A Rectilinear-Monotone Polygonal Fault Block Model for Fault-Tolerant Minimal Routing in Mesh

Dajin Wang, *Member*, *IEEE*

Abstract—We propose a new fault block model, Minimal-Connected-Component (MCC), for fault-tolerant adaptive routing in mesh-connected multiprocessor systems. This model refines the widely used rectangular model by including fewer nonfaulty nodes in fault blocks. The positions of source/destination nodes relative to faulty nodes are taken into consideration when constructing fault blocks. The main idea behind it is that a node will be included in a fault block only if using it in a routing will definitely make the route nonminimal. The resulting fault blocks are of the *rectilinear-monotone polygonal* shapes. A sufficient and necessary condition is proposed for the existence of the minimal "Manhattan" routes in the presence of such fault blocks. Based on the condition, an algorithm is proposed to determine the existence of Manhattan routes. Since MCC is designed to facilitate minimal route finding, if there exists no minimal route under MCC fault model, then there will be absolutely no minimal route whatsoever. We will also present two adaptive routing algorithms that construct a Manhattan route avoiding all fault blocks, should such routes exist.

Index Terms—Adaptive routing, fault model, fault tolerance, interconnection network, mesh.

1 Introduction

The *mesh* structure is one of the most important interconnection network models. As a topology to interconnect multiprocessor computer systems, it has been proven to possess many attractive properties. Parallel computers using mesh as their underlying architecture have been around for years [5], [12], [13]. Because of its importance to achieving high performance, fault-tolerant computing for mesh structures has been the focus of an extensive literature. A very important aspect of mesh fault tolerance is its ability to route from a source node to a destination, avoiding all faulty nodes. Routing is a process to send *messages*, which can be either data or instructions, from a source node to a destination node, passing some intermediate nodes. There are basically two types of routing: *deterministic* routing and *adaptive* routing. In deterministic routing, a fixed path is used to send/receive messages for a particular pair of source/destination. The obvious advantages of this routing are its simplicity and ease of implementation. However, it suffers the shortcoming of weak fault tolerability—when one or more nodes on the dedicated path fail, either routing cannot be carried out or only very limited "detours" can be taken. In adaptive routing, there is no dedicated path for a pair of source and destination. The path is adaptively constructed in the process of routing. A fully adaptive routing algorithm should allow the message to take any fault-free intermediate nodes to reach destination. As a result, adaptive routing can tolerate more faulty nodes than deterministic routing.

A natural goal in adaptive routing is to find a route as short as possible in the presence of faulty nodes, preferably the minimal route. Fault-tolerant routing has been studied extensively [1], [2], [3], [4], [7], [8], [9], [10], [14], [15], [16], [17]. In fault-tolerant routing on mesh, most work uses rectangular fault block model [1], [2], [3], [11], [14], [17]. In rectangular model, all faulty nodes are grouped in disjointed, rectangular areas, called *fault blocks*. A fault block is constructed in a way that includes as few nonfaulty nodes as possible while maintaining rectangular shape. All nodes in these fault blocks, whether they are faulty or nonfaulty, are to be bypassed in any routes. Rectangular fault block model is a simple and useful model, based on which many routing algorithms were developed. However, as we will show in Section 2, this model may not be optimal for the purpose of finding routes as short as possible. In constructing rectangular blocks, some nonfaulty nodes are unnecessarily included, decreasing the number of selectable nodes for routing, thus decreasing the chance for finding minimal routes.

In this paper, we propose a new fault block model for fault-tolerant adaptive routing in mesh. The proposed model is a refinement of the rectangular model. In construction of fault blocks, the positions of source and destination nodes are taken into consideration. A fault block so constructed is called a *Minimal-Connected-Component*, MCC for abbreviation. It has the shape of a *rectilinear-monotone polygon* instead of a rectangle. As a result, fewer nonfaulty nodes are included in fault blocks. Many nonfaulty nodes that would have been included in a rectangular fault block now can become candidate routing nodes, increasing the chance of obtaining minimal route(s) in the presence of faulty nodes. This is a noticeable overall

• *The author is with the State Key Laboratory for Novel Software Technology at Nanjing University, Nanjing 210093, China, and the Department of Computer Science, Montclair State University, Upper Montclair, NJ 07043. E-mail: wang@pegasus.montclair.edu.*

Manuscript received 28 June 2001; revised 18 Dec. 2001; accepted 4 Apr. 2002.
For information on obtaining reprints of this article, please send e-mail to: tc@computer.org, and reference IEEECS Log Number 114448.

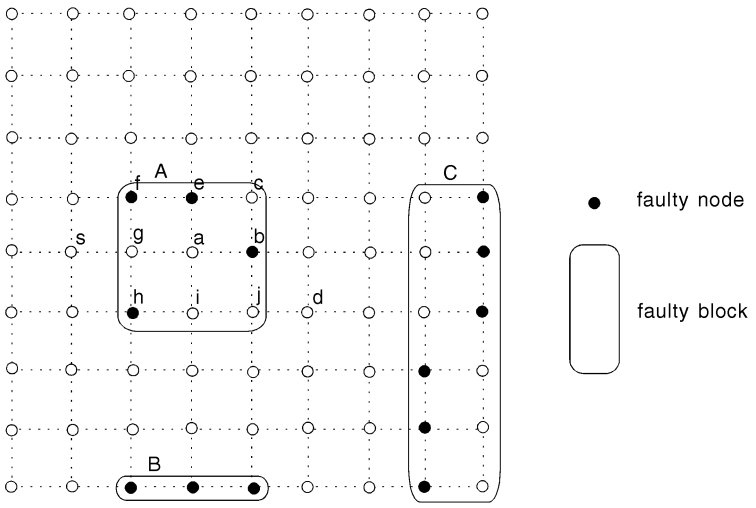


Fig. 1. An example of rectangular fault blocks. Fault-free but disabled nodes in a fault block will not be used in routing.

improvement in the fault-tolerability of the system. Since MCC is designed to facilitate minimal route finding, if there exists no minimal route under the MCC fault model, then there will be absolutely no minimal route whatsoever. What is more, the proposed scheme is cost effective for the benefits it provides—it is not necessary to compute a set of fault blocks for each pair of source/destination. For a given set of faulty nodes, we need to compute only *two* different sets of fault blocks.

The rest of this paper is organized as follows: In Section 2, we give a formal description of the Minimal-Connected-Component (MCC) fault block model. Also in Section 2, a sufficient and necessary condition is proposed for the existence of minimal Manhattan routes in presence of MCC fault blocks. In Section 3, we give an algorithm to determine the existence of Manhattan routes using the MCC fault block model. In Section 4, we will present two adaptive routing algorithms that construct a Manhattan route avoiding all fault blocks, in case such routes exist. Section 5 gives some concluding remarks.

2 Minimal-Connected-Component (MCC) Fault Block Model

2.1 The MCC Fault Blocks

It is convenient to represent a two-dimensional (2D) mesh with a 2D coordinate system in which we use a pair of ordered integers (x, y) to locate and identify a node. Routing is then to send messages from a source node (x_s, y_s) to a destination (x_d, y_d), passing some intermediate nodes. For the convenience of discussion and without loss of generality, we can always define the coordinate system so that $(x_s, y_s) = (0, 0)$. The destination node (x_d, y_d) will then fall in one of the four quadrants. Unless pointed out otherwise, we always assume that (x_d, y_d) is in the first quadrant, i.e., $x_d \geq 0$ and $y_d \geq 0$. The cases for the destination falling in other quadrants need only symmetric treatments.

