Hindawi Publishing Corporation
Journal of Applied Mathematics
Volume 2014, Article ID 592638, 10 pages
http://dx.doi.org/10.1155/2014/592638

Research Article

Adaptive Fault-Tolerant Routing in 2D Mesh with Cracky Rectangular Model

Yi Yang,[1] Meirun Chen,[2] Hao Li,[3] and Lian Li[1]

[1] *School of Information Science and Engineering, Lanzhou University, Lanzhou 730000, China*
[2] *School of Applied Mathematics, Xiamen University of Technology, Xiamen 361024, China*
[3] *Laboratoire de Recherche en Informatique, Bat 490, Universite Paris-Sud 11, 91405 Orsay Cedex, France*

Correspondence should be addressed to Yi Yang; yy@lzu.edu.cn

Received 7 February 2014; Accepted 9 March 2014; Published 7 April 2014

Academic Editor: X. Song

This paper mainly focuses on routing in two-dimensional mesh networks. We propose a novel faulty block model, which is cracky rectangular block, for fault-tolerant adaptive routing. All the faulty nodes and faulty links are surrounded in this type of block, which is a convex structure, in order to avoid routing livelock. Additionally, the model constructs the interior spanning forest for each block in order to keep in touch with the nodes inside of each block. The procedure for block construction is dynamically and totally distributed. The construction algorithm is simple and ease of implementation. And this is a fully adaptive block which will dynamically adjust its scale in accordance with the situation of networks, either the fault emergence or the fault recovery, without shutdown of the system. Based on this model, we also develop a distributed fault-tolerant routing algorithm. Then we give the formal proof for this algorithm to guarantee that messages will always reach their destinations if and only if the destination nodes keep connecting with these mesh networks. So the new model and routing algorithm maximize the availability of the nodes in networks. This is a noticeable overall improvement of fault tolerability of the system.

1. Introduction

In the last decades, the goal of many researchers was to study communication operations in networks with fixed topologies, including modeling architectures and routing algorithm of parallel computers and cluster or middle area communication networks (such as metropolitan networks covering a town or a small region). The quality of such networks strongly depends on correct and efficient execution of communication operations.

Direct networks [1] become a popular architecture for communication networks, especially in massively parallel computer system. In direct networks, nodes (computers) are connected to only a few nodes, that is, its neighbours, according to the topology of the networks and communicate with each other by exchanging messages. Moreover, the mesh structure is one of the most important topology of direct networks. Especially, low dimensional mesh networks, due to its low node degree, are more popular than the high dimensional mesh networks. Currently most of architecture of parallel computers is based on two-dimensional mesh topology, for example, Seitz et al. 1988 [2], Intel Touchstone DELTA [3, 4], and Intel paragon.

Several models based on direct networks have been studied ([5–9]), especially the two-dimensional mesh ([10–16], etc.) for communication operations. The purposes of these papers mainly focus on how to route messages in the two-dimensional mesh. Routing is the process to send messages from source nodes to destination nodes, passing some intermediate nodes. A very important aspect of message routing is its ability to route from a source node to a destination node, avoiding all faulty nodes or links.

Basically, there are two types of message routing:

(1) *deterministic routing* that is routing in which the routes between given pairs of nodes are determined in advance of transmission,

(2) *adaptive routing* that allows us to take any path between its source and its final destination; that is,

the path is adaptively constructed in the process of routing.

The deterministic routing algorithms are simple and ease of implementation, this is the advantage for deterministic routing. However, adaptive routing can reduce network latency and increase network throughput and the most attractive point is that it can tolerant more faults than deterministic routing [17]. Thus the latter one emerged as an attractive field. In most papers on this field, they often considered how to make a path between source and destination node pairs, avoiding the faulty nodes, and most work used the disconnected rectangular block fault model [11]. The disconnected rectangular blocks are composed of the faulty nodes and their neighboring nonfaulty nodes with the principle of maintaining rectangular shape. As a result, adaptive routing can tolerate faulty nodes by bypassing these rectangles. However, in order to maintain its rectangular shape, the block has to group some nonfaulty nodes inside, called unsafe nodes in these papers. Of course, these unsafe nodes will never be used until their corresponding blocks recovery, and the messages will never be sent to these nodes, while they should be (as illustrated in Figure 1).

Chien and Kim [18] present a partially adaptive algorithm for mesh networks. The basic idea is to use the algorithm to circumfuse any convex faulty regions. If faulty regions are not naturally convex, good nodes and links are marked as faulty until the regions become convex. However, once the faults are located on a boundary, in order to tolerate faults, all nodes form that boundary will become faulty. Boppana and Chalasani [10] use f-chain and f-ring, which is an extension of disconnected rectangular block fault model, to route the messages around them, and f-chain addresses the boundary problem in the Chien and Kim's paper. But the f-chain and f-ring may connect with each other; this makes the routing algorithm more complex than [18]. In [11], Su and Shin assume a node to be the basic fault element. They construct the blocks based only on the faulty nodes; thus they can only tolerate faulty nodes except the faulty links. Overall, the construction of these faulty regions is static; that is, once these regions are constructed, all nodes including the good ones in these regions cannot join in routing any more. The faulty regions are not self-adaptive; that is, if some of faulty nodes in these faulty regions are fixed well, then the faulty regions will be held as they were, but actually they can release some good nodes and become smaller ones keeping convex shape.

Adaptive fault-tolerance routing technologies are also using in WSN (Wireless Sensor Networks), MEMS (Micro-Electro-Mechanical Systems) and SoC (System on Chip) to increase the usability and robustness, as well as the whole performance. Most network topology adopted in those domains is 2D mesh. As a result, in recent years, there have been a number of researches focusing on fault-tolerance routing on wsn and Noc [19–22].

In this paper, we concentrate on the adaptive routing with fault-tolerant in two-dimensional mesh. Not only we do consider the situation of faulty nodes but also the situation of faulty links incident with any node. However, different from mentioned papers, the novel cracky rectangular block strategy introduced to tolerate faults can route messages both bypassing the cracky rectangular block and along the cracks in the rectangular block (just for a trope, actually they are routed along the connected links inside the faulty blocks). So we can route messages to the nodes both outside and inside the faulty blocks. This is a noticeable overall improvement of fault tolerability of the system. At the same time, the cracky rectangular block is fully self-adaptive. It can tolerate dynamic faults. For example, when some of faulty nodes or faulty links in a block are fixed well, the original block may become a smaller block or split to some smaller ones keeping their shape rectangular. Tolerating dynamic faults can enhance the run-time life of a multicomputer, thus increasing reliability.

