Origin-Based Fault-Tolerant Routing in the Mesh

Ran Libeskind-Hadas

Technical Report
HMC-CS-95-01

A preliminary version of this paper appears in the *Proceedings of the First IEEE Symposium on High Performance Architecture (HPCA '95).* This manuscript has been accepted for publication in *Future Generation Computer Systems.*

Origin-Based Fault-Tolerant Routing in the Mesh

Ran Libeskind-Hadas Eli Brandt
Department of Computer Science
Harvey Mudd College
Claremont, CA 91711

Abstract

The ability to tolerate faults is critical in multicomputers employing large numbers of processors. This paper describes a class of fault-tolerant routing algorithms for n-dimensional meshes that can tolerate large numbers of faults without using virtual channels. We show that these routing algorithms prevent livelock and deadlock while remaining highly adaptive.

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 [3, 11]. For example, a two-dimensional mesh is used in the Intel Paragon and Touchstone DELTA and a three-dimensional mesh is used in the MIT J-Machine.

In order to minimize network latency, the current generation of multicomputers employ the *wormhole routing* switching strategy [13]. Wormhole routing divides each packet to be sent through the network 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 to the destination processor. 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.

The path taken by a message from a source node to a destination node is dictated by a *routing algorithm*. In *source* routing algorithms, the source node completely specifies the routing path to the destination and stores this path in the header flit. This approach increases the size of the packet and disallows modification of the path once the header flit enters the network. Therefore, the current generation of multicomputers generally employ *distributed* routing algorithms. In these algorithms, each router examines the destination of an incoming header flit and decides on which output channel the message should be routed.

In order to minimize network latency, the routing algorithm must be fast. Moreover, to ensure scalability, each router must use at most a small constant number of state variables in making routing decisions. The use of routing tables, for example, is prohibitively expensive in large networks.

A routing algorithm may be either *deterministic* or *adaptive*. A deterministic algorithm allows only one path between every pair or processors whereas an adaptive algorithm allows a message to be routed on one of several possible paths. Similarly, a routing algorithm may be either *minimal* or *nonminimal*. A minimal routing algorithm restricts a message to be routed on a shortest path from the source to the destination, whereas a nonminimal algorithm allows longer paths to be used.

In order to ensure that all messages are eventually delivered to their destinations, a routing algorithm must be free from both *livelock* and *deadlock*. Livelock occurs when a message proceeds through the network but never arrives at its destination. Deadlock occurs when a message becomes permanently blocked in the network. A header flit becomes blocked when it cannot obtain a next channel because all such channels are currently owned by other messages. Deadlock arises when a cycle of blocked messages forms in which each blocked message on the cycle waits for a channel belonging to the next blocked message on the cycle. In this case all the messages along this cycle become permanently blocked.

A number of adaptive routing algorithms for the mesh have been proposed [2, 4, 7, 12]. Although adaptive algorithms allow some flexibility in routing, the routing restrictions still necessarily limit the number of paths that may be used. Consequently, certain patterns of faulty nodes will cause some messages to become undeliverable. As the number of nodes in multicomputers increases, the probability that some nodes will be faulty becomes significant. An important problem, therefore, is the design of fault-tolerant routing algorithms.

In this paper, we describe a new class of fault-tolerant routing algorithms for the mesh. These algorithms remain free from both livelock and deadlock even in the presence of a large number of faults. Moreover, these algorithms preserve a high level of adaptivity and can be implemented by fast routers. The remainder of this paper is organized as follows. In Section 2, we survey previous results on fault-tolerant wormhole routing. In Section 3, we introduce the notion of *origin-based* routing and show how this technique can be used to construct k^n distinct deadlock-free routing algorithms in an n-dimensional mesh with k processors along each dimension. In Section 4, we show how origin-based routing can be used to design effective fault-tolerant routing algorithms. In Section 5, we describe experimental results that indicate that origin-based fault-tolerant routing performs well even in the presence of a significant number of faults. Finally, we conclude in Section 6.

2 Previous Results

The objective of fault-tolerant routing is to maximize the number of non-faulty processors that can communicate with one another in the presence of faulty nodes and channels. Due to the routing restrictions required to enforce deadlock-freedom and the fact that certain fault patterns may isolate some non-faulty processors from the remainder of the network, in general it may not be possible to allow all non-faulty processors to communicate with one another.

Several approaches have been considered for achieving some level of fault-tolerance in wormhole-routed networks. These algorithms use the inherent fault-tolerance provided by adaptive routing. However, adaptive routing alone is not sufficient for fault-tolerance, because without additional information, a message may be routed along a path which is eventually blocked by faults. Linder and Harden [12] rectified this problem by augmenting the network with additional channels that allow a message to circumvent a single fault. Rather than using multiple physical channels between neighboring processors, this scheme uses *virtual channels* [5] to simulate multiple channels on a single physical channel. Although virtual channels are less expensive than physical channels, they do incur substantial overhead. A physical channel must be multiplexed to accommodate multiple virtual channels, resulting in a reduction in effective bandwidth. Moreover, each virtual channel requires its own flit buffers and other control hardware [3]. The algorithm proposed by Linder and Harden for n-dimensional meshes requires a number of virtual channels that grows exponentially with n, resulting in a substantial penalty in cost and performance. Moreover, this algorithm is not always able to tolerate multiple faults.

Chien and Kim [2] proposed an algorithm for fault-tolerant routing in meshes that uses at most 3 virtual channels per physical channel at the expense of being somewhat less adaptive than the algorithm of Linder and Harden. Chien and Kim's algorithm can tolerate multiple faults but a large fraction of non-faulty processors may be deactivated to ensure the correctness of the routing algorithm. For example, a single fault may require that $k-1$ non-faulty processors be deactivated, where k is the number of processors in each dimension of the mesh.

Glass and Ni [8] proposed a fault-tolerant routing algorithm for meshes that extends their earlier *turn-model* for adaptive routing [7]. Glass and Ni's algorithm is highly adaptive and, in contrast to previous approaches, requires no virtual channels. The routing algorithm tolerates at least $n-1$ faults in an n-dimensional mesh. However, there are many fault patterns with n or more faults for which this algorithm is unable to route between all non-faulty processors even though the non-faulty processors form a strongly connected region.

