A Fault-Tolerant Tree-Based Multicasting in Mesh Multicomputers

Jie Wu and Xiao Chen
Department of Computer Science and Engineering
Florida Atlantic University
Boca Raton, FL 33431
{jie, xchen}@cse.fau.edu

Abstract

We propose a fault-tolerant tree-based multicast algorithm for 2-dimensional (2-D) meshes based on the concept of the extended safety level which is a vector associated with each node to capture fault information in the neighborhood. In this approach each destination is reached through a minimum number of hops. In order to minimize the total number of traffic steps, three heuristic strategies are proposed. This approach can be easily implemented by pipelined circuit switching (PCS). A simulation study is conducted to measure the total number of traffic steps under different strategies. Possible extensions to 3-D meshes and deadlock-free multicasting are also included. Our approach is the first attempt to address the fault-tolerant tree-based multicast problem in 2-D meshes based on limited global information with a simple model and succinct information.

Keywords: *fault tolerance, faulty block, mesh, minimal routing, multicast, pipelined circuit switching (PCS), safety level*

1 Introduction

In many multicomputer systems, data must be redistributed periodically in such a way that all processors can be kept busy performing useful tasks. Because they do not physically share memory, nodes in multicomputers must communicate by passing messages through a communication network. Communication in multicomputers can be either *point-to-point* or *collective*. In point-to-point communication, only a single source and a single destination are involved. In collective communication, more than two nodes are involved in the communication. Examples of collective communication include *multicast*, *broadcast*, and *barrier synchronization*. The growing interest in the use of such routines is shown by their inclusion in many commercial communication libraries and in the *Message Passing Interface* (MPI) [23], an emerging standard for communication routines used by message-passing systems.

A multicast (one-to-many communication) facility sends messages from one node to multiple nodes. Multicast is an important system-level collective communication service. Several collective communication services such as broadcast and scatter in MPI are a subset or a derivation of multicast. Multicast is also essential in many other applications such as clock synchronization in distributed systems and cache coherency in distributed shared-memory systems. Due to the importance of multicast, efficient implementation of multicast has been extensively studied in the past ([1], [2], [4], [16], [22]). Some vendors are aware of the importance of multicast and have facilitated it by implementing multicast directly in hardware.

In a multicomputer system with hundreds and thousands of processors, fault tolerance is another issue which is defined as the ability of the system to function in the presence of component (processor or communication link) failures. The challenge is to realize fault tolerant communication without the expense of considerable performance degradation.

Multicast schemes can be classified into unicast-based, path-based, and tree-based. The unicast-based approach treats a multicast as a multiple-unicast. A unicast is a one-to-one communication. If there are n destinations in a multicast set, n worms are generated in a wormhole-routed system. Each worm goes to its destination separately as if there were n-unicast communications. As a result, substantial start-up delay could occur at the source. In the path-based approach, a path including all the destinations should be first established. The path-based approach uses only one or two worms which include all the destinations in the header(s). Each node is assumed to be able to store a copy of an incoming message (flit) and at same time forward it to the next node on the path. Like the path-based approach, each node in the tree-based approach is capable of storing an incoming message and forwarding it. In addition, it can split and replicate the message. In this way, the original worm is changed into a worm with multiple headers. Such multi-head branches can be dynamically generated at some intermediate nodes. It is believed that the tree-based approach offers a cost-effective multicasting [28].

Although many tree-based multicast algorithms have been proposed for store-and-forward networks, extensions to wormhole routing is still a challenging problem [20]. Our tree-based approach is based on a relatively new switching technique called *pipelined circuit switching* PCS [8] and multicast-PCS [31]. In a PCS (multicast-PCS), header (multiheader) flits are transmitted first during the path (tree) set-up phase. Once a path (tree) is reserved by the header, an acknowledgement is sent back to the source. When the source receives the acknowledgement, data flits are transmitted through the path (tree) in a pipelined fashion.

Unlike regular PCS that allows backtracking, our approach always establishes a minimal path to each destination. In our approach, fault information of a fault (faulty block) is distributed to limited number of nodes in the neighborhood so that multiheader flits can avoid the fault before reaching it. If the source satisfies certain conditions, our approach can set-up a multicast tree such that each destination (a leave node in the tree) is reachable through a minimal path in the tree. It is well-known that constructing a multicast tree with a minimum number of links (also called traffic steps) is an NP-complete problem. We present three heuristic strategies to minimize the total number of traffic steps. Our approach is illustrated using a 2-dimensional (2-D) mesh. The 2-D mesh topology is one of the most thoroughly investigated network topologies for multicomputer systems. It is important due to its simple structure and its good performance in practice and is becoming popular for reliable and high-speed communication switching. The multicomputers that use 2-D meshes including the Symult 2010 [26] and the Intel Touchstone [15]. Our approach can also be applied to 3-D mesh (the MIT J-machine [5]) and tori (Cray T3D [13] and Cray T3E []) which are meshes with wrap around links.

The rest of the paper is organized as follows. Section 2 reviews the related work. Section 3 introduces the notation and preliminaries. Section 4 proposes a multicast algorithm including three strategies. Section 5 discusses several results related to the proposed algorithm. Section 6 presents our simulation results. Section 7 discusses possible extensions to 3-D meshes and assurance of deadlock-freedom. Concluding remarks are made in Section 8.

2 Related Work

Multicast schemes can be classified into unicast-based, path-based, and tree-based. Unicast-based multicasting requires multiple start-ups (each of which needs several microseconds) while the channel propagation latency is on the order of nanoseconds. We discuss here only path-based and tree-based approaches.

In the path-based approach [17], a path including all the destinations should be first established.

The path-based approach uses only one or two worms which include all the destinations in the header(s). Each node is assumed to be able to store a copy of an incoming message (flit) and at same time forwards it to the next node on the path. Hamiltonian path-based routing is a path-based routing using Hamiltonian path. Eulerian trail-based routing [25] is a special path-based approach in which a Eulerian trail is established connecting all the channels in the network and each channel appears once and only once. To avoid generating an extremely long path, multiple paths can be constructed at the source that combine to reach all destinations [21]; however, multiple start-ups are needed. Among fault-tolerant path-based multicasting, Tseng et al's [30] Euler-path-based multicasting is based on constructing a Eulerian trail in faulty meshes with regular and even irregular fault patterns. Liberskind-Hadas et al's model [12] further relaxes the condition by constructing a pseudo-Hamiltonian path which visits each node at least once and each channel at most once. Both approaches require each node have global fault information (they are called *global-information-based multicasting*).

Tree-based multicasting has traditionally been considered a good mechanism for multicasting in store-and-forward networks to reduce the overall length from the source to destinations. However, extension to wormhole routing is still a challenging problem [20]. A routing algorithm that is proved to be deadlock-free in unicasting may still subject to deadlock in multicasting in a wormhole-routed system. The up*/down* routing [27] is such an example, although a restricted version of up*/down* routing for multicasting [11] has been proved to be deadlock-free. Deadlock is avoided by partitioning the network into two pre-defined subnetworks, A and B, with B being a tree. The multicast tree is constructed by visiting A first followed by B with the constraint that no branch (of the multicast tree) can be constructed before reaching subnetwork B.

Duato et al [19] presented a simple tree-based multicast algorithm for short messages which can be completely stored in a router. In this case, two branches in a multicast tree can be made independent to each other. Therefore, as long as the corresponding unicasting is deadlock-free, the multicast algorithm is also deadlock-free. Wang and Blough's approach [31] avoids deadlock through dynamic channel reservation using a relatively new switching technique called *pipelined circuit switching* PCS [8] and multicast-PCS [31]. In a PCS (multicast-PCS), header (multiheader) flits are transmitted first during the path (tree) set-up phase, just like a circuit switching, once acknowledgment is received at the source, data flits are transmitted through the path (tree) in a pipelined fashion, like a wormhole switching. A multicast may not be able to reserve a multicast tree, upon receiving a negative acknowledgment, the source will attempt to make a reservation at a later time.

During the path (tree) set-up phase, PCS (multicast-PCS) allows backtracking which may cause substantial delays. One reason for backtracking is the existence of faults in the system. Backtracking is sometimes unavoidable especially when each processor (node) knows only status of adjacent

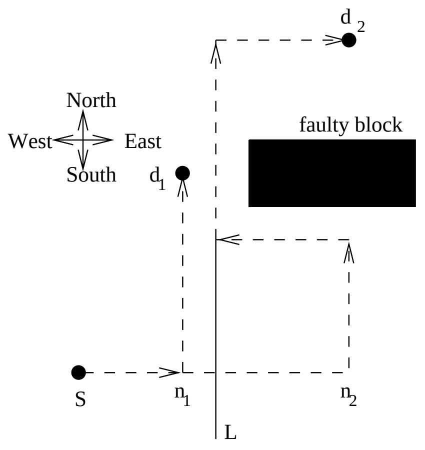


Figure 1: An example of multicast-PCS.

nodes (such approaches are called *local-information-based multicasting* [8]). The channel reservation algorithm in [31] follows the normal searching approach where each direction is attempted in a predefined order, say East, North, West, and South as in a 2-D mesh. In [8] history information is kept within the router to ensure that a particular physical path will not be searched twice. In [31], no history information needs to be kept if the search process following the order described below. Upon arrival of a header flit at a router, the possible output channels are attempted using the following rules. For forward header flits, the first channel searched is the one that has the same dimension and same direction as the incoming channel and then the next one in the order (wrapping around if needed). For a header flit entering router from local node, the starting channel is almost the one direction in the order, that is East. Finally, *profitable channels* (ones which would move a header flit closer to its destination) are attempted first according to the fixed order and then if needed, non-profitable channels are searched following the same order. (The rule for backtracking is omitted here.)

Consider an example shown in Figure 1 where a multicast tree is generated based on the above algorithm which connects s to two destination d_1 and d_2. Note that outgoing channels toward the East direction are reserved first. In this case, East and North are profitable directions for both d_1 and d_2. Once a node (n_1 for d_1 and n_2 for d_2) is reached where the East direction is no longer profitable for a destination, the output channel toward the North direction is attempted. Unfortunately, part of the North-directed channels from n_2 to d_2 are blocked by a faulty block. A detour path is generated that goes around the faulty block to reach destination d_2. Cases for backtracking are more complex and they could occur even when there is no fault but all output channels have been reserved by other multicasting.