A nonboundary node (x, y) has four immediate neighbors, located at $(x + 1, y)$, $(x - 1, y)$, $(x, y + 1)$, and $(x, y - 1)$, respectively. Following the convention in the literature on meshes, we say that node $(x + 1, y)$ is (x, y)'s neighbor in its *east* direction. Similarly, $(x - 1, y)$, $(x, y + 1)$, and $(x, y - 1)$ are called (x, y)'s *west*, *north*, and *south* neighbors. In each step of routing, a node can take one and only one direction to its immediate neighbor. So, a routing can be represented as a sequence of directions. If a route uses only one direction (e.g., all west or all north), then it is clearly a shortest possible route. In more general terms, we can define *Manhattan routing* as follows:

Definition 2.1. *A routing is called Manhattan routing if, during the routing, at most two directions are used.*

Obviously, a Manhattan route can always be effected for any pair of (x_s, y_s) and (x_d, y_d) if a mesh has no faulty nodes and it is the shortest possible route from (x_s, y_s) to (x_d, y_d).

However, if there are some faulty nodes in the mesh, in the sense that these nodes are unable to receive/send messages, a Manhattan route may or may not exist. In [17], a sufficient condition was proposed for the existence of Manhattan route using the commonly used *disconnected rectangular block* fault model [14]. While a useful model facilitating many routing algorithms [1], [2], [3], [11], [14], it is not an optimal one for the purpose of finding routes as short as possible. In forming the rectangular fault blocks, the positions of source and destination as relative to the faults are not taken into consideration. For example, as per the definition of rectangular fault block, in Fig. 1, the fault block A contains fault-free but "disabled" nodes a, g, i, and j. Since the whole block is to be bypassed, these disabled nodes will not be used in the process of routing, no matter where the source and destination are. However, if source s and destination d are as given in Fig. 1, using them in routing can generate a Manhattan route $(s \rightarrow g \rightarrow a \rightarrow i \rightarrow j \rightarrow d)$.

The above example motivates for a finer model of fault block so that the shortest possible routes can be found in the presence of faulty nodes. In this paper, we propose the Minimal-Connected-Component fault model, MCC for abbreviation, just for that purpose. The main idea behind this fault model is that a node will be included in a fault

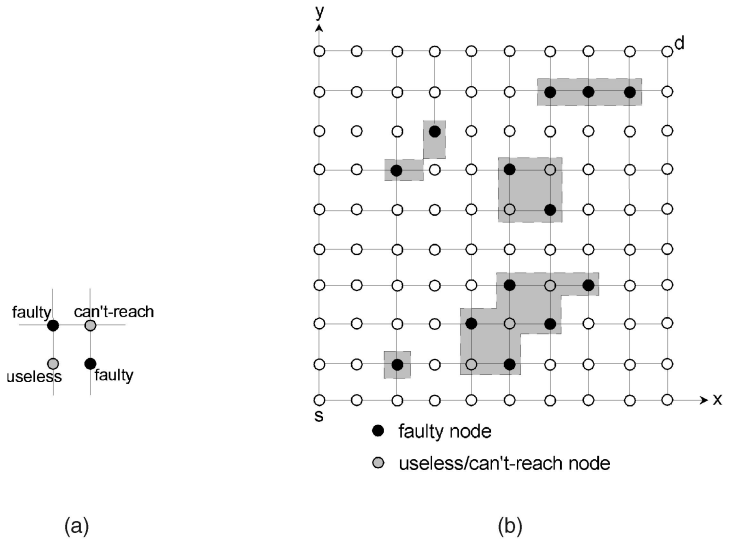


Fig. 2. (a) Definition of useless and can't-reach nodes. (b) Five example MCC fault blocks.

block only if using it in a routing will definitely make the resulting route non-Manhattan.

The construction of MCC blocks is a recursive procedure and is dependent on the relative positions of source and destination. The following construction procedure assumes $(x_s, y_s) = (0, 0)$ and $x_d, y_d \geq 0$, i.e., the destination is northeast of source.

1. Initially, label all faulty nodes as "faulty;"
2. If a node (x, y) is fault-free, but its north neighbor $(x, y+1)$ and east neighbor $(x+1, y)$ are faulty, (x, y) is labeled "useless;"
3. If a node (x, y) is fault-free, but its north and east neighbors are either faulty or useless, (x, y) is labeled "useless;"
4. If a node (x, y) is fault-free, but its south neighbor $(x, y-1)$ and west neighbor $(x-1, y)$ are faulty, (x, y) is labeled "can't-reach;"
5. If a node (x, y) is fault-free, but its south and west neighbors are either faulty or can't-reach, (x, y) is labeled "can't-reach;"
6. The nodes are recursively labeled until there are no new useless or can't-reach nodes;
7. Finally, if two adjacent nodes are faulty, useless, or can't-reach, then they are linked by an imaginary edge.

A node labeled "useless" is such a node that, once it is entered in a routing, the next move must take either west or south direction, making a Manhattan routing impossible. A node labeled "can't-reach" is such a node that, to enter it in a routing, a west or south move must be taken, making a Manhattan routing impossible.

We call the final set of nodes so connected an MCC fault block. Fig. 2a shows what useless and can't-reach nodes are; Fig. 2b shows five example MCC fault blocks. It can be seen that an MCC block in general covers a *rectilinear-monotone polygonal* area, of which the rectangle is a special case. Actually, an MCC block is obtained by removing the unnecessarily included fault-free nodes in the southeast and northwest corner sections of a rectangular block.

The MCCs for quadrant III destination (i.e., the destination is southwest of source) can be obtained just by exchanging the roles of useless and can't-reach nodes in the labeling procedure for quadrant I destination. In fact, MCCs generated from quadrant I labeling procedure and from quadrant III labeling procedure are of the same shape.

The labeling procedure for quadrant II destination (i.e., the destination is in northwest) can be derived from that for quadrant I by exchanging the roles of east and west neighbors:

2. If a node (x, y) is fault-free, but its north neighbor $(x, y+1)$ and *west* neighbor $(x-1, y)$ are faulty, (x, y) is labeled "useless;"
3. If a node (x, y) is fault-free, but its north and *west* neighbors are either faulty or useless, (x, y) is labeled "useless;"
4. If a node (x, y) is fault-free, but its south neighbor $(x, y-1)$ and *east* neighbor $(x+1, y)$ are faulty, (x, y) is labeled "can't-reach;"
5. If a node (x, y) is fault-free, but its south and *east* neighbors are either faulty or can't-reach, (x, y) is labeled "can't-reach."

Fig. 3 shows the quadrant II MCC blocks for the same faulty nodes in Fig. 2b.

Again, the labeling procedure for MCCs for quadrant IV destination is derived from the one for quadrant II by exchanging the roles of useless and can't-reach nodes. MCCs generated for quadrants II and IV are the same. Fig. 4 gives the general shapes of MCCs for routing in all four quadrants.

It can be seen that MCCs for d falling in quadrants I and III are of the same general shape, while MCCs for d falling in quadrants II and IV have the same shape. Therefore, it is not necessary to compute a set of fault blocks for *each* pair of source/destination. Instead, for given faulty nodes, only *two* different sets of fault blocks are computed: The destination d falling in quadrant I or III uses one set for routing and d falling in quadrant II or IV uses the other set.

