On Constructing the Minimum Orthogonal Convex Polygon for the Fault-Tolerant Routing in 2-D Faulty Meshes

Jie Wu, *Senior Member, IEEE,* and Zhen Jiang, *Member, IEEE*

***Abstract*—The rectangular faulty block model is the most commonly used fault model for designing fault-tolerant, and deadlock-free routing algorithms in mesh-connected multicomputers. The convexity of a rectangle facilitates simple, efficient ways to route messages around fault regions using relatively few or no virtual channels to avoid deadlock. However, such a faulty block may include many nonfaulty nodes which are disabled, i.e., they are not involved in the routing process. Therefore, it is important to define a fault region that is convex, and at the same time, to include a minimum number of nonfaulty nodes. In this paper, we propose an optimal solution that can quickly construct a set of minimum faulty polygons, called *orthogonal convex polygons*, from a given set of faulty blocks in a 2-D mesh (or 2-D torus). The formation of orthogonal convex polygons is implemented using either a centralized, or distributed solution. Both solutions are based on the formation of faulty components, each of which consists of adjacent faulty nodes only, followed by the addition of a minimum number of nonfaulty nodes to make each component a convex polygon. Extensive simulation has been done to determine the number of nonfaulty nodes included in the polygon, and the result obtained is compared with the best existing known result. Results show that the proposed approach can not only find a set of minimum faulty polygons, but also does so quickly in terms of the number of rounds in the distributed solution.**

***Index Terms*—Fault model, fault tolerance, orthogonal convex polygons, routing, 2-dimensional meshes (tori).**

ACRONYMS[1]

2-D meshes	2-dimensional meshes
EW (WE)	East-to-West (West-to-East)
NS (SN)	North-to-South (South-to-North)
FB	Faulty Block
FP	Faulty Polygon
MFP	Minimum Faulty Polygon
CMFP	Centralized Solution of Minimum Faulty Polygon
DMFP	Distributed Solution of Minimum Faulty Polygon

Manuscript received September 17, 2004. This work was supported in part by NSF grants CCR 0329741, CNS 0422762, CNS 0434533, ANI 0073736, and EIA 0130806. Associate Editor: M. Xie.

J. Wu is with the Department of Computer Science and Engineering, Florida Atlantic University, Boca Raton, FL 33431 USA (e-mail: jie@cse.fau.edu).

Z. Jiang is with the Department of Computer Science, Information Assurance Center, West Chester University, West Chester, PA 19383 USA.

Digital Object Identifier 10.1109/TR.2005.853039

[1]The singular and plural of an acronym are always spelled the same.

NOTATION

u, v	Nodes u, and v
ID	Node Identifier with an Encoded Address (x, y)
(E, S, W, N)	Boundary Entry with East, South, West, and North Boundary Node Information
$V[1 \ldots n]$	Boundary Array
$\min_x(C)$	Minimum Coordinate of Component C along X Dimension
$\max_x(C)$	Maximum Coordinate of Component C along X Dimension
$\min_y(C)$	Minimum Coordinate of Component C along Y Dimension
$\max_y(C)$	Maximum Coordinate of Component C along Y Dimension

I. INTRODUCTION

MESH-CONNECTED multicomputers, especially those with low-degree, are among the simplest, and least expensive structures for building a system using hundreds, and even thousands of processors. Low-degree mesh-connected multicomputers include meshes, and tori (which are meshes with wraparound connections). Dally [5], and Agarwal [1] recommended the use of low-degree networks, such as 2-dimensional (2-D), and 3-dimensional (3-D) meshes (tori); over high-degree networks, such as hypercubes, for better performance & cost-effectiveness. In a mesh-connected multicomputer, processors (also called nodes) exchange data, and coordinate their efforts by sending & receiving messages through the underlying mesh network. Thus, the performance of such a system depends heavily on the end-to-end cost of communication mechanisms. Routing is the process of transmitting data from one node to another node in a given system. As the number of nodes in a mesh-connected multicomputer increases, the chance of failure also increases. At the same time, applications that run on such a system are often critical, and may have real-time constraints. Therefore, the ability to tolerate failure is becoming increasingly important, especially for routing [6]. In this paper, we focus on 2-D meshes (tori), and in the subsequent discussion, we use meshes to represent both meshes & tori.

In designing a fault-tolerant routing algorithm, one of the most important issues is to select an appropriate fault model. A good fault model should accurately reflect fault situations in a real system. It should be defined in such a way that fault information can be easily established & maintained. The model

should not be too complex, and at the same time, not overly simplified so that certain objectives, such as optimization & deadlock-free requirements, are still obtainable. Most of the literature on fault-tolerant routing uses disjoint rectangular blocks [2]–[4], [7], [11], [20], [21] to model node faults (link faults can be treated as node faults), and to facilitate routing in 2-D meshes. First, a node labeling scheme that identifies nodes (faulty & nonfaulty) that cause routing difficulties is defined, and such nodes are called *unsafe nodes*. Connected unsafe nodes form a faulty rectangular region. Such a region is also called a rectangular faulty block, or simply a *faulty block*. A faulty block can be easily established & maintained through message exchanges among neighboring nodes.

The convexity of each faulty block facilitates a simple fault-tolerant & deadlock-free routing using either relatively few virtual channels [4], [11], [13] or no virtual channel [18]. The convex feature of a faulty block is a necessary condition for progressive routing, where the routing process never backtracks. The absence of backtracking in turn is a necessary condition for minimal routing, also called shortest path routing [9], [19], where the destination is reached through a minimal path from the source. The fault-tolerant minimal routing algorithm for 2-D meshes has been developed in [17] using the faulty block model. There are several studies on fault-tolerant routing that can handle nonrectangular fault regions, such as H-shape, L-shape, T-shape, U-shape, and $+$-shape fault regions [3], [12], [14].

Despite all the desirable features of the faulty block model, a major problem is that a faulty block may include many nonfaulty nodes treated as faulty nodes (with the unsafe label). A nonfaulty node in the faulty block will not be used in any routing process. This leads to the loss of the communication ability of such a node. In the worst case, a majority of nodes in a faulty block are nonfaulty. Although some efforts have been made either to enhance the faulty block definition to include fewer nonfaulty nodes in a faulty block [10], or to activate some boundary nonfaulty nodes in a faulty block as in [2], [11], the above problem still exists.

In this paper, we try to solve the above problem by providing an approach to find a set of disjoint convex polygons that cover all the faulty nodes with a minimum number of nonfaulty nodes included in these polygons. In this way, the fault situation in the networks can be described accurately in this new fault model without including any redundant nonfaulty node. It is the fundamental block of providing a fault-tolerant routing so that a robust network communication can be obtained at the most degree without extra loss of communication ability of nodes in a situation when some faults occur in the networks.

A convex region (polygon) is defined as a region (polygon) P for which a line segment connecting any two points in P lies entirely within P. If we change the "line segment" in the standard convex region definition to "horizontal or vertical line segment", the resultant region is called an *orthogonal convex region (polygon)* [8], [16]. Clearly, a faulty block is a special orthogonal convex region. In 2-D meshes, the boundary lines of a region are either horizontal or vertical; therefore, each region is a polygon. In the subsequent discussion, we use the terms polygon & region interchangeably.

