In Procs. of 1994 Int'l Symp. on Parallel Architectures, Algorithms, and Networks, Dec. 1994

Adaptive Fault-Tolerant Wormhole Routing Algorithms with Low Virtual Channel Requirements*

Suresh Chalasani
Electrical & Comp. Engr. Dept.
Univ. of Wisconsin-Madison
Madison, WI 53706-1691
suresh@cauchy.ece.wisc.edu

Rajendra V. Boppana
Div. of Math. and Computer Science
The Univ. of Texas at San Antonio
San Antonio, TX 78249-0664
boppana@ringer.cs.utsa.edu

Abstract. *We present simple methods to enhance a recently proposed class of fully-adaptive algorithms for fault-tolerant wormhole routing. These algorithms are based on Duato's theory and are being used in several research projects. We show that with three virtual channels per physical channel, mulitple rectangular shaped fault blocks can be tolerated in two-dimensional meshes. There is no restriction on the number of faults, and maintaining fault information locally is sufficient. The logic for fault-tolerant routing is used only in the presence of faults, and there is no performance degradation in the absence of faults. The proposed technique incorporates fault-tolerance into wormhole algorithms with simple logic and low virtual channel requirements.*

1 Introduction

Point-to-point k-ary n-cube and related networks are being used in many experimental and commercial parallel computers [1, 15, 14, 16]. A k-ary n-cube network has an n-dimensional grid structure with k nodes (processors) in each dimension such that every node is connected to two other nodes in each dimension by direct communication links.

The *wormhole* (WH) switching technique by Dally and Seitz [9] has been widely used in the recent multicomputers [15, 14, 16]. In the WH technique, a packet is divided into a sequence of fixed-size units of data, called *flits*. If a communication channel transmits the first flit of a message, it must transmit all the remaining flits of the same message before transmitting flits of another message. To avoid deadlocks among messages, multiple virtual channels are simulated on each physical channel and an order is enforced on the allocation of virtual channels to messages.

For fault-free networks, some of the most important issues in the design of a routing algorithm are high throughput, low-latency message delivery, avoidance of deadlocks, livelocks, and starvation, and ability to work well under various traffic patterns [12]. For networks with faults, a routing algorithm should exhibit a few additional features: graceful performance-degradation, and ability to handle faults with only a small increase in routing complexity and with local knowledge of faults, as per which each non-faulty processor knows only the status of its neighbors.

The well-known e-cube algorithm routes messages in a strictly ascending order of dimensions; that is, a message takes hops in dimension 0 (if any), then in dimension 1 (if any), and so on to reach its destination. Thus, the e-cube algorithm uses a fixed path to route messages between a pair of nodes even when multiple shortest paths are available, and is termed nonadaptive. Routing algorithms that permit the use of all the paths between a source-destination pair by messages are known as fully-adaptive routing algorithms.

Problem definition. In this paper, we address the issue of incorporating fault-tolerance into a class of adaptive routing algorithms proposed by Duato [11]. These adaptive algorithms have several advantages: low virtual channel requirements, high-performance, and ease of implementation. For example, based on Duato's theory, an adaptive algorithm with two virtual channels per physical channel can be designed for n-dimensional meshes. A variant of this algorithm is being used in the *reliable router* project at MIT [8]. Our routing techniques require only local knowledge of faults and work correctly when faulty components are confined to one or more rectangular blocks [2, 4]. Glass and Ni [13] present the *negative-first* algorithm, which tolerates up to $(n-1)$ faults in an n-dimensional mesh. Chien and Kim show that the planar adaptive routing algorithms can tolerate block faults in the mesh, if no faults are present on the boundaries of the mesh [5]. Dally and Aoki use dimension reversal schemes to provide adaptivity and fault-tolerance [7].

*Chalasani's research has been supported in part by a grant from the Graduate School of UW-Madison and the NSF grants CCR-9308966 and ECS-9216308. Boppana's research has been partially supported by NSF Grant CCR-9208784.

However, the number of virtual channels required by their schemes increases linearly with the number of faulty blocks to be tolerated. A more recent work [8] considers the issue of routing two classes of messages. Fully-adaptive routing is provided for each class of messages using two virtual channels; fault-tolerance is provided using an additional (fifth) virtual channel. However, their method tolerates at most one link or node fault. The work by Duato [10] uses four virtual channels per physical channel to tolerate up to $n-1$ faults in an n-dimensional mesh.