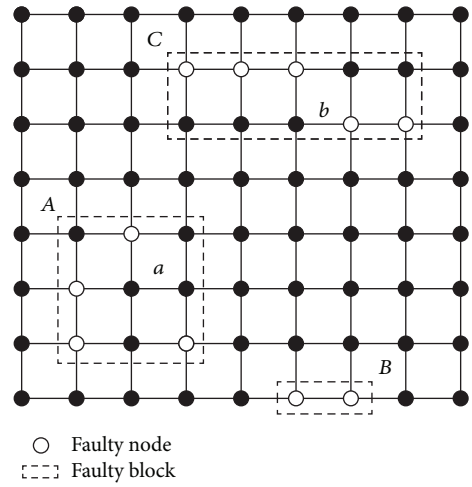


Figure 1: An example of disconnected rectangular blocks. Note that nonfaulty nodes, such as nodes a and b, in a block will never be used in routing any more.

The rest of this paper is organized as follows. Section 2 describes the basic routing algorithm in two-dimensional mesh. Section 3 introduces the cracky rectangular block strategy, including the cracky rectangular block model and the routing algorithm on it. This section also describes how the rectangular blocks adapt themselves depending on the situation of networks. Section 4 gives a proof that the message will be sent to any destination in the mesh as long as the mesh keep connecting. A conclusion will be given in Section 5, and it presents possible directions for future work.

2. Basic Routing Algorithm in Two-Dimensional Mesh

2.1. Two-Dimensional Mesh. It is convenient to represent a two-dimensional mesh with graph terminology. Let $G = (V, E)$ be undirected graph to represent a network. The set V of vertices of graph G represents nodes of the network. The set E of edges of graph G represents links between the nodes. Note that we keep using node and link in this paper.

The two-dimensional mesh $M(d_1, d_2)$, where $d_1 \geq 2$ and $d_2 \geq 2$ are positive integers, is defined as follows.

(i) A node X of $M(d_1, d_2)$ is represented by $X(x_1, x_2)$, $0 \le x_1 \le d_1 - 1$, and $0 \le x_2 \le d_2 - 1$.

(ii) There is a link between two different nodes $X(x_1, x_2)$ and $Y(y_1, y_2)$ if and only if $x_1 = y_1$ and $x_2 = y_2 \pm 1$ or $x_1 = y_1 \pm 1$ and $x_2 = y_2$. We denote this link by $\langle X, Y \rangle$.

For each k and k', with $0 \le x \le d_1 - 1$ and $0 \le x' \le d_2 - 1$, we call *row* k and *column* k' the subgraphs of $M(d_1, d_2)$, respectively, induced by nodes (x, k) and (k', x').

The *boundary* of $M(d_1, d_2)$ is the subgraph of $M(d_1, d_2)$ induced by the *rows* 0 and $d_1 - 1$ and the *columns* 0 and $d_2 - 1$.

Given any node v, let $\Gamma(v)$ be the set of nodes adjacent to v in $M(d_1, d_2)$ (called as *neighbours*). Given a nonboundary node $X = (x_1, x_2)$ of the two-dimensional mesh, the four neighbours of X are denoted by $W(X) = (x_1 - 1, x_2)$, $S(X) = (x_1, x_2 - 1)$, $E(X) = (x_1 + 1, x_2)$, and $N(X) = (x_1, x_2 + 1)$.

For each pair of nodes $X = (x_1, x_2)$ and $Y = (y_1, y_2)$, the distance between X and Y, denoted by $d(X, Y)$, is the length (number of links) of a shortest path between X and Y. We define the 1-distance and 2-distance between X and Y, respectively, by $d_1(X, Y) = |x_1 - y_1|$ and $d_2(X, Y) = |x_2 - y_2|$. From the above definition, we know that $d(X, Y) = d_1(X, Y) + d_2(X, Y)$.

2.2. A Basic Routing Function in Two-Dimensional Mesh. Consider a network $G = (V, E)$, in each node v, for each message m with final destination v_d, arriving on a link $\langle w, v \rangle$; we denote by $f_v(w, v_d) \subset \Gamma(v)$ the subset of v's neighbours bringing m closer to its destination if $v_d \neq v$; otherwise, the message is absorbed by v. Actually it is a routing function, this kind of routing is said to be local because it is independent of what happened in the rest of the network and can be computed locally by each router.

The basic routing function is a classical greedy routing function f_M in the two-dimensional mesh $M(d_1, d_2)$ as follows. Let $X = (x_1, x_2), Y = (y_1, y_2)$ be two different nodes in $M(d_1, d_2)$. A message m with destination Y received by the router of X arriving from node X' is sent to a node of $f_X(X', Y)$, that is, a set of at most two nodes $\{V_1, V_2\}$ (and at least one node) defined as follows. There are at most two nodes $V_1 \in \{W(X), E(X)\}$ and $V_2 \in \{N(X), S(X)\}$ at distance $d(X, Y) - 1$ from Y. Moreover, when $|f_X(X', Y)| = 2$, if $d_1(X, Y) \ge d_2(X, Y)$ (resp., $d_1(X, Y) < d_2(X, Y)$), then the routing function will choose to send the message to V_1 (resp., V_2). If this is not possible (e.g., the link incident with the chosen node is faulty), then the routing function tries to send the message to V_2 (resp., V_1). Moreover, if both the links $\langle X, V_1 \rangle$ and $\langle X, V_2 \rangle$ are faulty, then the router of X can route m to any node of $\Gamma(X) \setminus \{V_1, V_2\}$.

2.3. Blocking Situation and Its Traditional Solution. Consider now that a unique message m is transmitted to the two-dimensional mesh $M(d_1, d_2)$. As we will show below, in case of some link faults which do not disconnect the network, using the basic two-dimensional routing function does not guarantee that m will reach its destination. It can be blocked in a part of the network.