The challenge here is not only to conduct a theoretical study on the feasibility of finding such a smallest orthogonal convex polygon, but also to search for a practical, efficient distributed implementation in a system where each processor knows only the status of its neighbors. In [16], a simple & efficient distributed algorithm is presented which determines a set of small orthogonal convex polygons to cover all the faults in a given faulty block. This approach consists of two phases. First, a set of disjoint faulty blocks are constructed from a given set of faulty nodes. Each faulty block may contain many nonfaulty nodes. Then, each faulty block is partitioned into one or many disjoint orthogonal convex polygons by removing some nonfaulty nodes from the block. It is shown in [16] that a resultant region generated after removing nonfaulty nodes from the given faulty block is the minimum orthogonal convex polygon (simply called a *faulty polygon*) that covers all the faults in the region. Note that there may be several polygons generated from a given faulty block. In addition, it is shown that the number of nonfaulty nodes covered in these faulty polygons (from a given faulty block) is no more than that of the minimum orthogonal convex polygon that includes all the faulty nodes in the original faulty block. However, for certain cases, a faulty polygon can be further partitioned, and more nonfaulty nodes in the region can be removed. Therefore, such a faulty polygon is called a *sub-minimum faulty polygon*. This brings the following open problem: For a given faulty block, find a set of disjoint orthogonal convex polygons that covers all the faults in the faulty block, and contains a minimum number of nonfaulty nodes. Clearly, each faulty polygon is minimum (called a *minimum faulty polygon*) in the sense that it cannot be replaced by a set of faulty polygons that include fewer nonfaulty nodes.

In this paper, we first provide a centralized solution for the open problem, and then an efficient distributed solution is provided. This approach again consists of two phases. In the first phase, a set of components are formed, where each *component* consists of adjacent faulty nodes with no nonfaulty node. In the second phase, a minimum number of nonfaulty nodes are added to make each component a convex polygon. Extensive simulation has been done to determine the number of nonfaulty nodes included in the polygon, and the result obtained is compared with the best existing known result.

This paper is organized as follows. In Section II we provide some preliminaries including the definition of an orthogonal convex polygon, its application in fault-tolerant & deadlock-free routing in 2-D meshes, and a distributed formation of a sub-minimum faulty polygon. In Section III, we propose an optimal solution for constructing minimum faulty polygons. An efficient distributed solution is also provided. In Section IV, simulation results on the average number of rounds needed to construct minimum faulty polygons, and the number of nonfaulty nodes in the polygons are given. Finally, in Section V, we provide our conclusions & ideas for future work.

II. Preliminaries

A. Orthogonal Convex Polygons

We consider only node faults, and assume that faulty nodes just cease to work. Also, each nonfaulty node knows the status

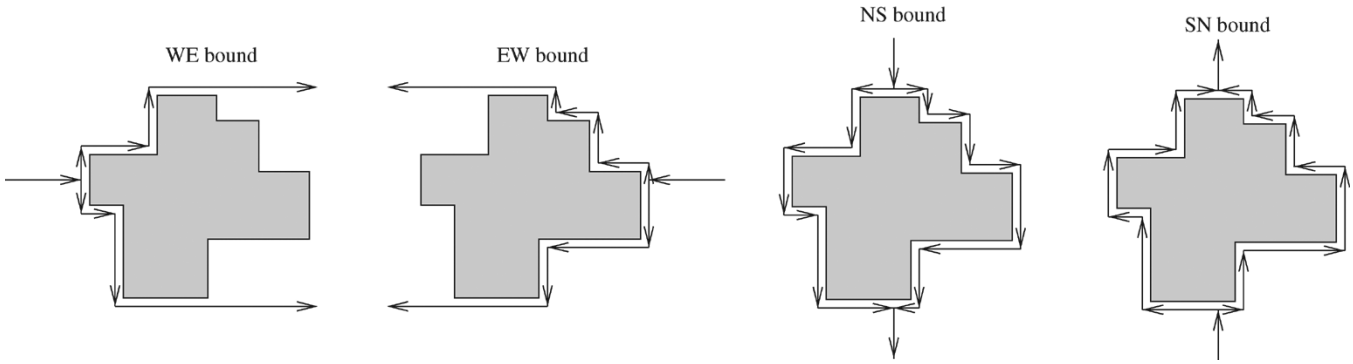

Fig. 1. Routing of messages around disabled regions.

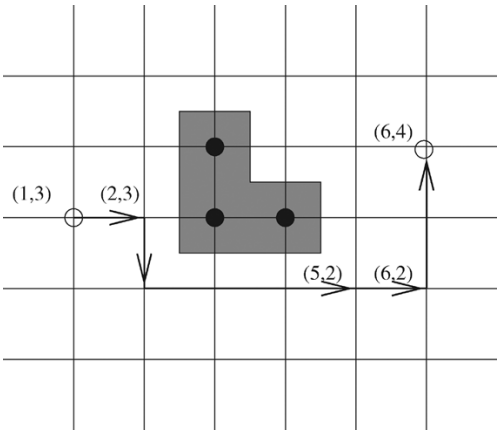

Fig. 2. A routing example from source (1,3) to destination (6,4).

of its neighbors only; that is, there is no priori global information of fault distribution. A 2-dimensional (2-D) $n \times n$ mesh with n^2 nodes has an interior node degree of 4, and a network diameter of $2(n-1)$. Each node u has an address (u_x, u_y), where $u_x, u_y \in \{0, 1, \ldots, n-1\}$. Two nodes u: (u_x, u_y), and v: (v_x, v_y) are connected if their addresses differ in one, and only one dimension, say dimension x. Moreover, $|u_x - v_x| = 1$. Similarly, if they differ in dimension y, then $|u_y - v_y| = 1$.

In the following, we give the definition of an orthogonal convex region.

Definition 1 [16]: A fault region is *orthogonal convex* iff the following condition holds: For any horizontal or vertical line, if two nodes on the line are inside the region, all the nodes on the line that are between these two nodes are also inside the region.

The difference between a standard convex region, and an orthogonal convex region is that the line in the latter is restricted to only horizontal & vertical, whereas the line in the former can be along any direction in a standard convex region. In 2-D meshes, each orthogonal convex region is also an orthogonal convex polygon. Clearly, T-shape, L-shape, and +-shape (the shadowed region in Fig. 1) fault regions are orthogonal convex polygons; whereas U-shape, and H-shape fault regions are nonorthogonal convex polygons. For example, the orthogonal convex region {(2,4), (3,4), (4,3)} is an L-shape polygon (the shadowed region in Fig. 2).

Note that an orthogonal convex polygon is the same as a solid fault in the solid fault model [3]. It is shown in [3] that only four virtual channels are needed around a convex region to support fault-tolerant & deadlock-free routing. However, finding an algorithm for computing a smallest solid fault was not included in [3]. Wang [15] independently proposed a routing-sensitive convex region where each convex region has two versions (shapes): one for west-east routing, and the other for east-west routing.