The class of algorithms described in this paper are highly adaptive and require no virtual channels. In contrast to previous approaches, these algorithms can tolerate a large number of faults with graceful degradation in performance; An n-dimensional mesh with k nodes along each dimension can tolerate at least $(k-1)^{n-1}$ faults.

3 Origin-Based Routing

In this section we describe the origin-based routing technique for fault-free meshes which underlies the class of fault-tolerant routing algorithms described in Section 4. For simplicity, in the remainder of this paper we restrict our attention to $k \times k$ meshes, although all results described here can be immediately extended to meshes of arbitrary dimension.

In origin-based routing, any one of the k^2 nodes in the mesh is designated as the *origin*. The origin node is given coordinates $(0,0)$ and the coordinates of the remaining nodes are assigned with respect to the origin. We will henceforth use the following terms and notation to simplify our discussion: We will let o denote the origin node, s denote a source node (the node sending the message), and t denote a destination node. The coordinates of s and t

will be denoted (x_s, y_s) and (x_t, y_t), respectively. Without loss of generality, we henceforth assume that $x_t \geq 0$ and $y_t \geq 0$ (all other cases are symmetric). The row and column containing the origin will be called the x- and y- axes, respectively. Finally, given a channel from node u to node v, u will be denoted the *head* of the channel and v will be denoted the *tail* of the channel.

In origin-based routing, the channels of the network are partitioned into two disjoint subnetworks, the *IN subnetwork* and the *OUT subnetwork*. The IN subnetwork contains all channels directed towards the origin and the OUT subnetwork contains all channels directed away from the origin. Thus a channel from node (x_1, y_1) to node (x_2, y_2) is in the IN subnetwork if $|x_1| > |x_2|$ (in which case $y_1 = y_2$) or $|y_1| > |y_2|$ (in which case $x_1 = x_2$) and is in the OUT subnetwork otherwise.

Definition 1 *The* outbox *for a destination node t with coordinates (x_t, y_t) $(x_t, y_t \geq 0)$ is the set of all nodes n with coordinates (x_n, y_n) such that $0 \leq x_n \leq x_t$ and $0 \leq y_n \leq y_t$.*

The origin-based routing algorithm with respect to a given origin node comprises the following two phases:

Phase 1: While the header flit is not in the outbox for its destination node t, route adaptively using the IN subnetwork.

Phase 2: The header flit is now in the outbox. Route adaptively towards t using the OUT subnetwork.

Origin-based routing is not restricted to minimal paths. In order to ensure that only minimal paths are used, Phase 1 of the algorithm can be easily modified. We define a channel to be *productive* with respect to a destination node t if its tail node is closer to t than is its head node, and *non-productive* otherwise. The minimal version of Phase 1 simply chooses a productive channel over a non-productive one when the router has two channels to choose from.

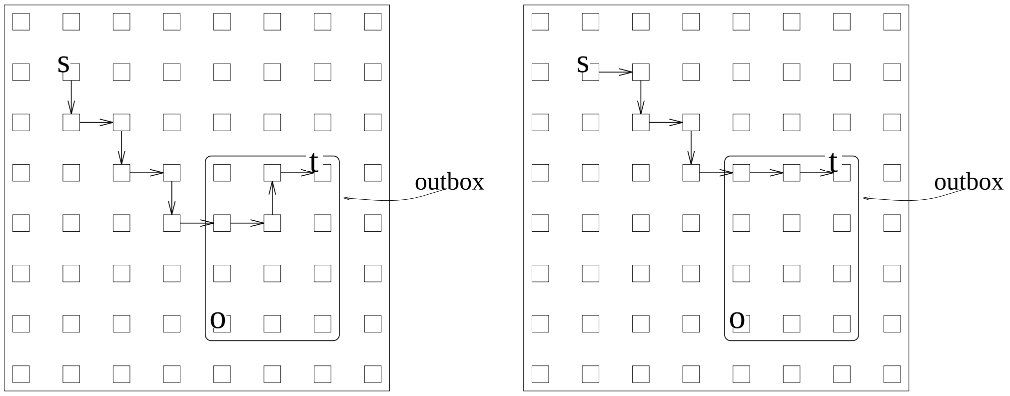
Figure 1a shows a nonminimal path found by an origin-based routing algorithm and Figure 1b shows a minimal path between the same two nodes using the minimal version of the algorithm.

We now show that origin-based routing is correct by proving that it is free from both livelock and deadlock.

Theorem 1 *Origin-based routing is livelock-free.*

Proof: The algorithm begins in Phase 1. By definition, each channel in the IN subnetwork reduces the distance from the header to the origin by exactly 1. Thus, in a finite number of steps the header either reaches the origin, which by definition is in the outbox for t, or reaches some other node in the outbox for t. At this point, the algorithm enters Phase 2. Phase 2 of the routing algorithm ensures that each traversed channel reduces the distance from the header flit to the destination t by exactly 1. Thus, the header reaches the destination in a finite number of steps. $\square$

Theorem 2 *Origin-based routing is deadlock-free.*



(a) (b)

Figure 1: Examples of (a) nonminimal origin-based routing and (b) minimal origin-based routing.

Proof: Dally and Seitz [5] have shown that a wormhole routing algorithm is deadlock-free if the channels can be labeled with integers in such that every path allowed by the routing algorithm has the property that its channels have strictly decreasing labels.

Let $distance(a, b)$ denote the rectilinear distance between two nodes a and b. Let δ denote the maximum rectilinear distance between the origin and every other node in the mesh. Let (u, v) denote the channel from node u to node v. If $distance(u, o) > distance(v, o)$ then, by definition, channel (u, v) is in the IN subnetwork and is labeled with the value $\delta + distance(u, o)$. Otherwise, $distance(u, o) < distance(v, o)$ and the channel is in the OUT subnetwork. (It is easily verified that $distance(u, o) \neq distance(v, o)$ for neighboring nodes u and v in a mesh.) In this case, the channel is labeled with the value $\delta - distance(u, o)$. Observe that every channel in the IN subnetwork has label at least $\delta + 1$ while every channel in the OUT subnetwork has label at most δ.

