Fault-Tolerant Multicast Routing in the Mesh with No Virtual Channels*

Ran Libeskind-Hadas, Kevin Watkins, and Thomas Hehre

Department of Computer Science
Harvey Mudd College
301 E. 12th St.
Claremont, CA 91711
e-mail: hadas@cs.hmc.edu

Abstract

This paper addresses the problem of fault-tolerant multicast routing in wormhole-routed multicomputers. We present a new pseudo-Hamiltonian *path-based routing methodology for constructing deadlock-free multicast routing algorithms requiring no virtual channels. This technique is applied to construct the first fault-tolerant multicast routing algorithm for the mesh that requires no virtual channels. Simulation results indicate that this technique results in minimal performance degradation in the presence of a large number of node and channel faults.*

1 Introduction

The distributed memory multiprocessor paradigm provides a promising means of constructing scalable parallel computers. These systems comprise a collection of *nodes*, where each node consists of a processor with its own local memory and a router which supports message communication between nodes. The routers are connected by *channels* according to a particular interconnection topology. Among the most common topologies for multicomputers are low-dimensional meshes. These topologies are scalable and have a number of features that make them particularly amenable to high-performance computing [7, 10]. For example, a two-dimensional mesh topology is used in the Intel Touchstone DELTA and Intel Paragon and a three-dimensional mesh is used in the MIT J-Machine and the Cray T3D.

* This work was supported in part by the National Science Foundation under grant CCR-9418311.

In order to minimize network latency, the current generation of multicomputers employ the *wormhole routing* switching strategy [17]. Wormhole routing divides each packet into a number of constant size components or *flits*. The *header flit* contains the routing information for the message and leads the remaining flits through the network in a pipeline fashion to the destination. Once the header flit gains access to a channel, the current message "owns" that channel until the tail flit passes through it and relinquishes ownership of the channel.

Communication in the network can be either *unicast* or *multicast*. In unicast communication a message is sent from a source processor to a single destination processor, whereas in multicast communication a message is sent from a source processor to an arbitrary set of destination processors. Multicast communication has applications in a number of fundamental operations such as barrier synchronization [22], cache coherency in distributed shared-memory architectures [11], and clock synchronization [1], among others.

A multicast routing algorithm must satisfy several fundamental requirements [13]. First, the algorithm must be free from both deadlock and livelock. Deadlock arises when a cycle of blocked messages forms in which each blocked message waits for a channel belonging to the next blocked message in the cycle. Livelock occurs when a message proceeds through the network indefinitely, never arriving at its destination. Second, the algorithm should minimize message delay from the source to each of the destinations. Third, the algorithm should be supported by fast and simple routing hardware. Finally, in applications requiring high reliability, the routing algorithm must be able to continue to function in the presence of faults.

Lin *et al.* [13] have proposed a hardware-supported multicast routing methodology called *path-based routing* A number of multicast routing algorithms for the mesh have been developed based on this technique [2, 12, 13, 19, 21]. These algorithms are free from deadlock and livelock and enjoy low message latency. Several of the path-based routing algorithms for fault-free meshes require no virtual channels [12, 13]. However, existing fault-tolerant path-based routing algorithms [2, 19, 21] require at least two virtual channels per physical channel.

In this paper we present a new fault-tolerant path-based routing algorithm for the mesh requiring no virtual channels. This algorithm is based on a new path-based routing technique called *pseudo-Hamiltonian path-based routing*. The remainder of this paper is organized as follows. In Section 2 we survey previous results on multicast wormhole routing. In Section 3 we describe the pseudo-Hamiltonian path-based routing technique for arbitrary networks. In Section 4 we show how this technique can be applied to the faulty mesh. In Section 5 we describe experimental results which indicate that pseudo-Hamiltonian path-based routing performs very well even in the presence of a significant number of faults. Finally, we conclude in Section 6.

2 Previous Results

Existing multicast routing algorithms can be classified as *unicast-based*, *tree-based*, and *path-based* [21]. In unicast-based routing algorithms, a source node sends messages to its set of destination nodes by sending a sequence of separate unicast messages to each of these destinations or, alternatively, by sending unicast messages to a subset of its destinations which then participate in sending the message to the remaining destinations. Examples of unicast-based multicast routing can be found in [16, 20]. These schemes are commonly employed in multicomputers supporting only unicast communication in hardware. The disadvantage of the unicast-based approach is that multiple copies of the same message are introduced into the network, resulting in increased network traffic. More importantly, each copy of the message incurs a substantial start-up latency at the source processor. In current generation machines this start-up latency is typically on the order of several microseconds, whereas once the message enters the network, the channel propagation latency is typically on the order of nanoseconds [18].

In tree-based schemes, a spanning tree is constructed in which the source processor is designated as the root. The message is then sent from the root down the tree. Upon receiving the message, a node absorbs the message if it is one of the destinations and, if it is not a leaf in the tree, forwards the message to all of its children. Although such schemes have been used effectively in networks employing store-and-forward and virtual cut-through routing, tree-based routing incurs high congestion in wormhole-routed networks [13]. In addition, enforcing deadlock-freedom in tree-based schemes requires additional constraints on the routing algorithm [13].

Path-based routing was introduced in [13] to address a number of these problems. In this scheme, a source node prepares a message for delivery to a set of destinations by first sorting the addresses of the destinations in the order in which they are to be delivered, and then placing this sorted list in the header flits of the message. When the header enters a router with address α, the router checks to see if α is the next address in the header. If so, the address α is removed from the message header and the data flits are forwarded both to the local processor at this node as well as to the next node on the path. Otherwise, the message is forwarded only to the next node on the path. In this way, the message is eventually delivered to every destination in the header.

An important feature of path-based routing is that a message can be delivered to multiple destinations with the same start-up latency as a message sent to a single destination [13, 19]. A number of path-based multicast routing algorithms have been proposed for the mesh to exploit this feature. Several algorithms based on the existence of Hamiltonian paths in the mesh were proposed in [12, 13]. While these algorithms require no virtual channels, they are not fault-tolerant since a faulty mesh does not necessarily contain a Hamiltonian path. Moreover, the problem of determining whether a faulty mesh contains a Hamiltonian path is NP-complete [9].