B. Fault-Tolerant & Deadlock-Free Routing

We first show the application of orthogonal convex polygon in achieving fault-tolerant & deadlock-free routing in 2-D meshes. It is assumed that faults in a given 2-D mesh are contained in a set of disjoint faulty polygons. Chalasani & Boppana's *extended e-cube routing* [3] is used to illustrate the routing process. The *e-cube routing*, also called $x-y$ routing, sends a message in a row (x-direction) until the message reaches a node that is in the same column as its destination, and then sends the message in the column (y-direction). Consider a routing process from node (1,3) to node (6,4); the message is first routed along the row to node (6,3), and then the message follows the column to reach destination (6,4). The extended e-cube routing follows the base e-cube routing until it hits a faulty polygon, and then sends the message around the region (the message is in an "abnormal" mode), clockwise or counterclockwise based on a set of rules (to be discussed shortly), until it becomes "normal" again, i.e., the region no longer has an effect on the routing process.

Messages are classified into one of four types: EW, WE, NS, or SN. A message is initially labeled as EW-bound, or WE-bound, depending on its direction of travel along the row. Once a message completes its row hops, it becomes a NS-bound, or SN-bound message. For the routing example from node (1,3) to node (6,4), the message is WE-bound initially. The message becomes SN-bound once it reaches node (6,3). Once a message hits a faulty polygon, it has to route around the region. Fig. 1 shows ways to route around a faulty polygon for different types of messages. The orientation (clockwise or counterclockwise) is inconsequential for an NS- or SN-bound message. The orientation for a WE-bound (EW-bound) message is the following: clockwise (counterclockwise) if the message is in a row above its row of travel, counterclockwise (clockwise) if the message is in a row below its row of travel, and inconsequential (don't care) if the message is in the same row of travel.

Consider again the routing process from (1,3) to (6,4). The message is routed along the row toward the east, and it is

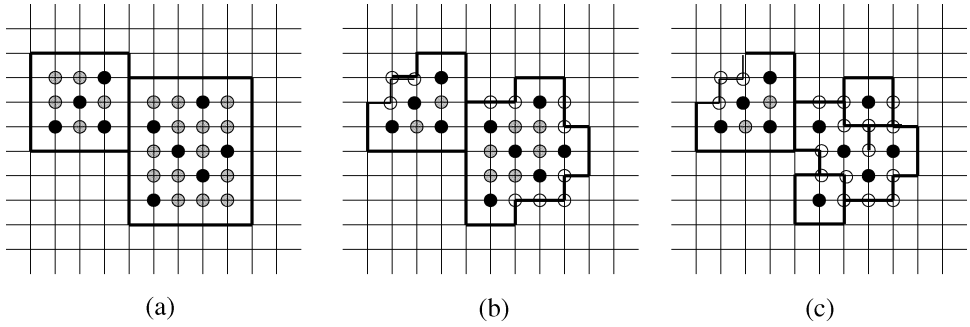

Fig. 3. (a) Rectangular faulty blocks, (b) sub-minimum faulty polygon, and (c) minimum faulty polygon.

WE-bound. Suppose there is a faulty polygon {(2,4), (3,4), (4,3)} (see Fig. 2). Once it reaches the boundary of the faulty polygon at node (2,3), the message is routed along the faulty polygon. Because node (2,3) is in a row that is below the row of travel (or the row of the destination), the message is routed around the faulty polygon counterclockwise as shown in Fig. 2. Once the message reaches node (5,2), it becomes "normal" again. The rest of the routing process follows the base e-cube routing along the row to node (6,2), and then along the column to reach node (6,4).

To ensure freedom from deadlock & livelock, four virtual channels, vc_0, vc_1, vc_2, vc_3, are used for channels around faulty polygons. Note that more virtual channels are needed if the region is concave. The use of these virtual channels is as follows: EW-bound messages use vc_0 for all hops around faulty polygons, WE-bound messages use vc_1, NS-bound messages uses vc_2, and SN-bound messages use vc_3. Other details related to the extended e-cube routing can be found in [3].

C. Distributed Formation of Sub-Minimum Faulty Polygon

Wu's sub-minimum faulty polygon construction [16] is based on two labeling schemes.

- **Labeling scheme 1**: All faulty nodes are *unsafe*, and all nonfaulty nodes are *safe* initially. A nonfaulty node is changed to *unsafe* if it has a faulty or unsafe neighbor in both dimensions; otherwise, it remains *safe*.
- **Labeling scheme 2**: All faulty nodes are marked *disabled*. All safe nodes are marked *enabled*. An unsafe node is initially marked *disabled*, but it is changed to *enabled* if it has two or more enabled neighbors.

Based on the above labeling schemes, a faulty node must be unsafe & disabled. For a nonfaulty node, there are three possible cases: (1) safe & enabled, (2) unsafe & enabled, and (3) unsafe & disabled. The labeling scheme 1 corresponds to a *growing phase* that includes nonfaulty nodes in the block, and as a result, rectangular faulty blocks are generated. The labeling scheme 2 corresponds to a *shrinking phase* that removes nonfaulty nodes from rectangular faulty blocks.

Fig. 3(a) shows rectangular faulty blocks generated from ten faulty nodes (black nodes). Gray nodes are unsafe nodes. Other safe nonfaulty nodes are not shown. Fig. 3(b) shows two orthogonal convex polygons generated from Fig. 3(a). Gray nodes are unsafe, and disabled nodes (i.e., nonfaulty nodes contained in the polygon); and white nodes are unsafe but enabled nodes (i.e., nonfaulty nodes that are originally in rectangular faulty block, but are removed from the orthogonal convex polygon). Clearly, the right polygon in Fig. 3(b) is not minimum, and it can be further partitioned into three polygons as shown in Fig. 3(c).

Wu [16] shows that under the condition that each polygon cannot be further partitioned (such as the left polygon in Fig. 3(b)), the polygon is minimum. That is, the polygon contains the minimum number of nonfaulty nodes unless the original faulty block can be further partitioned.

III. Minimum Orthogonal Convex Polygon

We start with a centralized solution for determining the minimum faulty polygons, followed by a distributed solution. Both solutions consist of two phases. First, faulty nodes are grouped into a set of components, where faulty nodes in each component are adjacent. Then, a minimum number of nonfaulty nodes are included to make each component a polygon.

Minimum Orthogonal Convex Polygons. (Centralized Solution based on Labeling Schemes 1 & 2.)

1. Construct components using the merge process. Each component C maintains minimum, and maximum coordinates along X, and Y dimensions: $\min_x(C)$, $\min_y(C)$, $\max_x(C)$, and $\max_y(C)$.
2. Apply labeling scheme 1 to each component C to generate a virtual faulty block. Basically, all nodes inside the boundary of $X : [\min_x(C), \max_y(C)]$, and $Y : [\min_y(C), \max_y(C)]$ other than those of C are treated as nonfaulty nodes, and then become unsafe after the labeling process.
3. Apply labeling scheme 2 to enable some nonfaulty but unsafe nodes in the virtual faulty block.
4. Use the superseding rule to resolve conflicting node status for final status of each nonfaulty node.

A. Centralized Solutions

In the centralized solution, first we define the notion of *node adjacency*.

Definition 2: For a given node (x, y), the adjacent nodes of (x, y) are $(x-1, y-1)$, $(x-1, y)$, $(x-1, y+1)$, $(x, y-1)$, $(x, y+1)$, $(x+1, y-1)$, $(x+1, y)$, and $(x+1, y+1)$.