We must now show that in any path selected by the origin-based routing algorithm, the channels are strictly decreasing. Let v denote the location of the header flit, let (u, v) denote the input channel holding the header flit, and let (v, w) denote the output channel selected by the router. We consider routing in Phase 1 and Phase 2 separately.

Case 1: The algorithm is in Phase 1. Therefore, (u, v) and (v, w) are both in the IN subnetwork and, by definition, have labels $distance(u, o) + \delta$ and $distance(v, o) + \delta$, respectively. Since (u, v) is in the IN subnetwork, $distance(u, o) > distance(v, o)$ and thus the label on (u, v) is strictly larger than the label on (v, w).

Case 2: The algorithm is in Phase 2 and, thus, the header flit is located at a node in the outbox for t. Channel (v, w) is necessarily in the OUT subnetwork while channel (u, v) is in the IN subnetwork if the flit has just arrived at the outbox and is in the OUT subnetwork otherwise. In the first case, the label on (u, v) is at least $\delta+1$ which is larger than the label on any channel in the OUT subnetwork. In the second case, channels (u, v) and (v, w) have labels $\delta - distance(u, o)$ and $\delta - distance(v, o)$, respectively. Since

(u, v) is in the OUT subnetwork, $distance(u, o) < distance(v, o)$ and thus the label on (u, v) is again strictly larger than the label on (v, w). □

We conclude with several observations on origin-based routing. First, we note that the negative-first routing algorithm proposed by Glass and Ni [7] is, in fact, a special case of origin-based routing where the origin is placed at the southwest corner of the mesh. Second, we observe that origin-based routing allows high adaptivity between most pairs of nodes. For example, if minimum routing is used, the only messages that must be routed deterministically are those from s to t where s and t are in the same row or column or s and t are in the same quadrant with respect to the origin node and either $|x_s| < |x_t|$ and $|y_s| > |y_t|$, or $|x_s| > |x_t|$ and $|y_s| < |y_t|$. The number of pairs of processors that must route deterministically is, therefore, a function of the location of the origin. It is easily verified that this function achieves its maxima when the origin is placed at one of the four corners of the mesh (negative-first routing or its isomorphs) and is minimized when the origin is at the center of the mesh [1]. On the other hand, there is a hot-spot effect at the origin since a message sent between nodes in opposite quadrants must pass through the origin. For example, for uniform traffic the hot-spot effect is minimized when the origin is placed at a corner and is maximized at the center. The decision of how to optimally balance these opposing factors largely depends on the traffic workload of the particular application.

4 Fault-Tolerant Routing Algorithms

In this section we show how the origin-based routing technique can be extended to routing in the presence of faulty nodes. The algorithm is described in Subsection 4.1. Subsection 4.2 describes extensions to the algorithm that increase its fault-tolerance and performance.

4.1 Fault-Tolerant Origin-Based Routing

We assume that node faults are *total*: neither the processor nor the router at a faulty node can be used [9]. In the interest of simplicity, faulty channels are not considered separately, but instead can be modeled by introducing faults at the endpoints of each faulty channel. In addition, in the interest of simplifying our discussion, we assume that all node faults are detected using a built-in self-test mechanism [10] and that the reconfiguration process is controlled by a central host processor (although the techniques described here can be implemented by a distributed algorithm as well). Finally, we assume that faults can occur at any time. When a fault occurs, the host processor (or the distributed protocol) establishes new routing rules for each non-faulty node, each node rolls back its computation to the most recently checkpointed state (assuming that faulty nodes may inject incorrect messages into the network), and computation resumes [6].

In order to maintain scalability and to maximize performance, each node may use only a small number of state variables in making routing decisions. Fault patterns that include connected concave regions will require some nodes to maintain a large amount of information about the network in order to navigate messages in and around complex concavities. Therefore, our algorithm begins by introducing *pseudo-faults*: non-faulty processors that are disabled in order to make all connected faulty regions convex. However, even rectangular fault regions can, in pathological cases, form maze-like patterns that require some nodes to store considerable amounts of information for routing. By making all faulty regions square, we are able to show that routing can be performed by a fast and simple router storing only four values: the distance to the nearest fault directly to the east, west, north, and south. Faulty regions can be made square by an efficient algorithm that first computes the connected components using breadth-first search, fills the bounding rectangle of each component with pseudo-faults, and finally extends each bounding rectangle into a square.

Pseudo-faults are also used by Chien and Kim [2] in their planar-adaptive routing algorithm. However, the planar-adaptive algorithm requires that all nodes on an edge of the mesh be disabled if a single fault occurs on that edge. In contrast, our algorithm typically requires only a few neighboring nodes to be disabled in order to form square regions, as indicated by the experimental results in Section 5.

Once pseudo-faults are introduced and each node has obtained the distance to the nearest faulty node in each direction, an origin is selected for the routing algorithm. For simplicity, we will assume in this subsection that an origin is selected with the property that there is no faulty node in the row or column containing the origin. Observe that in a $k \times k$ mesh, any $k-1$ or fewer faults allow such an origin to be found. In Subsection 4.2 we show how this constraint on origin-placement can be relaxed to allow potentially larger numbers of faults.

The fault-tolerant routing algorithm begins routing exactly as in Phase 1 of the origin-based routing algorithm described in the previous section. In other words, a message is routed first on the IN subnetwork. Observe that in this phase, each router that receives the header flit has at least one possible output channel on which to forward the message: First, every node that is not on one of the two axes has two output channels that are in the IN subnetwork. At most one of the nodes at the tails of these two channels can be faulty since otherwise there would be a concave faulty region. A node on one of the axes has only one output channel in the IN subnetwork. However, by our selection of the origin the axes contain no faulty elements, and thus this output channel can always be used.