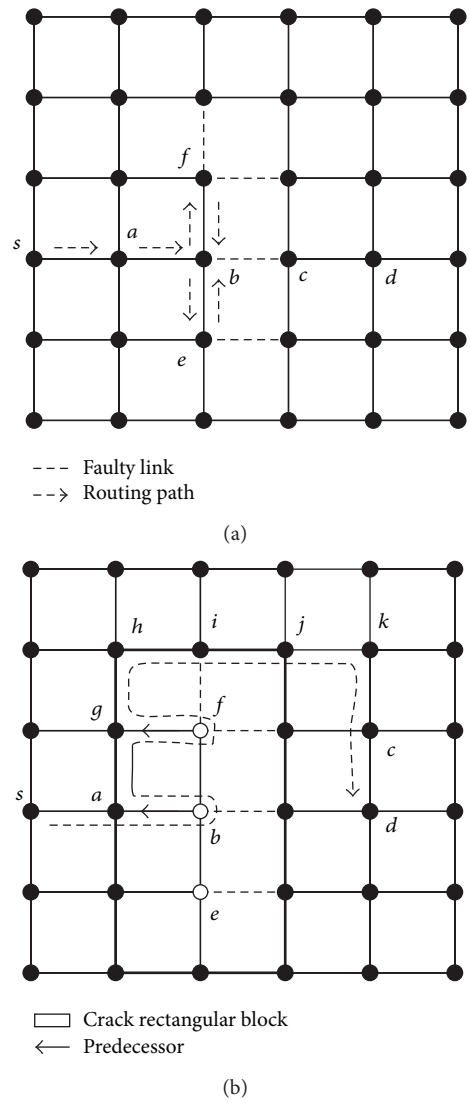


Figure 2: (a) Livelock situation in $M(d_1, d_2)$: node s tries to send a message m to node d, but m is in a livelock induced by nodes b, e, and f caused by the faulty links. (b) Cracky rectangular block solves the livelock problem, and the message m will be sent to its destination along the node sequences $s, a, b, a, g, f, g, h, i, j, k, c$, and d.

As shown in Figure 2(a), the basic routing function can unfortunately lead to blocking situations due to some properties of the structure of the faulty links. Clearly, in this example the message will always in the subgraph induced by nodes b, e, and f and will never reach its destination node d. Actually, this message m is in livelock situation, which keeps a message moving indefinitely without reaching the destination.

It is well known that the adaptive routing may cause livelock problems. Therefore routing without livelock is one of the most important design issues for communication operations in multicomputer systems (note that we only consider the livelock situation in this paper, and we can solve the deadlock problem with some sophisticated methods [1, 23, 24]). Contemporary, this livelock situation is well addressed by the traditional disconnected rectangular block

faulty model (rectangular block for short). However, the usability and robustness of the mesh network will gradually decrease, while the number of faulty nodes increases in this model. As [25]'s experiment shows that the distribution of faulty nodes has the tendency to make the whole mesh to be one "big block." It can be seen from the experiments that, with the rectangular model, there is only one faulty block left when the faulty rate of nodes is 15 percent and the size of two-dimensional mesh is 100×100. In consequence the whole mesh becomes useless because this big faulty block occupies the entire mesh region, and we call this as "big block" problem. The novel cracky rectangular faulty block strategy, which we will introduce in the next section, makes full use of nonfaulty nodes/links in the mesh. All the nonfaulty nodes/links that would have been included in original rectangular faulty blocks now can become candidate routing nodes/links.

3. Adaptive Fault-Tolerant Strategy with Cracky Rectangular Block

In order to solve the livelock situation and the big block problem, we propose a novel strategy for fault-tolerant routing. We use the cracky rectangular block to avoid livelock and traverse block's every connecting internal node if needed. Therefore, we can transmit each message to any node not only outside of a block but also inside of the block like Figure 2(b), and the message can reach the inside nodes f, b, and e which are forbidden in the original rectangular block.

Formally, a rectangular block $C((l_1, h_1), (l_2, h_2))$ is a submesh $M(h_1 - l_1 + 1, h_2 - l_2 + 1)$ of the mesh $M(d_1, d_2)$ induced by the nodes $x = (x_1, x_2)$ with $l_i \le x_i \le h_i$, for each $i \in \{1, 2\}$. Let $u = (u_1, u_2)$ be a node. By definition, if, for each $i \in \{1, 2\}$, we have $l_i - 1 \le u_i \le h_i - 1$, then u belongs to the inside part of the rectangular block C. Else, if $i = 1$, $j = 2$ (or $i = 2$, $j = 1$) and $u_i \in \{h_i, l_i\}$ and $l_j \le u_j \le h_j$, then u belongs to the border of the rectangular block.

A *cracky rectangular block* (*cracky block* for short) is a rectangular block with spanning forest internal induced by all the connecting nodes inside of this block, all the roots of that forest belong to the border of the cracky block, and the spanning forest connects all the internal nodes to their roots if and only if those nodes still keep connecting.

Figure 3 presents two instances of the cracky block in a two-dimensional mesh, which are A and B, respectively. A is a general cracky block, while B is a cracky block which is induced by the faulty links on the boundary of the mesh, and it is an incomplete cracky block.

3.1. Construction of the Cracky Rectangular Block. Each node's activities are based on message-driven mechanism. There are two types of messages routed in mesh. One is *entity message* (message for short), which is routed between any node pair. The other one is *system message*, this type of message can only be sent between neighbours, and their contents are mainly about the status of themselves, such as the node's faulty degree and it's detailed situation of faulty links. The first one is the entity for computing or communication, and the later one concentrates on maintaining the usability and robustness of networks; in other words, it is for constructing the cracky block when some faults occur in this mesh in order to avoid the livelock situation as mentioned above.

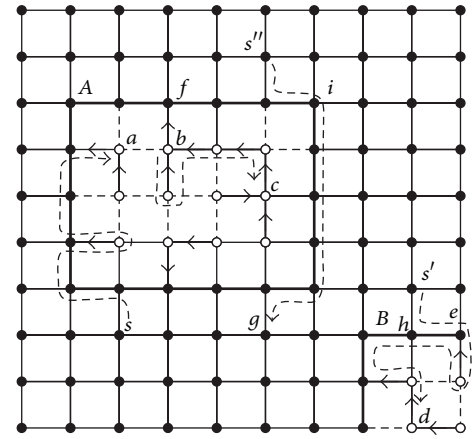


Figure 3: A and B are two cracky rectangular blocks. A is a general cracky block, while B is a cracky block which is induced by the faulty links on the boundary of the mesh, and it is an incomplete cracky block. As nodes a, b, and d in the cracky blocks A and B are faulty nodes, f and i are border nodes, and any node outside of A and B is good node. With the cracky rectangular block, not only good node s'' keeps connecting with another good node g as general, but also the good nodes s, s' can send messages to faulty nodes a, d, respectively; what is more, those faulty nodes can communicate with each other, such as nodes b and c.