In our approach, fault information of a fault (faulty block) is distributed to limited number of nodes in the neighborhood so that multiheader flits can avoid the fault before reaching it. In the example of Figure 1, fault information (about the faulty block) is distributed to nodes along the adjacent line L which is one unit distance away from the faulty block so that the searching process for d_2 will never enter a detour region (the region directly to the south of the faulty block). Actually, our approach tries to share a common path for all the destinations in a multicast set as much as possible without generating another tree branch. Because fault information is distributed to limited number of nodes, our approach is called *limited-global-information-based* multicasting which is a compromise of local-information-based approach and global-information- based approach.

In this paper we show that once the source satisfies certain conditions, a multicast tree can be set up such that each destination (a leave node in the tree) can be reached through a minimal path in the tree. The minimal path feature ensures that the corresponding unicasting is deadlock-free; therefore, during the set-up phase, a header is allowed to wait for a channel as long as the original header has not generated a new branch.

3 Notation and Preliminaries

3.1 K-ary n-dimensional meshes

A k-ary n-dimensional (n-D) mesh with k^n nodes has an interior node degree of $2n$ and the network diameter is $k(n-1)$. Each node u has an address $(u_1, u_2, ..., u_n)$, where $u_i = 0, 1, \cdots, k-1$. Two nodes $(v_1, v_2, ..., v_n)$ and $(u_1, u_2, ..., u_n)$ are connected if their addresses differ in one and only one dimension, say dimension i; moreover, $|v_i - u_i| = 1$. Basically, nodes along each dimension are connected as a linear array. Each node in a 2-D mesh is simply labeled as (x, y).

Routing is a process of sending a message from a source to a destination. A routing is *minimal* if the length of the routing path from the source to the destination is the minimal distance between these two nodes. For example, a routing is minimal between (x_1, y_1) and (x_2, y_2) if the length of its path is $|x_1 - x_2| + |y_1 - y_2|$. In a system with faults, minimal routing may not be possible if all the minimal paths are blocked by faults. A multicasting is minimal if the length of the routing path from the source to each destination is the minimal distance between these two nodes.

The simplest routing algorithm is *deterministic* which defines a single path between the source and the destination. The X-Y routing in 2-D meshes is an example of deterministic routing in which the message is first forwarded along the X dimension and then routed along the Y dimension. *Adaptive* routing algorithms, on the other hand, support multiple paths between the source and the destination. *Fully adaptive minimal* routing algorithms allow all messages to use any minimal

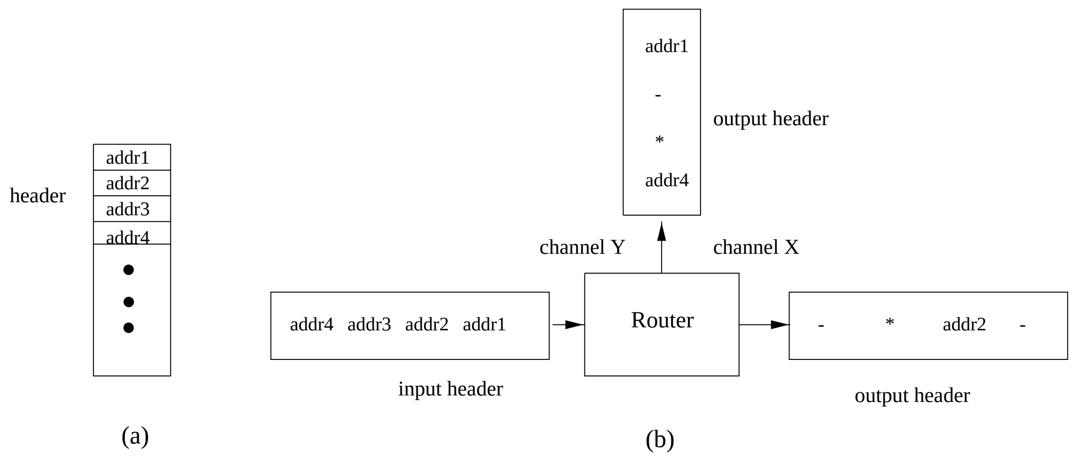


Figure 2: (a) header structure and (b) asynchronous multiheader routing

paths. In addition to the optimality requirement, we try to maintain maximum adaptivity in the routing process.

3.2 Multicast-PCS

Each multicast message has two parts: *message header* and *message information*. Message header contains the current list of destination addresses (also called a *multicast set*). To support the tree-based multicast algorithm in this paper, we use the basic pipelined circuit switching mechanism (PCS) [8] and the asynchronous multiheader routing in multicast-PCS.

At an intermediate node, which can be either a destination node or a *forwarding node* not in the multicast set, during the set-up phase, if the header must be split into two in order to reach different destinations, asynchronous multi-header routing [31] is used. When the header reaches a router, the router makes the routing decision and deposits each split address (now becomes a new header) from the original header into one output channel (either X-directed channel or Y-directed channel), the other output channels reserved for the same multicast are idle. We fill this idle time with a special symbol - as shown in Figure 2 which illustrats an example of asynchronous multiheader routing. When one destination matches the local processor, teh corresponding addresses is replaced by another symbol *, as shown in Figure 2 where $addr_3$ is the local processor. The reason for using two different symbols is to avoid deadlock which will be discussed later.

Our approach here differs from Wang and Blough's multicast-PCS. During the set-up phase, only profitable channels are reserved, i.e., neither backtracking nor detouring is allowed. However, a header is allowed to wait for a busy profitable channel as long as the original header has not generated a new tree branch, i.e., the header does not contain symbol - [31]. Otherwise, a negative acknowledgment is sent back to the source.

In the multicast tree, all the directed channels that are away from the source are p-channels (positive channels); otherwise, they are n-channels (negative channels). When a negative acknowledgment is sent back through the n-channels of the multicast tree established so far, each channel is released as the acknowledgment passes. When a node (in the tree) receives a negative acknowledgment through an incoming channel (n-channel or p-channel), it releases the corresponding channel (which is a pair, one n-channel and one p-channel) and forwards the negative acknowledgment to all its outgoing (reserved) channels. When a header becomes empty, i.e., it reaches a leave node of the multicast tree, it sends a positive acknowledge through an outgoing n-channel. When a node receives a positive signal (through an incoming n-channel), it forwards the signal to its outgoing n-channel. Note that each node in the tree has one outgoing n-channel unless it is the source or the channel has been released by a negative acknowledgment. In the later case, the branch rooted at the node will be eventually released. During the channel release process, the original tree is reduced to a set of disjoint subtrees, with each having at least one negative acknowledgment.

In a faulty mesh, when all the minimal paths from a source to a destination are blocked by faults, no multicast tree can be established by our approach. We should provide a simple mechanism so that the source can easily detect this situation and stop attempting to establish minimal paths. In this case, the source can switch back to the normal multicast-PCS, like the one proposed in [31]. In addition, another mechanism is needed to prevent the header from reaching a region where a destination cannot be reached through a minimal path. These two mechanisms are discussed in the next section.

3.3 Extended Safety Levels

Let's first discuss the fault model used in our approach. Most literature on fault-tolerant routing in 2-D meshes uses disconnected rectangular blocks ([1], [2], [3], [10], [29]) to model node faults (link faults are treated as node faults) to facilitate routing in 2-D meshes. First, a node labeling scheme is defined and this scheme identifies nodes that cause routing difficulties. Adjacent nodes with labels (including faulty nodes) form faulty rectangular regions [2].

Definition 1: *In a 2-D mesh, a healthy node is* disabled *if there are two or more disabled or faulty neighbors in different dimensions. A* faulty block *contains all the connected disabled and faulty nodes.*

For example, if there are three faults $(1,1)$, $(2,2)$, and $(4,2)$ in a 2-D mesh, two faulty blocks are generated. One contains nodes $(1,1)$, $(1,2)$, $(2,1)$, and $(2,2)$ and the other one contains $(4,2)$.

Each faulty block is a rectangle. The convex nature of a rectangle simplifies the routing process by avoiding backtracking during the set-up phase. The block fault model has the following interesting property: The distance between any two faulty blocks is at least two [34].

In our approach two types of limited global information are used: *safety information* and *faulty block information*. Safety information is used for the source to determine the feasibility of establishing a minimal path to each destination in a multicast set. Safety information is represented as a vector associated with each node. This vector includes four elements indicating the distance to the closest faulty block to the East, South, West, and North of the current node. Faulty block information is used to facilitate the process of setting up a multicast tree and it is stored in nodes that are along four adjacent lines of each faulty block. In the following we discuss each type of information one by one.

Safety Information. In a 2-D mesh with faulty blocks, we use node $(0,0)$ as the source node and (i,j) as one of the destinations with $i > 0$ and $j > 0$. Other cases can be treated in a similar way. There may not always exist a minimal path from the source to the destination. To facilitate the discussion of minimal unicasting and multicasting in 2-D meshes with faulty blocks, Wu [34] proved the following theorem.

Theorem 1 [34]: *Assume that node $(0,0)$ is the source and node (i,j) is the destination. If there is no faulty block that goes across the X or Y axis, then there exists at least one minimal path from $(0,0)$ to (i,j), i.e., the length of this path is $|i| + |j|$. This result holds for any location of the destination and any number and distribution of faulty blocks in a given 2-D mesh.*

Definition 2 [34]: *In a 2-D mesh, a node (x, y) is safe if there is no faulty block along the xth column or the yth row.*

Based on Theorem 1, as long as the source node is safe, minimal paths exist for each destination in any multicast set. To decide the safety status of a node, each node is associated with a safety vector (E, S, W, N) with each element corresponding to the distance to the closest faulty block directly to its East, South, West, and North, respectively. Alternatively, (E, S, W, N) can be represented as (+X, -Y, -X, +Y) where +X corresponds to the distance to the closest faulty block along the positive X direction. A node is safe if each element in the vector is an infinite number (a default value). The safety condition can be weakened while still guaranteeing optimality. Specifically, a source node $(0,0)$ is said to be *extended safe* to a destination (i,j) if and only if there is no faulty block along the north (+Y) and east (+X) directions within the rectangle formed by the source and the destination. Clearly, a minimal routing is possible if a given source is extended safe with respect to a given destination. A source node is said to be extended safe to a multicast set if it is extended safe to each destination in the set. Throughout the paper, we assume that the source is extended safe with respect to a given multicast set.