After proceeding on the IN subnetwork, the header flit eventually arrives at an edge of the outbox for its destination, t. (In the worst case, the header flit arrives at the origin, which is by definition in the outbox.) However, unlike the fault-free case, we cannot simply enter Phase 2 at this point. To see this, consider for example the situation in Figure 2. In this case, any path taken from the current node along the OUT subnetwork will cause the message to "overshoot" node t. The message, therefore, will be required to reenter the IN subnetwork to get to t. However, recall from the proof of Theorem 2 that channels in the OUT subnetwork are prohibited from routing to channels on the IN subnetwork in order to prevent deadlock.

In order to ensure that every message can be delivered to its destination once it arrives in the outbox, we introduce the concept of *safe* nodes.

Definition 2 *A node v is* safe *with respect to destination node t if the following conditions are satisfied:*

1. *Node v is in the outbox for t.*

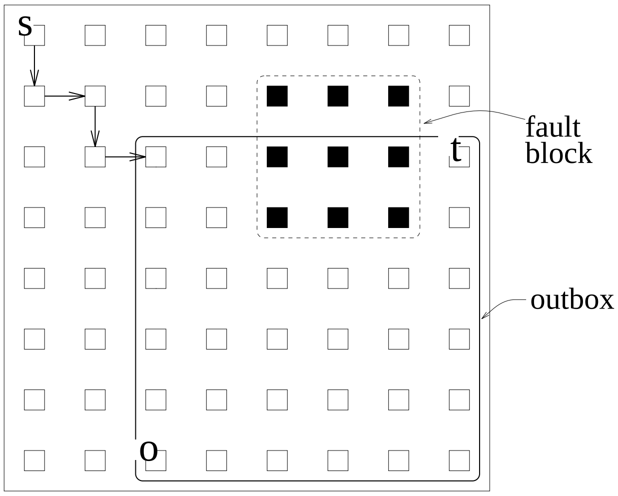


Figure 2: A faulty region blocking a message in the outbox.

2. *For any pattern of faults in which and v and t are non-faulty, there exists a fault-free path in the OUT subnetwork from v to t.*

In other words, although not every node in the outbox has the property that there exists a path from this node to t using only channels in the OUT subnetwork, this property is satisfied for safe nodes. Surprisingly, there are large number of nodes that are always safe and these nodes can be found easily.

Definition 3 *The* diagonal band *for destination node t is the set of all nodes v satisfying the following properties:*

1. *Node v is in the outbox for t.*

2. *If (x_t, y_t) and (x_v, y_v) denote the coordinates of t and v, respectively, then $x_t - x_v = y_t - y_v + e$ where $e \in \{-1, 0, 1\}$.*

An example of a diagonal band is shown in Figure 3.

Theorem 3 *The nodes on the diagonal band for t are safe with respect to t.*

Proof: Let v be a node on the diagonal band with coordinates (x_v, y_v) and let destination node t have coordinates (x_t, y_t). The proof is by strong induction on the ℓ^∞ distance from v to t: $\max\{|x_t - x_v|, |y_t - y_v|\}$. Since we are assuming here that $x_t \geq x_v$ and $y_t \geq y_v$, absolute values are not needed here and will henceforth be omitted. The induction basis is $n = 0$ in which case $v = t$ and thus v is safe by definition. Assume that all nodes on the diagonal band at ℓ^∞ distance n or less from t are safe. Consider a node v on the diagonal band at distance $n+1$ from t. Thus, $x_t - x_v = y_t - y_v + e$, $e \in \{-1, 0, 1\}$ and $\max\{x_t - x_v, y_t - y_v\} = n + 1$. First, consider the case that $e = -1$. In this case, $x_t - x_v = n$ and $y_t - y_v = n+1$. The node v_n immediately to the north of v has coordinates $(x_v, y_v + 1)$, is therefore in the diagonal

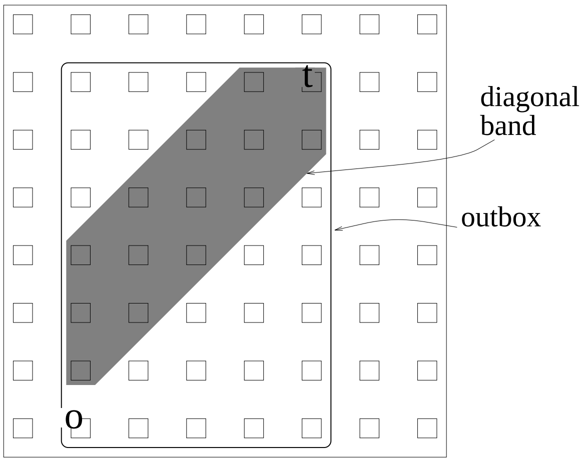


Figure 3: The diagonal band.

band, and is at distance n from t. If v_n is non-faulty then, by the induction hypothesis, there exists a path in the OUT subnetwork from v_n to t and thus from v to t. If, however, v_n is faulty then it is an element of a faulty square region of width at least 1. In this case, a flit can be routed from v eastwards along the bottom edge of this faulty square region. None of the nodes along this edge can be faulty because faulty regions are convex. After some number of eastward steps, $k \geq 1$, the flit must reach a node to the southeast of the faulty square. This node must still be in the outbox for t, since otherwise t itself would be in the faulty square region, contradicting the assumption that t is non-faulty. At this point the flit is routed northwards k steps along the right edge of the faulty square, which is again necessarily fault-free. The flit is now at node $(x_v + k, y_v + k)$ which is on the diagonal band at distance $n + 1 - k$ from t. Since $n + 1 - k \leq n$, by the induction hypothesis there exists a path on the OUT subnetwork from this node to t and thus from v to t. We conclude that v is safe.

By symmetry, v is also safe when $e = 1$. Therefore, the only remaining case is that $e = 0$. In this case, the nodes immediately to the north and east of v have coordinates $(x_v, y_v + 1)$ and $(x_v + 1, y_v)$, respectively, and are therefore also in the diagonal band and at distance $n + 1$ from t. These nodes have just been shown to be safe. These nodes cannot both be faulty, since by convexity this would imply that v is faulty. Therefore, v is adjacent to at least one safe node and is therefore also safe. □

The proof of Theorem 3 suggests that once the header flit reaches a safe node v in the outbox, continuing to the destination node is simple: If v has a safe non-faulty neighbor, then forward the flit to that node. Otherwise, v must be on the edge of a faulty region. In this case v advances along the edge of the faulty region towards t and turns towards the diagonal band when it reaches the corner of the faulty square. The proof of Theorem 3 shows that the flit is guaranteed to return to a safe node using this simple routing strategy. We call this routing procedure *safe routing*. An example of safe routing is illustrated in Figure 4.