The trip-based scheme proposed by Tseng *et al.* [21] generalizes the Hamiltonian path-based approach by using *trips* rather than Hamiltonian paths. A trip is defined to be a path in a graph that visits every vertex at least once. Using a special type of trip called a *skirt*, Tseng *et al.* show how multicast can be supported in an arbitrary connected network using two virtual channels per physical channel. In contrast to Hamiltonian paths, a polynomial-time algorithm is known for constructing skirts in any connected network [21]. Boppana and Chalasani [2] have also proposed fault-tolerant path-based algorithms that employ two or three virtual channels per physical channel, depending on the pattern of faults in the mesh.

In this paper we present a new generalization of the Hamiltonian path-based methodology called pseudo-Hamiltonian path-based routing. We then apply this technique to develop a fault-tolerant path-based multicast routing algorithm for the mesh that requires no virtual channels.

3 Pseudo-Hamiltonian Path-Based Routing

Each node of a multicomputer consists of a processor, with its own local memory, and a router responsible for message communication. A router is connected to its processor by one or more pairs of *internal channels*. One channel in each pair is an *internal input channel* used for input from the processor to the router and one is an *internal output channel* used for output from the router to the processor. In this paper we assume an *all-port architecture* in which each node in the mesh has four pairs of internal channels.

Each router is connected to neighboring routers via *external channels* according to some interconnection topology. Throughout this paper, we will refer to external channels as simply "channels". Interconnection networks will be represented by undirected graphs $G = (V, E)$ with each vertex $v \in V$ representing a node and each undirected edge (u, v) representing a pair of oppositely directed channels between u and v. We let $ch(u, v)$ denote the channel from u to v and $ch(v, u)$ denote the channel from v to u.

We now describe a generalization of Hamiltonian path-based routing called pseudo-Hamiltonian path-based routing which supports multicast routing with no virtual channels. In the next section we apply this technique to faulty meshes.

Definition 1 *A* pseudo-Hamiltonian path *in an undirected graph is a path which visits each vertex at least once and each edge at most once.*

To illustrate the difference between Hamiltonian paths, skirts, and pseudo-Hamiltonian paths, consider the graphs in Figure 1. The graph in Figure 1(a) contains a Hamiltonian path, indicated by dotted edges. The graph in Figure 1(b) is easily shown to contain no Hamiltonian path. However, a skirt in this graph is indicated by dotted edges in Figure 1(b). This graph also contains a pseudo-Hamiltonian path as indicated by dotted edges in Figure 1(c).

We now show how pseudo-Hamiltonian paths can be used to partition the channels into two subnetworks and then show how deadlock-free multicast routing can be achieved using these subnetworks.

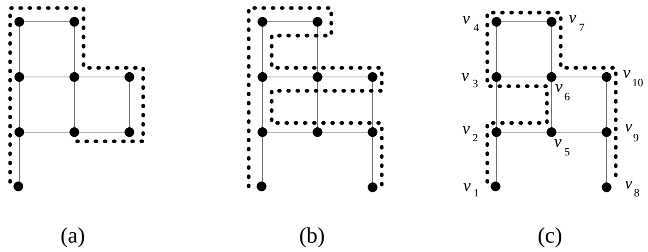

Figure 1: (a) A Hamiltonian cycle. (b) A skirt. (c) A pseudo-Hamiltonian path.

3.1 Construction of the Subnetworks

Let $G = (V, E)$ be an undirected graph representing an interconnection network and let $p = [v_{i_1}, v_{i_2}, \ldots, v_{i_m}]$ denote a pseudo-Hamiltonian path in G. With each vertex $v \in V$ we associate a set $S_p(v)$ of positive integers, called the *index set* for v with respect to pseudo-Hamiltonian path p, as follows:

$$S_p(v) = \{j | v_{i_j} \in p \text{ and } v = v_{i_j}\}.$$

In other words, $S_p(v)$ contains the indices corresponding to the occurrences of v on path p. For example, the pseudo-Hamiltonian path in Figure 1(c) is

$$p = [v_1, v_2, v_5, v_6, v_3, v_4, v_7, v_6, v_{10}, v_9, v_8]$$

and the corresponding index sets are as follows: $S_p(v_1) = \{1\}$, $S_p(v_2) = \{2\}$, $S_p(v_3) = \{5\}$, $S_p(v_4) = \{6\}$, $S_p(v_5) = \{3\}$, $S_p(v_6) = \{4, 8\}$, $S_p(v_7) = \{7\}$, $S_p(v_8) = \{11\}$, $S_p(v_9) = \{10\}$, and $S_p(v_{10}) = \{9\}$.

Next, we partition the channels of the network into two subnetworks, a *high-channel subnetwork* and a *low-channel subnetwork*. Each channel $ch(u, v)$ is also given an integer label denoted $label(u, v)$. For each edge $(u, v) \in E$ we consider two cases:

1. If there exist integers $i \in S_p(u)$ and $j \in S_p(v)$ such that $|i - j| = 1$, then by construction of the index sets, u and v occur as consecutive vertices in path p. Without loss of generality, assume that $i < j$, and thus u is visited immediately before v in p. Then the channel from u to v, $ch(u, v)$, is in the high-channel subnetwork and $label(u, v) = j$ while $ch(v, u)$ is in the low-channel subnetwork and $label(v, u) = i$. We note that this construction is well-defined: There cannot exist more than one pair $i \in S_p(u)$ and $j \in S_p(v)$ such that $|i - j| = 1$ since if this were the case then u and v appear as consecutive nodes at least twice in path p. This, however, implies that edge (u, v)

is used at least twice in p, contradicting the assumption that p is pseudo-Hamiltonian.

2. If for every $i \in S_p(u)$ and $j \in S_p(v)$, $|i-j| \neq 1$, then u and v do not occur as consecutive vertices in p. In this case, an arbitrary element i is chosen from $S_p(u)$ and an arbitrary element j is chosen from $S_p(v)$. Without loss of generality, let $i < j$. Then $ch(u,v)$ is in the high-subnetwork and $label(u,v) = j$ while $ch(v,u)$ is in the low-subnetwork and $label(v,u) = i$.