In the beginning, all the nodes work well; that is, there does not exist any faulty node or faulty link. Any node can both receive the message from any of its neighbours, and vice versa, of course, depending on the basic routing strategy. When some nodes or links are ruined because of some reasons, these failed nodes or nodes incident with failed links will judge their current status immediately, and then they send the system messages as soon as possible including their status to its connected neighbours to tell them what have happened in detail. For neighbours, once they receive the system messages, they judge their current status depending on their latest status and the received system messages at once. Of course, they will notice their connected neighbours about current status if and only if the current status is different from previous status. Finally, the construction of a stable cracky block is implemented by the above system messages exchange.

Before exposing our distributed algorithm to construct the cracky block, we will give some definitions first. For a two-dimensional mesh $M(d_1, d_2)$, the faulty degree of a node $X = (x_1, x_2)$ is the number of failed links incident with X, and we denote it by $f(X)$. From the observation of a cracky block, there are three types of nodes in a mesh network: *faulty node*, *good node*, and *border node*. A faulty node X belongs to the interior of a cracky block and $0 \le f(X) \le 4$, and oppositely, a good node allocates outside of any cracky blocks and $f(X) = 0$. Of course a border node belongs to the border of a cracky block with $0 \le f(X) \le 1$. For example, in Figure 3, a, b, and d in the cracky blocks A and B are faulty nodes, f

Input: $f(v)$: faulty degree of node v.
Output: $s(v)$: status of node v.
(1) **procedure** INITIAL_STATUS(v)
(2) **if** $f(v) = 0$ **then**
(3) $s(v) \leftarrow 1$
(4) **else if** $f(v) = 1$ **then**
(5) $s(v) \leftarrow$ depending on Table 1
(6) **else if** $f(v) \geq 2$ **then**
(7) $s(v) \leftarrow \emptyset$
(8) **end if**
(9) *notice_status*(v)
(10) **end procedure**
Input: $s(v)$: the current status of node v, $M(v)$: the system message received by node v.
Output: $s(v)$: the updated status of node v.
(11) **procedure** UPDATE_STATUS(v)
(12) **if** $s(v) \neq M(v) \cap s(v)$ **then**
(13) $s(v) \leftarrow M(v) \cap s(v)$
(14) *notice_status*(v)
(15) **end if**
(16) **end procedure**
Input: $s(v)$: the updated status of node v.
Output: $M(v)$: the system message to be sending to $N^*(v)$.
(17) **procedure** NOTICE_STATUS(v)
(18) **if** $s(v) = \emptyset$ **then**
(19) send system message $M(v)$ to $N^*(v)$
(20) **else if** $s(v) \in \{\{E\}, \{W\}\}$ **then**
(21) send system message $M(v)$ to both $N(v)$ and $S(v)$ if exist
(22) **else if** $s(v) \in \{\{N\}, \{S\}\}$ **then**
(23) send system message $M(v)$ to both $E(v)$ and $W(v)$ if exist
(24) **end if**
(25) **end procedure**

ALGORITHM 1: Let v be any node in the two-dimensional mesh, let $f(v)$ be the faulty degree, let $s(v)$ be the status, and let $M(v)$ be the content of received system message.

TABLE 1: Given a node v with $f(v) = 1$, judging its $s(v)$ according to its incident failed links.

$s(v)$	$f(v) = 1$
$\{N\}$	The link $\langle v, S(v)\rangle$ failed
$\{E\}$	The link $\langle v, W(v)\rangle$ failed
$\{S\}$	The link $\langle v, N(v)\rangle$ failed
$\{W\}$	The link $\langle v, E(v)\rangle$ failed

and i are border nodes, and any node outside of A and B is good node.

Given any node X, let $N^*(X)$ be the set of neighbors Y of X such that the link $\langle X, Y\rangle$ is not faulty. Moreover, we set an order in $N^*(X)$ as follows: $u_0, u_1, \ldots, u_{|N^*(X)|-1}$. Let X_p be the node who sends message to node X, and let X_s be the node who will receive the message sent by X.

We denote by $s(X)$ the status of X, and $s(X)$ is one of the elements of status set $STATUS = \{\{E\}, \{S\}, \{W\}, \{N\}, \{N, E\}, \{S, E\}, \{S, W\}, \{N, W\}\}$. The status of a node will indicate which type of node it is. In detail, there are two more status of X, which are empty set $\emptyset$ and universal set 1. And $s(X) = \emptyset$ shows that the node is a faulty node, $s(X) = 1$ identifies a good node, and if $s(X) \in STATUS$, then X must be a border node. For example, if $s(X) = \{N\}$, then the node X locates at the north border of a cracky block, like f in Figure 3, or if $s(X) = \{N, E\}$, then X is at the northeast corner, just like i. Totally, the system message can be sent to four neighbours, and X_s will be E, S, W, and N according to $E(X), S(X), W(X)$, and $N(X)$. A system message sent by node X is $M(X) = \{X_s\} \cup s(X)$ which includes the destination neighbour and sender's current status. For example, $M(X) = \{E, N\}$ means that this message will send to $E(X)$ and $s(X) = N$. We define an operation to implement the status judgment of a node who receives a novel system message. The corresponding algorithm to update the status for any node in mesh network is given by Algorithm 1.

At the beginning of the construction, every node should run the procedure initial_status respectively to make sure its status $s(v)$ according to its faulty degree $f(v)$. After finishing the above procedure, node will run the procedure notice_status to send system messages to neighbours according to its status $s(v)$. Once a neighbour node receives this type of message, it will run the procedure update_status to refresh its latest status. Actually, this process will be repeated until every node's status getting stable. Finally, there will emerge some cracky blocks in the mesh. For example, Figure 4 shows

Input: v: node v is in a cracky rectangle block.
Output: $l(v)$: $\{h, f\}$, $pred(v)$: predecessor of v.
(1) **procedure** HungNodeOnForest(v)
(2) **if** v is a *good* or *border* node **then**
(3) $l(v) \leftarrow h$
(4) return
(5) **else if** v is a faulty node **then**
(6) $l(v) \leftarrow f$
(7) **end if**
(8) $u_i \in N^*(v)$
(9) check $l(u_i)$ for all u_i until $\exists u \in N^*(v)$ s.t. $l(u) = h$
(10) $l(v) \leftarrow h$
(11) pred(v) $\leftarrow u$
(12) **end procedure**

Algorithm 2: Let v be a node in a cracky rectangle block, making it hung when it is possible.

