no restriction on the number of faults. We show that the proposed algorithm does not lead to deadlock and livelock with any number of nonoverlapping f-regions.

Reference:

[1] R.V. Boppana and S. Chalasani, "Fault-tolerant wormhole routing algorithms for mesh networks," *IEEE Trans. Computers*, vol. 44, no. 7, pp. 848-864, July 1995.

[2] Y. M. Boura and C. R. Das, "Fault-toelrant routing in mesh networks", *Proceedings of the 1997 International Conference on Parallel Processing*, vol I, pp. 106-109, August 1995.

[3] S. Chalasani and R.V. Boppana, "Communication in multicomputers with nonconvex faults," *IEEE Trans. Computers*, vol. 46, no. 5, pp. 616-622, May 1997.

[4] A. Chien and J. Kim, "Planar-adaptive routing: low-cost Adaptive networks for multiprocessors", *Proc., 19th Ann., Int'lSymp. Computer Architecture*, pp. 268-277, 1992

[5] J. Duato, S. Yalmanchili and L. Ni, "Interconnection networks an engineering approach", Los Alamitos, California, IEEE Computer Society Press, 1997.

[6] C. Glass and L. M. Ni, "Fault-tolerant wormhole routing in meshes without virtual channels", *IEEE Trans. Parallel and Distributed Systems*, vol. 7, no. 6, pp. 620-636, June 1996.

[7] T. Lee and J.P. Hayes, "A fault-tolerant communication scheme for hypercube computers", *IEEE Transactions on Computer*, vol. C-41, no. 10, pp. 142-1256, October 1992.

[8] S. Park, J. Youn and B. Bose, "Fault-tolerant wormhole routing algorithms in the presence of concave faults", *International Parallel and Distributed Processing Symposium*, pp. 633-638, May 2000.

[9] S. Park, J. Youn and B. Bose, "Wormhole routing in faulty mesh networks", *International Conference on Parallel and Distributed Processing Techniques and applications*, pp. 1007-1012, June 2000.

[10] C. Su and K. Shin, "Adaptive fault-tolerant deadlock-free routing in meshes and hypercubes," *IEEE Trans. Computers*, vol. 45, no. 6, pp. 666-683, June 1996.

[11] P. Sui and S. Wang, "An improved algorithm for fault-tolerant wormhole routing in meshes," *IEEE Trans. Computers*, vol. 46, no. 9, pp. 1040-1042, September 1997.

[12] J. Youn, B. Bose and S. Park, "Fault-Tolerant Communication in Meshes with Some Nonconvex Faults", *International Conference on Communications in Computing*, pp. 233-239, June 2000.