The fault-tolerant version of origin-based routing is the following:

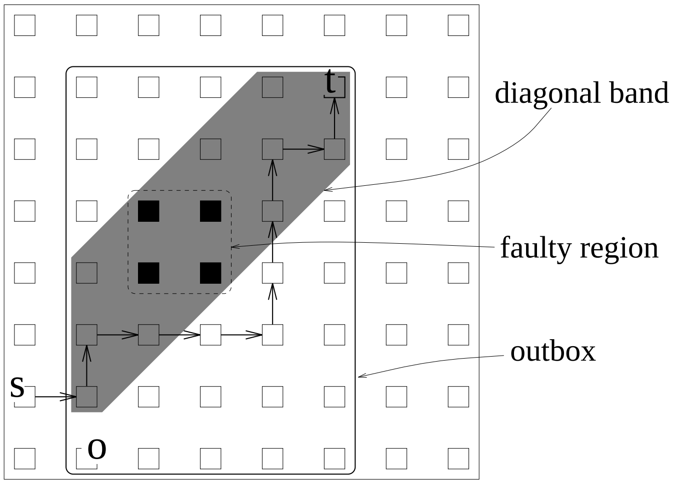


Figure 4: An example of safe routing.

Phase 1: While the header flit is not in the outbox for its destination node t, route adaptively using the IN subnetwork.

Phase 2: The header flit is now in the outbox. While the header flit is not at a safe node, compute the distance to the nearest safe node in each direction. Compare each such distance to the distance to the nearest fault in that direction. If the safe node is closer than the fault, route directly to the safe node. Otherwise, continue routing on the IN subnetwork.

Phase 3: The header flit is now at a safe node. Apply safe routing to reach t.

In general, minimal routing is not possible in the presence of faulty nodes since a fault between two nodes in the same row or column necessitates nonminimal routing. However, the routing algorithm can again be easily modified to avoid unnecessarily long paths (at the expense of reducing adaptivity) by modifying Phase 1 to select productive channels over non-productive ones.

Figure 5a shows a path found by a fault-tolerant origin-based routing algorithm and Figure 5b shows a path found by the productive version of this algorithm.

We now show that fault-tolerant origin-based routing is correct by proving that it is free from both livelock and deadlock.

Theorem 4 *Fault-tolerant origin-based routing is livelock free.*

Proof: The algorithm begins in Phase 1. As we have already shown, there must exist at least one output channel that may be used at each node in this phase. Each channel in the IN subnetwork reduces the distance from the header to the origin by exactly 1. Thus, in a finite number of steps the header either reaches the origin, which by definition is in the outbox for t, or reaches some other node in the outbox for t. At this point, the algorithm enters Phase 2.

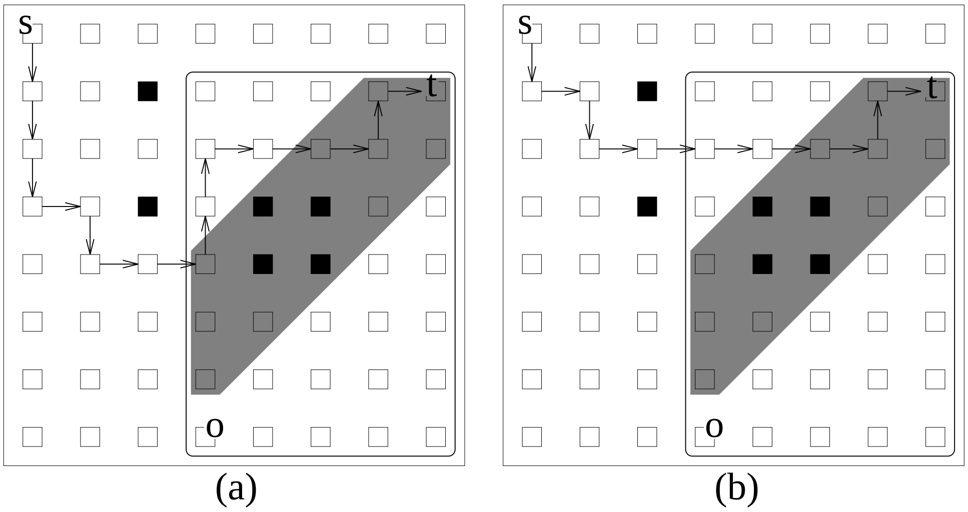


Figure 5: Examples of (a) fault-tolerant origin-based routing and (b) productive fault-tolerant origin-based routing.

In Phase 2 the header flit is either routed to a safe node or, by staying in the IN subnetwork, arrives at the origin in a finite number of steps. By the definition of the diagonal band, there exist safe nodes on either the x- or y- axes. Since the axes contain no faults by our choice of the origin, the header flit can be routed directly from the origin to a safe node. The algorithm now enters Phase 3.

In Phase 3 the algorithm routes from one safe node to another on the OUT subnetwork. Since each channel in the OUT subnetwork reduces the rectilinear distance to t by exactly 1, the header flit reaches t in a finite number of steps. □

Theorem 5 *The fault-tolerant origin-based routing algorithm is deadlock-free.*

Proof: The channels of the mesh are labeled as in the proof of Theorem 1. We must show that in any path selected by the fault-tolerant origin-based routing algorithm, the channels are strictly decreasing. We consider routing in each of the three phases separately.

Case 1: The algorithm is in Phase 1. The argument is identical to that of **Case 1** in the proof of Theorem 2.

Case 2: The algorithm is in Phase 2. While routing is from an input channel in the IN subnetwork to an output channel in the IN subnetwork, correctness follows from Case 1. When the router determines that the header can be routed towards a safe node, the path to the safe node is either entirely in the IN subnetwork or entirely in the OUT subnetwork. Therefore, channel numbers along this path are strictly decreasing.