As an example of this construction, consider the pseudo-Hamiltonian path in Figure 1(c). The resulting high-channel and low-channel subnetworks are shown in Figure 2(a) and Figure 2(b), respectively. Finally, we note that step 2 in the above construction allows some arbitrary choices in selection of indices from the index sets. This flexibility can be exploited to adapt the subnetworks to different message traffic patterns if these pattern are known at the time that the subnetworks are established.

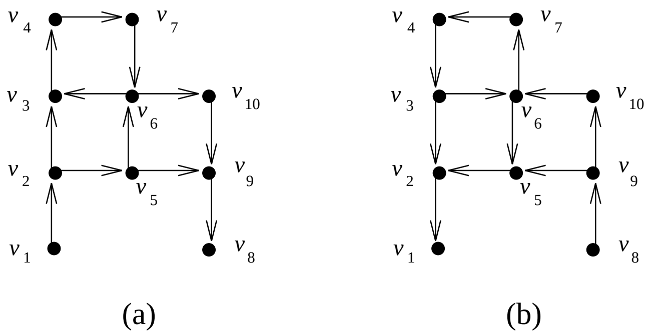

Figure 2: (a) High-channel subnetwork. (b) Low-channel subnetwork.

3.2 The Routing Algorithm

We now show how the high-channel and low-channel subnetworks can be employed to achieve deadlock- and livelock-free multicast routing. A message originating at node v with destination set D executes the algorithm in Figure 3 to initiate the message. The *Message Initiation* algorithm partitions the destinations with respect to one of the indices of the source node and creates two worms containing the message, one to be sent in the low-channel subnetwork and one to be sent in the high-channel subnetwork. The ability to choose arbitrary indices in step 2 of this algorithm can be exploited by heuristics to balance the number of messages in the two worms as evenly as possible or to partition the destinations according to some other metric.

Algorithm: ***Message Initiation***
Input: Source node v and destination set D.

Procedure:

1. Select an arbitrary index $i_v \in S_p(v)$.
2. For each destination node $u \in D$ select an arbitrary index $i_u \in S_p(u)$.
3. Partition the selected indices into two sets, L and H, such that L comprises the indices less than i_v and H comprises the indices greater than i_v.
4. Sort the indices in L in decreasing order and sort the indices in H in increasing order.
5. Create two copies of the message. One message contains the sorted indices in L in its header and one message contains the sorted indices in H in its header.
6. Using two internal input channels, send the message with index list L in the low-channel subnetwork and send the message with index list H in the high-channel subnetwork.

Figure 3: Message initiation algorithm.

The routing protocol executed by each router upon receiving a header of a message in the high-channel subnetwork is shown in Figure 4. The algorithm for messages in the low-channel subnetwork is analogous.

We now show that pseudo-Hamiltonian path-based routing is correct by proving that it is both deadlock-free and livelock-free.

Theorem 1 *Pseudo-Hamiltonian path-based multicast routing is deadlock-free.*

Proof: Since the high-channel and low-channel subnetworks are disjoint and there are no channel dependencies between these networks, it suffices to consider each of the subnetworks individually. We consider the high-channel subnetwork since the argument for the low-channel subnetwork is analogous. In multicast wormhole routing, deadlock can arise due to cyclic dependencies on external channels as well as internal output channels (or *consumption channels*) [4].

Dally and Seitz [8] have shown that a wormhole routing algorithm is deadlock-free with respect to external channels if the channels can be labeled such that every path allowed by the routing algorithm has the property that its channels have strictly increasing labels (or, equivalently, all paths have strictly de-

Algorithm: ***Message Routing*** (for the high-channel subnetwork)
Inputs:

v = current node,
M = message entering node v,
i = first index in header of message M.

Procedure:

1. If M originates at node v then select any output channel $ch(v,w)$ such that $label(v,w) \leq i$, forward the message along this channel, and exit.
2. Otherwise, if $i \in S_p(v)$ then copy the message to the local processor and remove i from the header.
3. Let j denote the next index in the header. Note that $j = i$ if the message was not sent to node v. Let (u,v) denote the input channel on which message M arrived at node v. Select any output channel $ch(v,w)$ such that
   (a) $label(u,v) < label(v,w)$ and
   (b) $label(v,w) \leq j$.

   and forward the message along the selected channel.

Figure 4: High-channel subnetwork routing algorithm.

creasing labels). The *Message Routing* algorithm requires that $ch(u,v)$ can be followed by $ch(v,w)$ only if $label(u,v) < label(v,w)$ and thus all paths in the high-channel subnetwork have strictly increasing labels. Therefore, the algorithm is deadlock-free.

Next, we consider deadlock involving consumption channels. One consumption channel at each node is reserved for each of the four external channels entering that node. Assume that there exists deadlock in the high-channel subnetwork. Let M be a deadlocked message in the high-channel subnetwork owning the external channel with the largest label among all deadlocked messages in this subnetwork. Denote this channel by $ch(u,v)$. Since messages in the high-channel subnetwork acquire external channels with strictly increasing labels, $ch(u,v)$ holds the header of message M. The header cannot be waiting for a consumption channel, since $ch(u,v)$ has exclusive access to a consumption channel. Therefore, the header must be waiting for an external output channel $ch(v,w)$. Since $label(u,v) < label(v,w)$, $ch(v,w)$ is not involved in deadlock and thus becomes available to message M in a finite amount of time. Thus M is not deadlocked, contradicting our assumption. □

Theorem 2 *Pseudo-Hamiltonian path-based multicast routing is livelock-free.*

Proof: Since the two subnetworks are analogous, we again consider only messages in the high-channel subnetwork. To prove freedom from livelock we must show that the routing algorithm satisfies two properties: First, we must show that when a message M arrives at a node v, there exists at least one output channel on which the message is allowed to continue if the header is not empty. Secondly, we must show that regardless of which of these valid output channels is selected, the message will eventually be delivered to the next destination in the header.