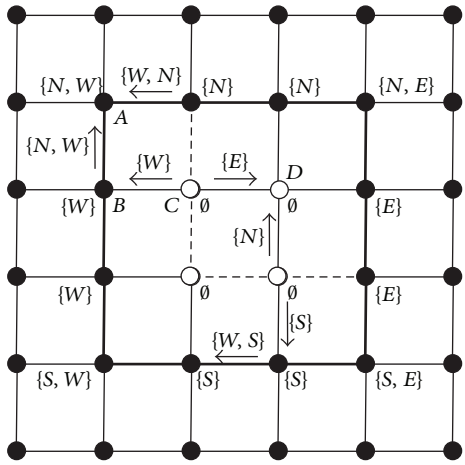


Figure 4: Construction of a cracky block depends on system message exchange. The dash line represents faulty link, and the bold line makes up the border of a cracky block. The arrow refers to the system message.

a distributed process to construct a cracky block. We just pick up four nodes, $A, B, C,$ and D, to describe how the algorithm performs. During the first phase, node C will initial its status $s(C) = \emptyset$ because of $f(C) = 2$; meanwhile $s(A) = 1, s(B) = 1, s(D) = 1$ as a result of $f(A) = 0, f(B) = 0,$ and $f(C) = 0$ separately. In the second phase, according to the algorithm only node C will send system message to its connected neighbours which are nodes B and D. Finally node A receives two system messages and will refresh its new status by $s(A) = (1 \cap \{N, W\}) \cap \{W, N\} = \{N, W\}$, so it will be the northwest of a cracky block for this moment. For nodes B and D, $s(B) = 1 \cap \{W\} = \{W\}$ and $s(D) = (1 \cap \{E\}) \cap \{N\} = \emptyset$ are a west border node and a faulty node. When the algorithm stops, there is a border of crack block as shown in Figure 4 by the bold line. The macroconstruction of a block depends on the microdistributed message exchange activities of relative nodes.

Then we will construct the spanning forest for the faulty nodes. We say that the faulty node is *hung* if and only if it chooses exactly one neighbor as predecessor. Denote by pred(v) the predecessor of v, and Succ(v) = $\{v' \in \Gamma(v) : v'$ is a faulty node and $v = \text{pred}(v')\}$. We consider an order over the elements of Succ(v). We denote by $\text{succ}_i(v)$ the ith element of Succ(v), with $1 \le i \le k_v = |\text{Succ}(v)|$. A node v is said to be *final* if Succ(v) = $\emptyset$. A node which is not hung is *free*. We denote by $l(v)$ these two boolean states $\{h, f\}$, which refers to the hung and free status for each node inside of the block. After running Algorithm 2, the spanning forest for the block will be accomplished; like A and B in Figure 3, every node inside of blocks A and B will find only one predecessor and be marked as hung.

3.2. The Cracky Rectangular Block Is Stable. A node v is said to be *stable* if pred(v)'s status can never change to a free status, during the running time of the algorithm. In particular the nodes of the cracky block are stable. A cracky block C is said to be stable if all the nodes belonging to C are stable.

To prove that the cracky block is stable, we assume that there exists a set S of nonstable nodes. If $S = \{v\}$, then v is free and can never become stable during the running time of Algorithm 2, so in this case there is no stable node in the neighborhood of v because otherwise v will choose this node as predecessor. Using the same argument for each node w of $N^*(v)$, there is no stable node in the neighborhood of w. But since the graph is connected, v is necessarily joined by a path to a node u of the border of the block. So we are in contradiction with the fact that the nodes of the border are stable. If $|S| \ge 2$, then since the graph is connected there exists at least one node u in S which is adjacent to a node $v \notin S$. Clearly, v is stable. From the second loop of the algorithm and since there is an order in the neighbors of u for the choice of its predecessor, there exists a step in the algorithm which leads the node u to choose v as predecessor. After this step, let $S \leftarrow S \setminus \{u\}$. Using the same arguments, after some steps of the algorithm, the set S would be empty. So all the nodes are stable.

3.3. Adaptive Routing with the Cracky Rectangular Blocks. In this section, we will give the global fault-tolerant routing

Input: v: node v routing messages, v_p: the node who sends messages to node v.
Output: v_s: the node who will receive the messages.

```
(1) procedure ROUTING(v)
(2)     if v is a good node then
(3)         basic routing function with node v
(4)     else if v is a faulty node then
(5)         if (v is final) or (v_p) = succ_{κ_v}(v) then
(6)             v_s ← pred(v)
(7)         else if v_p = succ_i(v) and i < κ_v then
(8)             v_s ← succ_{i+1}(v)
(9)         else if v_p = pred(v) then
(10)            v_s ← succ_1(v)
(11)        end if
(12)    else if v is a border node then
(13)        routing according to Table 2.
(14)    else if v is a node belongs to the border of mesh then
(15)        if v is final then
(16)            v_s ← v_p
(17)        else
(18)            v_s ← succ_1(v)
(19)            succ_1(v) ← v_p
(20)        end if
(21)    end if
(22) end procedure
```

ALGORITHM 3: The novel routing algorithm based on cracky rectangular blocks.

strategy. Primarily, once a message encounters a cracky block, this message will bypass the cracky block, which encloses the faulty nodes/links, along its border node in a clockwise (or counter-clockwise) manner. Especially, the message should traverse the interior spanning tree rooted with the border node by Depth-First-Search, while it bypasses the cracky block. Finally, the message will leave the cracky block from one of its corners which is the nearest from the destination and keeps going with the basic routing function; otherwise the message will be absorbed by the interior node which must be the destination node.