Case 3: The algorithm is in Phase 3. The argument is identical to that of **Case 2** in the proof of Theorem 2. □

4.2 Extensions

We begin by showing how an origin can be established even if there is a fault in every row and column of the mesh. Recall that once pseudo-faults are introduced to form square faulty regions, an origin is selected such that there are no faults on the axes. This property guarantees that Phase 2 of the fault-tolerant routing algorithm terminates with the header flit at a safe node. If more than $k-1$ faults are present, it may not be possible to find such a location for the origin. In this case, we can select an origin such that the bounding box induced by the faults directly to the east, west, north, and south of the origin contains the largest number of non-faulty nodes. This bounding box is then the submesh that is actually used, and all non-faulty nodes outside the bounding box are disabled.

Next, we show how the region of safe nodes can be expanded substantially beyond the diagonal band in 2-dimensional meshes. This result has two consequences. First, it allows much greater adaptivity in Phase 3 of the algorithm. Second, fault-tolerant origin-based routing using the diagonal band differs from fault-free origin-based routing, even when there are no faults. This is because the fault-tolerant algorithm requires that safe routing be performed in Phase 3. The extended safe region described here adjusts dynamically to the fault pattern in the mesh. In particular, when there are no faults, the safe region becomes the entire outbox. Therefore, by using the extended safe region rather than the diagonal band, the fault-tolerant algorithm reduces to standard origin-based routing when there are no faults.

The size of the extended safe region is a function of the width of the widest faulty square and the ℓ^∞ distance between the nearest faulty squares[1]. Observe that since two faulty squares must be separated by at least one non-faulty node, the minimum ℓ^∞ distance is at least 2. Intuitively, wide faulty squares that are close to one another pose the most serious challenge to routing and thus require the most restrictive safe zones. Similarly, in the case of no faulty nodes, the entire outbox should be safe.

Recall that without loss of generality, we are assuming that the destination node t has coordinates (x_t, y_t) such that $x_t \geq 0$ and $y_t \geq 0$. We define the *width* of the safe zone at $(x_t - j, y_t - j)$ to be the number of safe nodes on and below (as well as on and to the west of) this node. Let $S(j)$ represent the width of the safe zone at $(x_t - j, y_t - j)$.

Theorem 6 *Let w denote the width of the largest faulty square region and let d denote the minimum ℓ^∞ distance between the two nearest faulty regions. Then*

$$S(j) = h_j(w, d) \equiv \lfloor \frac{j}{w} \rfloor \cdot (d-1) + 2$$

Proof: The proof is by strong induction on j. The induction basis is $j = 0$, for which $h_j(w, d) = 2$. Since t is on the diagonal band and the width of the diagonal band is 2, the basis is trivial. For the induction hypothesis, we assume that $S(i) = h_i(w, d)$ for $0 \leq i < j$. In the induction step we wish to show that $S_j = h_j(w, d)$.

[1]Recall that the ℓ^∞ distance between points (x_1, y_1) and (x_2, y_2) is defined as $\max\{|x_1 - x_2|, |y_1 - y_2|\}$. The ℓ^∞ distance between two regions is the minimum ℓ^∞ distance between two nodes, one from each of the two regions.

Assume the contrary. That is, $S(j) = h' < h_j(w, d)$. Hence, node $(x_t - j, y_t - j - h')$ is unsafe, while nodes $(x_t - j, y_t - j - i)$ are safe for $0 \le i < h'$. Thus, there exists a fault pattern F with fault blocks of width w and with ℓ^∞ distance at least d between blocks such that there does not exist a path in the OUT subnetwork from $(x_t - j, y_t - j - h')$ to t. Under F, node $(x_t - j, y_t - j - h' + 1)$ must be faulty, since otherwise there would be a path in the OUT subnetwork from $(x_t - j, y_t - j - h')$ to t. Therefore, node $(x_t - j, y_t - j - h' + 1)$ is part of a faulty square block of width ℓ, $1 \le \ell \le w$. Let $x_t - j + m$ be the x-coordinate immediately to the left of this faulty square block. Note that $1 \le m \le \ell \le w$.

The faulty square induces a fault-free path in the OUT subnetwork from $(x_t - j, y_t - j - h')$ to $(x_t - j + m, y_t - j - h' + \ell + (d - 1))$. Since $m \ge 1$, the induction hypothesis can be invoked to give $S(j - m) = (d - 1)\lfloor \frac{j-m}{w} \rfloor + 2$. Therefore, if the induced path reaches node $(x_t - j + m, y_t - j + m - [(d - 1)\lfloor \frac{j-m}{w} \rfloor + 1])$, then node $(x_t - j, y_t - j - h')$ is safe. In other words, we must only show that

$$y_t - j - h' + \ell + (d - 1) \ge y_t - j + m - [(d - 1)\lfloor \frac{j - m}{w} \rfloor + 1]$$

or equivalently

$$j + h' - \ell - (d - 1) \le j - m + [(d - 1)\lfloor \frac{j - m}{w} \rfloor + 1].$$

We now construct this inequality.

We begin by observing that since $m \le w$,

$$\lfloor \frac{j}{w} \rfloor - 1 \le \lfloor \frac{j}{w} - 1 \rfloor \le \lfloor \frac{j}{w} - \frac{m}{w} \rfloor = \lfloor \frac{j - m}{w} \rfloor.$$

Multiplying the leftmost and rightmost sides of this inequality by $(d - 1)$, we obtain

$$(d - 1)\lfloor \frac{j}{w} \rfloor - (d - 1) \le (d - 1)\lfloor \frac{j - m}{w} \rfloor.$$

Since $h' \le (d - 1)\lfloor \frac{j}{w} \rfloor + 1$ by assumption, the inequality above implies that

$$h' - d \le (d - 1)\lfloor \frac{j}{w} \rfloor - (d - 1) \le (d - 1)\lfloor \frac{j - m}{w} \rfloor.$$

Subtracting ℓ from the left hand side and m from the right hand side ($m \le \ell$) and adding $j + 1$ to both sides results in the desired inequality and the induction is complete. □

An example of an extended safe region given by Theorem 6 is shown in Figure 6. Note that in this example, $w = 1$ and $d = 3$. The extended safe region is substantially wider than the diagonal band, allowing very high adaptivity in Phase 3 of the routing algorithm.