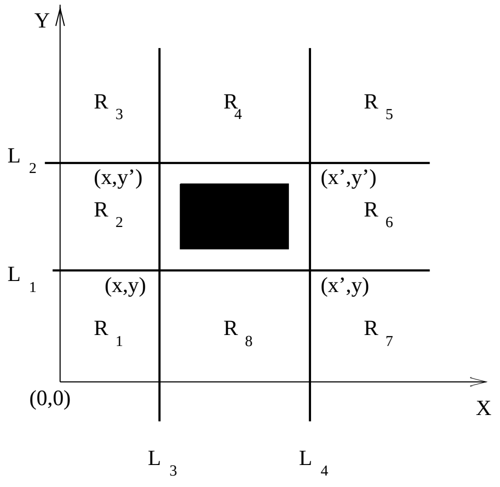


Figure 3: The four adjacent lines and eight regions of a faulty block

Faulty Block Information. Safety information of each node is used just to check the feasibility of establishing a minimal path from the source to each destination in a multicast set. In order to facilitate the channel reservation process by avoiding faulty blocks before reaching it, we need to distribute faulty block information to appropriate nodes. To minimize the distribution of fault information, the distribution is limited to nodes on four adjacent lines of each faulty block [34]. Figure 3 shows eight regions generated from the four adjacent lines of a faulty block. The four adjacent lines are parallel to the four sides of the faulty block, one-unit distance away. The limited global information (faulty block information) is kept on these four adjacent lines, except for nodes that are adjacent to the faulty block (since all nodes know their adjacent faulty blocks). Assume (x, y) is the coordinate of the intersection node of lines L_1 and L_3. (x, y'), (x', y') and (x', y) are the coordinates of the other intersection nodes of these four adjacent lines (see Figure 3). To obtain a minimal routing, a header should not cross L_3 from region R_1 (where the source is located) to R_8 if a destination is in region R_4 (see Figure 3). Simiarly, a header should not cross L_1 from region R_1 to R_2 if the destination is in region R_6. For each faulty block as shown in Figure 3, faulty information is stored at each node on L_1, i.e., the section between $(-\infty, y)$ and (x, y). Also, it is stored at each node between $(x, -\infty)$ and (x, y) on L_3. To minimize path information, only locations of two opposite corners of a faulty block are essential, say, (x, y) and (x', y') as shown in Figure 3. Note that teh address of a faulty block is given, each region can be easily determined. For example, R_4 can be represented as $x \leq X \leq x'$ and $y' \leq Y$ and R_6 as $x' \leq X$ and $y \leq Y \leq y'$. Note that the distribution of fault information along each line ca nbe treated as a multicasting. Since all the destinations can be reached through a minimal path without generationg any new branch, this process resembles a unicasting problem.

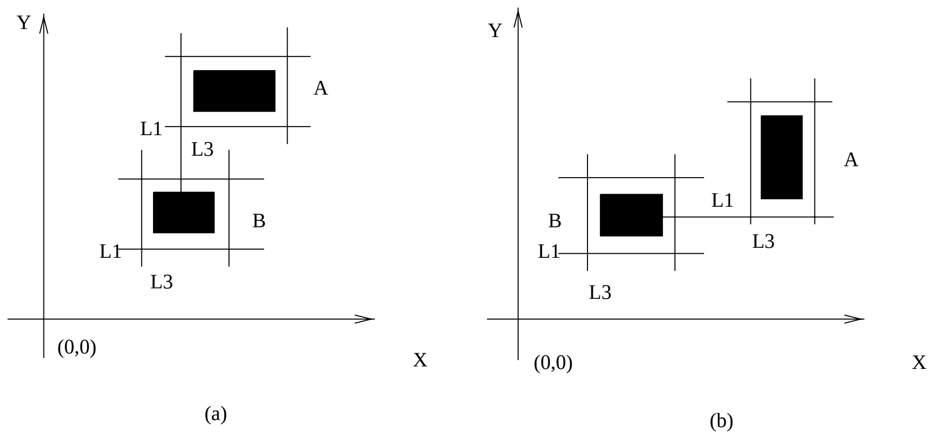


Figure 4: Two paths of multiple faulty blocks: (a) Two faulty blocks intersect with each other vertically and (b) Two faulty blocks intersect with each other horizontally

When there are multiple faulty blocks in the network, they may be intersected or independent. Two faulty blocks are *intersected* if one of the four adjacent lines of a faulty block intersects with another faulty block. In this case, faulty block information is transferred between these two blocks. In Figure 4 (a), the header should not cross L_3 of faulty block B (from R_1 to R_8) if a destination is in region R_4 of B or in region R_4 of A. We say fault information of faulty block A is transfered to nodes along line L_3 of faulty block B (because line L_3 of A intersects with faulty block B). However, there is no information transfer from A to B for nodes along line L_1 of B. Similarly, in Figure 4 (b), the header should not cross L_1 of faulty block B from region R_1 to region R_2 if a destination is in region R_6 of B or in region R_6 of A. However, there is no information transfer (from A to B) for nodes along line L_3 of B.

3.4 Unicasting in 2-D meshes with faulty blocks

In [34], Wu proposed the following unicast algorithm: the routing starts from the source, using any adaptive minimal routing until L_1 (or L_3) of a faulty block is met. Such a line can be either *noncritical* or *critical*. If the selection of two profitable channels, one along $+X$ and the other along $+Y$, does not affect the minimal routing, then the path is noncritical; otherwise, it is critical. L_1 (L_3) is critical to a multicast set if a destination in the multicast set is region R_6 (R_4). In case of noncritical, the adaptive minimal routing continues by randomly selecting a profitable channel. In case of a critical path, the selection should be done based on the relative location of the destination to the path:

- (L_1 is met) If the destination in region R_6, the header should stay on line L_1 until reaching node $(x^{'}, y)$ (the intersection of L_1 and L_4 of the faulty block); otherwise, the selection is random.

- (L_3 is met) If the destination is in region R_4, the routing message should stay on L_3 until reaching node $(x, y^{'})$ (the intersection of L_3 and L_2 of the faulty block); otherwise, the selection is random.

Multicasting can be considered as multiple minimal unicasts, i.e., each unicast is minimal optimal. To reduce traffic, messages intended for different destinations should share as many common path(s) as possible. In the next section, we propose a *minimal multicast* algorithm which is minimal unicasting for each destination and has as few number of traffic steps as possible.

4 Multicasting in 2-D Meshes with Faulty Blocks

4.1 Minimal multicast algorithm

In the set-up phase, the header is *2d-free* at a given position if the message can take either the $+X$ or $+Y$ direction in the next step; a message is *1d-free* if the message can only take the $+X$ or $+Y$ direction but not both in the next step; and a message is *in conflict* if the message should take both the $+X$ and $+Y$ directions in the next step. See Figure 5 (a) for an example, there are three destinations $d_1(x_1, y_1)$, $d_2(x_2, y_2)$ and $d_3(x_3, y_3)$ in a multicast set. Starting from source node $(0, 0)$, the next step should be taken only along the $+Y$ direction because there is a destination $d_3(x_3, y_3)$ on the Y axis. Therefore, the header at source node $(0, 0)$ is said to be 1d-free. At node v, the next step can be taken along either the $+X$ or $+Y$ direction, the header at node v is said to be 2d-free. At node u, there are destinations along both the $+X$ and $+Y$ directions. The next step should be taken along both the $+X$ and $+Y$ directions. Therefore a conflict occurs at node u. To solve this conflict, the header should be split into two: one gets destination address (x_1, y_1) of d_1 and the other gets destination address (x_2, y_2) of d_2. We then continue routing each message individually.

Figure 5 (b) shows another multicast example with destinations d_1, d_2, and d_3 in the multicast set. At node w the next step to take is along the $+X$ direction, so node w is 1d-free. At node v, either direction can be taken in the next step, so the message node v is said to be 2d-free. At node u, because on the critical line with respect to destination d_2. At the same time, node u it is on the critical line with respect to destination d_1. A conflict occurs at node u because d_1 requires that the next step to be taken along the $+Y$ direction and d_2 requires that the next step to be taken along the $+X$ direction. Note that fault information (two opposite corners of the faulty block) is stored

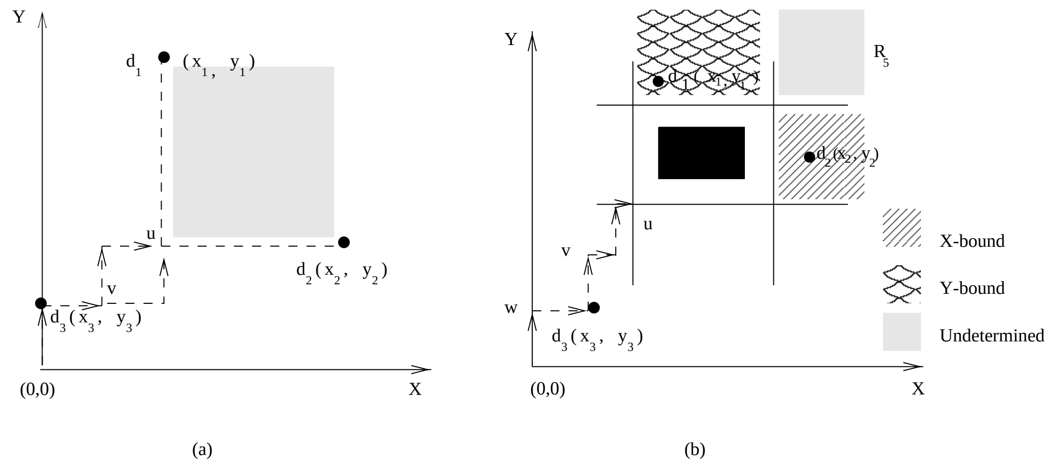


Figure 5: (a) An example of conflict in the next step and (b) Another example of conflict in the next step

at the router of u. It is easy to determine the region of each destination.

The following definition provides a formal definition of these concepts.

Definition 3: *A multicast header is X-bound (Y-bound) at node u if at least one of the following conditions is true:*

- *Node u has the same Y (X) coordinate as at least one of its destinations.*
- *Node u is on the L_1 (L_3) of a faulty block and it is a critical path of at least one of its destinations.*

Definition 4: *A multicast header at node u is in-conflict if it is both X-bound and Y-bound, 1d-free if it is either X-bound or Y-bound but not both, 2d-free if it is neither X-bound nor Y-bound.*

We will focus on the situation when a multicast header is in-conflict and the corresponding location (node) is called a *separating point.* To resolve a conflict, the message has to split into two. Each copy follows either the $+X$ or $+Y$ direction. At a separating point, some of the destinations should be grouped into the X-bound group or Y-bound group depending on which direction to take in the next step to ensure minimal steps for each destination. But for some destinations, this grouping can not be done in an obvious way at this point. These destinations are called *undetermined.*

In the following we examine several cases of separating points. We classify them based on the number of faulty blocks involved.