For the first requirement, assume that message M arrives at the router at node v and let i denote the first index in the header of M when the *Message Routing* algorithm is invoked. If M originates at v then step 1 of the algorithm applies. By the *Message Initiation* algorithm, there exists $i_v \in S_p(v)$ such that $i > i_v$. By construction of the high-channel subnetwork, there exists an output channel $ch(v,w)$ such that $label(v,w) = i_v + 1$. Thus, $label(v,w) \leq i$ as required by step 1 of the *Message Routing* algorithm.

Next, we consider the case that M does not originate at v. Let $ch(u,v)$ denote the input channel on which M arrives at v. Note that since M was forwarded from node u, the *Message Routing* algorithm executed at node u ensures that $label(u,v) \leq i$. Observe also that by construction of the high-channel subnetwork there exists an output channel $ch(v,w)$ with $label(v,w) = label(u,v) + 1$, unless v is the last node on the pseudo-Hamiltonian path. Thus, condition (a) in step 3 of the *Message Routing* algorithm is satisfied by channel $ch(v,w)$. Next, we show that channel $ch(v,w)$ also satisfies condition (b) in step 3.

If $i \in S_p(v)$ then step 2 of the *Message Routing* algorithm is executed and the message is delivered to the processor at node v. If the header is not empty after delivering the message to node v, let j denote the next index in the header. We note that $i < j$ since the indices are sorted in strictly increasing order. Therefore,

$$label(v,w) = label(u,v) + 1 \leq i + 1 \leq j$$

and condition (b) in step 3 is satisfied.

If $i \notin S_p(v)$ then $label(u,v) \neq i$ and thus $label(u,v) < i$. Since M is not delivered to the local processor at node v, $j = i$. So, $label(u,v) < j$ and thus $label(v,w) = label(u,v) + 1 \leq j$, again satisfying condition (b) in step 3.

For the second requirement, we must show that if i is the first index in the header of message M then message M is eventually delivered to the node containing this index in its index set. To see this, observe that the constraints in step 3 of the *Message Routing* algorithm ensure that while i is the first index in M, the channels reserved by M have strictly increasing labels that are less than or equal to i. Therefore, message M is delivered within finite time to the node corresponding to index i. □

4 Multicast Routing in the Mesh

In this section we show how the pseudo-Hamiltonian path technique can be applied to obtain fault-tolerant multicast routing algorithms for the mesh using no virtual channels. In order to simplify our discussion, we restrict our attention here to the *block-fault model* [2, 3, 5, 6, 14], although the technique described here can be applied to a number of other fault models as well.

The block-fault model accounts for both faulty nodes and faulty channels with the assumption that all faulty nodes and channels are contained in rectangular blocks. The block-fault model has been widely used in fault-tolerant wormhole unicast routing to model a number of common fault scenarios at the chip, multichip module, and board levels [2, 3, 5]. Block-faults have also been used to model independent node faults by disabling or *pseudo-faulting* some non-faulty nodes [6, 14]. This approach allows for simple routers [6] and eliminates hot-spot effects that otherwise arise at the entrances to concave fault regions [14]. In addition, to illustrate our technique with minimal case analysis, we assume here that the minimum ℓ^∞ distance between two blocks is at least 3. That is, we assume that for any two nodes with coordinates (x_1, y_1) and (x_2, y_2) inside different fault blocks, $\min\{|x_1 - x_2|, |y_1 - y_2|\} \geq 3$. An example of a fault pattern in a 10×10 mesh satisfying these requirements is shown in Figure 5. The darkened nodes in this figure indicate nodes in each of the block-faults. This fault pattern will be used throughout this section to illustrate the construction of pseudo-Hamiltonian paths.

A pseudo-Hamiltonian path in a mesh containing rectangular block-faults separated by ℓ^∞ distance 3 or more is constructed using an algorithm comprising six steps. This algorithm, in fact, constructs a pseudo-Hamiltonian *cycle* which can then be converted into a path by simply removing any single edge from the cycle.

In the first step of the algorithm, a *perimeter cycle* is constructed around the boundary of the mesh, circumventing faulty blocks when necessary. The fact that fault blocks are separated by ℓ^∞ distance of at least 3 ensures that this is possible. The result of applying this step to the fault pattern in Figure 5 is shown if Figure 10(a).

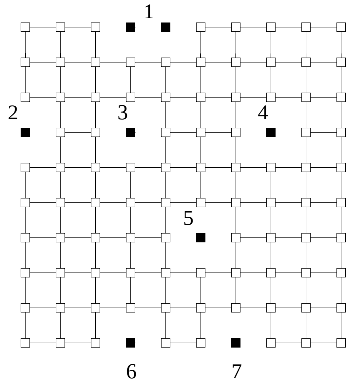

Figure 5: A 10×10 mesh with seven labeled block-faults.

In the second step of the algorithm, *block cycles* are constructed around each of the rectangular blocks. This is illustrated in Figure 10(b). The fact that all fault blocks are ℓ^∞ distance 3 or more apart ensures that all such cycles are disjoint.

In the third step, the perimeter cycle and the block cycles constructed in the previous two steps are joined into a single cycle. For each block cycle, a "connector" is constructed consisting of two parallel vertical paths. The first path in the connector originates at the southwest corner of the block cycle and the second path originates one vertex to the right of the southwest corner. These two parallel paths are extended to the south until one of them encounters a vertex on the perimeter cycle or another block cycle. The connector is then used to merge the two cycles into a single cycle. There are four cases that arise in this construction. The cases are shown in Figure 6(a), (b), (c), and (d) where the darkened squares indicate vertices in a faulty block and the southwest portion of the block cycle is indicated by the L-shaped strip.