We conclude with several remarks and observations about Theorem 6. First, note that each non-faulty node need only keep the values of w and d for the current fault pattern. No specific information about the location of faults is maintained by the nodes. This is critical in ensuring scalability of the network and performance of the routers. The values of w and d are obtained during reconfiguration when a new fault occurs. With these values alone, each router can determine if a given node is safe. Next, note that when there are no faults, $w = 0$ and $d = \infty$ and thus Theorem 6 tells us that all nodes in the outbox are safe. Finally, we observe that regardless of the values of w and d, the safe zone given by Theorem 6 always includes the diagonal band because $\lfloor \frac{j}{w} \rfloor \cdot (d - 1) + 2 \ge 2$. In most cases, the extended safe zone is larger than the diagonal band and smaller than the entire outbox.

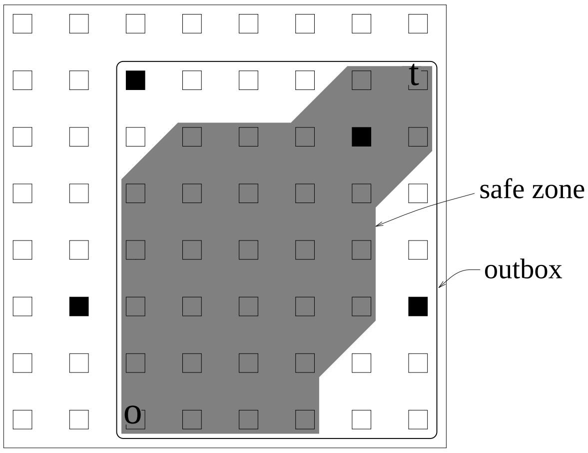


Figure 6: An extended safe zone.

5 Experimental Results

In order to measure the fault-tolerance and performance of origin-based routing, several simulations were performed for a 10×10 mesh. Since pseudo-faults are introduced by origin-based routing, our first experiments measure the number of usable nodes as a function of the number actual faulty nodes. The next set of experiments measure the performance degradation incurred by this class of algorithms. Since a number of factors other than the routing algorithm itself contribute to the performance of the network, these experiments attempt to isolate some of the features of origin-based routing that are correlated to network performance. Specifically, we study the effect of faults on adaptivity and path length. In all the experimental results described here, faults are introduced at the nodes with equal probability and all results reported here are averaged over at least 2000 trials. When there is a choice of location for the origin, the origin is placed as close as possible to one of the corners of the mesh. Thus, in the case of no faults, the algorithm reduces to negative-first routing.

Our first experiments examine the number of faults that can be tolerated by origin-based routing. Recall that the algorithm begins by introducing pseudo-faults to make all faulty regions square. As the number of real faults increases, we expect the number of pseudo-faults to increase as well, eventually disabling the entire mesh. Our experimental results indicate that this occurs when slightly more than 20% of the nodes have real faults. When 10% of the nodes are faulty, for example, on average less than 12% of the non-faulty nodes are disabled with pseudo-faults and all remaining nodes can be used. In contrast, with 10% faulty nodes, the fault-tolerant version of negative-first routing [8] is able to ensure deadlock-free routing with probability less than 0.32 in a 10×10 mesh. The fraction of non-faulty nodes disabled by origin-based routing as a function of the number of real faults is given in Figure 7.

Next, we examine the performance of fault-tolerant origin-based routing. The first performance indicator we consider is adaptivity. Our implementation uses productive routing. That is, the router selects output channels that reduce the distance to the destination when-

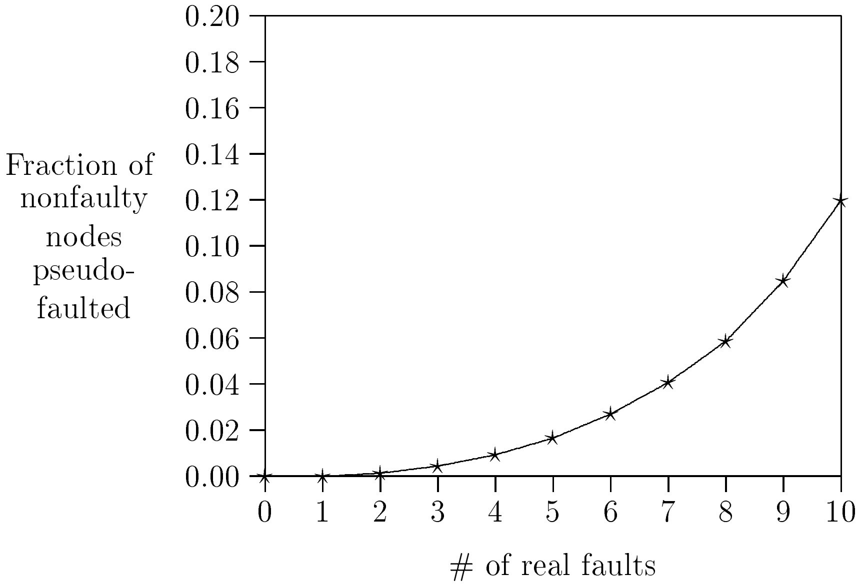


Figure 7: Fraction of non-faulty nodes disabled with pseudo-faults as a function of the number of real faults.

ever possible. Thus, each router has either 1 or 2 output channels from which to select. Figure 8 shows the average number of output channels available as a function of the number of real faults in the mesh. In the case of 0 faults, the algorithm behaves exactly like the negative-first routing algorithm and has an average of about 1.34 output channels available at each router. (A combinatorial argument shows that this value approaches 1.5 as the width of the mesh goes to infinity.) Figure 8 suggests that adaptivity degrades gracefully with the number of faults.