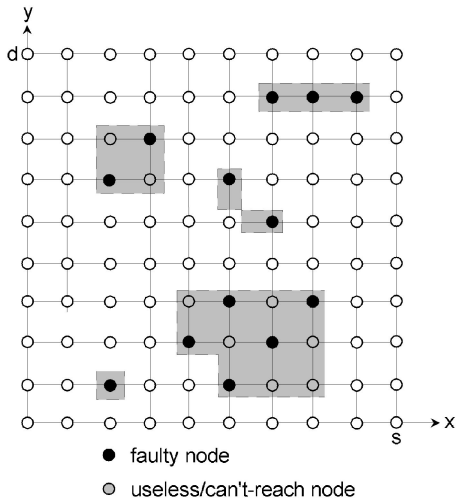


Fig. 3. Example quadrant II MCC blocks for the same faulty nodes in Fig. 2b.

The following two obvious observations of MCC are important:

Proposition 2.1. *All MCCs in a mesh are disjoint to each other.*

Proposition 2.2. *Any two MCCs are separated by at least two Hamming distance, i.e., one can always find a route going "in-between" two MCCs.*

The construction of MCCs can be implemented by exchanging/setting the status ("good"/"faulty"/"useless"/"can't-reach") among neighboring nodes until no nodes change their status. This kind of approach is called *localized algorithm* [6], [16], in which a set of processors perform simple operations (such as exchange of status of neighboring nodes) distributively and independently, in such a way that the collective outcome renders a result of global nature (such as construction of fault blocks). Adopting localized algorithm, it is not difficult to see that the complexity of MCC construction, i.e., the number of rounds needed to produce the set of MCCs, is linearly proportional to the diameter of the largest block.

2.2 Experiment Results

As we claimed earlier, the MCC model includes fewer nonfaulty nodes in fault blocks. Many nonfaulty nodes that would have been included in rectangular fault blocks now can become candidate routing nodes. Experiments were conducted to compare the two models in terms of 1) the total number of nodes included in fault blocks and 2) the total number of fault blocks. The experiment results are shown in Fig. 5. In experiment (a), faulty nodes are randomly generated on a 50×50 mesh. The rate of faulty nodes ranges from 1 percent to 15 percent. Data used in the table is the average of 1,000 runs. As can be seen in Fig. 5a, nodes (faulty and nonfaulty) included in rectangular blocks always outnumber nodes in MCC blocks. The outnumbering becomes sharper and sharper as fault rate grows. In experiment (b), the number of blocks in the two models is compared. The rate of faulty nodes is fixed at 10 percent. The size of mesh ranges from 10×10 to 100×100. The result in Fig. 5b shows that the number of MCC blocks is always higher than that of rectangular blocks. That means MCC is a much "finer" grouping of faulty nodes than that of rectangular blocks. Fig. 5c does a similar comparison with fault rate 15 percent. It can be seen that, in the rectangular model, the number of blocks first grows as mesh size grows. However, when mesh size grows further, the number of blocks decreases. That is, the distribution of faults has the tendency to make the whole mesh one "big block." On the other hand, the number of MCC blocks grows nicely as mesh size increases.

To measure how well MCC facilitates finding minimum routes among faults, another set of simulations was conducted in which the availability of Manhattan routes using MCC is examined. Using different fault rates as the parameter, the simulation showed that Manhattan routes always have 100 percent availability when the fault rate is up to 23 percent. Sub-100 percent availability is first observed when the fault rate reaches 25 percent, at which rate the availability is about 98.9 percent. Note that, given today's technology, the node fault rate of 23 percent is highly unlikely. In comparison with the rectangular model, recall that, in Fig. 5c at fault rate of 15 percent, the distribution of faults has already shown the tendency to make the whole mesh one big block, blocking all feasible routes.

2.3 The Existence of Manhattan Routes in a Faulty Mesh

As has been mentioned, MCC is designed to facilitate minimal route finding. An MCC block is a *minimum* connected block so that a node v of the MCC is either faulty or, if fault-free, then entering v would need a step that violates the requirement for a Manhattan route (i.e., a third direction would be used in the route). In other words, the MCC is a fault

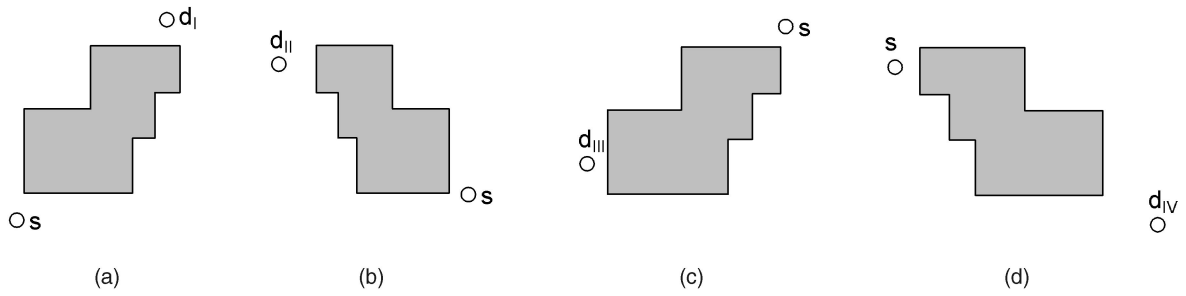
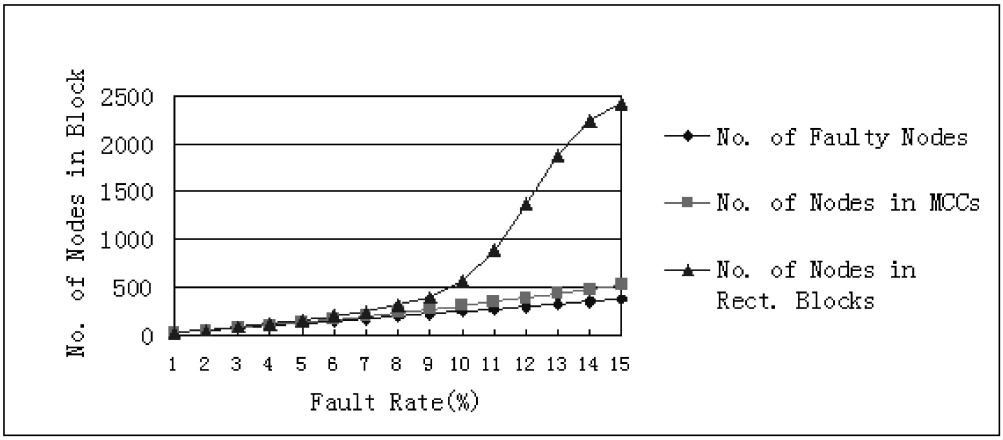
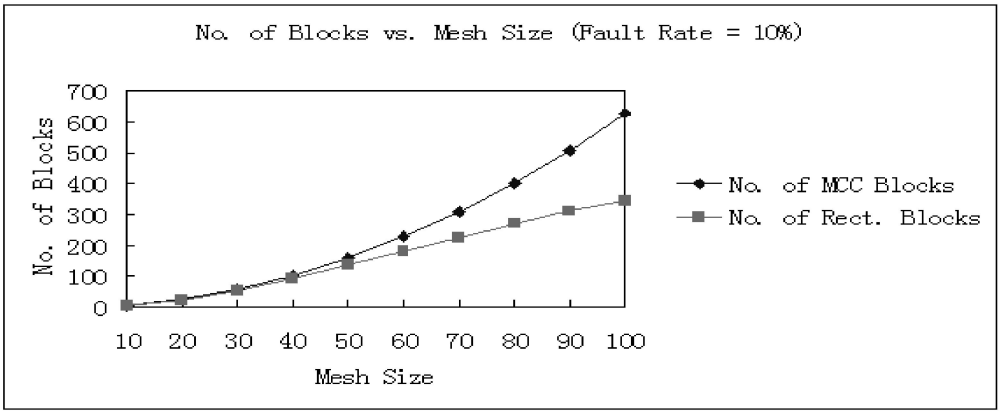