1. If separating point u does not involve any faulty block, u has the same X coordinate as some of the destinations and the same Y coordinate as some other destinations (see Figure 5 (a)). The destinations that have the same Y coordinate as the one for u are X-bound. Similarly, the destinations that have the same X coordinates as the one for u are Y-bound. All the other destinations (in the shadow region excluding the boundaries of Figure 5 (a)) are undetermined destinations.

2. If separating point u involves one (independent) faulty block, there are three cases.

   (a) u is on both L_1 and L_3 of the faulty block and both are critical. Destinations in R_4 are Y-bound and destinations in R_6 are X-bound the rest are undetermined.

   (b) u is on L_1 of the faulty block and it is critical, but not on L_3; however, some destinations and u have the same X coordinate. Destinations in region R_6 are X-bound and those having the same X coordinate as the one for u are Y-bound. The remaining destinations are undetermined.

   (c) u is on L_3 of the faulty block and it is critical, but not on L_1; however, some destinations and u have the same Y coordinate. Destinations in region R_4 are Y-bound and those having the same Y coordinate as the one for u are X-bound. The remaining destinations are undetermined.

3. If separating point u involves multiple faulty blocks, let's first consider three cases for each direction.

   - Y-bound:

     (a) One or more faulty blocks intersect with each other vertically and u is on path L_1 (it is critical) of a faulty block.

     (b) u and some destinations have the same Y coordinate.

     (c) Combination of the above two.

   - X-bound:

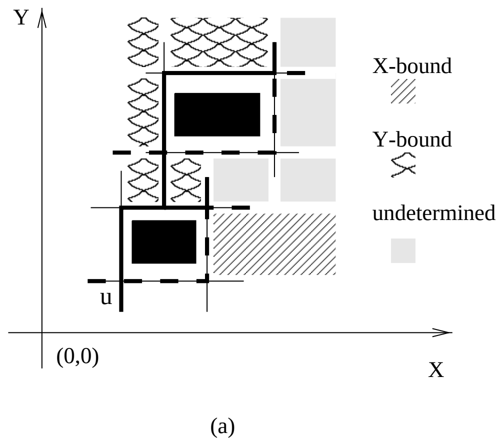
     (a) One or more faulty blocks intersect with each other horizontally and u is on path L_3 (it is critical) of a faulty block.

     (b) u and some destinations have the same X coordinate.

     (c) Combination of the above two.

   - Nine possible cases generated from combining one case from the Y-bound category and the other case from the X-bound category.

Figure 6 (a) shows a case of combining 3Y(a) and 3X(a) and Figure 6 (b) shows a case of combining 3Y(a) and 3X(b).



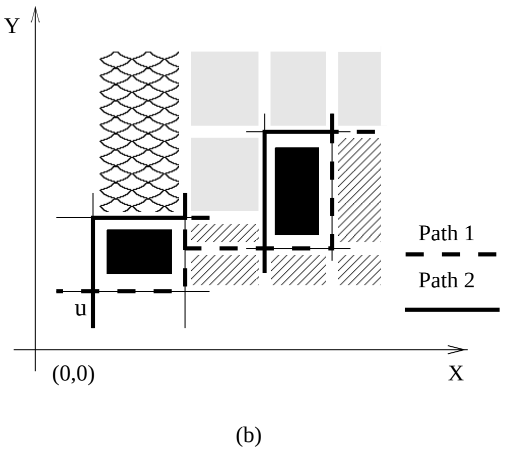


Figure 6: The X-bound, Y-bound and undetermined regions of separating point u if (a) u hits both X and Y paths and two faulty blocks intersect vertically and (b) u has the same X coordinate as some of the destinations and hits path 1 two faulty blocks intersect horizontally

Therefore, at a separating point u, each destination belongs to one of the three groups: X-bound, Y-bound and undetermined. In order to make a decision for undetermined destinations at a separating point, i.e., to place them in either the X-bound or Y-bound group, heuristic strategies have to be used. Originally, the multicast process starts at the source node.

Multicast algorithm for 2-D mesh with faulty blocks:

1. If the current node u is a destination, keep a copy of the message to its local memory and remove the current node from the message header. If the current node is a forwarding node, go directly to the next step.

2. If the message is in conflict at node u, i.e., it reaches a separating point, use one of three strategies (to be discussed in Section 3.3) to split the message. If the message is 1d-free at node u along the X (Y-direction), it should take the next step along the X (or Y) direction. If the message is 2d-free at node u, use a minimal adaptive routing algorithm to take the next step in either the X or Y direction.

3. Treat each message (new or old) at a next node as a new multicast with this next node as the new source. Repeat the above steps until each destination in the message header is reached.

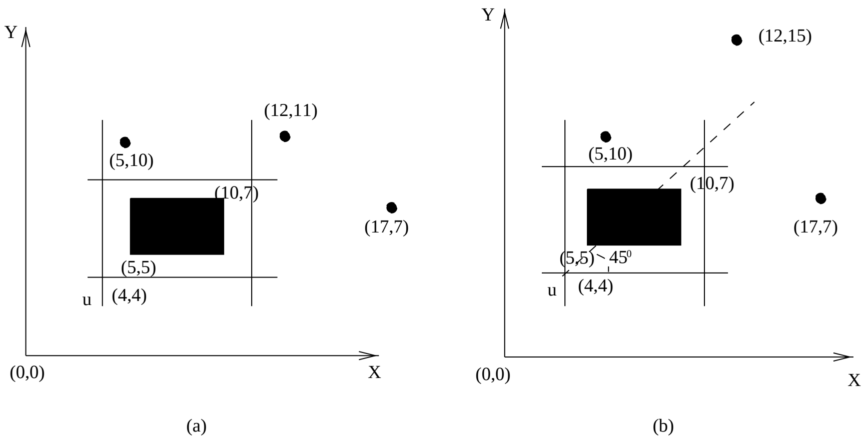


Figure 7: (a) An example of using Strategy 1 and (b) An example of generating bad traffic step using Strategy 2

4.2 Strategies to resolve a conflict

We propose three strategies to resolve a conflict at a separating point.

Strategy 1: *At a separating point u, X-bound destinations go along the X direction and Y-bound destinations go along the Y direction in the next step. For undetermined destinations, randomly pick a direction group (X- or Y-bound) to join.*

This strategy is simple, but it does not achieve good traffic step if most (or all) of the destinations are placed in the Y-bound group, but they are closer to the nodes in the X-bound group or vice versa. In Figure 7 (a), there is one faulty block with $(5,5)$ and $(10,7)$ as its two opposite corners. At point $u(4,4)$, destination $(17,7)$ will take the next step in the X-bound group and $(5,10)$ in the Y-bound group. For the undetermined destination $(12,11)$, if we use Strategy 1 to put it in the Y-bound group, the total number of traffic steps is the sum of the steps from the source $(0,0)$ to point u, from u to $(5,10)$, from $(5,10)$ to $(12,11)$, and from u to $(17,7)$, which is 39. Therefore the number of traffic steps can be reduced to 35 if $(12,11)$ joins the X-bound group. To reduce the number of traffic steps for this type of situations, we have the following Strategy 2.

Strategy 2: *At separating point $u(u_x,u_y)$, $+X$-bound destinations go along the X direction and Y-bound destinations go along the $+Y$ direction in the next step. For an undetermined destination $v(x_v,y_v)$, let $x_{off}=x_v-xu$ and $y_{off}=y_v-y_u$. If $x_{off}>y_{off}$, then place v in the X-bound group. If $x_{off}<y_{off}$, then place v in the Y-bound group. If $x_{off}=y_{off}$, then place v arbitrarily.*

According to Strategy 2, destination $(12,11)$ in the above example will join the X-bound group which results in fewer number of traffic steps. But this strategy is still not effective for cases like Figure 7 (b). In this figure, according to Strategy 2, the undetermined destination $(12, 15)$ should join the Y-bound group. The total number of traffic steps is 43. This number can be reduced to 39 if destination $(12, 15)$ joins the X-bound group. Note that Strategy 2 can be easily implemented in hardware. Basically, only two subtractors and one comparator needed in the router to determine the output channel for each undetermined destination.

If we take a closer look at the problem, the grouping problem leading to fewer number of traffic steps in addition to minimal multicasting resembles the *optimal multicast tree* (OMT) problem defined in [14]. To model the OMT problem, the graph model [9] can be used. Let graph $G(V, E)$ denote a graph with vertex (node) set V and edge (link) set E. When G is known from context, sets $V(G)$ and $E(G)$ will be referred to as V and E, respectively. A tree $T(V, E)$ is a connected graph that contains no cycles.

In our model, graph G is a 2-D mesh; however, tree T has to be defined differently. The following defines a *virtual tree* T in 2-D meshes.

Definition 5: *Let $T(V, E)$ be a virtual tree in a 2-D mesh, where a node $u(x_u, y_u) \in V(T)$ is a regular node (x_u, y_u) in the 2-D mesh. For any edge $(u, v) = ((x_u, y_u), (x_v, y_v)) \in E(T)$, it is a minimal path from $u(x_u, y_u)$ to $v(x_v, y_v)$ in the 2-D mesh, i.e., $|x_u - x_v| + |y_u - y_v|$. An edge in the virtual tree is called a virtual edge. A path in T is a sequence of virtual edges. For any two nodes u and v which may or may not be connected by an virtual edge in T, $dis_T(u, v)$ denotes the length (the number of edges) of a minimal path from u to v in T.*

Figure 8 (a) shows an example of a virtual edge between nodes u and v. Any node (represented by unfilled circles in the graph) in the rectangle formed by nodes u and v as two opposite corners can be on the minimal path.