In this paper, we present new techniques to tolerate a large number of faulty blocks using just three virtual channels and only local fault information. Our results show that the two-channel adaptive algorithm by Duato can be enhanced to tolerate any number and combination of faulty blocks using at most one additional virtual channel. This result compares favorably with the results in [8, 10], which use three or four channels to handle, for example, one fault in 2D meshes.

The rest of this paper is organized as follows. Section 2 describes the fault-model used in this paper. Section 3 discusses the two-channel adaptive routing algorithm proposed by Duato for 2D meshes. Section 4 presents a fault-tolerant version of Duato's algorithm that tolerates multiple faulty blocks using three virtual channels. Section 5 summarizes the work reported in this paper.

2 The Fault Model

In this paper we consider only 2-dimensional (2D) $N = k \times k$ mesh networks. Our results can be extended to tori and higher-dimensions. The two dimensions of mesh are denoted DIM_0 and DIM_1. Each node is uniquely indexed by a 2-tuple, say, (x_1, x_0), $0 \leq x_1, x_0 \leq k-1$. A node in the mesh, $x = (x_1, x_0)$, has up to four neighbors given by the valid 2-tuples from the set $\{(x_1 \pm 1, x_0 \pm 1)\}$. Neighbor nodes are connected by bidirectional links, implemented using two unidirectional physical communication channels. We denote the link between nodes x and y by $< x, y >$ and virtual channels of class i as c_i.

A message that reaches its destination is consumed in finite time. Using extra logic and buffers, multiple virtual or logical channels can be simulated on a physical channel in a time-demand multiplexed manner [6, 3]. We always specify the number of virtual channels on per physical channel basis. The channel dependency graph is formed as follows. The virtual channels of the network form the nodes of the channel dependency graph; there is an edge from virtual channel u to virtual channel v if the routing algorithm allows the use of v after using u for any message.

In the remainder of this section, we describe the fault model considered in this paper and the concept of fault-rings, which are created by faults. To simplify presentation, we discuss these concepts for two-dimensional (2D) meshes. These results can be extended to multidimensional meshes and torus networks with suitable modifications [4]. We label the sides of a 2D mesh as North, South, East and West.

We consider both node and link faults. Link faults can also be used to model partial faults of routers, for example, when the buffer space associated with a channel is faulty in a node that is otherwise perfect. A node fault is modeled by making all links incident on it faulty. A node with no nonfaulty links incident on it is considered faulty. We assume that faults are nonmalicious—a failed component simply ceases to work. Therefore, only non-faulty processors generate messages. Furthermore, messages are destined only to fault-free processors. The fault information is kept only locally—each fault-free node knows the status of its neighbors only.

A *fault set* is a set of faulty nodes and links. A set F of faulty nodes and links indicates a (rectangular) faulty block, or f-region, if there is a rectangle connecting various nodes of the mesh such that (a) the boundary of the rectangle has only fault-free nodes and channels and (b) the interior of the rectangle contains all and only the components given by F. A fault set that includes a component from one of the four boundaries—top and bottom rows, left most and right most columns—of a 2D mesh denotes a rectangular faulty block, if the above definition is satisfied when the mesh is extended with nonfaulty *virtual* rows and columns on all four sides. Figure 1 indicates three rectangular faulty blocks: $F_1 = \{(3,3),(3,4),(4,3),(4,4)\}$, $F_2 = \{< (1,1),(2,1) >, < (1,2),(2,2) >\}$, and $F_3 = \{< (0,4),(0,5) >\}$.

We use the *block-fault* model, in which each fault belongs to exactly one faulty block. Under the block-fault model, the complete set of faults in a 2D mesh is the union of multiple faulty blocks. For example, the complete fault set for the network in Figure 1 is $F_1 \cup F_2 \cup F_3$.