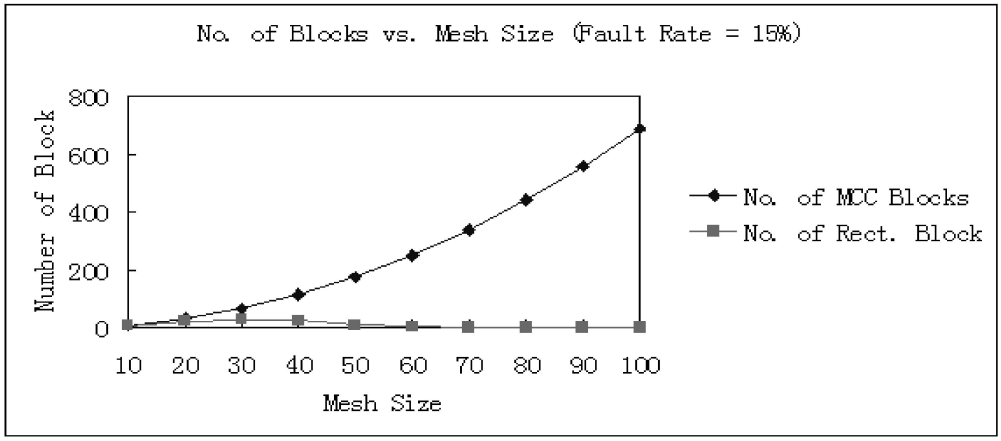
Fig. 4. The shape of MCC for d falling in (a) first quadrant, (b) second quadrant, (c) third quadrant, (d) fourth quadrant.



(a)



(b)



(c)

Fig. 5. Simulation results.

block model that provides the *maximum* possibility of finding a Manhattan route among faults. Therefore, if there exists no Manhattan route under the MCC fault model, there will be absolutely no Manhattan route.

We now propose a sufficient and necessary condition for the existence of Manhattan routes in a mesh with faulty nodes. To check the existence of Manhattan routes in a faulty mesh, MCCs in the mesh are constructed first. All the nodes in MCCs are not to be entered or a Manhattan route cannot be effected. The sufficient and necessary condition for the existence of Manhattan routes is given in the following theorem. An obvious assumption is that neither source nor destination node belongs to any MCC.

Theorem 2.1. *A Manhattan route can be found if and only if neither of the following two statements holds:*

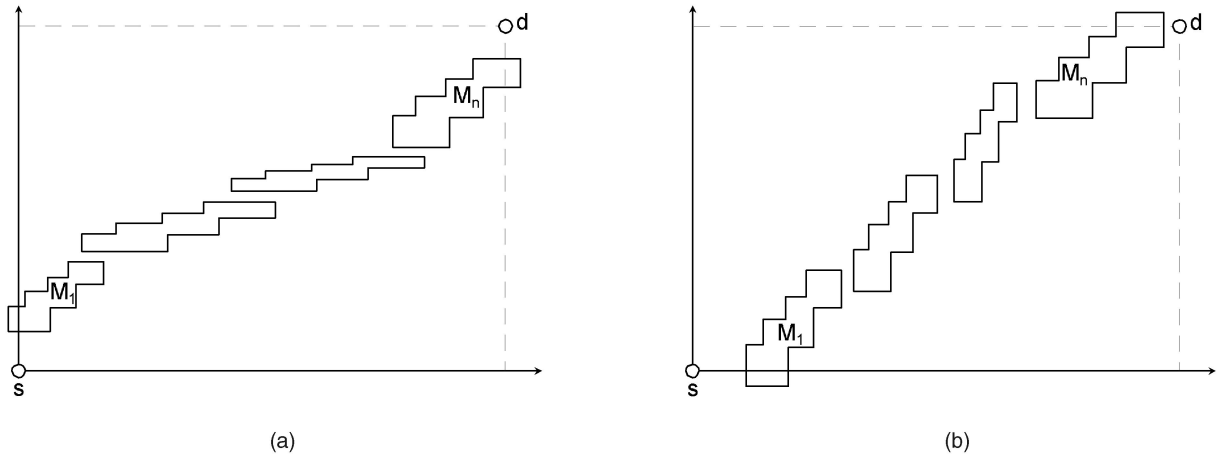


Fig. 6. (a) Type-I MCC sequence, (b) Type-II MCC sequence.

1. *There exists a sequence of MCCs* $(M_1, M_2, \ldots, M_n)$ *such that*
   - M_1 *contains a node* $(0, z_1)$ *such that* $0 < z_1 < y_d$;
   - M_n *contains a node* (x_d, z_n) *such that* $0 < z_n < y_d$;
   - *For all* M_i, M_{i+1}, $1 \le i \le n-1$,

$$\min\{a : (a,b) \in M_{i+1}\} - 1 \le \max\{u : (u,v) \in M_i\} \le \max\{x : (x,y) \in M_{i+1}\} - 1$$

and

$$\max\{v : (u,v) \in M_i\} < \max\{y : (x,y) \in M_{i+1}\}.$$

We call this type of sequence a Type-I sequence.

2. *There exists a sequence of MCCs* $(M_1, M_2, \ldots, M_n)$ *such that*
   - M_1 *contains a node* $(z_1, 0)$ *such that* $0 < z_1 < x_d$;
   - M_n *contains a node* (z_n, y_d) *such that* $0 < z_n < x_d$;
   - *For all* M_i, M_{i+1}, $1 \le i \le n-1$,

$$\min\{b : (a,b) \in M_{i+1}\} - 1 \le \max\{v : (u,v) \in M_i\} \le \max\{y : (x,y) \in M_{i+1}\} - 1$$

and

$$\max\{u : (u,v) \in M_i\} < \max\{x : (x,y) \in M_{i+1}\}.$$

We call this type of sequence Type-II sequence.

Fig. 6 gives an illustration of the two types of MCC sequences to help convey the idea. A proof of the theorem is provided below.

Proof. The proof is in two parts. The first part will show that, if a Manhattan route exists at all, then there cannot exist any MCC sequence of the types described in Theorem 2.1. In the second part, we show that, if there does not exist any MCC sequence of the kinds described in Theorem 2.1, then we can always find a Manhattan route from source to destination.

Part I. Suppose a Manhattan route R exists from source to destination, in the presence of MCCs of the mesh. By the definition of Manhattan route, its general shape should look like "stairs," as shown in Fig. 7. Observe from Fig. 7 that R divides the rectangular region defined by s and d into two parts, denoted as U and L, respectively. Note that any MCC should be in either the U part or the L part, not intersecting with R. For the Type-I MCC sequence to exist, M_1 must fall in U, and M_n in L. From the description of the first type sequence, there must be an M_j in the sequence intersecting with R, contradicting the assumption.

Similarly, for the Type-II MCC sequence to exist, M_1 must fall in L and M_n in U. There must be an M_j in the sequence intersecting with R, contradicting the assumption.