For a multicast set, let d_0 denote the source node and d_1, d_2, $\cdots$, d_k denote k destination nodes, where $k \geq 1$. The set $K = \{d_1, d_2, \cdots, d_k\}$, which is a subset of $V(G)$, is called a *multicast set*. Each node d_i in the set has an address (x_i, y_i), $0 \leq i \leq k$ and G is the given 2-D mesh. The definition of the OMT problem is as follows:

Definition 6: *An optimal multicast tree (MT), $T(V, E)$, for multicast set K is a virtual tree of G such that*

1. *$\{d_0\} \cup K \subseteq V(T)$.*
2. *$dis_T(d_0, d_i) = dis_G(d_0, d_i)$, for $1 \leq i \leq k$.*
3. *$|E(T)|$ is as small as possible.*

Note that set $V(T) - K$ includes all the forwarding nodes of an MT. When $V(T) - K =$, the

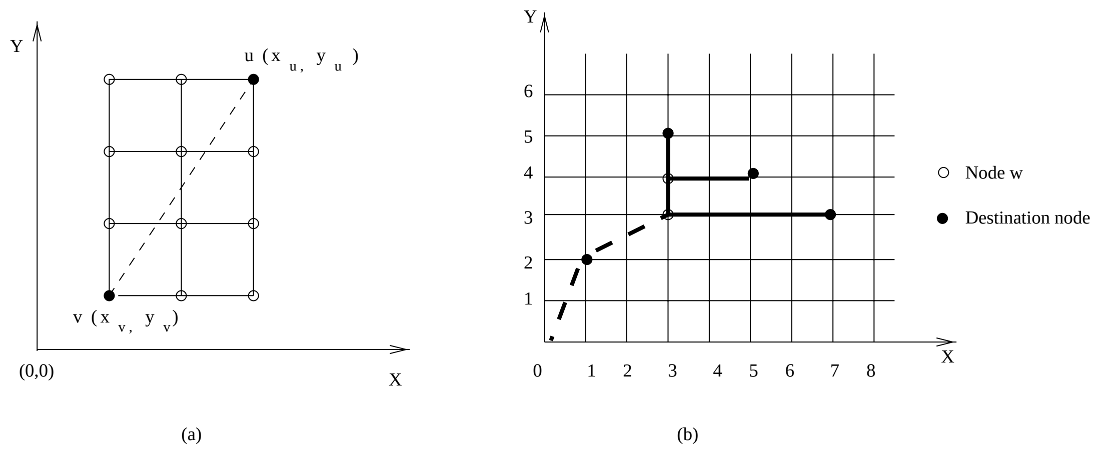


Figure 8: (a) The virtual edge between two nodes u and v and (b) The construction of an MT tree

corresponding MT contains destination nodes only. In the OMT problem, not only the number of time steps to each destination should be minimal but also the total number of traffic steps should be reduced as small as possible. The next question is how to construct such a minimal multicast tree (MT). The method we use is the greedy method derived from [16] in a system without faulty blocks. This greedy algorithm uses the concept of *split-and-sort* function to prepare a multicast. Then it is extended to cover cases with faulty blocks. In the original algorithm, the condition for time-step optimal is not required. The following algorithm makes some changes to achieve time-step optimal. Constructing an MT tree consists of two parts: the preparation part and the construction part. The MT tree constructed is represented by a virtual tree defined above.

Greedy algorithm:

(Preparation part):

Sort all the destinations $d_1, \cdots, d_k$ in ascending order with $dis_G(d_0, d_i)$ where G is a given 2-D mesh, $1 \leq i \leq k$, as the key. Without loss of generality, suppose $dis_G(d_0, d_1) \leq dis_G(d_0, d_2) \leq \cdots \leq dis_G(d_0, d_k)$ after sorting.

(Construction part):

1. Construct a virtual tree T with source d_0 as the root and by setting $V(T) = \{d_0, d_1\}$ and $E(T) = \{(d_0, d_1)\}$ initially.
2. Add the rest of nodes $d_i(x_i, y_i)$ $(2 \leq i \leq k)$ one by one to the tree as follows:
    (a) Among all $(u, v) \in E(T)$, find a $w(x_w, y_w)$ which satisfies the following conditions:
        i. on a minimal path from u to v,

ii. $w \leq d_i$, that is, $x_w \leq x_i$ and $y_w \leq y_i$, and

iii. $dis(d_i, w)$ is minimal.

(b) $V(T) \leftarrow V(T) \bigcup \{d_i\}$. If $w \notin V(T)$, then $V(T) \leftarrow V(T) \bigcup \{w\}$.

(c) If $w \neq u$ and $w \neq v$, then $E(T) \leftarrow E(T) \bigcup \{(u,w),(w,v)\} - \{(u,v)\}$.

(d) If $d_i \neq w$, then $E(T) \leftarrow E(T) \bigcup \{(w, d_i)\}$.

The basic idea of the MT construction part is to build an MT tree that always adds a closest remaining destination node to it until all the destination nodes are covered by the resultant MT tree.

Figure 8 (b) shows an example of applying this greedy algorithm. Suppose the original header at node $(0,0)$ includes destinations $(1,2)$, $(3,5)$, $(5,4)$ and $(7,3)$ which are represented by the filled circles in the figure. By applying the preparation part of the algorithm, we have the sorted destinations: $(0,0)$, $(1,2)$, $(3,5)$, $(5,4)$ and $(7,3)$. If two destinations have the same distance to the source, they are placed in an arbitrary order.

Now let's apply the construction part. In Step 1, $(0,0)$ is the root. $V(T) = \{(0,0),(1,2)\}$ and $E(T) = \{((0,0),(1,2))\}$. The rest of the destination nodes are added to the tree one by one. The next node to add is $(3,5)$. In Step 2(a), since $E(T)$ has only $((0,0),(1,2))$, node w which satisfies the three conditions of Step 2(a) is node $(1,2)$. In Step 2(b), $(3,5)$ is added to $V(T)$ and it is not necessary to add $(1,2)$ to $V(T)$ because it is already in the set. In Step 2(c), because $(3,5) \neq (1,2)$, $((1,2),(3,5))$ is added to $E(T)$. The next node to add is $(5,4)$. In Step 2(a), $E(T)$ has $((0,0),(1,2))$ and $((1,2),(3,5))$. For $((0,0),(1,2))$, the node satisfies the above three conditions is node $(1,2)$. For $((1,2),(3,5))$, the node satisfies the three conditions is node $(3,4)$. The distance between $(1,2)$ and $(5,4)$ is 6 and the distance between $(3,4)$ and $(5,4)$ is 2, we select node $(3,4)$ as w which is represented by an unfilled circle. In Step 2(b), both $(5,4)$ and w are added to $V(T)$. In Step 2(c), because $(3,4) \neq (1,2)$ and $(3,4) \neq (3,5)$, we delete $((1,2),(3,5))$ from $E(T)$ and add $((1,2),(3,4))$, $((3,4),(3,5))$ to $E(T)$. In Step 2(d), $((3,4),(5,4))$ is also added to $E(T)$. Now $E(T)$ has $((0,0),(1,2))$, $((1,2),(3,4))$, $((3,4),(3,5))$ and $((3,4),(5,4))$. The next node to add is $(7,3)$. Among all the pairs in $E(T)$, node $(3,3)$ should be w.

The node and edge sets of the resultant multicast tree are the following:

$$V(T) = \{(0,0),(1,2),(3,5),(5,4),(7,3),(3,4),(3,3)\}$$

and

$$E(T) = \{((0,0),(1,2)),((1,2),(3,3)),((3,3),(3,4)),((3,4),(3,5)),((3,4),(5,4)),((3,3),(7,3))\}$$

To distinguish a virtual edge that corresponds to one single minimal path from the one that corresponds to many minimal paths. We use a solid line to represent the former case and a dashed line to represent the later case. Since there exist multiple paths in virtual edges $((0,0),\ (1,2))$ and $((1,2),\ (3,3))$, these edges are represented by dashed lines in the figure.

Theorem 2: *The greedy algorithm guarantees a minimal path to each destination.*

Proof: We first prove the following result: For any edge (u, v) in the MT, $u \leq v$. Recall that $u \leq v$ is defined as $x_u \leq x_v$ and $y_u \leq y_v$. We prove this result by induction on the number of destination nodes in the MT. Clearly the result holds initially when there are two nodes d_0, d_1, and one edge (d_0, d_1) in the MT with $d_0 \leq d_1$. Suppose the result holds when there are k destination nodes in the MT, now a new destination d_k is added to the tree and it is connected to node w which is on the minimal path of (u, v) and it meets the conditions specified in the greedy algorithm. At most three new edges are added, (u, w), (w, v), and (w, d_i). $w \leq d_i$ clearly holds based on the selection procedure for w in the greedy algorithm. In addition, $u \leq v$ based on the induction assumption and w is on the minimal path between u and v, we have $u \leq w$ and $w \leq v$.

For any destination d_i, we can always find a path from source d_0 to d_i in the MT:

$$v_0(d_0) \to v_1 \to v_2 \to \ldots \to v_l \to v_{l+1}(d_i)$$

Based on the about result, $v_k \leq v_{k+1}$ $(0 \leq k \leq l)$, hence the above path a minimal path from d_0 to d_i in the corresponding 2-D mesh. $\square$

Although some changes are made to the approach in [16] to achieve time-step optimal, the above algorithm still maintains the same complexity as the original one. The computation induced by the split-and-sort function (the greedy algorithm) is called *off-line computation time* [7] and it can be used to estimate the complexity of the multicast algorithm.

Theorem 3: *Consider the greedy MT algorithm with k destinations. The time complexity for the preparation part is $O(k \log k)$. The time complexity for the construction part is $O(k^2)$.*

Proof: Since the distance between any two nodes can be calculated in a constant time for 2-D meshes, the preparation part takes $O(k \log k)$ time to sort the destination nodes. For the construction part, Step 1 takes a constant time. Step 2(a) can be done in $O(i)$ time with $1 \leq i \leq k$ (its proof is shown in the next paragraph). Both Steps 2(b) and 2(c) take $O(1)$ time. There are $k-1$ iterations of Step 2. Thus, the time complexity of the construction part is $\sum_{i=1}^{k-1}(O(1)+O(i)+O(1)+O(1)) = O(k^2)$.