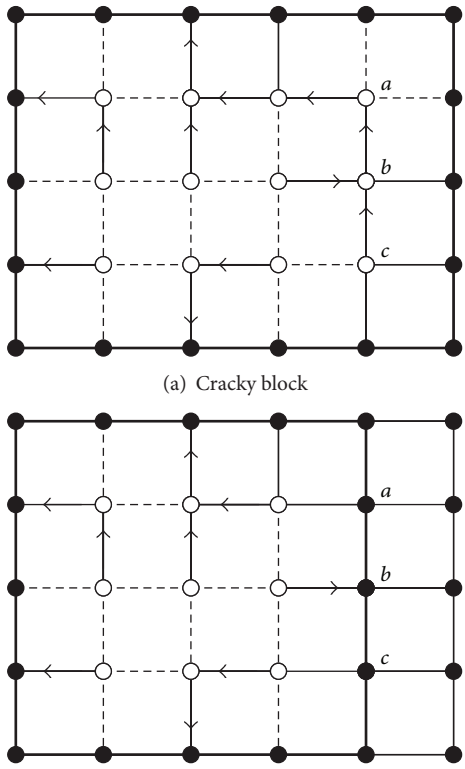
We now give the complete local routing function we run in each node v of $M(d_1, d_2)$, as shown in Algorithm 3. This algorithm is based on the basic routing function we have defined in Section 2.

The cracky rectangular block and the adaptive fault-tolerant algorithm make up the fault-tolerant strategy, and we can use Algorithm 3 to send a message from any connected node to arbitrary connected node. For example, in Figure 3, the good node s wants to send a message to the node a, but node a is a faulty node and locates interior the cracky block A, the algorithm will send this message along the path shown in the figure, and the faulty node b sending message to another faulty node c also can be accomplished by the algorithm; if a good node s'' wants to communicate with another good node g, the routing path will like the situation depicted in the figure.

3.4. Self-Adaptive and Faulty Boundary Independency of Cracky Rectangular Block. For high performance and usability, the cracky blocks should be self-adaptive. As we know, the emergency of cracky blocks in a mesh is the result of nodes managing themselves distributedly and independently. The status of an isolated node is closely related to their neighbours. Therefore the size and shape of a block are dynamic according to faulty nodes. In other words, if some of the faulty nodes have been fixed, the original block may become a smaller one or split up into smaller ones. On the contrary, if some good nodes or links fail, there will be some new cracky blocks or some of the original cracky blocks grow huge as a result.

Given a two-dimensional mesh $M(d_1, d_2)$, let $v(v_1, v_2)$ be a faulty node in cracky block $C((l_1, h_1), (l_2, h_2))$. Let $X(x_i, v_1)$ with $l_1 \le x_i \le h_1$, and let $Y(v_2, x_j)$ with $l_2 \le x_j \le h_2$. There is a fact that when v has been repaired such that $f(v) = 0$, then we should make sure if $f(X) = 0$ (resp., $f(Y) = 0$). If they are, then X (resp., Y) may be cancelled from the block C and C will become four smaller ones at most. In addition, these new cracky blocks still keep stable. The cancelled row or column may becoming the new border belonging to those new cracky blocks, alternatively becoming the good ones outside any blocks, so they will still keeping hung, certainly their successors will also keeping hung. To implement the above, when a node with its incident links is fixed well, we just send a recovery signal to its four neighbours to rerun the procedure initial_status in Algorithm 1. Recursively, the recovery signal will be sent to nodes which connected with the faulty nodes received the signal until it meets the good node outside the cracky block.

For example, Figure 5(a) shows a cracky block, and a, c are two faulty nodes with $f(a) = 2$, $f(c) = 1$, and $f(b) = 0$. When the two nodes a and c have been repaired, they all changed to good nodes with $f(a) = f(c) = 0$. The cracky block will become like Figure 5(b), and a, b, and c become the



(a) Cracky block

(b) Cracky block after nodes a, b, and c fixed well

Figure 5: A sample of self-adaptive block.

new border and the cracky still keeps stable (note that we do not give the detail because of the page limitation).

Our adaptive fault-tolerant routing strategy is faulty boundary independency; that is, if there exists a fault occurring on the boundary of the mesh, the strategy is still running. Lines 14 to 21 in Algorithm 3 give a solution to this situation. When some of the boundary nodes of mesh have failed, then the corresponding cracky block will be constructed like B in Figure 3. As shown, it is an incomplete cracky block. If s' wants to send a message to d, the message will first go to node h according the basic routing function. Because h is the border node of B, the message will be sent to e which is a boundary node of the mesh. When the message traverses all the successors of e, it will be rebound to the node h to continue routing and finally find its destination. To sum up, the message will continue routing when it encounters a mesh boundary because of the rebound function.

4. The Cracky Rectangular Model Is Creditable

The next two propositions show that, with the above algorithm, each message will reach its destination, if a message arrives in a node of a cracky block C;

(i) if its destination is in C, it will reach it;

(ii) if its destination is out of C, it will leave C closer to its destination than before.

This is shown by the next two results.

Proposition 1. *Consider a cracky block C. By using Algorithm 3, if a message m has its destinations Y in C and if it arrives in a node of the cracky block, then it will reach its destination.*

Proof. Consider a subgraph S of the mesh induced by the nodes of a cracky block $C = C((l_1, h_1), (l_2, h_2))$. By definition of Algorithm 3, a message moving on S follows a circuit crossing each node at least once.

Consider a message m moving in the mesh, with destination $X(x_1, x_2) \in V(C)$, reaching a node $X' \in V(C)$. By definition of the routing function, since $l_1 \le x_1 \le h_1$ and $l_2 \le x_2 \le h_2$, then m will never want to take an arc out from the C. Thus, it follows the circuit of S induced by the algorithm. So, m will arrive in node X. □

Proposition 2. *Consider a cracky block C. By using the algorithm, if a message m has its destination Y outside C and if it arrives in node of the cracky block, then it will leave C and be closer to its destination than before.*

Proof. Let $m(Y)$ be a message with destination $Y(y_1, y_2)$. Suppose that $m(Y)$ moves from a node $X(x_1, x_2)$ to another node $X' = (x_1', x_2')$ by the dimension i according to the routing algorithm; that is, $|x_i - y_i| = \max\{|x_j - y_j| : 1 \le j \le 2\}$ and $|x_i - y_i| - 1 = |x_i' - y_i|$. We claim that, in the following routings, except the special case that m moves along a quasi-Hamiltonian cycle of a cracky block, the routing of m will never augment the i-dimension distance.