Part II. Suppose there does not exist any type of MCC sequences described in Theorem 2.1. We will show that a Manhattan route from source s to destination d can always be constructed.

By the above assumption, there does not exist any Type-I MCC sequence (shown in Fig. 6a). Then, there will be a stair-case-like *gap*, shown in Fig. 8a, which is composed of a sequence of alternative, connected south, west, south, west, ... directions, originating from node d, and ending somewhere east of node s at the *x*-axis. We call this gap a *Type-I gap*. Note that, if there were a Type-I

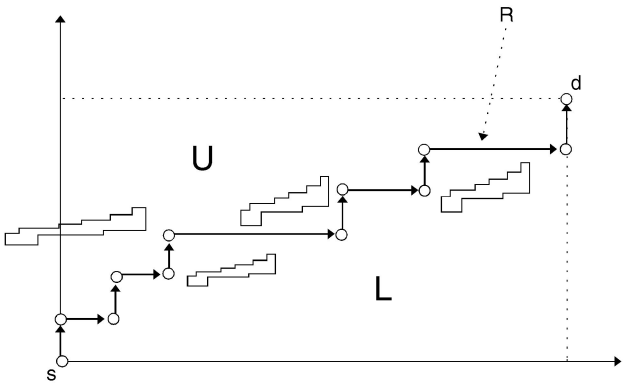


Fig. 7. Manhattan route R exists. MCCs should be in either U or L, not intersecting with R.

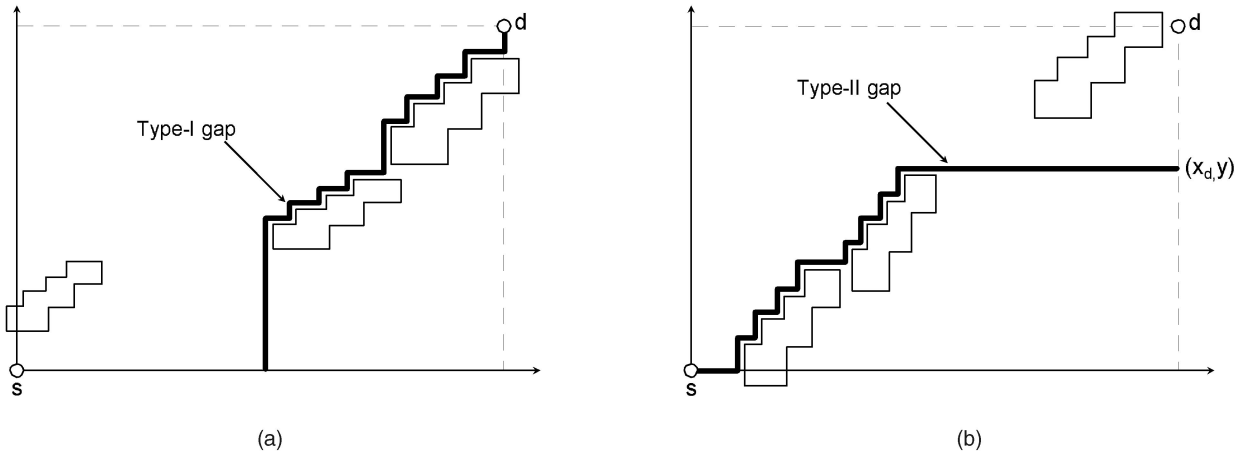


Fig. 8. (a) Type-I gap, (b) Type-II gap.

MCC sequence, such a gap could not be established (the sequence of alternative directions would not be able to reach the x-axis at east of s).

Similarly, by the assumption that there does not exist any Type-II MCC sequence (shown in Fig. 6b), we can establish a *Type-II gap*, composed of a sequence of alternative, connected east, north, east, north, ... directions, originating from s, and ending somewhere south of d at (x_d, y), where $y < y_d$. See Fig. 8b.

It is obvious that there must exist a node c shared by both Type-I and Type-II gaps and c can be viewed as a node that divides a gap into two portions, i.e., southwest and northeast portions. Combining the southwest portion of the Type-II gap and the northeast portion of the Type-I gap, we get the desired Manhattan route. The construction is shown in Fig. 9. □

3 Existence Algorithm for Manhattan Routes

3.1 Algorithm Description

Based on Theorem 2.1, we propose an algorithm to determine whether there exists a Manhattan route between two given nodes. This existence information can be built into and then quickly retrieved from the transitive closure on the *MCC overlapping graph*.

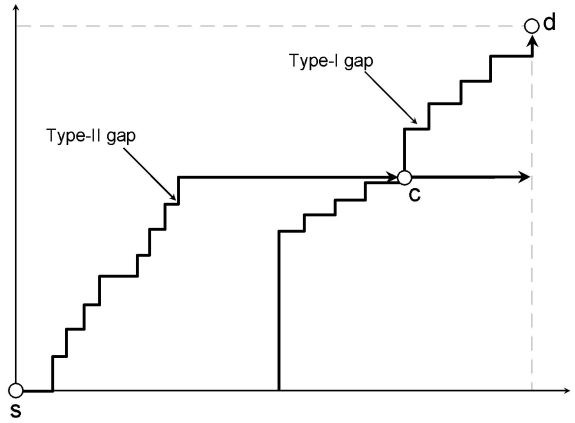


Fig. 9. Combining Type-II and Type-I gaps to construct a Manhattan route.

An MCC overlapping graph is constructed out of the layout of MCCs. Given an MCC set $S = \{M_1, M_2, \ldots, M_n\}$ in a mesh, we construct a *directed graph* $G_S^I = (V, A^I)$ with vertex set $V = \{v_1, v_2, \ldots, v_n\}$, where v_i represents MCC M_i and the "I" superscript in G_S^I and A^I refers to Type-I. The set of arrows (directed edges) A^I is defined as follows: If M_i's northeast-most node, denoted U_i, is "immediately below" M_j and "properly covered" by M_j's west-east span (see Fig. 10a for illustration), then there is an arrow pointing from v_i to v_j. Using $x(U_i), y(U_i)$ to denote U_i's x and y-coordinates, respectively, and letting L_j represent the southwest-most node of M_j, we have,

$$A^I = \{(v_i, v_j) \mid \text{going north from } U_i, M_j \text{ is the first encountered MCC such that } y(U_i) < y(U_j) \wedge x(L_j) - 2 < x(U_i) < x(U_j)\}.$$

An obvious choice of data structure for G_S^I is an $n \times n$ 2D array, denoted g_S^I. The value of an element $g_S^I(i,j)$ will be either 0 or 1:

$$g_S^I(i,j) = \begin{cases} 1 & \text{if } (v_i, v_j) \in A^I \text{ or } i = j \\ 0 & \text{if } (v_i, v_j) \notin A^I. \end{cases}$$

The *transitive closure* of g_S^I, denoted $T(g_S^I)$, tells whether there exists a directed path from one vertex to another in G_S^I, i.e.,

$$T(g_S^I)(i,j) = \begin{cases} 1 & \text{if there exists a directed path from } v_i \text{ to } v_j \\ 0 & \text{otherwise.} \end{cases}$$

Fig. 11 illustrates an example set of MCCs, the corresponding overlapping graph G_S^I, g_S^I, and $T(g_S^I)$.