Now we show that Step 2(a) can be done in $O(i)$ time. Clearly, there are at most $O(i)$ edges in the MT with i destinations, since at most one additional node w is added for the inclusion of each destination in the multicast set. Next we show that it takes $O(1)$ to select w from the minimal path of (u, v) that satisfies the conditions, i.e., $w \leq d_i$ and $dis(d_i, w)$ is minimal. The selection procedure can be done using the example in Figure 2 by treating two opposite corner nodes (x, y)

and $(x^{'}, y^{'})$ as u and v, respectively. The faulty block contains all the nodes on a minimal path from u to v. The new destination node $d_i(x_i, y_i)$ is in regions R_4, R_5, or R_6 (if we assume source d_0 is the origin). If d_i is in region R_4, w will be $(x_i, y^{'})$. If d_i is in region R_6, w will be $(x^{'}, y_i)$. If d_i is in region R_5, w will be $(x^{'}, y^{'})$ which is v itself. All three cases can be done in constant time and the selected w clearly meets the conditions for w. The final w is selected from w's selected for each edge in the MT. Since there are $O(i)$ edges in the MT, it takes $O(i) \times O(1) = O(i)$ in time. $\square$

Based on the above greedy algorithm, we have the following Strategy 3 to resolve conflict in a system with faulty blocks as follows:

Strategy 3: *At separating point u, construct two MT trees using the greedy algorithm, one along the $+X$ direction and the other along the $+Y$ direction. The X-bound destinations are inserted to the X-direction tree with u as its root. The Y-bound destinations belong to the Y-direction tree with u as its root. Once these two trees are constructed, they are combined into one through the common root. The undetermined nodes are inserted to the resultant MT tree using the greedy algorithm.*

The key to sort destinations in the greedy algorithm is the distance between separating point u and destinations. Finally, based on the construction of the MT tree, destination nodes are divided into two groups: X-bound group and Y-bound group. Note that the MT tree constructed at each separating point is an auxiliary tool to help determine the next forwarding node for each undetermined destination. No actual tree is constructed in the routing process and the final multicast tree does not necessarily match an MT constructed at a particular separating point. This is because when the multicast message reaches the next separating point, the same process is repeated with probably more fault information in the neighborhood, that is, the X-bound (or Y-bound) group determined from a previous separating point is farther partitioned at this new separating point.

Unlike nodes in a fault-free 2-D mesh, two nodes in a faulty 2-D mesh may not have a virtual edge between them, because faulty blocks may block all the minimal paths between them. In this case, one node is *ineligible* to the other. For example, in the selection of w in the greedy algorithm, a potential w is ineligible to a destination d_i if virtual edge (w, d_i) does not exist. Therefore, when we add a node to an MT tree using the proposed greedy method, we will not consider ineligible nodes (with respect to a destination node under consideration) as a potential w.

Now we give an example to illustrate the proposed multicast algorithm using Strategy 3 (see Figure 9) with two faulty blocks: one with $(5, 6)$ and $(9, 8)$ as its two opposite corners and the other with $(8, 13)$ and $(12, 15)$ as its two opposite corners. Initially, the message is at $(0, 0)$ and it is 2d-free, so it can take the next step in the X or Y direction. Assume the message reaches separating point $u(2, 3)$ (see Figure 9 (a)), since this node has the same Y coordinate as some of the destinations and the same X coordinate as some other destinations. Strategy 3 is used to resolve the conflict. This situation of point u belongs to Figure 5 (a). At this point, the X and Y direction

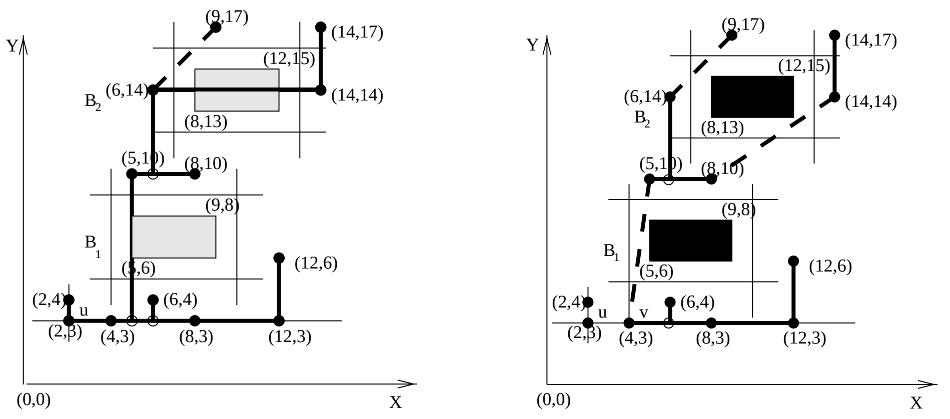


Figure 9: (a) MT trees constructed at separating point u and (b) MT trees constructed at the next separating point v

MT trees are constructed separately but with the same root node u. The X-bound destination nodes $(4, 3)$, $(8, 3)$ and $(12, 3)$ which have the same Y coordinate as u are added in sequence to the X direction tree using the greedy method. The Y-bound destination $(2, 4)$ is added to the Y direction tree. For the undetermined nodes, sort them as $(6, 4)$, $(5, 10)$, $(12, 6)$, $(8, 10)$, $(6, 14)$, $(9, 17)$, $(14, 14)$ and $(14, 17)$ based on their distances to u. Since node u does not involve any faulty block, i.e., it does not have any fault information, the undetermined destinations are added to the X or Y direction tree as if there were no faulty blocks in the system. The result of the construction is shown in Figure 9 (a). The faulty blocks are colored grey meaning their existence is not known to node u.

Now the header is split to two new messages. One new message takes the X direction in the next step and another one takes the Y direction. We treat each new message as a new source and the same process is repeated. Let's follow the X direction message since the Y direction message can be done easily. The new source $(2, 3)$ is 1d-free, so it can take the next step in the X or Y direction. Assume this message reaches separating point $v(4, 3)$ (see Figure 9 (b)), since this node has the same Y coordinate as some of the destinations and it hits path 2 of faulty block B_1, this situation of point v belongs to Figure **??** (a). Again X and Y direction MT trees are constructed separately. The X-bound destination nodes $(8, 3)$ and $(12, 3)$ which have the same Y coordinate as u are added to the X direction tree using the greedy method. Note that node v not only has the faulty block information of faulty block B_1 but also has the faulty block information of faulty block B_2. Thus, destinations $(5, 10)$, $(8, 10)$, $(6, 14)$ and $(9, 17)$ which are on path 2 or at the north of path 2 are Y-bound and are added to the Y direction tree using the greedy method. For the undetermined destinations, sort them as $(6, 4)$, $(12, 6)$, $(14, 14)$ and $(14, 17)$ based on their distances to point v. Next try to add them in both trees (now the two branches of the merged tree) and choose the closer one to join. The first node to add is $(6, 4)$, it is in region R_8 of faulty block B_1, it is added to the X direction tree (branch) because it is closer to it. Destination $(12, 6)$ is added to the X direction tree. Next node to add is $(14, 14)$. When we apply the greedy method, the nodes on some edge may not be eligible. For example, along edge $((6, 10), (6, 14))$, nodes $(6, 13)$, $(6, 14)$ are not eligible because all the minimal paths to $(14, 14)$ are blocked by B_2. They can not be selected as w. Node $(14, 14)$ is added to the Y direction tree with $(8, 10)$ as w (although $(6, 12)$ can also be selected as w). Destination $(14, 17)$ is added to the Y direction tree via node $(14, 14)$. The resultant MT tree is shown in Figure 9 (b). The faulty blocks are colored black because their faulty information is known to node v.

Notice the difference between Figures 9 (a) and (b). In Figure 9 (a), node $(5, 10)$ is directly linked to the X direction tree regardless of the faulty block below it. In Figure 9 (b), a dashed line is used because it is now aware of the faulty block B_1. A similar situation happens to node $(14, 14)$.

5 Discussion

The following theorem shows that a 1d-free destination will never be converted into a 0d-free destination at the next node. That is, the set-up process following Strategy 3 always finds a minimal path to each destination if there exists one from the original source node.

Theorem 4: *The set-up process following Strategy 3 generates a minimal path to each destination in a multicast set.*

Proof: This process resembles the original greedy algorithm. However, destination nodes are grouped into three sets. Nodes in each set are inserted following the sorting order within each set but may not following the global sorting order. The MT is then constructed set by set. We can easily prove that each edge (u, v) in the MT still meet the condition $u < v$. The only difference is that w may not exist along (u, v) such that $w < d_i$. However, based on the above argument, such w exists for at least one edge in the MT. □

In the proposed set-up process based on Strategy 3, we try to postpone the message splitting as late as possible to lower the total number of traffic steps. Thus, the message to different destinations can share as many paths as possible. Note that the optimal multicast problem is NP-complete in 2-D meshes without faulty components. The problem of reducing the number of traffic steps becomes more difficult in a faulty environment with faulty blocks. In our model, it can achieve time-step optimal but can not guarantee traffic-step optimal. We try to lower the number of traffic steps as much as possible and also make the complexity of the algorithm acceptable.

Theorem 5: *Splitting a message later generates fewer number of traffic steps than splitting earlier.*

Proof: In the algorithm, the message is split only if there exists a conflict (i.e., the message is split at a separating point). That is, the splitting of the message is postponed until it is a must. Suppose the locations of the remaining destinations are $d_1(x_1, y_1)$, $d_2(x_2, y_2)$, $\cdots$, $d_k(x_k, y_k)$, there are two splitting points: node $u(x_u, y_u)$ and node $v(x_v, y_v)$ with $u \leq v$. For each destination node $d_i(x_i, y_i)$ $(1 \leq i \leq k)$, the conditions $x_u \leq x_i$, $x_v \leq x_i$, $y_u \leq y_i$ and $y_v \leq y_i$ hold true. One routing strategy splits the message at node (x_u, y_u) and another routing strategy splits the message at node (x_v, y_v). Now we prove that splitting point (x_v, y_v) achieves fewer number of traffic steps than splitting point (x_u, y_u). Based on the assumption, we know that node v is closer to all the destinations than node u. If the message is split at v then the path from u to v can be shared by all the destinations. If the message is split at node u, then at least one message can not share the whole path from u to v which has a length of $x_v - x_u + y_v - y_u$. Therefore, we prove that splitting the message later is better than splitting the message earlier. □

Theorem 6: *In an $n \times n$ mesh, if there are k destinations, the time complexity of the set-up process based on Strategy 3 is $O(k^3)$ (if $k \leq n$) or $O(nk^2)$ (if $k \geq n$).*