To prove the claim, it is clear that by the routing algorithm, if m do not meet a cracky block, it cannot move from a node $Z = (z_1, z_2)$ to another one $Z' = (z_1', z_2')$ with $|z_i - y_i| = |x_i' - y_i| = |x_i - y_i| - 1 = |z_i' - y_i| - 1$. Suppose now that m moves from $U' = (u_1', u_2')$ to meet a node $U = (u_1, u_2)$, with $|u_i - y_i| \le |x_i - y_i| - 1$, of a cracky block $C = ((a_1, b_1), (a_2, b_2))$ by the dimension s and suppose that it leaves C by a node $W = (w_1, w_2)$ to a node $W' = (w_1', w_2')$ by the dimension t. According to the routing algorithm, without loss of generality, we may assume that the movement of m from X' to U' does not augment the s-dimension distance; that is, $|u_s' - y_s| \le |x_s' - y_s|$. By the routing function, we have

$$\begin{aligned} \left|w_i' - y_i\right| \le\ & |w_i - y_i| \le |w_t - y_t| \le |u_t - y_t| \le \left|u_t' - y_t\right| \\ & \le \left|u_s' - y_s\right| \le \left|x_s' - y_s\right| \le |x_s - y_s| \le |x_i - y_i|. \end{aligned} \quad (*)$$

Assume that $|w_i' - y_i| = |x_i - y_i|$. We have all equalities in $(*)$, in particular $|u_t - y_t| = |u_s' - y_s|$. For any l, $1 \le l \le 2$, it follows that $|u_l - y_l| \le |u_l' - y_l| \le |u_s' - y_s| = |u_t - y_t|$, which implies that m would leave C from the node U by the t-dimension and hence $W = U$. This gives

$$\left|w_i' - y_i\right| \le |w_i - y_i| = |u_i - y_i| \le |x_i - y_i| - 1, \quad (1)$$

a contradiction. Therefore we have $|w_i' - y_i| \le |x_i - y_i| - 1$ and the claim holds.

Consequently, if m leaves a cracky block C, it will never be sent back to C. □

Table 2: The routing table for border nodes of cracky blocks.

	v's border position is west	v's border position is east
$v_p = W(v)$	($v_s \leftarrow E(v)$ and $\text{succ}_1(v) \leftarrow N(v)$) if (a) $v_s \leftarrow N(v)$ if (b)	$v_s \leftarrow \text{succ}_1(v)$
$v_p = S(v)$	($v_s \leftarrow E(v)$ and $\text{succ}_1(v) \leftarrow N(v)$) if (a) $v_s \leftarrow N(v)$ if (b)	($v_s \leftarrow W(v)$ and $\text{succ}_1(v) \leftarrow N(v)$) if (a) $v_s \leftarrow N(v)$ if (b)
$v_p = E(v)$	$v_s \leftarrow \text{succ}_1(v)$	($v_s \leftarrow W(v)$ and $\text{succ}_1(v) \leftarrow S(v)$) if (a) $v_s \leftarrow S(v)$ if (b)
$v_p = N(v)$	($v_s \leftarrow E(v)$ and $\text{succ}_1(v) \leftarrow S(v)$) if (a) $v_s \leftarrow S(v)$ if (b)	($v_s \leftarrow W(v)$ and $\text{succ}_1(v) \leftarrow S(v)$) if (a) $v_s \leftarrow S(v)$ if (b)
	v's border position is south	v's border position is north
$v_p = W(v)$	($v_s \leftarrow N(v)$ and $\text{succ}_1(v) \leftarrow E(v)$) if (a) $v_s \leftarrow E(v)$ if (b)	($v_s \leftarrow S(v)$ and $\text{succ}_1(v) \leftarrow E(v)$) if (a) $v_s \leftarrow E(v)$ if (b)
$v_p = S(v)$	($v_s \leftarrow N(v)$ and $\text{succ}_1(v) \leftarrow W(v)$) if (a) $v_s \leftarrow W(v)$ if (b)	$v_s \leftarrow \text{succ}_1(v)$
$v_p = E(v)$	($v_s \leftarrow N(v)$ and $\text{succ}_1(v) \leftarrow W(v)$) if (a) $v_s \leftarrow W(v)$ if (b)	($v_s \leftarrow S(v)$ and $\text{succ}_1(v) \leftarrow W(v)$) if (a) $v_s \leftarrow W(v)$ if (b)
$v_p = N(v)$	$v_s \leftarrow \text{succ}_1(v)$	($v_s \leftarrow S(v)$ and $\text{succ}_1(v) \leftarrow E(v)$) if (a) $v_s \leftarrow E(v)$ if (b)

	v's border position is NE corner	v's border position is SE corner	v's border position is SW corner	v's border position is NW corner
$v_p = W(v)$	$v_s \leftarrow v_i$ if (c) $v_s \leftarrow S(v)$	$v_s \leftarrow v_i$ if (c) $v_s \leftarrow N(v)$	(d)	(d)
$v_p = S(v)$	$v_s \leftarrow v_i$ if (c) $v_s \leftarrow W(v)$	(d)	(d)	$v_s \leftarrow v_i$ if (c) $v_s \leftarrow E(v)$
$v_p = E(v)$	(d)	(d)	$v_s \leftarrow v_i$ if (c) $v_s \leftarrow N(v)$	$v_s \leftarrow v_i$ if (c) $v_s \leftarrow S(v)$
$v_p = N(v)$	(d)	$v_s \leftarrow v_i$ if (c) $v_s \leftarrow W(v)$	$v_s \leftarrow v_i$ if (c) $v_s \leftarrow E(v)$	(d)

(a): is not final.
(b): is final.
(c): let the destination node of the message be Y, if $\exists v_i \in \Gamma(v)$ and v_i's status is good, s.t. $d(v_i, Y) = d(v, Y) - 1$.
(d): using the basic routing function.

5. Concluding Remarks

In this paper, we propose a cracky rectangular fault block model for faulty-tolerant adaptive routing in two-dimensional mesh interconnection networks. This model improves the widely used rectangular model by taking into consideration the faulty links instead of faulty nodes in the process of constructing cracky blocks. It has been shown that we construct the spanning forest, which rooted with the border node, for all connected node in the cracky blocks. Thus the message can traverse all the nodes inside of the block by a kind of Depth-First-Search. As a result in the cracky block model, all faulty nodes that would have been useless now can be used for routing. Meanwhile the cracky block manages the size and scale in a self-adaptive mode; that is, the number or size of cracky block will gradually grow huge because of the increasing of faulty nodes/links, and contrarily, they will decrease for the fixed nodes/links. Based on the cracky block model, an algorithm is proposed to route message in the two-dimensional mesh without livelock. The novel strategy for fault-tolerant routing is faulty boundary independency, and it can apply the faulty occurring on the mesh boundary. The novel strategy for fault-tolerant routing improves the robustness and performance of two-dimensional mesh interconnection networks.