Symmetrically, we construct overlapping graph $G_S^{II} = (V, A^{II})$ for Type-II MCC sequence, where A^{II} is defined as follows: If M_i's northeast node, U_i, is "immediately left of" M_j and "properly covered" by M_j's south-north span (see Fig. 10b for illustration), then there is an arrow pointing from v_i to v_j, i.e.,

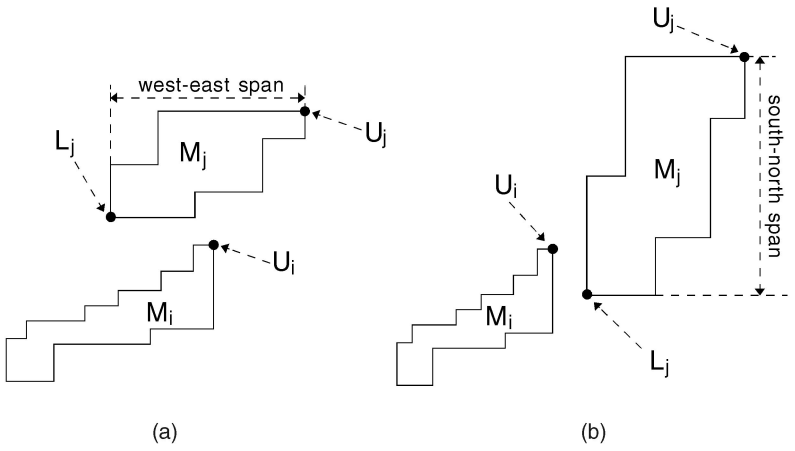
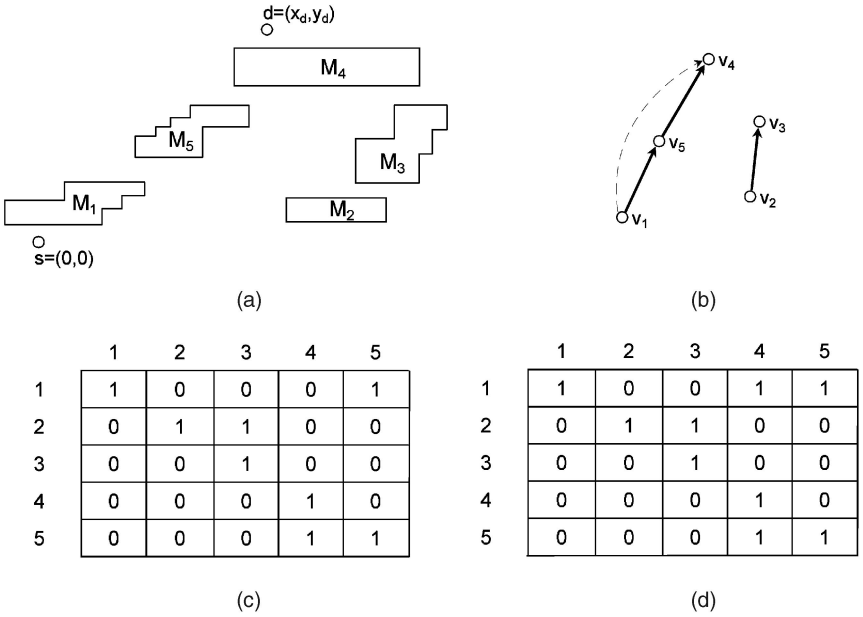


Fig. 10. (a) If that's the case, there will be an arrow pointing from v_i to v_j in G_S^I. (b) If that's the case, there will be an arrow pointing from v_i to v_j in G_S^{II}.



	1	2	3	4	5
1	1	0	0	0	1
2	0	1	1	0	0
3	0	0	1	0	0
4	0	0	0	1	0
5	0	0	0	1	1

	1	2	3	4	5
1	1	0	0	1	1
2	0	1	1	0	0
3	0	0	1	0	0
4	0	0	0	1	0
5	0	0	0	1	1

Fig. 11. (a) A set S of five MCCs, (b) G_S^I, (c) g_S^I, (d) $T(g_S^I)$.

$$A^{II} = \{(v_i, v_j) \mid \text{going east from } U_i, M_j \text{ is the first encountered MCC such that } x(U_i) < x(U_j) \wedge y(L_j) - 2 < y(U_i) < y(U_j)\}.$$

Once $T(g_S^I)$ and $T(g_S^{II})$ are computed, based on Theorem 2.1, the existence problem for Manhattan route between a particular pair of nodes, say $(0,0)$ and (x_d, y_d), can be solved by the following algorithm.

Algorithm *Manhattan(source-node, destination-node)*

{Purpose: Determine whether there exists a Manhattan route for a pair of given nodes}

Input: Source node (x_s, y_s), destination node (x_d, y_d)
Output: YES/NO

1. **if** $\exists i, j$ such that
   M_i's west-east span properly covers (x_s, y_s)
   and M_i is north of (x_s, y_s)
   and M_j's west-east span properly covers (x_d, y_d) **and** M_j is south of (x_d, y_d)
   and $T(g_S^I)(i, j) = 1$
   then return NO;
   else goto 2;
2. **if** $\exists k, l$ such that
   M_k's south-north span properly covers (x_s, y_s)
   and M_k is east of (x_s, y_s)
   and M_l's south-north span properly covers (x_d, y_d) **and** M_l is west of (x_d, y_d)
   and $T(g_S^{II})(k, l) = 1$
   then return NO;
   else goto 3;
3. return YES;

Refer to Fig. 11 again for an example. The condition check in Step 1 of algorithm *Manhattan* will turn out "true" with $i = 1$ and $j = 4$. Therefore, there will be no Manhattan route from s to d.

3.2 Implementation Issues and Cost Analysis

For given MCC set $S = \{M_1, M_2, \ldots, M_n\}$, g_S^I, $T(g_S^I)$, g_S^{II}, and $T(g_S^{II})$ are all computed in a preprocessing phase. It is easier

to appoint one processor to compute g_S^I (g_S^{II}). The information needed to carry out this computation is the two "corners" (U_i, L_i) of all MCCs. A simple check of U_i against the west-east spans of all other MCCs will work out one arrow of G_S^I. Therefore, the time complexity to compute g_S^I (g_S^{II}) is $O(n^2)$, where n is the number of MCCs. After g_S^I (g_S^{II}) is established, $T(g_S^I)$ ($T(g_S^{II})$) can be computed using Floyd-Warshall algorithm with complexity $O(n^3)$.

The existence algorithm for a pair of source/destination can be run by any node. Naturally, it can be run by the source node. For a node to run the existence algorithm, the two matrices $T(g_S^I), T(g_S^{II})$ must be residing at the node. Also needed at each node (x, y) are the ids of MCCs whose east-west/north-south spans properly cover (x, y). To get this information, (x, y) simply sends out four signals in parallel along the four directions. We discuss in detail how to collect information of MCCs *above* (x, y). (For the other three directions, the process is similar.) (x, y) sends out a request signal toward north. If an MCC is encountered, the signal goes around the boundary of the MCC to restore its north track. At the same time, the id of the encountered MCC is sent back to (x, y) along the path just traveled by the signal. The north-bound signaling/reporting continues until the boundary of mesh is reached. The information collection process, like the computation of g_S and $T(g_S)$, is performed in the preprocessing phase.