For each f-region in a network with faults, it is feasible to connect the fault-free components around the region to form a ring or chain. This is the fault ring, f-ring, for that region and consists of the fault-free nodes and channels that are adjacent (row-wise, column-wise, or diagonally) to one or more components of the fault region. The f-ring of an f-region is of rectangular shape. For example, the f-ring associated with the f-region F_1 in Figure 1 has nodes $(2,2),(2,3),(2,4),(2,5),(3,5),(4,5),(5,5),(5,4),(5,3),(5,2),(4,2),(3,2)$ on its boundary. Notice that a fault-free node is in the f-ring only if it is at most two hops away from a faulty component. There can be several fault rings, one for each f-region, in a network with multiple faults. In a (k,n)-mesh, a link may be common to up to n f-rings and a node common to up to $2n$ f-rings. A set of fault rings are said to overlap if they share one or more links. For example, the f-rings of F_1 and F_2

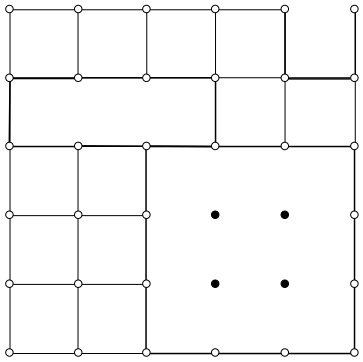

Figure 1: Examples of f-rings and f-chains in a mesh. Faulty nodes are shown as filled circles, and faulty links are not shown. There are three f-regions, and the corresponding f-rings and f-chain are indicated by thick lines.

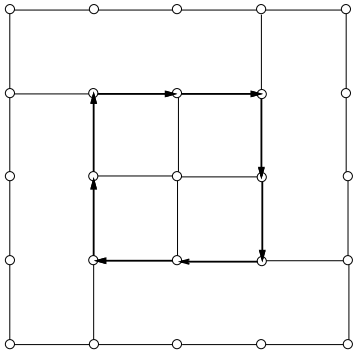

Figure 2: Deadlock with a weakly adaptive algorithm. Directed lines indicate the physical channels used by the six messages in the deadlock. Messages using adaptive (e-cube) channels are indicated to the left (right) of directed edges.

in Figure 1 overlap with each other, since they share link $< (2,2),(2,3) >$. Fault rings are used to route messages around faults.

Forming a fault-ring around an f-region is not possible when the f-region touches one or more boundaries of the network (e.g., F_3 in Figure 1). In this case, a *fault chain*, f-chain, is formed around the f-region. There are four basic types of fault chains that are formed when an f-region touches exactly one of the (2D) network boundaries. The fault chain of an f-region that touches more than one edge of the network can be synthesized from these four basic f-chains. The nodes at which an f-chain touches the network boundaries are the *end nodes* of the f-chain. Since the links in an f-chain are undirected, we can form a directed ring, which spans from one end of the f-chain to the other end and then back. An algorithm to form f-rings and f-chains using local knowledge of faults in a distributed manner is presented in [2].

3 A class of adaptive Routing Algorithms

Recently Duato [11] showed a way to design fully-adaptive algorithms with low virtual channel requirements. The algorithm uses a known deadlock- and livelock-free algorithm and one or more extra virtual channels to increase the adaptivity. Depending on the base routing algorithms used, a class of fully-adaptive algorithms can be developed. A particularly simple and effective fully-adaptive algorithm (denoted $\mathcal{A}$) thus derived uses the well-known e-cube as the base routing algorithm and two virtual channels: c_n, the nonadaptive channel, and c_a, the adaptive channel. The first component of algorithm $\mathcal{A}$ is a deterministic base algorithm that delivers messages between each source-destination pair using c_n. For simplicity, we assume that this base algorithm is the e-cube algorithm which routes messages in the order of increasing dimensions (for example, routing in rows followed by columns in 2D meshes); the e-cube algorithm uses only one virtual channel per physical channel and correctly delivers messages in 2D meshes without any deadlocks. The second component of $\mathcal{A}$ specifies the interaction between the base algorithm and the adaptive channel c_a. This interaction is explained in Figure 3. The following definition is used to distinguish between adaptive and nonadaptive hops by messages.

Definition 1 (E-cube hop) *At any given time, the path specified by* e*-cube from the current host to the destination of the message is called its* e*-cube path; the first hop in that path is its* e*-cube hop from the current host node.*

Algorithm $\mathcal{A}$ first tries to route a message M using c_a along any of the dimensions that take M closer to the destination (Step 2 of Adaptive-Algorithm). If this fails, $\mathcal{A}$ tries to route M using c_n along the least dimension in which the current host and the destination differ (Step 3). If this step also fails, the same sequence of events is tried after a delay of one cycle. Here indefinite starvation of M is avoided by assigning the virtual channels that become free to messages in a fair manner (using, for example, FIFO order). Alternatively, message M can wait for the nonadaptive channel along its e-cube hop, if steps 2–3 in Adaptive-Algorithm fail.