In phase 1, a *merge process* is used that groups faulty nodes into components, where each component consists of adjacent faulty nodes only. Clearly, a sub-minimum faulty polygon may include multiple components. In addition, for each component, four coordinates, $\min_x$, $\min_y$, $\max_x$, and $\max_y$, representing

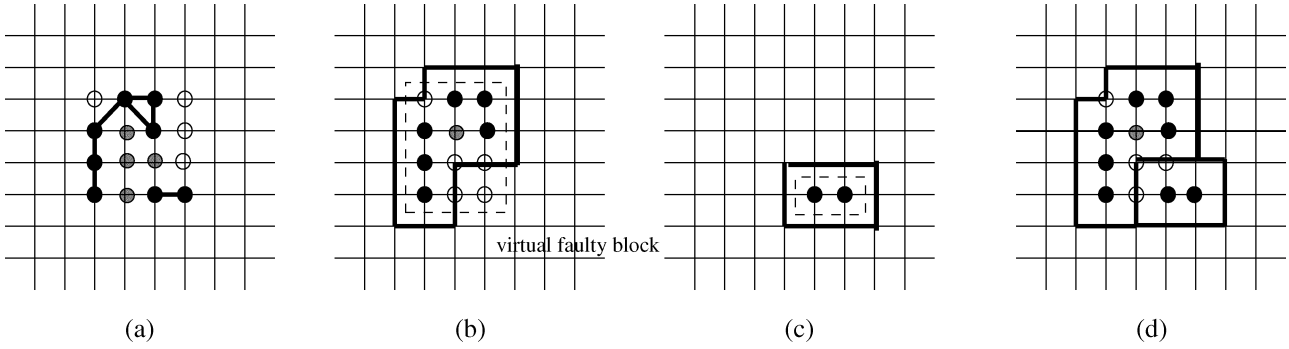

Fig. 4. Centralized solution: (a) components, (b) & (c) faulty polygons, and (d) "piling" all faulty polygons.

the minimum, and maximum coordinates of any nodes in this component along X, and Y dimensions, are maintained in a merge process.

In phase 2, there are two possible solutions. One is based on applying labeling schemes 1 & 2 to each component. The other is based on the definition of orthogonal convex polygon to "fill in" each concave region of a component with nonfaulty nodes.

In the first solution of phase 2, for each component, a *virtual faulty block* is constructed by applying the labeling scheme 1. In the virtual faulty block, four corners are $(\min_x, \min_y)$, $(\min_x, \max_y)$, $(\max_x, \min_y)$, and $(\max_x, \max_y)$. Such a rectangle can be simply represented by two opposite corners $[(\min_x, \min_y), (\max_x, \max_y)]$. Applying labeling scheme 2, minimum faulty polygons are derived, one for each virtual faulty block.

Fig. 4 shows an example of two components, and the corresponding virtual faulty blocks. In Fig. 4(a), white nodes are unsafe but enabled nodes in the sub-minimum faulty polygon. In Fig. 4(b) & (c), white nodes are unsafe but enabled nodes in the virtual faulty block, but removed from the minimum faulty polygon. Gray nodes are unsafe & disabled nodes. Virtual faulty blocks (Figs. 4(b) & (c)) are encircled by rectangles with dashed Lines, and minimum faulty polygons are represented by rectangles with thick lines. The final diagram (Fig. 4(d)) is constructed by "piling" all the diagrams on top of each other, with the following superseding rule for node status:

- **Superseding rule**: black nodes overwrite gray & white nodes, and gray nodes overwrite white nodes.

Note that using the sub-minimum faulty polygon construction, two components in Fig. 4 will be placed in one faulty block after the labeling scheme 1. Also, the white node on the north-west corner will be enabled. The resultant polygon, shown in Fig. 4(a), still includes two components plus one gray node, and three white nodes in Fig. 4(d); and it is not minimum.

Theorem: Each polygon P derived based on the proposed process is a minimum faulty polygon.

Proof: We need only to prove that there exists no set of disjoint convex polygons that covers all the faulty nodes in polygon P, and that in addition, the number of nonfaulty nodes included in these polygons is less than that of P.

Suppose P is constructed out of k components $C_1, C_2, \ldots, C_k$, and the corresponding polygons are $P_1, P_2, \ldots, P_k$. Based on the definition of component, each P_i, $1 \leq i \leq k$, cannot be partitioned. Let $P'_1, P'_2, \ldots, P'_m$ be any set of disjoint faulty polygons that covers all faults in P. Then for each P_i, there is a P'_j, $1 \leq j \leq m$, that covers all faults in P_i. Because P_i is derived from labeling schemes 1 & 2 on C_i, and it cannot be further partitioned (see in Theorem 2 in [16]), P'_j includes also all nonfaulty nodes in P_i. Therefore, $P : P_1, P_2, \ldots P_i$ contains no more nonfaulty nodes than those of $P'_1, P'_2, \ldots P'_m$. ■

The first solution is based on emulating labeling scheme 1, growing each component into a virtual faulty block; and labeling scheme 2, shrinking each virtual faulty block into a minimum faulty polygon. In the second solution, nonfaulty nodes are directly added to the concave region of a component to convert it to a convex polygon. The correctness of this approach is clear because it is equivalent to the labeling schemes 1 & 2 applied on each component. We first define the notions of *concave row section*, and *concave column section*.

Definition 3: Given a component, for a horizontal (vertical line) where two end nodes on the line are inside the component, each section of the line that is outside the component is called a concave row (column) section.

Minimum Orthogonal Convex Polygons (Centralized Solution based on Concave Row & Column Sections.)

1. Construct components using the merge process. Each component C maintains minimum, and maximum coordinates along X, and Y dimensions: $\min_x(C)$, $\min_y(C)$, $\max_x(C)$, and $\max_y(C)$.
2. Identify all concave row, and column sections by scanning each component twice, one horizontally from $\min_x$ to $\max_x$, and one vertically from $\min_y$ to $\max_y$.
3. Assign disable status to all nodes in concave row, and column sections.
4. Use the superseding rule to resolve conflicting node status for final status of each node.

To find minimum faulty polygons, we only need to disable all nodes on the concave row (column) section. To identify all concave row (column) sections, we need to "scan" each component twice, horizontal (from $\min_x$ to $\max_x$), and vertical (from $\min_y$ to $\max_y$). Each nonfaulty node is enabled by default, and the final status of each nonfaulty node is decided based on the same superseding rule discussed in the last subsection.