Proof: At a separating point, the time complexity for sorting the destination nodes is $O(k \log k)$. The time complexity for the construction part is $O(k^2)$ as shown in Theorem 3 when there are no faulty blocks. Since there are only three potential w for each virtual edge, their eligibility can be determined in a constant time (assume that edge (w, d_i) for each of the three potential w intersects with a constant number of faulty blocks), the time complexity remains the same as in fault-free meshes. Also, if $k \leq n$, at each separating point at least one destination will be split out. Since there are at most $k-1$ separating points, the overall complexity is $O(k^2)+O((k-1)^2)+ \cdots +O(1^2)$ $= O(k^3)$. When $k \geq n$, since the longest distance between the source and a destination in an $n \times n$ mesh can not exceed $2n$, we can have at most $2n$ separating points, i.e., each intermediate step is a separating point. Therefore, the time complexity is at least $O(k^2)+O((k-1)^2)+ \cdots +$ $O(k-2n-1)^2) = O(nk^2)$. □

Note that the above result is based on the worst case, that is, a case with a maximum number of separating points. In a real system, the average of separating points is much less than n (the number of destinations). Although, in a 2-D mesh with faulty blocks, there are more separating points than a 2-D mesh without faulty blocks. Still, the number of separating points is much less than n. The simulation results in Figure 10 (1) confirm this observation. In this figure, simulation of multicasting is done in a 50×50 mesh under different distributions of faults and destinations. The number of separating points is recorded in four curves: one for the theoretical upper bound, one for the fault-free case (the theoretical lower bound), the rest two are for 50-fault and 100-fault cases. Results show that for both 50-fault and 100-fault cases stay close to the fault-free case. That is, the fault-tolerant time-optimal multicasting does not introduce much additional off-time

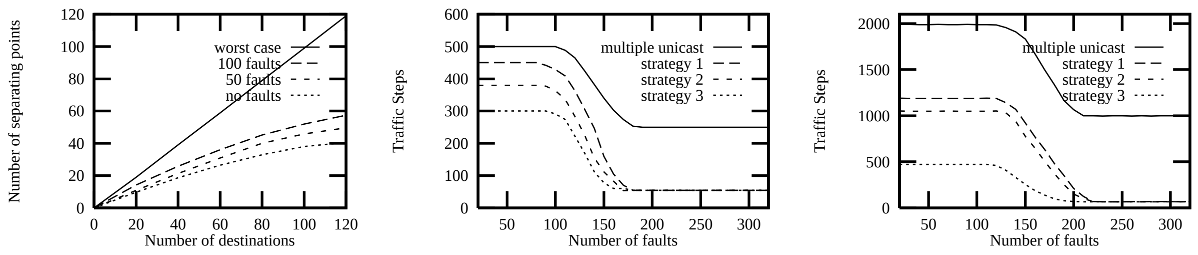


Figure 10: (1)The number of separating points in multicasting in a 50 × 50 mesh (2) Total number of traffic steps vs. number of faults using different strategies with (a) destination number 10 and (b) destination number 40

computation complexity compared with the split-and-sort function in a fault-free 50 × 50 mesh under the same distribution of destination nodes.

6 Simulation

A simulation study has been conducted to test the proposed multicast algorithm. We use a 50×50 mesh and randomly generate faults and destinations. First we calculate the total number of traffic steps when the number of destinations is fixed and then calculate the total number of traffic steps when the number of faults is fixed. We try different routing strategies and compare them with the multiple-unicast approach, i.e., unicasting to each destination without considering of sharing path(s).

When the number of destinations is fixed, we try two numbers of destinations 10 and 40. For each case, the number of faults goes from 0 to 320 (see Figures 10 2(a) and 10 2(b)). In both graphs, we can see that Strategies 1, 2 and 3 can significantly reduce the total number of traffic steps compared with the one derived from the multiple-unicast approach. Strategy 2 is better than Strategy 1 and Strategy 3 is much better than both Strategies 1 and 2. In these two graphs with the number of faults ranging from 0 to 100, the total number of traffic steps remains stable because the number of faults is not large enough to affect the multicast process. From the cases with the number of faults ranges from 210 to 320, the total number of traffic steps remains the same again because the system is saturated. However, such a large number of faults rarely happens in a real system. In the range from 110 to 200 destinations, with the increase of the number of faults, all three strategies save more traffic steps than the multiple-unicast approach. This means that these strategies are effective in saving traffic steps when the number of faults increases.

Comparing two graphs in Figures 13 and 14, all strategies save more traffic steps if the des-

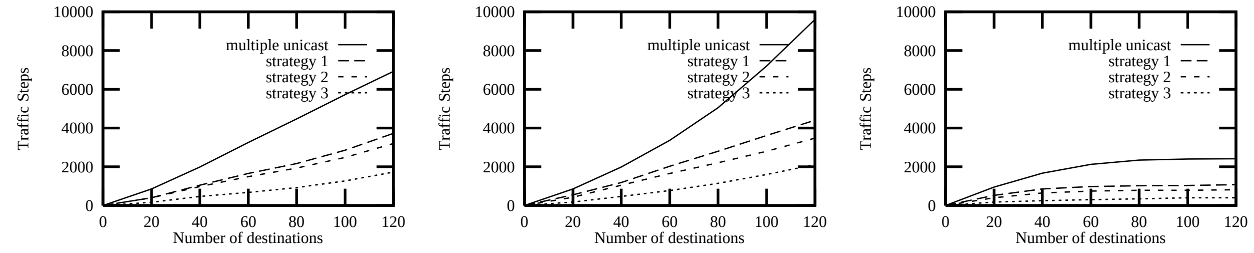


Figure 11: Total number of traffic steps vs. number of destinations using different strategies with (a) 50 faults and (b) 100 faults (c) 150 faults

tination number is higher. For example, Strategy 3 generates four times fewer number of traffic steps in the 40-destination graph than the multiple-unicast approach while it generates 1.7 times fewer number of traffic steps in the 10-destination graph than the multiple-unicast approach with the number of faults from 0 to 100. This is also true to Strategies 1 and 2. That means that all three strategies are more effective when the number of destinations is higher.

When the number of faults is fixed, we try three numbers of faults 50, 100 and 150. For each case, the number of destinations ranges from 0 to 120 (see Figures 11 (a), 11 (b) and 11 (c)). ¿From the graphs in Figures 15, 16 and 17, we can see that Strategies 1, 2 and 3 can significantly reduce the total number of traffic steps. Strategy 2 is better than Strategy 1 and Strategy 3 is much better than both Strategies 1 and 2.

If the number of faults is very large (say 150), the number of traffic steps will reach a constant with the increase of the number of destinations because the system is saturated with faulty nodes. If the number of faults is not too large, the total number of traffic steps continues to increase with the increase of the number of destinations. In these graphs, we observe that all strategies can save more number of traffic steps in the 50-fault graph than in the 100-fault graph. For example, if the number of destinations is 120, Strategy 3 can generate four times fewer number of traffic steps than the multiple-unicast approach in the 50-fault graph while it can only generate 2.1 times fewer number of traffic steps than the multiple-unicast approach in the 100-fault graph. It explains the more number of faults, the more difficult the routing process.

¿From the above simulation, we conclude that in a real system in which the number of simultaneous faults is usually low, with the increase of the number of destinations, all strategies can significantly reduce the total number of traffic steps, especially Strategy 3, although Strategies 1 and 2 can be implemented much easier. Therefore, a choice should be made depending on different objectives of various applications.

7 Extensions

We consider two possible extensions: fault tolerant multicasting in 3-D meshes and deadlock-free multicasting in 2-D meshes.

7.1 Fault-tolerant multicasting in 3-D meshes

Our approach can be extended to fault-tolerant multicasting in 3-D meshes based on limited global information. In 3-D meshes, *faulty cubes* are used as the fault model. More formally, a healthy node in a 3-D mesh is disabled if there are two or more disabled or faulty neighbors. A faulty cube contains all the connected unsafe and faulty nodes. Each faulty cube is a cube and the distance between any two faulty cubes is at least three.

Theorem 1 for 2-D meshes can be extended to be applied to 3-D meshes as follows:

Theorem 1a: *Assume that node (0, 0, 0) is the source and node (i, j, k) is the destination in a 3-D mesh. If there is no faulty cube that intersects with the XY plane (a plane with z=0), the YZ plane (a plane with x=0), and the XZ plane (a plane with y=0), there exists at least one minimal path from (0, 0, 0) to (i, j, k), i.e., the length of this path is $|i| + |j| + |k|$. This result holds for any location of the destination and any number and distribution of faulty cubes.*

To support minimal unicasting and multicasting in 3-D meshes, two types of fault information is used. The safety information is used to determine the safety status of a given source and destination pair. Each node in a 3-D mesh is associated a vector of six elements $(+X, -X, +Y, -Y, +Z, -Z)$. $+X$ represents the distance to the closest faulty cube along the positive X direction and other elements are defined in a similar way. The fault-cube-information is used to guide the routing message to each destination through a minimal path. Specifically, the faulty cube information is distributed to six adjacent planes of each faulty cube.

The same multicast algorithm for 2-D meshes can be extended to 3-D meshes. Types of multicast are now extended to 1d-free, 2d-free, 3d-free, and in-conflict. More cases have to be considered in the greedy algorithm to resolve a conflict. All these details will be our future work.

7.2 Deadlock-free multicasting

Deadlock due to dependencies on consumption resources (such as channels) are a fundamental problem in multicasting [6]. A deadlock involving several multicast processes occurs when there is a cyclic dependency for consumption channels.