The power of algorithm $\mathcal{A}$ arises from the fact that, at any intermediate host, M can be routed along any of the *profitable* paths—paths which take M closer to its destination. This is accomplished using one

Procedure Adaptive-Algorithm(M, x, d)
/* Comment: Current host is x. Destination is d, and $d \neq x$. This procedure specifies how to route the message M by one step from x to its neighbor such that M moves nearer to its destination. */

1 Determine all the neighbors of x that are along a shortest path from x to d. Let S be the set of such neighbors.

2 If virtual channel c_a is available from x to a neighbor $y \in S$, route M from x to y using c_a; return.

3 If virtual channel c_n is available from x to a neighbor z along the e-cube hop of M, route M from x to z using c_n; return.

4 /* No virtual channel is available along a shortest path. */
Return and try this procedure one cycle later.

Figure 3: Pseudocode for an adaptive algorithm, $\mathcal{A}$.

Procedure Set-Message-Type(M)
/* Comment: When a message is generated, it is labeled as EW if $x_0 \geq y_0$ and as WE otherwise. */

If M is an EW or WE message and $x_0 = y_0$, change its type to NS if $x_1 < y_1$ or SN if $x_1 > y_1$.

Procedure Set-Message-Status(M)
/* Comment: x is the current host of M. Let y be neighbor of x reached when the e-cube hop of the message is used. */

1 If M is a row message and x is on an f-ring, set the status of M to **misrouted** and return.

2 If M is a column message and y is faulty or $d_0 \neq x_0$, set the status of M to **misrouted** and return.

3 Set the status of M to **normal** and return.

Figure 4: Procedures to set the status and type of a message. The current host of M is $x = (x_1, x_0)$ and destination is $d = (d_1, d_0)$.

adaptive channel. In other words, a message can reserve c_a at any step without any restrictions, even if it was forced to use c_n in the previous step(s).

Algorithm $\mathcal{A}$ has been shown be to be deadlock-free by Duato, using a general theory [11]. Using simulations, it was also shown to perform well for uniform traffic.

Deadlocks in the presence of faults. We now show that algorithm $\mathcal{A}$ cannot tolerate faults in the network. Figure 2 shows a 5×5 mesh with multiple faults. Link faults in this figure give rise to four f-rings. There are six messages in the deadlock cycle, with Mi originating at Si and destined for Di, $1 \leq i \leq 6$. Further, message Mi is waiting for $M(i+1)$, $1 \leq i \leq 5$, and $M6$ is waiting for $M1$. It is noteworthy that each and every message in the deadlock has at least one minimal path to its destination, which could be used by the adaptive algorithm in the absence of other messages.

In this example, $M1$ reserved nonadaptive channels in the first two hops, and the adaptive channel in the third hop to reach node $S2$. At $S2$, it cannot reserve the nonadaptive channel from $S2$ to x, since this channel is faulty. Hence, $M1$ tries indefinitely to reserve the adaptive channel from $S2$ to $S3$, which is currently held by $M2$. Messages $M2$ to $M6$ are also in a similar scenario. Notice that this deadlock arises because some of the nonadaptive channels are faulty. If no nonadaptive channel is faulty, this deadlock cannot arise. In the next section, we indicate how to modify $\mathcal{A}$ to tolerate non-overlapping f-rings and f-chains.

4 Fault-Tolerant Adaptive Routing

We incorporate fault-tolerance into the adaptive algorithm $\mathcal{A}$, we enhance the routing logic and use three virtual channels: c_0, c_1 and c_2. The resulting algorithm, denoted $\mathcal{A}_f$, can tolerate multiple f-rings and f-chains under the block fault model.

To route messages around the f-rings, messages are classified into one of the following types: EW (East-to-West), WE (West-to-East), NS (North-to-South), or SN (South-to-North). A message is labeled as either an EW or WE message when it is generated, depending on its direction of travel along the row. Once a message completes its row hops, it becomes a NS or a SN message depending on its direction of travel along the column. Thus, EW and WE messages can become NS or SN messages; however, once a message becomes a NS or SN message, it cannot change its type. These rules are summarized in Procedure Set-Message-Type() of Figure 4. EW and WE messages are collectively known as *row* messages and NS and SN as *column* messages.