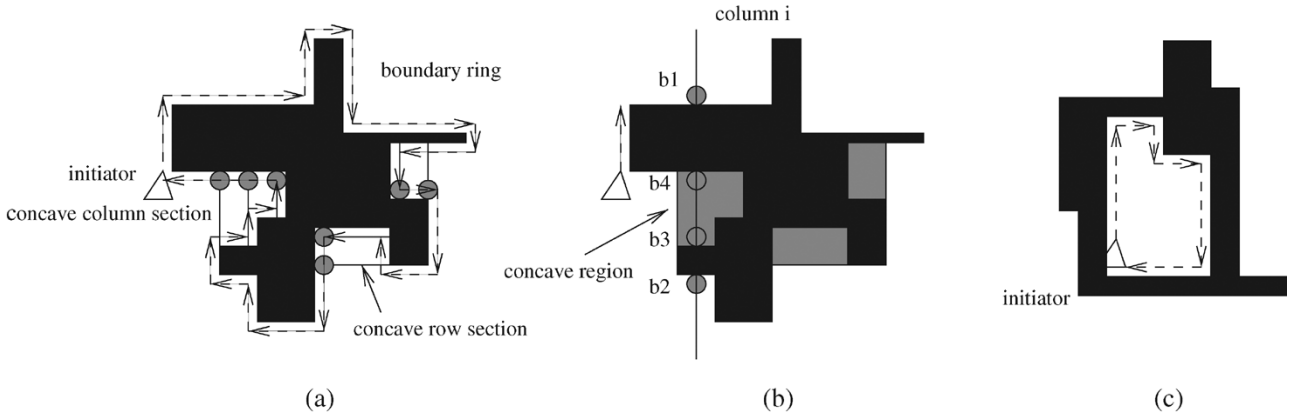

Fig. 5. Distributed solution for (a), and (b) open concave region; and for (c) closed concave region.

B. Distributed Solutions

The distributed solution is more involved. The first challenge is the construction of a component. This is done by *boundary nodes* of a component, i.e., nodes outside any faulty components, but adjacent to this component. If a boundary node is at the north-side of a component, it is called a *north boundary* node with respect to the component. Other adjacent nodes toward south, east, and west are defined in a similar way. Note that each node may have multiple boundary status (e.g., a node can be a south, and north boundary node if it is at the south, and north of the component). Boundary nodes form a "ring" surrounding the faulty component. A boundary node is the *south-west outer corner* (see the triangle node in Fig. 5(a)) if it has a west boundary neighbor, and a south boundary neighbor. A boundary node is the *south-west inner corner* (see the triangle node in Fig. 5(c)) if it is an east, and a north boundary node at the same time. Note that the south-west outer corner belongs to the boundary ring, but it is not an east, south, west, or north boundary node. Once the boundary ring is constructed, it is assumed that each node knows the next boundary node, along the clockwise direction.

We assume that both the west-most south-west outer corner, and the west-most south-west inner corner (if any) initiate the ring construction process, and these corners are also called the *initiators*. The initiation message goes around the faulty component in a clockwise direction. See Fig. 5, where an initiator is represented by a triangle. The situation for multiple initiations will be discussed later.

The next challenge is the distributed method of determining the status of each nonfaulty node. Because a concave row/column section may overlap with another faulty component, as will be seen later with an example, the centralized approach based on labeling schemes 1 & 2 cannot be directly applied. A disabled node in a concave row/column section of one faulty component can be a faulty node in another faulty component. Therefore, such a node cannot participate in neighbor information exchange, as required in labeling schemes 1 & 2. Instead of involving all nodes in the concave row & column section, we use only boundary nodes that are end nodes of concave row/column sections, because a boundary node cannot be a faulty node in any faulty component. Each concave row/column section is defined during the boundary ring construction. The following information needs to be piggybacked with the initiation message:

- Initiator ID (x, y)
- Boundary 2-D array $V[1 \ldots n](E, S, W, N)$

Initiator ID is used as the message identification, and it is also used to determine when the message should terminate, i.e., when a node receives a message with its own identification. Boundary array ($n \times 4$ 2-D array) is used to determine concave row & column sections, where E, S, W, N are used to store east, south, west, and north boundary node information. This array includes *row number* for the most recently visited north & south boundary node at each column, and *column number* for the most recently visited east & west boundary node at each row. Basically, in a given $n \times n$ 2-D mesh, one entry is used for south (north) boundary nodes on each row (from row 1 to row n), and one entry is used for east (north) boundary nodes on each column (from column 1 to column n). Initially, all entries in the boundary array are initialized to "$-$" (undefined). It should be stressed that during the ring formation, multiple boundary nodes of the same type (east, south, west, or north) may appear. There is only one entry for each type per row (column). During the ring formation of the example in Fig. 5(b), there are four boundary nodes, b_1 (north), b_2 (south), b_3 (north), and b_4 (south), in column i. $V[i](S)$ stores the row number of b_1, and later the row number of b_3. Similarly, $V[i](N)$ stores the row number of b_2, which is later substituted by the row number of b_4.

In the following algorithm, one end node of each concave row (column) section is selected to be in charge of the notification of disable status to all nodes in the corresponding section (see gray nodes in Fig. 6). Such an end node is also called *notification end node*, which is responsible for notification, and maintains the positions of two end nodes of the corresponding concave section.

Concave Row & Column Section. (Distributed Solution)

1. Upon receiving the initiation message, the current node does the following if it is an east, south, west, or north boundary node:
   - Update the corresponding entry $V[1 \ldots n](E, S, W, N)$ based on the type of

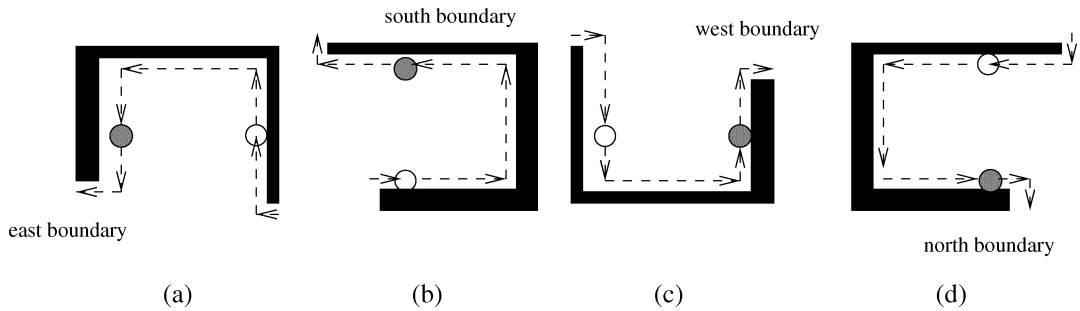

Fig. 6. Four different cases of concave sections: (a) & (c) concave row section, and (b) & (d) concave column section.

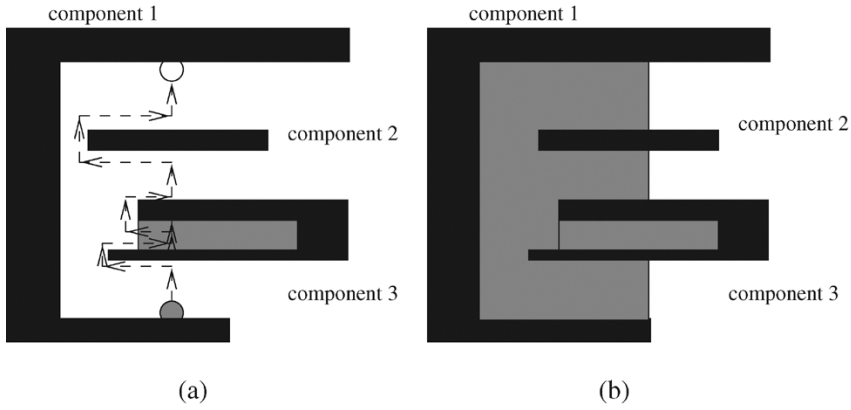

Fig. 7. Notification in a concave column section with blocking polygons.

boundary node, and the current node position. In the case of multiple boundary status, multiple entries are updated.