In the future, we will extend this strategy to multidimensional mesh networks, we have already testified that the construction method is suitable for multidimensional mesh networks, and then we will attempt to extend the routing algorithm to find a circuit on the multidimensional cracky blocks. In addition, we will add routing table to the cracky blocks to minimize the totally routing hops. These will come up in our next paper.

Conflict of Interests

The authors declare that there is no conflict of interests regarding the publication of this paper.

Acknowledgments

The authors would like to thank the Natural Science Foundation of P. R. of China (61300230), the Key Science and Technology Foundation of Gansu Province (1102FKDA010), Natural Science Foundation of Gansu Province (1107RJZA188),

and the Fundamental Research Funds for the Central Universities for supporting this research.

References

[1] M. N. Lionel and K. M. Philip, "A survey of wormhole routing techniques in direct networks," *Computer*, vol. 26, no. 2, pp. 62–76, 1993.

[2] C. L. Seitz, W. C. Athas, C. M. Flaig, A. J. Martin, J. Seizovic, and W. K. Su, "The architecture and programming of the ametek series 2010 multicomputer," in *Proceedings of the 3rd Conference on Hypercube Concurrent Computers and Applications*, pp. 33–37, 1988.

[3] Intel Corporation, *A Touchstone DELTA System Description*, 1991.

[4] S. L. Lillevik, "The touchstone 30 gigaflop DELTA prototype," in *Proceedings of the 6th Distributed Memory Computing Conference*, pp. 671–677, May 1991.

[5] P. T. Gaughan, B. V. Dao, S. Yalamanchili, and D. E. Schimmel, "Distributed, deadlock-free routing in faulty, pipelined, direct interconnection networks," *IEEE Transactions on Computers*, vol. 45, no. 6, pp. 651–665, 1996.

[6] Y. J. Suh, B. V. Dao, J. Duato, and S. Yalamanchili, "Software based fault-tolerant oblivious routing in pipelined networks," in *Proceedings of the International Conference on Parallel Processing*, pp. I101–I105, 1995.

[7] G. M. Chiu and S. P. Wu, "A fault-tolerant routing strategy in hypercube multicomputers," *IEEE Transactions on Computers*, vol. 45, no. 2, pp. 143–155, 1996.

[8] T. C. Lee and J. P. Hayes, "A fault-tolerant communication scheme for hypercube computers," *IEEE Transactions on Computers*, vol. 41, no. 10, pp. 1242–1256, 1992.

[9] A. C. Liang, S. Bhattacharya, and W. T. Tsai, "Fault-tolerant multicasting on hypercubes," *Journal of Parallel and Distributed Computing*, vol. 23, no. 3, pp. 418–428, 1994.

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

[11] C. C. Su and K. G. Shin, "Adaptive fault-tolerant deadlock-free routing in meshes and hypercubes," *IEEE Transactions on Computers*, vol. 45, no. 6, pp. 666–683, 1996.

[12] C. Glass and L. Ni, "Maximally fully adaptive routing in 2D meshes," in *Proceedings of the International Conference on Parallel Processing*, vol. 1, pp. 101–104, August 1992.

[13] Y. M. Boura and C. R. Das, "Fault-tolerant routing in mesh networks," in *Proceedings of the International Conference on Parallel Processing*, pp. I106–I109, 1995.

[14] R. Libeskind-Hadas and E. Brandt, "Origin-based fault-tolerant routing in the mesh," in *Proceedings of the 1st International Symposium on High Performance Computer Architecture*, pp. 102–111, 1995.

[15] J. Wu, "Fault-tolerant adaptive and minimal routing in mesh-connected multicomputers using extended safety levels," *IEEE Transactions on Parallel and Distributed Systems*, vol. 11, no. 2, pp. 149–159, 2000.

[16] D. Xiang, J.-G. Sun, J. Wu, and K. Thulasiraman, "Fault-tolerant routing in meshes/tori using planarly constructed fault blocks," in *Proceedings of the International Conference on Parallel Processing*, pp. 577–584, June 2005.

[17] W. Chou, A. W. Bragg, and A. A. Nilsson, "The need for adaptive routing in the chaotic and unbalanced traffic environment," *IEEE transactions on communications systems*, vol. 29, no. 4, pp. 481–490, 1981.

[18] A. A. Chien and J. H. Kim, "Planar-adaptive routing: low-cost adaptive networks for multiprocessors," in *Proceedings of the19th International Symposium on Computer Architecture*, pp. 268–277, May 1992.

[19] M. Ebrahimi, M. Daneshtalab, J. Plosila, and F. Mehdipour, "MD: minimal path-based fault-tolerant routing in on-chip networks," in *Proceedings of the 18th Asia and South Pacific Design Automation Conference (ASP-DAC '13)*, pp. 35–40, Yokohama, Japan, January 2013.

[20] M. Ebrahimi, M. Daneshtalab, and J. Plosila, "High Performance Fault-Tolerant Routing Algorithm for NoC-Based Many-Core Systems," in *Proceedings of the 21st Euromicro International Conference on Parallel, Distributed and Network-Based Processing (PDP '13)*, pp. 462–469, Belfast, UK, February-March 2013.

[21] M. Ebrahimi, H. Tenhunen, and M. Dehyadegari, "Fuzzy-based adaptive routing algorithm for networkson-chip," *Journal of Systems Architecture*, vol. 59, no. 7, pp. 516–527, 2013.

[22] S. S. Alamian, R. Fallahzadeh, S. Hessabi, and J. Alirezaie, "A novel test strategy and fault-tolerant routing algorithm for NoC routers," in *Proceedings of the 17th CSI International Symposium on Computer Architecture and Digital Systems (CADS '13)*, pp. 133–136, Tehran, Iran, October 2013.

[23] X. Lin and L. M. Ni, "Deadlock-free multicast wormhole routing in multicomputer networks," in *Proceedings of the 18th International Symposium on Computer Architecture*, pp. 116–125, May 1991.

[24] W. J. Dally and C. L. Seitz, "Deadlock-free message routing in multiprocessor interconnection networks," *IEEE Transactions on Computers*, vol. 36, no. 5, pp. 547–553, 1987.

[25] D. Wang, "A rectilinear-monotone polygonal fault block model for fault-tolerant minimal routing in mesh," *IEEE Transactions on Computers*, vol. 52, no. 3, pp. 310–320, 2003.