The next performance indicator we examine is the average length of a path taken by the origin-based routing algorithm. More precisely, we measure the increase in the length of a path from a random source to a random destination with respect to the length of a shortest path from the source to the destination. Experimental results suggest that this increase is quite modest, with approximately a 9% increase in path length over shortest paths when as many as 10% of the nodes are faulty. These results are summarized in Figure 9.

6 Conclusions

In this paper we have described a new class of fault-tolerant routing algorithms for wormhole-routed meshes. Unlike many previous techniques, fault-tolerant origin-based routing does not require virtual channels. Moreover, origin-based routing algorithms are able to sustain a substantial number of faults, with graceful degradation in adaptivity and path length. If available, virtual channels could be used to further increase adaptivity and reduce path

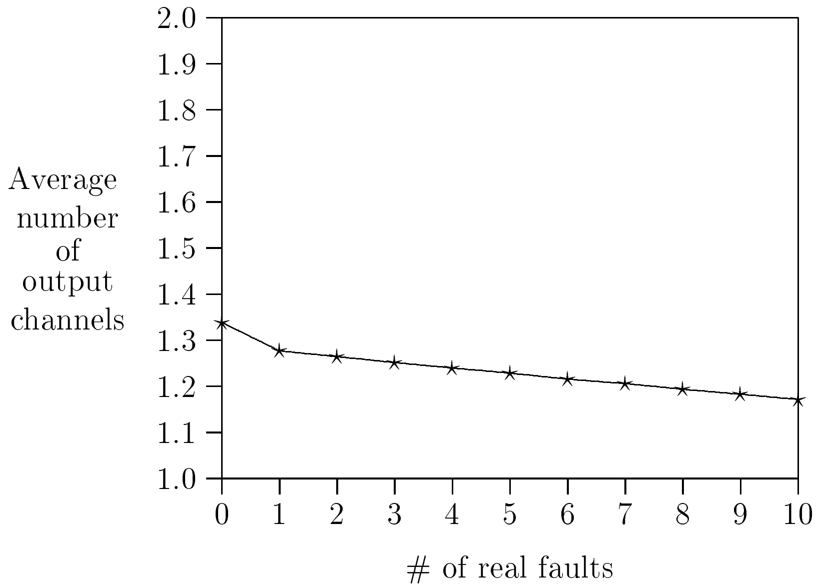


Figure 8: Average number of output channels available at each router as a function of the number of real faults.

length.

The fault-tolerant negative-first routing algorithm proposed by Glass and Ni [8] also provides fault-tolerance without virtual channels. Negative-first routing and origin-based routing both re-label the channels of the mesh in order to circumvent faulty nodes. Negative-first routing performs re-labeling in a purely local fashion, resulting in fast and simple reconfiguration. Origin-based routing requires that either a centralized or distributed algorithm be invoked to globally re-label the channels of the network. This process is likely to result in slower reconfiguration but is able to tolerate a larger number of faults.

Acknowledgements

The authors wish to thank Mr. Daniel Kelley for many valuable discussions and expert assistance in preparation of the manuscript.

References

[1] Andrew Basile. Optimizing mesh network communication. Senior Honors Thesis in Mathematics, Harvey Mudd College, 1994.

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

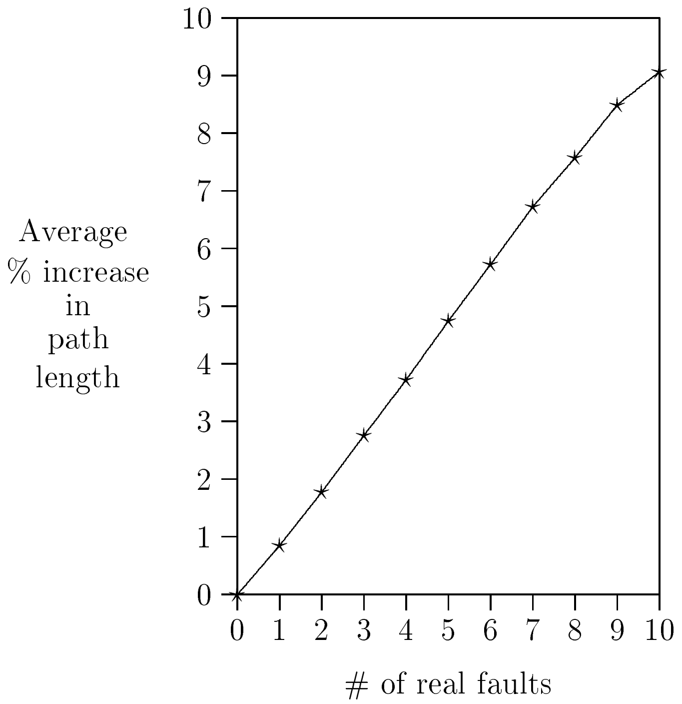


Figure 9: Average percentage increase in path lengths over shortest paths as a function of the number of real faults.

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

[4] W. Dally and H. Aoki. Deadlock-free adaptive routing in multicomputer networks using virtual channels. Technical report, Massachusetts Institute of Technology, Laboratory for Computer Science, Sept. 1990.

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

[6] S. Dutt and J. Hayes. Some practical issues in the design of fault-tolerant multiprocessors. *IEEE Transactions on Computers*, 41(5):588–598, May 1992.

[7] C. Glass and L. Ni. Adaptive routing in mesh-connected networks. In *Proc. 12th Int. Conf. on Distributed Computing Systems*, pages 12–19, June 1992.

[8] C. Glass and L. Ni. Fault-tolerant wormhole routing in meshes. In *FTCS23*, pages 240–249, June 1993.

[9] J. Hastad, T. Leighton, and M. Newman. Reconfiguring a hypercube in the presence of faults. In *Proc. 19th ACM Symp. Theory of Computing*, pages 274–284, 1987.

[10] J. Kuhl and S. Reddy. Distributed fault-tolerance for large multiprocessor systems. In *Proc. 7th Int. Symp. on Computer Architecture*, pages 23–30, May 1980.

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

[12] D. Linder and J. Harden. An adaptive and fault tolerant wormhole routing strategy in k-ary n-cubes. *IEEE Transactions on Communications*, 40:2–12, Jan. 1991.

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