- The current node is a notification end node for concave row or column section.
  if the current node is an east (west) boundary, and the west (east) boundary of the same row has a record with no smaller (no larger) a column number [see Figs. 6(a) & (c)], or
  if the current node is a south (north) boundary, and the north (south) boundary of the same column has a record with no smaller (no larger) a row number [see Figs. 6(b) & (d)].

- If the current node is a notification end node, it records the positions (stored in V) of two end nodes of the corresponding concave row (column) section.

2. Pass on the initiation message to the next node in the boundary ring with the updated boundary array.

In the example of Fig. 5(b), the initiation message visits node b_1 first, and stores its row information in $V[i](N)$, which is initiated to "$-$". When b_2 is visited, its row information is stored in $V[i](S)$ (which is also initiated to "$-$"). Because the row number of b_2 is smaller than the row number of b_1 (i.e., $V[i](S) < V[i](N)$), no action is needed. When b_3 is visited, its row information is stored in $V[i](N)$, i.e., the row number of b_1 is replaced. Again, no action is needed because $V[i](S) < V[i](N)$. Finally, when b_4 is visited, its row information is stored in $V[i](S)$. Because the row number of b_4 is no smaller than the row number of b_3 (i.e., $V[i](S) \geq V[i](N)$), b_4 is the notification end node for a concave section in column i. The row positions of two end nodes are recorded in $V[i](S)$, and $V[i](N)$.

Various optimizations are possible on reducing memory space by combining $V[i](S)$ with $V[i](N)$, and $V[i](E)$ with $V[i](W)$; and removing redundant portions in different concave sections by also holding the second most recently visited boundary node information. Details are more involved, and are skipped here.

One more challenge is that a concave region (consisting of concave rows, and/or columns) may actually contain faults (it is fault-free only with respect to the current component), and the corresponding faulty polygons are called *blocking polygons*. In the absence of blocking polygons, forwarding status along the concave row (column) section is straightforward along a row (column); otherwise, the message that carries node status must route around blocking polygons. Fig. 7(a) shows an example where a concave column section in a concave region (component 1) has to bypass two blocking polygons, components 2 & 3. Note that, when routing around blocking polygons, node status for portions (of the section) that also belong to concave regions of blocking polygons (such as that of component 3 in Fig. 7(a)) is determined multiple times (i.e., colored twice in Fig. 7(a), one by component 1, and one by component 3). The end result of the concave region is shown in Fig. 7(b).

In a component, there are many south-west corners (inner or outer). It is possible that many or all of them can initiate the process. Therefore, the procedure starts whenever a new south-west corner is formed, and we have the following overwriting rule in the priority order of (a) & (b): When a node receives more than one initiation message, (a) the one with a smaller x value

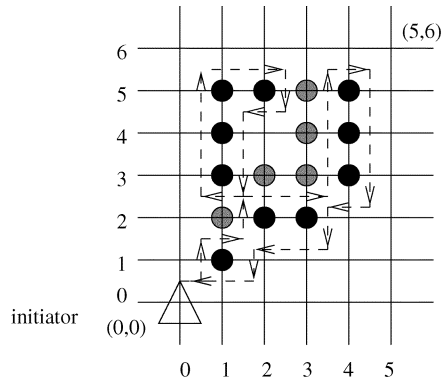

Fig. 8. An example.

in initiator ID overwrites the rest, and then, (b) the one with a smaller y value in initiator ID overwrites the rest. A south-west corner is "awakened" by an incoming initiation message if it has not started its process, and its ID overwrites that of the incoming message. In this way, the west-most south-west (inner or outer) corner will dominate even if it is not the initiator originally.

Fig. 8 shows an example of one component with ten faulty nodes (black nodes). Gray nodes in the figure correspond to notification end nodes of concave row/column sections. The remaining disabled nodes are not shown. Based on the distributed solution, node (1, 2) is a south boundary node responsible for concave column section [1, 2...2] (a column section with one row). In fact, node (1, 2) has three statuses: north, west, and south boundary node; this node updates three entries in the boundary array with the same timestamp. Node (2, 3) is a north boundary node for column section [2, 3...4]. Node (2,3) is also an east boundary node. Nodes (3, 3), (3, 4), and (3, 5) are all west boundary nodes for concave row sections [2...3, 3], [2...3, 4], and [3...3, 5], respectively.

IV. Simulation

A simulation has been conducted in a 100×100 mesh to test the effectiveness of the new faulty polygon model. To simplify the simulation, we assume that the number of faults is no more than 800. After the occurrence of faults, faulty blocks, sub-minimum faulty polygons, and minimum faulty polygons (in both centralized, and distributed solutions) are constructed through rounds of information exchanges, and updates between neighbors. We consider two fault distributions, assuming all faults are sequentially added to the network: First, we randomly generate the positions of these faults in the random fault distribution model. Second, to test cases with large faulty components, a clustered fault distribution model is adopted here. In this model, all points have the same failure rate initially. After a fault (x, y) is inserted, the failure rate of its adjacent neighbors (eight in all) is doubled. Therefore, there are two different failure rates, one for nodes that are not adjacent neighbors of existing faults, and the other for nodes that are adjacent neighbors of existing faults. Under the clustered fault distribution model, faults tend to form clusters.

Fig. 9 shows the average number of nonfaulty nodes contained in faulty blocks (FB), sub-minimum faulty polygons (FP), and minimum faulty polygons (MFP) in the whole network. The results under the random fault distribution model

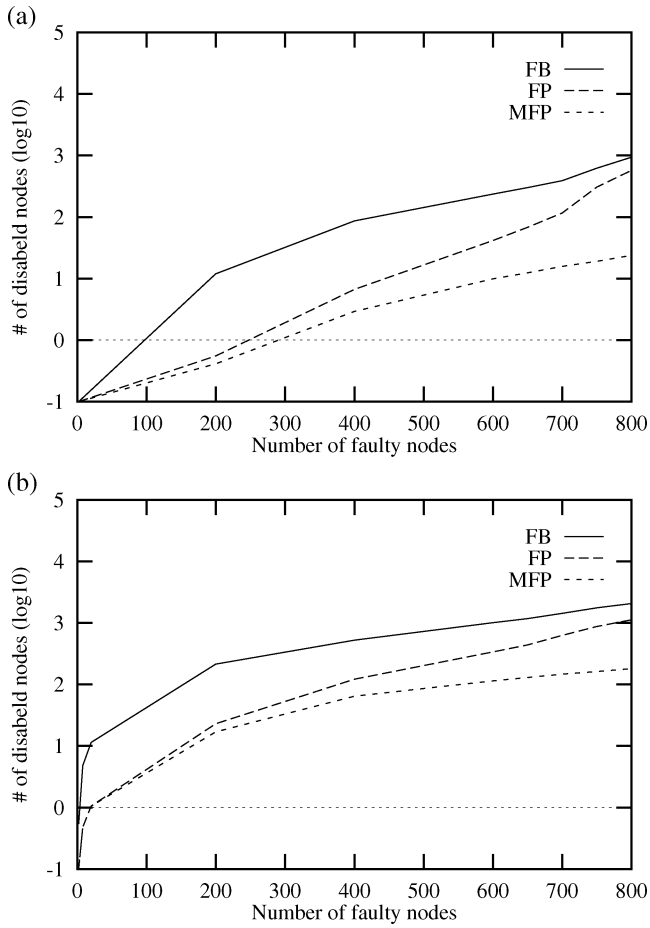

Fig. 9. Average number of nonfaulty but disabled nodes contained in faulty blocks (FB), sub-minimum faulty polygons (FP), and minimum faulty polygons (MFP) in the whole network: (a) the random fault distribution model, and (b) under the clustered fault distribution model.

are shown in Fig. 9(a), and those under the clustered fault distribution model are shown in Fig. 9(b). The faulty polygon (FP or MFP) covers all the faults in the FB model, but contains fewer nonfaulty nodes than does FB. Under the FP model, 50% of nonfaulty nodes contained in FB can be enabled. Under the MFP model, 90% of nonfaulty nodes contained in FB can be enabled.