In the first case, shown in Figure 6(a), the connector encounters two vertices connected by a horizontal edge on another cycle. In this case, both this edge and the edge between the two origin vertices on the block cycle are removed and the connector merges the two cycles into a single cycle as shown in Figure 6(e). In the second case, shown in Figure 6(b), the left path of the connector encounters a corner of a cycle but the right path does not encounter a corner at this row. In

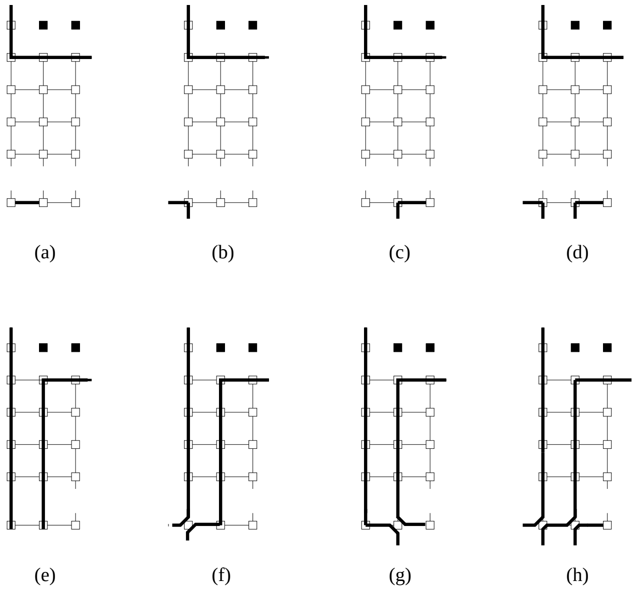

Figure 6: The four cases arising in step 3 of the algorithm. (a) Case 1. (b) Case 2. (c) Case 3. (d) Case 4. (e) Solution to case 1. (f) Solution to case 2. (g) Solution to case 3. (h) Solution to case 4.

this case the two cycles may again be merged into a single cycle as shown in Figure 6(f). In the third case, shown in Figure 6(c), the right path of the connector encounters a corner of a cycle but the left path does not encounter a corner at this row. In this case the cycles are merged as shown in Figure 6(g). Finally, the last case is that both paths of the connector encounter corners in the same row. This case and its solution are shown in Figure 6(d) and (h), respectively. It is easily verified that these are the only possible cases. The result of applying this step to the cycles in Figure 10(b) is shown in Figure 10(c).

After step 3 of the algorithm, a single cycle has been constructed. The vertices on this cycle are said to be *visited* and the non-faulty vertices not on the cycle are said to be *unvisited*. In the last three steps of the algorithm, the unvisited vertices are added to the cycle. In step 4, any pair of unvisited vertices connected by a horizontal edge are selected. Such vertices are said to be *horizontally adjacent*. A connector similar to the one used in the previous step of the algorithm is used to connect these horizontally adjacent vertices to the current cycle and the procedure is repeated until no unvisited horizontally adjacent vertices remain. The four possible cases are analogous to those in step 3 of the algorithm and are shown in Figure 7(a), (b), (c), and (d) with their solutions in Figure 7(e), (f), (g),

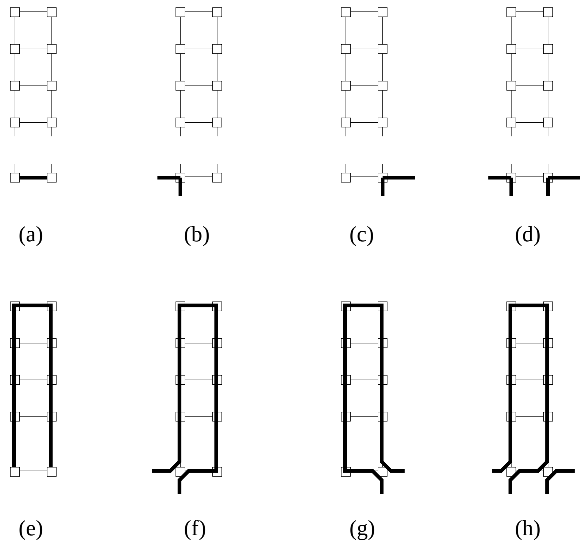

Figure 7: The four cases arising in step 4 of the algorithm. (a) Case 1. (b) Case 2. (c) Case 3. (d) Case 4. (e) Solution to case 1. (f) Solution to case 2. (g) Solution to case 3. (h) Solution to case 4.

and (h), respectively. The result of applying this step to the cycles in Figure 10(c) is shown in Figure 10(d).

Since after step 4 there remain no horizontally adjacent unvisited vertices, the only remaining unvisited vertices occur in vertical strips of length one or more. For example, in Figure 10(d) there remains one vertical strip of length four, one vertical strip of length 3, and three vertical strips of length one. Step 5 of the algorithm appends unvisited vertices to the cycle such that upon the termination of this step the only remaining vertical strips of unvisited vertices have length 1. Finally, step 6 adds these last remaining vertices to the cycle.

In step 5 we consider each vertical strip of length two or more. In each such strip, vertically adjacent vertices are partitioned into pairs, resulting in at most one unpaired vertex in each strip. Let u and v denote a pair of vertices where u is directly above v. Let w and x denote the vertices immediately to the left of u and v, respectively. Both w and x must already be visited since otherwise there would exist a pair of horizontally adjacent unvisited vertices, contradicting our assumption. Vertices u and v are now added to the cycle. The two possible cases are shown in Figure 8(a) and (b), with their solutions shown in Figure 8(c) and (d), respectively. The result of applying this step to the cycles in Figure 10(d) is shown in Figure 10(e).

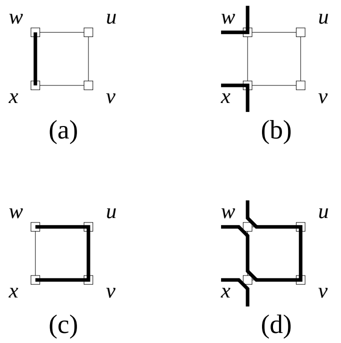

Figure 8: (a) The two cases arising in step 5 of the algorithm. (a) Case 1. (b) Case 2. (c) Solution to case 1. (d) Solution to case 2.