When a node (x_s, y_s) is about to perform the existence algorithm for node (x_d, y_d), it needs to know the ids of MCCs under (x_d, y_d). There are two possible ways this information can be made available. One way is use the idea of *global fault information*, i.e., the information of MCCs to a node's east/west/north/south directions is made known to all other nodes. The global information is convenient for any routing algorithm. However, since every node has to store the covering MCC information of all other nodes, the overhead is quite large. For that reason, many known routing algorithms choose not to use global fault information. Another approach is get this information *on-demand*. That is, in the beginning, all nodes know nothing about other nodes' covering MCCs. When (x_s, y_s) wants to perform existence algorithm for node (x_d, y_d), it first sends a small signal to (x_d, y_d) along a path not necessarily minimal. Upon receiving the signal, (x_d, y_d) sends back to (x_s, y_s) the information about its covering MCCs. Once node (x_s, y_s) gets the information about (x_d, y_d)s covering MCCs, it is kept in (x_s, y_s) for possible later use.

Once the information about (x_d, y_d)s covering MCCs is available at (x_s, y_s), algorithm *Manhattan* can be performed by 1) checking $T(g_S^I)$ for an MCC directly above (x_s, y_s) and all MCCs below (x_d, y_d) and 2) checking $T(g_S^{II})$ for an MCC directly east of (x_s, y_s) and all MCCs west of (x_d, y_d). The complexity is bounded by $O(n)$, where n is the number of MCCs.

4 Minimal Routing Algorithms

If algorithm *Manhattan* returns a YES answer to the question "Does there exist a minimal route from (x_s, y_s) to (x_d, y_d)?", the next step would be to work out such a route. Without loss of generality, we assume $(x_s, y_s) = (0, 0)$ and $x_d, y_d \geq 0$. There are two schemes to do the minimal routing. One scheme initiates the routing at source node (x_s, y_s). That is, node (x_d, y_d) has no sense it is the destination of a message (or messages) sent by another node. Therefore, (x_d, y_d) is not involved in the whole routing process. The other scheme involves (x_d, y_d) in the routing process. The routing is initiated at both (x_s, y_s) and (x_d, y_d). In the first scheme, to assist in minimal routing, some relevant MCC information has to be distributed in certain nodes or a minimal route cannot be guaranteed. In the second scheme, no MCC information needs to be distributed in any node. However, a small premessage has to be sent to (x_d, y_d) (via a path not necessarily minimal) to inform it to start routing together with (x_s, y_s). If in existence algorithm *Manhattan*, (x_s, y_s) has sent a signal to (x_d, y_d) (for request of (x_d, y_d)s covering MCCs), then there is no need to send a premessage again —(x_d, y_d) already knows it will be the destination of a routing process.

4.1 Routing Initiated at s

The idea of the algorithm is to always look for a Type-I gap with respect to d. When a node is met that belongs to a Type-I gap, the route can just go along the gap to reach d. The problem is, however, since d does not know it is the target of a routing, it will not establish a Type-I gap.

The solution is to preestablish Type-I gap information for an L'-corner with respect to all its "potential targets." This is a preprocessing, performed one time after the formation of MCC set. Once established, it can be used to guide all future routings. The preprocessing performs a south-first-west-second traversal from all nodes: A node always sends a signal to its south neighbor as long as it can, until an MCC boundary is hit, at which time it sends a signal to its west neighbor along the boundary of MCC. The signal just contains the coordinates of the originator node, say (x, y). When a corner node L' is reached, (x, y) will be deposited in L'. At the end of preprocessing, the L' contains *every* (x, y) with respect to which L' belongs to the Type-I gap.

As illustrated in Fig. 12, the locations of all white nodes are stored in L'_j, which means L'_j lies on the Type-I gap of any white node. Similarly, L'_i lies on the Type-I gap of any black or white node. An L'-node then distributes its target list southward along vertical direction. The distribution stops when a node of MCC is met.

The size of target list introduced above can be greatly reduced so that the search time of the target list can be minimized. For example, instead of storing every individual target node, we can store a vertical range of targets. See Fig. 13 for illustration. The routing procedure is described below.

The routing begins at source s.

1. Go east, until a node c is met that contains a target list or an MCC is hit. If a *c*-node is met, goto 2a, if an MCC is hit goto 2b.
2.
   a. A *c*-node is met. Search the target list to see if it contains d. If YES (the *c*-node is on the gap leading to *d*), run *Gap_Traversal* and terminate; if NO goto 1.

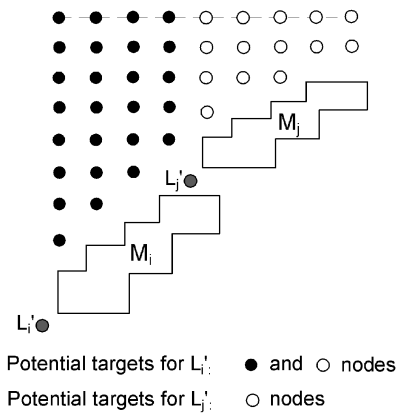


Fig. 12. Corner stores "potential targets" information.

b. An MCC M_i is hit. Go along the northwest (NW) boundary of M_i; then goto 1.

(Note that Step b is actually the construction of a Type-II gap.)

The *Gap_Traversal* process is described as follows: First, go north toward the L'-node. Starting at the L'-node, go along the NW boundary until the upper corner U' is reached. Then, go east, until hitting the NW boundary of another MCC. Repeat above boundary-east-boundary-east routing until $x = x(d)$ is reached. Then, go north to reach d. The success of the *Gap_Traversal* process is guaranteed by the fact that its starting node is indeed on the (Type-I) gap with respect to d. Also, the existence of a gap guarantees that the gap will be reached sooner or later.

Note that, as soon as the gap node is hit, there will be no more target list search. Before reaching the gap, the search is only performed at several "crucial" nodes. The major cost for this approach is the storage of search lists.

4.2 Routing Initiated at Both s and d

This routing scheme is based on the second part of the proof for Theorem 2.1. It proceeds by working out the Type-I gap (from destination) and Type-II gap (from source) defined in the proof.

The routing process is begun at both s and d.

From node s:

- Initialization: s marks itself as "s-route-node."
- A newly marked s-route-node always sends a signal to its east neighbor as long as it can (i.e., as long as the east neighbor does not belong to an MCC).
- A node receiving a signal marks itself as s-route-node.
- If a newly marked s-route-node cannot send east ("hitting an MCC"), it sends signal to its north neighbor (going along the boundary of MCC).
- The preceding process is repeated until a node $a = (a_x, a_y)$, where $a_x = x_d$, is marked s-route-node for the first time. (Being able to reach such a node is guaranteed by Theorem 2.1.)

From node d:

- Initialization: d marks itself as "d-route-node."
- A newly marked d-route-node always sends a signal to its south neighbor as along as it can.
- A node receiving a signal marks itself as d-route-node.
- If a newly marked d-route-node cannot send southward, it sends signal to its west neighbor (going along the boundary of MCC).
- The preceding process is repeated until a node $b = (b_x, b_y)$, where $b_y = 0$, is marked d-route-node for the first time. (Being able to reach such a node is guaranteed by Theorem 2.1.)

Final construction:

The Manhattan route from s to d can be constructed as follows: Starting from s, the route goes along the s-route-nodes. When it reaches the node that was marked as both s-route-node and d-route-node, the route goes along the "upper-east" part of d-route-nodes. Fig. 14 illustrates the routing process.