Fig. 10 shows the average size of FB, FP, and MFP by the number of faulty, and nonfaulty nodes they contain. The results under the random fault distribution model are shown in Fig. 10(a), and those under the clustered fault distribution model are shown in Fig. 10(b). The average size of both FP & MFP is smaller than that of FB. The average size of MFP is the least of the three (FB, FP, and MFP). Because of the scattered distribution of faults under the random fault distribution model, most faulty blocks (under FB or FP model) contain no more than four nodes, and most virtual faulty blocks (under MFP model) contain only one faulty node. Under the clustered fault distribution model, the size of each faulty block becomes large, and the average size can be six times that of the random faulty distribution model. However, the average size of MFP does not increase much, even when the number of faults reaches 800.

Fig. 11 shows the average number of rounds of information exchanges between neighbors for status determination in the

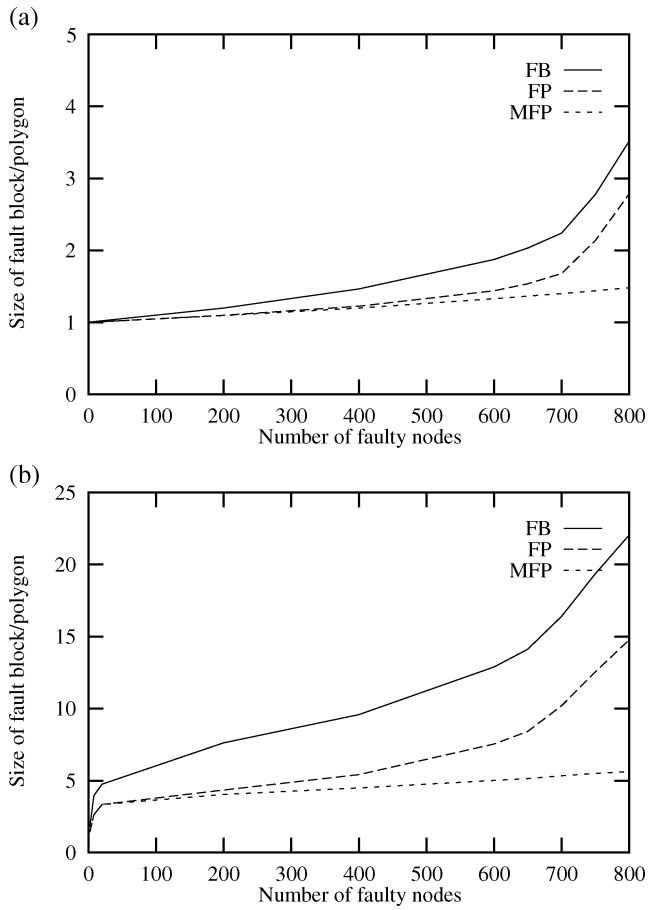

Fig. 10. Average size of FB, FP, and MFP: (a) the random fault distribution model, and (b) under the clustered fault distribution model.

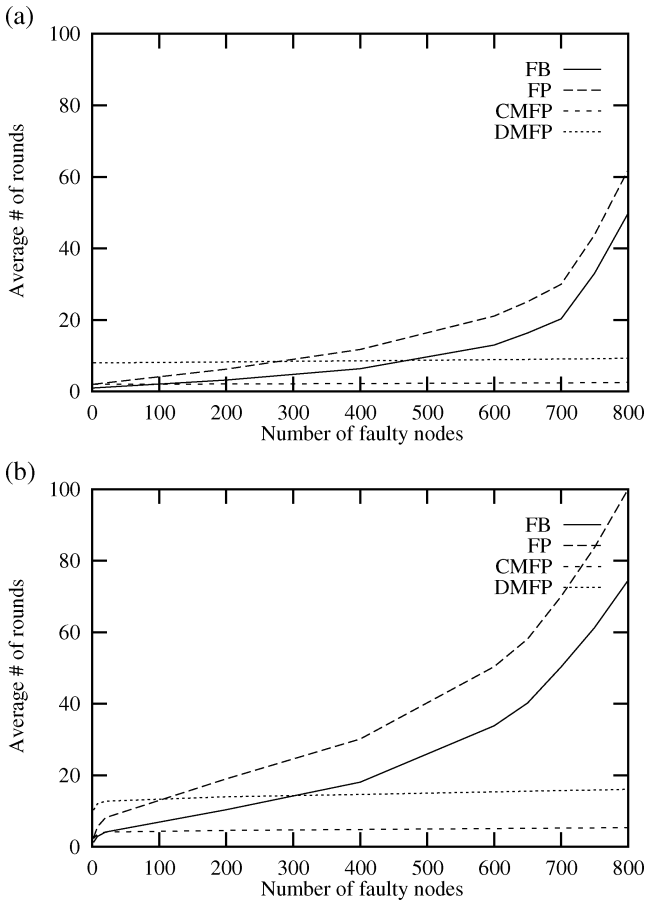

Fig. 11. Average number of rounds for status determination under FB, FP, and MFP (the centralized solution CMFP, and the distributed solution DMFP): (a) under the random fault distribution model, and (b) under the clustered fault distribution model.

networks under FB, FP, and MFP (the centralized solution CMFP, and the distributed solution DMFP) model. More specifically, under the FB model, numbers of rounds are needed for applying the labeling scheme 1. Under the FP model, extra rounds are needed for applying labeling scheme 2. Under the CMFP model, we "emulate" labeling schemes 1 & 2 on faulty components, where a virtual faulty block is generated from each component. Rounds of information exchanges are needed when labeling schemes 1 & 2 are applied on each faulty component. Under the DMFP model, we use the boundary ring construction followed by notification from one end node in each concave row/column section. Rounds of information propagation are needed in boundary ring construction, and the notification process. That is, the number of rounds for status determination of FB or FP depends on the maximum size of faulty blocks, and that of MFP (CMFP or DMFP) depends on the maximum size of faulty components. Because in the presence of a large number of faulty nodes, the average size of faulty blocks is significantly larger than that of components, the number of rounds needed for MFP (DMFP or CMFP) is much less than that of FB or FP. In MFP, the construction of DMFP needs more rounds than that of CMFP, because the boundary ring construction needs to circle around each faulty component, and to notify nodes in the concave region. However, the number of rounds in DMFP is still much less than that of FB or FP. This result confirms the effectiveness of our minimum faulty polygon (MFP) model because the process is more parallel in nature, and the status of nodes in MFP can be determined more quickly.