Procedure Fault-Tolerant-Adaptive-Algorithm(M, x, d)
/* Comment: Current host is x. Destination is d, and $d \neq x$. This procedure specifies how to route the message M by one step from x to its neighbor such that M moves nearer to its destination. */

1 Set-Type(M).

2 Set-Status(M).

3 If M is **normal**,
Route M using
Modified-Adaptive-Algorithm().

4 If M is **misrouted**,
Route M using Route-Around-Frings() in Figure 6.

Figure 5: Fault-tolerant adaptive routing algorithm.

Procedure Route-Around-Frings(M, x, d)
/* Comment: The current host of M is (x_1, x_0) and destination is (d_1, d_0). Virtual channel usage for all messages is as shown in Table 1. */

EW messages Route around the f-ring in the clockwise orientation if $d_1 > x_1$, and in the counter-clockwise orientation if $d_1 < x_1$. Either orientation can be chosen if $d_1 = x_1$.

WE messages Route around the f-ring in the clockwise orientation if $d_1 < x_1$, and in the counter-clockwise orientation if $d_1 > x_1$. Either orientation can be chosen if $d_1 = x_1$.

NS and SN messages Route around the f-ring in either orientation.

Figure 6: Rules for routing misrouted messages around an f-ring.

Table 1: Usage of Virtual Channels by Misrouted Messages

Message type	Channel	Used for
EW	c_0	all hops
NS	c_1	all hops
SN	c_2	all hops
WE	c_0	row hops
WE	c_1	column hops in the south-to-north direction
WE	c_2	column hops in the north-to-south direction

The type of a message M is used to determine its status. A row message (EW or WE) is termed **misrouted** if it is on an f-ring or f-chain. In addition, a column message (NS or SN) whose head flit is either blocked by a fault or not in the same column as its destination is also labeled as **misrouted**. All other messages are termed **normal**. These rules are described in Procedure Set-Message-Status().

Normal messages have adaptivity, and misrouted messages do not. Procedure Fault-Tolerant-Adaptive-Algorithm() provides a high-level description of how messages are routed. Step 1 of this procedure determines the type of the message M (EW, WE, NS, or SN). Step 2 determines the status — normal or misrouted — of M. If M is normal, step 3 routes M using a modified version of $\mathcal{A}$. The Modified-Adaptive-Algorithm differs from the Adaptive-Algorithm (Figure 3) as follows: it uses c_1 or c_2 in place of c_a and c_0 in place of c_n. If the status of the message is misrouted, then it is routed using the algorithm Route-Around-Faults() (Figure 8) discussed below.

4.1 Routing on f-chains

Since we assume only local knowledge of faults, a message may be routed in the direction of the f-chain that actually leads to a dead end. As an example, consider the routing of a WE message, M, from s to d on the f-chain in Figure 7. When M touches the f-chain at node a, it is routed in clockwise orientation, since d is in a row above a. As a result, M reaches node b at one end of the f-chain; it needs to take a *u-turn* and travel on the f-chain in the opposite direction to reach the other side of the f-region. Finally when it reaches node c on the f-chain, the SE corner node, the message leaves the f-chain and completes its journey using the e-cube algorithm. A similar scenario may be constructed for EW messages. It is noteworthy that column messages are not blocked by this type of f-region; that is, column messages never need to take a u-turn on such f-chains. When an f-region touches more than one boundary of the network, even fewer types of misrouted messages encounter the f-chain. For example, the only misrouted messages that travel on an f-chain of an f-region touching the North and East boundaries of the network are WE misrouted messages.

For routing on f-chains, the virtual channels used are just as shown in Table 1. For example, the WE message in Figure 7 will use c_1 for its journey from a to b, c_2 from b to c, and c_0 from c to e. Other types

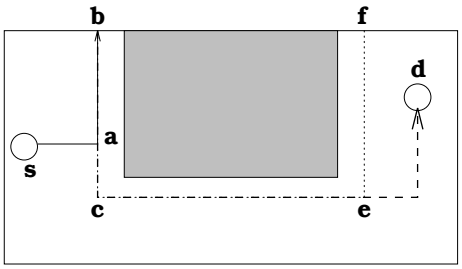

Figure 7: Example of routing on an f-chain. The shaded area indicates an f-region and dotted lines the corresponding f-chain. The path of the message after u-turn is indicated by a dashed line. The u-turn path overlaps with the path taken on the f-chain from a to b before the u-turn.