5 Conclusion

In this paper, we proposed a rectilinear-monotone shaped fault block model for fault-tolerant adaptive routing in mesh interconnection networks. This model improves the widely used rectangular model by taking into consideration the positions of source and destination relative to faulty nodes in the process of constructing fault blocks. It has been shown that the MCC model is a finer grouping of faulty nodes than rectangular model. Many fewer nonfaulty nodes are included in fault blocks. Many nonfaulty nodes that would have been included in rectangular fault blocks now can be used in routing, thus increasing the chances of

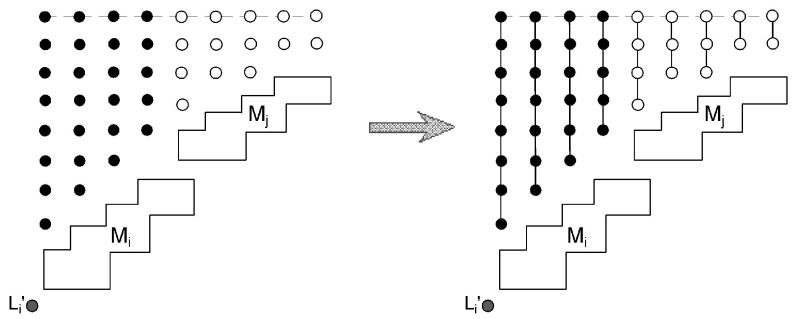


Fig. 13. We can store a vertical range of targets to reduce space cost and search time.

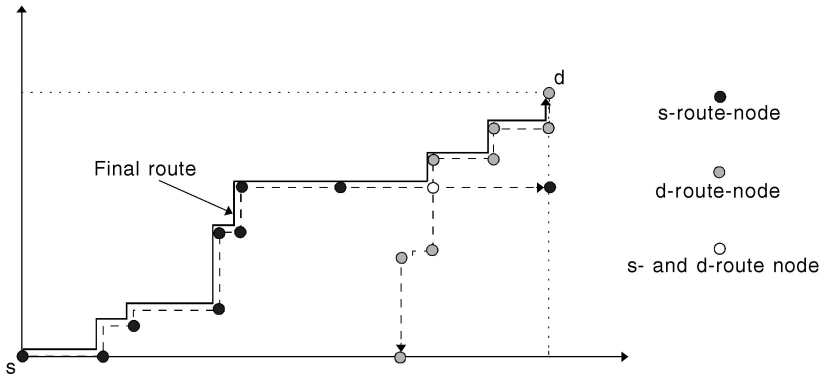


Fig. 14. Manhattan routing: the dashed lines show the "s-route" and "d-route." The solid line is the constructed Manhattan route.

obtaining optimal routes among faulty nodes. We proposed a sufficient and necessary condition for checking the existence of the minimal Manhattan routes in presence of such fault blocks. Based on the condition, an algorithm was proposed to determine the existence of Manhattan routes in a faulty mesh. We also gave two adaptive routing algorithms that construct a Manhattan route avoiding all fault blocks.

If the determination algorithm returns a NO answer for the faulty mesh, we still need to find a route, preferably as short as possible. A natural direction in which the work of this paper can be extended is to find a shortest route among MCC fault blocks if a minimal route does not exist.

REFERENCES

[1] R.V. Boppana and S. Chalasani, "Fault-Tolerant Wormhole Routing Algorithms for Mesh Networks," *IEEE Trans. Computers,* vol. 44, no. 7, pp. 848-864, July 1995.

[2] Y.M. Boura and C.R. Das, "Fault-Tolerant Routing in Mesh Networks," *Proc. 1995 Int'l Conf. Parallel Processing,* pp. I 106-I 109, 1995.

[3] A.A. Chien and J.H. Kim, "Planar-Adaptive Routing: Low Cost Adaptive Networks for Multiprocessors," *Proc. 19th Int'l Symp. Computer Architecture,* pp. 268-277, 1992.

[4] G.M. Chiu and S.P. Wu, "Fault-Tolerant Routing Strategy in Hypercube Multicomputers," *IEEE Trans. Computers,* vol. 45, no. 2, pp. 143-155, Feb. 1996.

[5] W.J. Dally, "The J-Machine: System Support for Actors," *Actors Knowledge-Based Concurrent Computing,* C. Hewitt and G. Agha, eds., MIT Press, 1989.

[6] D. Estrin, R. Govindan, J. Heidemann, and K. Kumar, "Next Century Challenges: Scalable Coordination in Sensor Networks," *Proc. IEEE/ACM Mobicom '99,* pp. 263-270, 1999.

[7] P.T. Gaughan, B.V. Dao, S. Yalamanchili, and D.E. Schimmet, "Distributed, Deadlock-Free Routing in Faulty, Pipelined, Direct Interconnection Networks," *IEEE Trans. Computers,* vol. 45, no. 6, pp. 651-665, June 1996.

[8] G.J. Glass and L.M. Ni, "Fault-Tolerant Wormhole Routing in Meshes without Virtual Channels," *IEEE Trans. Parallel and Distributed Systems,* vol. 7, no. 6, pp. 6201-636, June 1996.

[9] T.C. Lee and J.P. Hayes, "A Fault-Tolerant Communication Scheme for Hypercube Computers," *IEEE Trans. Computers,* vol. 41, no. 10, pp. 1242-1256, Oct. 1992.

[10] A.C. Liang, S. Bhattacharya, and W.T. Tsai, "Fault-Tolerant Multicasting on Hypercubes," *J. Parallel and Distributed Computing,* vol. 23, no. 3, pp. 418-428, Dec. 1994.

[11] R. Libeskind-Hadas and E. Brandt, "Origin-Based Fault-Tolerant Routing in the Mesh," *Proc. First Int'l Symp. High Performance Computer Architecture,* pp. 102-111, 1995.

[12] S.L. Lillevik, "The Touchstone 30 Gigaflop DELTA Prototype," *Proc. Sixth Distributed Memory Computing Conf.,* pp. 671-677, 1996.

[13] C.L. Seitz, "The Architecture and Programming of the Amete Series 2010 Multicomputer," *Proc. Third Conf. Hypercube Concurrent Computers and Applications,* pp. I 33-I 36, Jan. 1988.

[14] C.C. Su and K.G. Shin, "Adaptive Fault-Tolerant Deadlock-Free Routing in Meshes and Hypercubes," *IEEE Trans. Computers,* vol. 45, no. 6, pp. 666-683, June 1996.

[15] Y.-J. Suh, B.V. Dao, J. Duato, and S. Yalamanchili, "Software Based Fault-Tolerant Oblivious Routing in Pipelined Networks," *Proc. 1995 Int'l Conf. Parallel Processing,* pp. I 101-I 105, 1995.

[16] J. Wu, "Reliable Unicasting in Faulty Hypercubes Using Safety Levels," *IEEE Trans. Computers,* vol. 46, no. 2, pp. 241-247, Feb. 1997.

[17] J. Wu, "Fault-Tolerant Adaptive and MinimalRouting in Mesh-Connected Multicomputers Using Extended Safety Levels," *IEEE Trans. Parallel and Distributed Systems,* vol. 11, no. 2, pp. 149-159, Feb. 2000.

Dajin Wang received the BEng degree from Shanhai University of Science and Technology, China, in 1982, the MS degree and the PhD degree, both in computer science, from the Stevens Institute of Technology, Hoboken, New Jersey, in 1986 and 1990, respectively. Since then, he has been with the faculty of the Department of Computer Science at Montclair University, Upper Montclair, New Jersey, where he is currently a professor. He has also been a visiting research professor at the State Key Laboratory for Novel Software Technology at Nanjing University, China, and has held the position of visiting scholar at Hong Kong Polytechnic University. His research interests include fault-tolerant computing, algorithmic robotics, and parallel processing. He is a member of the IEEE.