V. Conclusion

In this paper, we provided a solution to an open problem posed in [16]. That is, given a set of faulty nodes, find a set of disjoint orthogonal polygons to cover these faulty nodes such that the number of nonfaulty nodes in these orthogonal polygons is minimized. We proposed a centralized solution, and its distributed counterpart. Simulation has been conducted to compare the proposed fault model with previous ones, including the rectangular faulty block, and sub-optimal orthogonal convex polygon, in terms of the average size of the block, and the average number of nonfaulty nodes included in the block. Results have shown that the proposed approach can not only find a set of minimum faulty polygons, but also does so quickly in terms of the number of rounds in the distributed solution. Our future work will focus on extending the proposed method to higher dimension meshes (tori).

References

[1] A. Agarwal, "Limits on interconnection network performance," *IEEE Trans. Parallel Distrib. Syst.*, vol. 2, no. 4, pp. 398–412, Oct. 1991.

[2] Y. M. Boura and C. R. Das, "Fault-tolerant routing in mesh networks," in *Proc. 1995 Int. Conf. Parallel Processing*, Aug. 1995, pp. I 106–I 109.
[3] S. Chalasani and R. V. Boppana, "Communication in multicomputers with nonconvex faults," *IEEE Trans. Comput.*, vol. 46, no. 5, pp. 616–622, May 1997.
[4] A. A. Chien and J. H. Kim, "Planar-adaptive routing: low-cost adaptive networks for multiprocessors," *J. ACM*, vol. 42, no. 1, pp. 91–123, Jan. 1995.
[5] W. J. Dally, "Performance analysis of k-ary n-cube interconnection networks," *IEEE Trans. Comput.*, vol. 39, no. 6, pp. 775–785, Jun. 1990.
[6] J. Duato, S. Yalamanchili, and L. Ni, *Interconnection Networks: An Engineering Approach*: IEEE Computer Society, 1997.
[7] R. Libeskind-Hadas and E. Brandt, "Origin-based fault-tolerant routing in the mesh," in *Proc. 1st Int. Symp. High Performance Computer Architecture*, Jan. 1995, pp. 102–111.
[8] F. P. Preparata and M. I. Shamos, *Computational Geometry: An Introduction*: Springer Verlag, 1985.
[9] L. Sheng and J. Wu, "Maximum-shortest-path (msp) is not optimal for a general n/spl times/n torus," *IEEE Trans. Reliab.*, vol. 52, no. 1, pp. 22–25, 2003.
[10] J. D. Shih, "Adaptive fault-tolerant wormhole routing algorithms for hypercube and mesh interconnection networks," in *Proc. 11th Int. Parallel Processing Symp.*, Apr. 1997, pp. 333–340.
[11] C. C. Su and K. G. Shin, "Adpative fault-tolerant deadlock-free routing in meshes and hypercubes," *IEEE Trans. Comput.*, vol. 45, no. 6, pp. 672–683, Jun. 1996.
[12] Y. J. Suh, B. V. Dao, J. Duato, and S. Yalamanchili, "Software based fault-tolerant oblivious routing in pipelined networks," in *Proc. 1995 Int. Conf. Parallel Processing*, Aug. 1995, pp. I 101–I 105.
[13] P. H. Sui and S. D. Wang, "An improved algorithm for fault-tolerant wormhole routing in meshes," *IEEE Trans. Comput.*, vol. 46, no. 9, pp. 1040–1042, Sep. 1997.
[14] Y. C. Tseng, M. H. Yang, and T. Y. Juang, "An Euler-path-based multicasting model for wormhole-routed networks with multi-destination capability," in *Proc. 1998 Int. Conf. Parallel Processing*, Aug. 1998, pp. 366–373.
[15] D. Wang, "Minimal-connected-component (MCC)—a refined fault block model for fault-tolerant minimal routing in mesh," in *Proc. IASTED Int. Conf. Parallel and Distributed Computing and Systems*, Nov. 1999, pp. 95–100.
[16] J. Wu, "A distributed formulation of smallest faulty orthogonal convex polygons in 2-D meshes," in *Proc. Int. Parallel and Distributed Processing Symp. (IPDPS)*, 2001.
[17] ——, "Fault-tolerant adaptive and minimal routing in mesh-connected multicomputers using extended safety levels," *IEEE Trans. Parallel Distrib. Syst.*, vol. 11, no. 2, pp. 149–159, Feb. 2000.
[18] ——, "A fault-tolerant and deadlock-free routing protocol in 2-D meshes based on odd-even turn model," *IEEE Trans. Comput.*, vol. 52, no. 9, pp. 1154–1169, Sep. 2003.
[19] ——, "Maximum-shortest-path (msp): an optimal routing policy for mesh-connected multicomputers," *IEEE Trans. Reliab.*, vol. 48, no. 3, pp. 247–255, 1999.
[20] D. Xiang, "Fault-tolerant routing in hpercube multicomputers using local safety information," *IEEE Trans. Parallel Distrib. Syst.*, vol. 12, no. 9, pp. 942–951, 2001.
[21] D. Xiang, A. Chen, and J. Wu, "Reliable broadcasting in wormhole-routed hypercube-connected networks using local safety information," *IEEE Trans. Reliab.*, vol. 52, no. 2, pp. 245–256, 2003.

Jie Wu received the B.S. & M.S. degrees from Shanghai University of Science and Technology (now Shanghai University) in 1982 & 1985, respectively; and the Ph.D. degree from Florida Atlantic University in 1989. He is currently a Professor at the Department of Computer Science and Engineering, Florida Atlantic University. He has published over 250 papers in various journals & conference proceedings. His research interests are in the area of ad hoc & sensor networks, routing protocols, fault-tolerant computing, and interconnection networks. Dr. Wu served on many conference committees, including HPCA, ICDCS, ICPP, INFOCOM, IPDPS, and MASS; and editorial boards, including IEEE TRANSACTIONS ON PARALLEL AND DISTRIBUTED SYSTEMS. He was a co-guest-editor of IEEE COMPUTER, and Journal of Parallel and Distributing Computing. He is the author of the text "Distributed System Design" published by the CRC Press. Dr. Wu was the recipient of the 1996–1997, and 2001–2002 Researcher of the Year Award at Florida Atlantic University. He served as an IEEE Computer Society Distinguished Visitor. Dr. Wu is a member of ACM and a senior member of IEEE.

Zhen Jiang received the B.S. degree in 1992 from Shanghai Jiaotong University, Shanghai, P. R. China; the Master degree in 1998 from Nanjing University, Nanjing, Jiangsu, P. R. China; and the Ph.D. degree in 2002 from Florida Atlantic University, Boca Raton, Florida. Currently, he is an associate professor in the Department of Computer Science, West Chester University of Pennsylvania, West Chester, Pennsylvania. He is also a faculty member in the Information Assurance Center at West Chester University, which is one of 59 NSA Centers of Academic Excellence in Information Assurance Education in the nation. His research interests are in the areas of interconnection networks, sensor networks, fault-tolerant routing, performance evaluation, and object-oriented design. He is a member of IEEE.