Finally, in step 6 the remaining unvisited vertices are added to the cycle. Let v denote an unvisited vertex at the beginning of step 6 and let $u_0, u_1, \ldots, u_7$ denote the eight vertices at ℓ^∞ distance 1 from v, as shown in Figure 9(a). Both u_3 and u_7 must already be visited since otherwise one of these vertices and v would constitute a pair of horizontally adjacent unvisited vertices, which is not possible after step 4. Similarly, both u_1 and u_5 must be visited, since otherwise one of these vertices and v would constitute a pair of vertically adjacent unvisited vertices, which is not possible after step 5. There are two possible cases. In the first case, at least one of the eight edges $(u_i, u_{(i+1)\bmod 8})$ is not yet used in the cycle. Without loss of generality, we consider the case that (u_1, u_2) is unused. Since u_1 is already visited, the current cycle must include the edge (u_0, u_1) and the edge from u_1 to its north neighbor, as shown in Figure 9(b). If edge (u_2, u_3) is not used in the cycle then vertex v is added to the cycle as shown in Figure 9(c). If edge (u_2, u_3) is used, as shown in Figure 9(d), then vertex v is added to the cycle as shown in Figure 9(e). The result of applying this step to the cycles in Figure 10(e) is shown in Figure 10(f). The last remaining case is that all edges $(u_i, u_{(i+1)\bmod 8})$ are used. The analysis for this case is rather involved and the reader is referred to [15] for details. However, in over 100,000 fault patterns generated by our simulator for each of 16×16, 32×32, and 64×64 meshes, this case never arose. Finally, since this algorithm considers each node only a finite number of times, the total time to construct the pseudo-Hamiltonian path is linear in the number of nodes in the mesh.

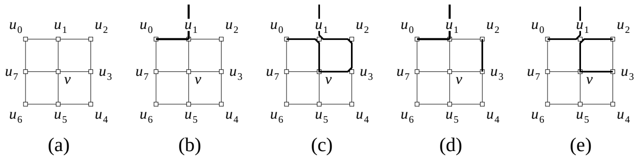

Figure 9: (a) An unvisited vertex v and its eight neighbors. (b) The case that (u_1, u_2) is unused. (c) The solution when (u_2, u_3) is unused. (d) The case that (u_2, u_3) is used. (e) The solution when (u_2, u_3) is used.

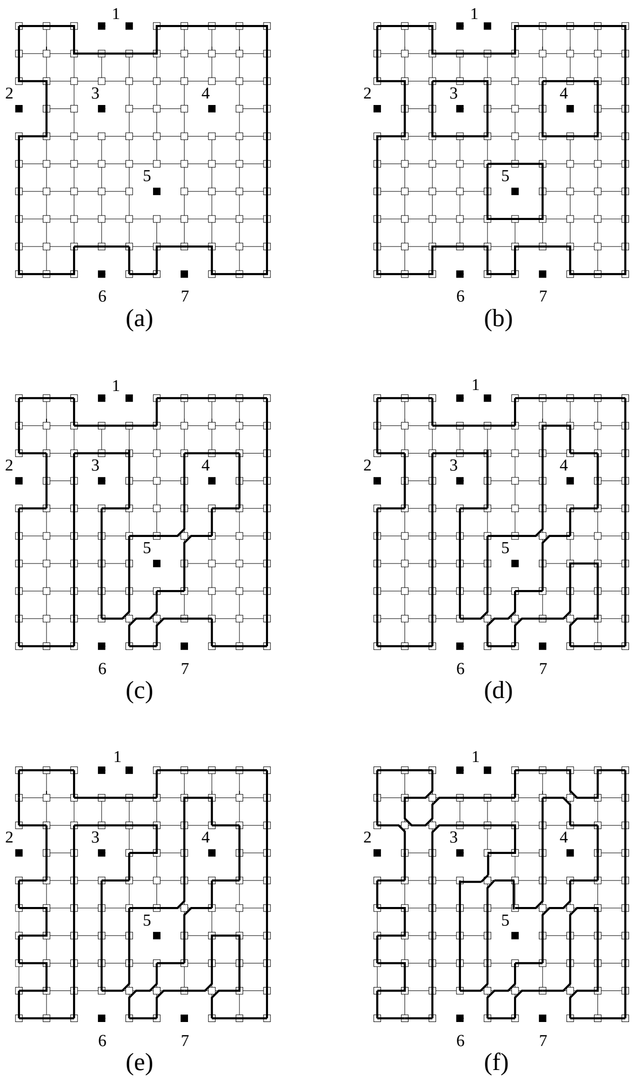

Figure 10: (a) The cycle after step 1. (b) After step 2. (c) After step 3. (d) After step 4. (e) After step 5. (f) After step 6.

5 Simulation Results

In this section we describe simulation results for fault-tolerant pseudo-Hamiltonian path-based routing in the mesh. For comparison, the skirt-based routing algorithm [21] employing two virtual channels per physical channel was also simulated. The simulations were conducted for 16×16 meshes using the event-driven Mesh Architecture Routing Simulator (MARS) developed at Harvey Mudd College. All simulations were executed until the confidence interval was smaller than 1% of the mean using 95% confidence intervals.

The skirt-based routing algorithm requires that a spanning tree be constructed in the network and the choice of this spanning tree may affect the performance of the algorithm [21]. A number of spanning tree construction heurstics were considered and a heuristic was selected which attempts to minimize branches in the spanning tree and consequently reduces the number of hops between nodes. Both algorithms were implemented with a *productive* channel selection policy that favors channels that reduce the distance to the next destination [21].

In the first simulations, the average hop counts per message were computed for a single source in the network with the number of destination nodes ranging from 0 to 100. These measurements were taken for the cases of 0%, 5%, and 10% faulty nodes, as shown in Figure 11.

In the fault-free case, both the pseudo-Hamiltonian path-based and the skirt-based algorithms find Hamiltonian paths, and therefore the hop counts for both algorithms are essentially identical to those found by the Hamiltonian path-based algorithm [13]. In our simulations, we observed two competing effects that accounted for the number of hops used by both algorithms. First, as the number of faults increases, the number of nodes that may participate in routing decreases, resulting in pseudo-Hamiltonian paths and skirts with fewer edges. On the other hand, the subnetworks induced by the pseudo-Hamiltonian paths and skirts may require messages to use more of these edges to circumvent faults. In our simulations of pseudo-Hamiltonian path-based routing, these two effects largely offset one another, with the number of hops actually decreasing slightly with an increasing number of faults. In contrast, the hop counts for skirt-based routing increased when faults were introduced. For example, for the case of 10% faulty nodes and 100 destinations for the message, the paths taken by the skirt-based algorithm were on average approximately 38% longer than those taken by the pseudo-Hamiltonian path-based algorithm.