We first show that the proposed routing algorithm is deadlock-free for unicasting. Unlike many

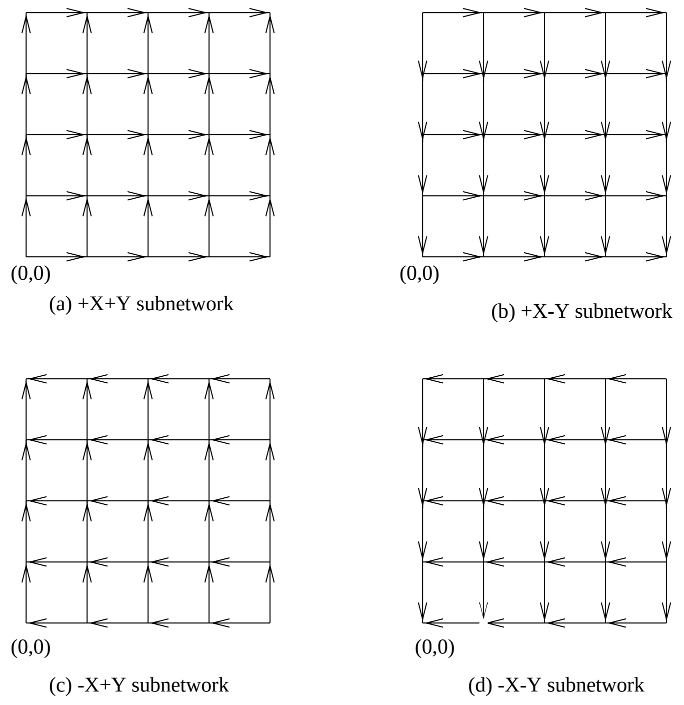


Figure 12: (a) $+X+Y$ subnetwork (b) $+X-Y$ subnetwork (c) $-X+Y$ subnetwork (d) $-X-Y$ subnetwork

non-minimal fault-tolerant unicasting, the deadlock issue can be easily solved through the concept of *virtual network* [18] where a given physical network is partitioned into several virtual networks. Each virtual network is partitioned into several virtual channels arranged in such a way that no cycle exists among channels, i.e., there is no *intra-virtual-network cycle*.

Figure 12 shows a partition of a 2-D mesh into four virtual subnetworks +X+Y, +X-Y, -X+Y, and -X-Y. Depending on the relative location of the source and destination, one of the four virtual subnetworks is selected and the corresponding multicast can be completed within the selected subnetwork without using any other subnetworks. In this way, any *inter-virtual-network cycle* is avoided. Throughout this paper, we assume that source is $(0,0)$ and (i,j) is one of the destination with $i > 0$ and $j > 0$. Therefore, only the +X+Y subnetwork is used. We can easily extend this case to cover other three cases.

During the set-up phase, a header is allowed to wait for a (profitable) channel if the orginial header has not generated a new branch. This case resembles multiple unicasting, and therefore, no deadlock will occur. During the data transmission phase, all needed channels have been reserved, and therefore no deadlock will occur.

The distributions of both safety information and faulty block information can be considered as a routing process. Clearly, deadlock can be easily avoided if they are based on PCS. However, since both distributions can be viewed as minimal unicasting, no deadlock would occur if other switching mechanisms are used as long as the partition approach shown in Figure 12 is followed.

8 Conclusions

In this paper, we have proposed a fault-tolerant tree-based multicast algorithm for 2-D meshes based on the concept of a faulty block and extended safety levels. The algorithm has been proved to achieve minimal multicast, i.e., each destination is reached through a minimal path. Three heuristic strategies proposed in this paper can significantly reduce the total number of traffic steps based on the results of our simulation. The first two strategies can be easily implemented through hardware without much additional cost and delay. Possible extensions to 3-D meshes and assurance of deadlock-freedom have also been discussed. Our approach is the first attempt to address the fault-tolerant multicast problem in 2-D meshes based on limited global information with a simple model and succinct information.

References

[1] R. V. Boppana and S. Chalasani. “Fault tolerant wormhole routing algorithms for mesh networks”, *IEEE Transactions on Computers*, Vol. 44, No. 7, July 1995, 848-864.

[2] Y. M. Boura and C. R. Das, “Fault-tolerant routing in mesh networks”, *Proc. of 1995 International Conference on Parallel Processing*, 1995, Vol. 1, 106-109.

[3] A. A. Chien and J. H. Kim. “Planar-adaptive routing: low cost adaptive networks for multiprocessors”, *Proc. of the 19th International Symposium on Computer Architecture*, 1992, 268-277.

[4] C. M. Cunningham and D. R. Avresky, “Fault-tolerant adaptive routing for two-dimensional meshes”, *Proc. of the 1st IEEE Symposium on High Performance Computer Architecture*, Jan. 1995, 122-131.

[5] W. J. Dally, “The J-machine: System support for Actors”, *Actors: Knowledge-Based Concurrent Computing*, MIT Press, 1989.

[6] J. Duato, S. Yalamanchili, and L. M. Ni, *Interconnection Networks: An Engineering Approach*, IEEE Computer Society, 1997.

[7] E. Fleury and P. Fraigniaud, “Multicasting in meshes”, *Proc. of the 1994 International Conference on Parallel Processing*, 1994, III 151-III 158.

[8] P. T. Gaughan and S. Yalamanchili, "A family of fault-tolerant routing protocols for direct multiprocessor networks", *IEEE Transactions on Parallel and Distributed Systems*, Vol. 6, No. 5, May 1995, 482-495.

[9] F. Harary, *Graph Theory.* Readings, MA: Addison-Wesley, 1972.

[10] R. Libeskind-Hadas and E. Brandt, "Origin-based fault-tolerant routing in the mesh", *Future Generation Computer Systems*, Vol. 11, No. 6, Oct. 1995, 603-615.

[11] R. Libeskind-Hadas, D. Mazzoni, and R. Rajagopalan, "Tree-based multicasting in wormhole routed irregular topologies," *Proc. of the first IPPS/SPDP*, April 1998, 244-249.

[12] R. Libeskind-Hadas, K. Watkins and T. Hehre, "Fault-Tolerant Multicast Routing in the Mesh with No Vitual Channels", *Proc. of the 2nd IEEE Symp. on High-Performance Computer Arch.*, January 1996, 180-190.

[13] R. E. Kessler and J. L. Schwarzmeier, "CRAY T3D: A new dimension for Cray research", *Compcon*, Spring, 176-182.

[14] Y. Lan, A. H. Esfahanian, and L. M. Ni, "Distributed multi-destination routing in hypercube multiprocessors", *Proc. of the 3rd Conference on Hypercube Concurrent Computers and Applications*, Vol. 1, 1988, 631-639.

[15] S. L. Lillevik, "The Touchstone 30 gigaflop DELTA prototype", *Proc. of the 6th Distributed Memory Computing Conference*, 1991, 671-677.

[16] X. Lin and L. M. Ni, "Multicast communication in multicomputer networks", *IEEE Transactions on Parallel and Distributed Systems*, Vol. 4, No. 10, Oct. 1993, 1105-1117.

[17] X. Lin, P. McKinley, and L. M. Ni, "Deadlock-free Muticast Wormhole Routing in 2D-Mesh Multicomputers", *IEEE Transactions on Parallel and Distributed Systems*, Vol. 5, Aug. 1994, 793-804.

[18] D. H. Linder and J. C. Harden, "An adaptive and fault tolerant wormhole routing strategy for k-ary n-cubes", *IEEE Transactions on Computers*, vol. C-40, no. 1, Jan 1991, 2-12.

[19] M. P. Malumbres, J. Duato and J. Torrellas, "An Efficient Implementation of Tree-Based Multicast Routing for Distributed Shared-Memory Multiprocessors", *in Proc. of Eighth IEEE Symp. on Parallel and Distributed Processing*, Oct. 1996, 186-189.

[20] L. M. Ni, "Should Scalable Parallel Computers Support Efficient Hardware Multicast?", *Proc. of the 1995 ICPP Workshop on Challenges for Parallel Processing*, 1995, 2-7.

[21] D. K. Panda, S. Singal, and P. Prabhakaran, "Multidestination Message Passing Mechanism Conforming to Base Wormhole Routing Scheme", *Proceedings of the Parallel Computer Routing and Communication Workshop*, LNCS 853, 1994, 131-145.

[22] D. K. Panda, “Issues in designing efficient and practical algorithms for collective communication on wormhole-routed systems”, *Proc. of the 1995 ICPP Workshop on Challenges for Parallel Processing*, Aug. 1995, 8-15.

[23] P. Pacheco, *Parallel Programming with MPI*, Morgan Kaufmann Publishers, 1997.

[24] D. K. Panda and S. Singal and P. Prabhakaran, “Multidestination message passing mechanism comforming to base wormhole routing scheme”, *Proc. of the First International Workshop on Parallel Computer Routing and Communication (PCRCW'94)*, K. Bolding and L. Snyder,Eds., Springer-Verlag, May 1994, 131-145.

[25] W. Qiao and L. M. Ni, “Adaptive routing in irregular networks using cut-through switches,” *Proc. of the 1996 International Conference on Parallel Processing*, Aug. 1996, I 52-I 60.

[26] C. L. Seitz, “The architecture and programming of the Ameteck Series 2010 multicomputer”, *Proc. of the 3rd Conference on Hypercube Concurrent Computers and Applications*, Vol. 1, Jan. 1988, 33-36.

[27] M. Shroeder and et al. “Autonet: A high-speed, self- configuration local area network using point-to-point links”, *IEEE Journal of Selected Areas in Communications*, Vol. 9, No. 10, Oct. 1991, 1318-1335.

[28] R. Sivaram, D. K. Panda, and C. Stunkel, “Multicasting in irregular networks with cut-through switches using tree-based multidestination worms,” *Proc. of the 2nd Parallel Computing, Routing, and Communication Workshop*, June 1997.

[29] C. C. Su and K. G. Shin, “Adaptive fault-tolerant deadlock-free routing in meshes and hypercubes”, *IEEE Transactions on Computers*, Vol. 45, No. 6, June 1996, 672-683.

[30] Y. C. Tseng, M. H. Yang and T. Y. Juang, “An Euler-Path-Based Multicasting Model for Wormhole-Routed Networks with Multi-Destination Capability”, *Proc. of 1998 Int'l Conf. on Parallel Processing*, Aug. 1998, 366-373.

[31] H. Wang and D. Blough, ”Tree-Based Fault-Tolerant Multicast in Multicomputer Networks Using Pipelined Circuit Switching,” *Department of Electrical and Computer Engineering, University of California, Irvine, Technical Report ECE 97-05-01*, May 1997.

[32] J. Wu and K. J. Yao, “A limited-global-information-based multicasting scheme for faulty hypercubes”, *IEEE Transactions on Computers*, Vol. 44, No. 9, Sept. 1995, 1162-1166.

[33] J. Wu, “On Constructing Faulty Orthogonal Convex Polygons in 2-D Meshes”, Technical Report, TR-CSE-98-5, Department of Computer Science and Engineering, Florida Atlantic University, Jan. 1998.

[34] J. Wu, “Fault-tolerant adaptive and minimal routing in mesh-connected multicomputers using extended safety levels,” *Proc. of the 18th International Conf. on Distributed Computing Systems*, May 1998, 428-435.