In the next set of simulations, network latencies for both pseudo-Hamiltonian path-based and skirt-based algorithms were studied for varying network loads. The processors generated messages according to a negative binomial distribution with message lengths chosen uniformly from 10 to 100 flits and the number of destinations per source chosen uniformly from 5 to 15. In addition, the following system parameters were assumed [19]: The start-up latency was 1 microsecond, link propagation delay was 5 nanoseconds, and router delay was 20 nanoseconds. Latency measurements were taken for the cases of 0%, 5%, and 10% faulty nodes, as shown in Figure 12.

In the fault-free case, both algorithms incurred latencies essentially identical to those of the Hamiltonian path-based routing algorithm [13]. At high loads, however, the multiple virtual channels used in skirt-based routing resulted in somewhat slower saturation. In the case of faulty nodes, the two algorithms performed similarly at lowe loads. At higher loads, the longer paths taken by skirt-based routing resulted in high congestion and thus earlier saturation.

Finally, Figure 13 shows the effect of the number of faults on network latency for three fixed average arrival rates: 0.006, 0.008, and 0.0010 arrivals per μsec. In each case the number of faults ranged from 0 to 25. For small numbers of faults, both algorithms incurred essentially identical latencies. As the number of faults increased, the longer paths incurred by skirt-based routing resulted in considerably higher latencies.

6 Conclusions

In this paper we have introduced a new technique for constructing fault-tolerant path-based routing algorithms requiring no virtual channels and this technique was applied to develop a fault-tolerant routing algorithm for the mesh. Simulation results indicate that this algorithm enjoys very modest performance degradation, even in the presence of a substantial number of faults. If virtual channels are available, they can be used to implement other subnetworks for even higher performance. We are currently investigating the applicability of pseudo-Hamiltonian path-based routing to a number of other topologies.

References

[1] M. Azevedo and D. Blough. Fault-tolerant clock synchronization of large multicomputers via mul-

tidimensional interactive convergence. Technical Report ECE 94-12-03, University of California, Irvine, Dec. 1994.

[2] R. Boppana and S. Chalasani. Fault-tolerant multicast communication in multiprocessors. In *Proc. 1995 Int. Conf. on Parallel Processing*, August 1995.

[3] R. Boppana and S. Chalasani. Fault-tolerant wormhole routing algorithms for mesh networks. *IEEE Transactions on Computers*, To appear.

[4] R. Boppana, S. Chalasani, and C. Raghavendra. On multicast routing in multicomputers. In *Proc. 1994 IEEE Symp. on Parallel and Distributed Processing*, 1994.

[5] S. Chalasani and R. Boppana. Adaptive fault-tolerant wormhole routing algorithms with low virtual channel requirements. In *Proc. 1994 Int. Symp. on Parallel Architectures, Algorithms, and Networks*, Dec. 1994.

[6] A. Chien and J. Kim. Planar-adaptive routing: Low-cost adaptive networks for multiprocessors. In *Proc. of the Int. Symp. on Computer Architecture*, 1992.

[7] W. Dally. Virtual channel flow control. In *Proc. of the 17th Int. Symp. on Computer Architecture*, pages 60–68, May 1990.

[8] W. Dally and C. Seitz. Deadlock-free message routing in multiprocessor interconnection networks. *IEEE Transactions on Computers*, 36(5):547–553, May 1987.

[9] A. Itai, C. Papadimitriou, and J. Szwarcfiter. Hamilton paths in grid graphs. *SIAM J. Computing*, 11(4):676–686, 1982.

[10] F. T. Leighton. *Parallel Algorithms and Architectures: Arrays, Trees, and Hypercubes.* Morgan Kaufmann, 1992.

[11] K. Li and R. Schaefer. A hypercube shared virtual memory. In *Proc. 1989 Int. Conf. Parallel Processing*, volume 3, pages 125–132, 1989.

[12] X. Li, P. McKinley, and A-H. Esfahanian. Adaptive multicast wormhole routing in 2D mesh multicomputers. In *Proc. 1993 Parallel Architectures and Languages Europe Conference (PARLE'93)*, pages 228–241, June 1993.

[13] X. Li, P. McKinley, and L. Ni. Deadlock-free multicast wormhole routing in 2-D mesh multicomputers. *IEEE Transactions on Parallel and Distributed Systems*, 5(8):793–804, Aug. 1994.

[14] R. Libeskind-Hadas and E. Brandt. Origin-based fault-tolerant routing in the mesh. In *Proc. of the First IEEE Symposium on High-Performance Computer Architecture*, pages 102–111, Jan. 1995.

[15] R. Libeskind-Hadas, K. Watkins, and Tom Hehre. Fault-tolerant multicast routing in the mesh with no virtual channels. Technical Report HMC-CS-95-03, Harvey Mudd College, Department of Computer Science, June 1995.

[16] P. McKinley, H. Xu, A-H. Esfahanian, and L. Ni. Unicast-based multicast communication in wormhole-routed networks. *IEEE Transactions on Parallel and Distributed Systems*, 5(12):1252–1265, Dec. 1994.

[17] L. Ni and P. McKinley. A survey of wormhole routing techniques in direct networks. *IEEE Computer*, 26(2):62–76, Feb. 1993.

[18] D. Panda. Global reduction in wormhole $k-$ary n-cube networks with multidestination worms. In *Proc. 1995 Int. Parallel Processing Symposium*, pages 652–659, April 1995.

[19] D. Panda, S. Singal, and P. Prabhakaran. Multidestination message passing mechanism conforming to base wormhole routing scheme. In *Proc. 1994 Parallel Computer Routing and Communication Workshop*, number 853 in Lecture Notes in Computer Science, pages 131–145. Springer-Verlag, 1994.

[20] Y. Tsai and P. McKinley. A dominating set model for broadcast in all-port wormhole-routed 2D mesh networks. In *Proc. Eighth ACM Int. Conf. on Supercomputing*, July 1994.

[21] Y-C. Tseng, D. Panda, and T-H Lai. A trip-based multicasting model in wormhole-routed networks with virtual channels. *IEEE Transactions on Parallel and Distributed Systems*, To appear.

[22] H. Xu, P. McKinley, and L. Ni. Efficient implementation of barrier synchronization in wormhole-routed hypercube multicomputers. *Journal of Parallel and Distributed Computing*, 16:172–184, 1992.

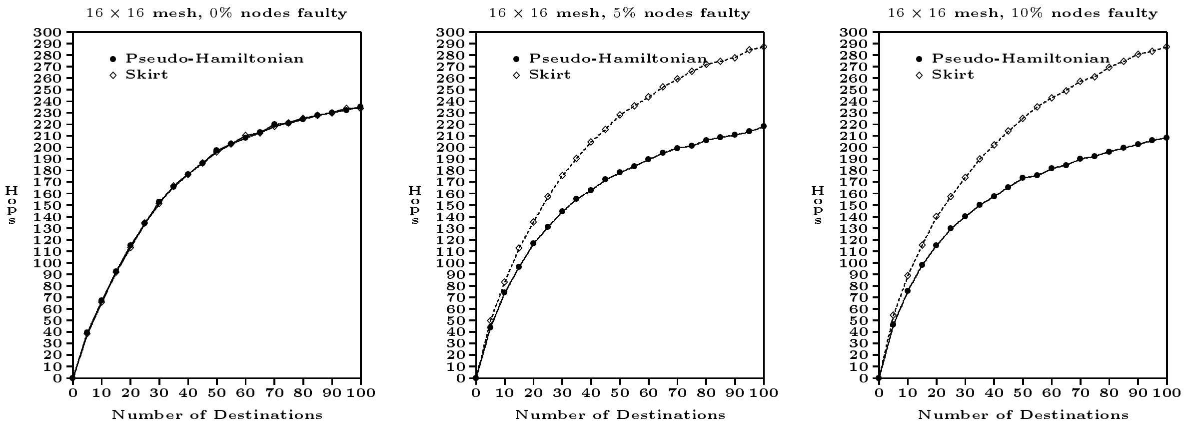

Figure 11: Hop count versus number of destinations in a 16×16 mesh with 0%, 5%, and 10% faulty nodes.

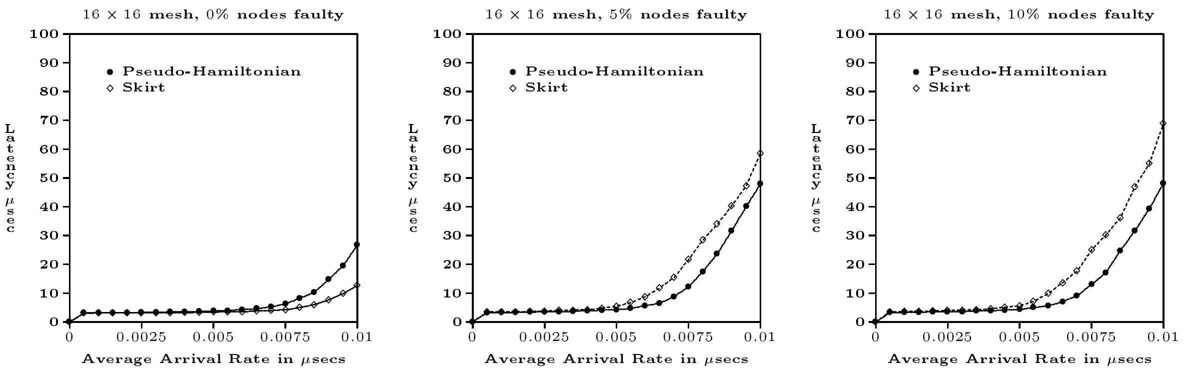

Figure 12: Latency in microseconds versus injection rate in a 16×16 mesh with 0%, 5%, and 10% faulty nodes.

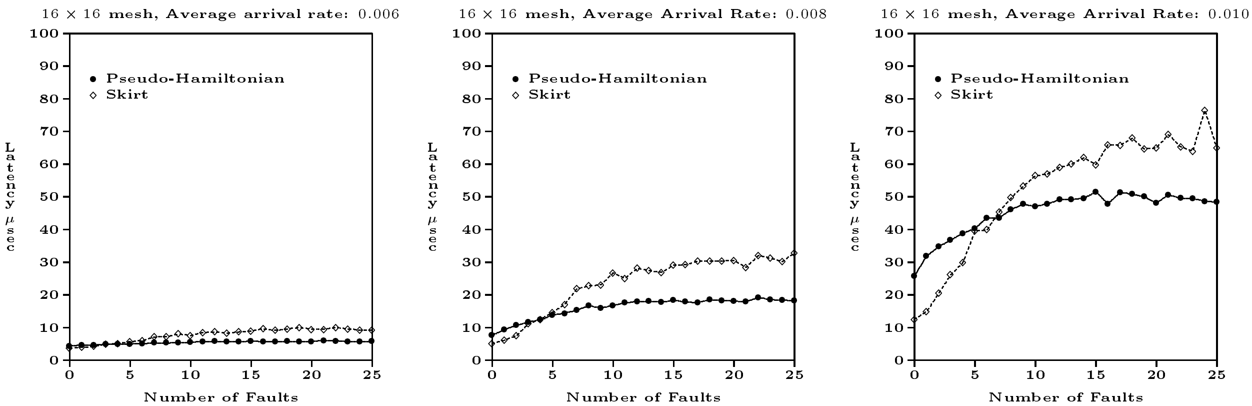

Figure 13: Latency in microseconds versus number of faults in a 16×16 mesh at average arrival rates of 0.006, 0.008, and 0.010 arrivals per μsec.