Procedure Route-Around-Faults-Algorithm(M)
/* Comment: The current host of M is (a_1, a_0) and destination is (b_1, b_0). */

1 Select the orientation of M using the rules mentioned in Figure 6. For column messages on overlapping f-rings (f-chains), the orientation of M is chosen only for the first f-ring (f-chain) and it alternates for each successive f-ring (f-chain).

2 Route M along the fault-ring in the specified direction. The virtual channels usage by M is as shown in Table 1.

3 In Step 2, if M reaches the end-node of an f-chain, its orientation is reversed and Step 2 is used again.

Figure 8: Pseudocode for routing around faults.

.

of messages, EW, NS and SN, use the same virtual channels before and after the u-turn.

To summarize the discussion in this section, we present Procedure Route-Around-Faults-Algorithm() which is used to route misrouted messages until they become normal.

4.2 Routing in the presence of non-overlapping f-rings and f-chains

To simplify the discussion, we first consider the case in which only nonoverlapping f-rings exist in the network.

Each misrouted message is routed around the f-ring using a specific orientation, *until it becomes normal again*. Let us first consider a WE misrouted message M with destination d. Assume that it becomes misrouted when it arrives at a node on an f-ring. M is routed around the f-ring in the clockwise (respectively, counter-clockwise) orientation if d is in a row below (respectively, above) x; if both x and d are in the same row, any orientation can be chosen. These rules are summarized in Figure 6. For misrouted EW messages, the direction is chosen in a manner similar to EW messages. Either clockwise or counter-clockwise orientation can be chosen arbitrarily for misrouted NS and SN messages.

The virtual channels used by misrouted messages is shown in Table 1. Note that normal messages always use c_0 as the non-adaptive channel, and c_1 and c_2 as adaptive channels. Misrouted messages have no adaptivity and can use only one virtual channel class for each hop. Misrouted EW messages use c_0 for all hops. Similarly, misrouted NS and SN messages use c_1 and c_2 respectively, for all hops. Misrouted WE messages use c_0 for all row hops and c_1 (respectively, c_2) for column hops in the south-to-north (respectively, north-to-south) direction. As we shall see later, this virtual channel assignment is crucial to prove deadlock-freedom.

4.3 Extension to routing on overlapping f-rings

Now we consider routing in overlapping f-rings. For row (EW and WE) messages, routing on overlapping f-rings is the same as routing on nonoverlapping f-rings. That is, each time a messages encounters a new f-ring, it chooses its direction of travel using the rules in Figure 6. However, to route a column message on a series of overlapping f-rings, its orientation is chosen only once (using the rules in Figure 6); further, the orientation of a message on a pair of successive overlapping f-rings is not the same. For example, if there are three overlapping f-rings, and if a message chooses the clockwise orientation for the first f-ring, then it will travel in the counter-clockwise orientation on the second and in the clockwise orientation on the third.

We next prove that the fault-tolerant adaptive algorithm described in this section is correct and deadlock- and livelock-free.

4.4 Proof of deadlock and livelock freedom

Lemma 1 *The fault-tolerant adaptive algorithm described in this section provides correct and livelock- and deadlock-free routing of messages in 2D meshes with any number of f-regions, regardless of whether they overlap or not.*

Proof. First, normal messages use c_0 as the deterministic channel, and can never wait for channels c_1 or c_2. In other words, normal messages can only wait for c_0 channels. Misrouted messages can wait for c_0, c_1 or c_2 on f-rings or f-chains only.

The original adaptive algorithm $\mathcal{A}$ avoids deadlocks by ensuring that an exclusive set of channels are available for each class of messages at any time. The adaptive channels are simply shared by multiple classes of messages.

Dependencies of EW messages. Normal EW messages can wait only for c_0 channels in the east-to-west direction. Misrouted EW messages can wait for c_0 channels along the columns as well. However, no WE message can reserve a c_0 channel in a column or in the east-to-west direction. Hence, EW messages cannot wait for WE messages.

Dependencies of WE messages. Normal WE messages can wait only for c_0 channels in the west-to-east direction. Misrouted WE messages take row hops using channel c_0 only in the west-to-east direction. Misrouted WE messages use channels c_1 or c_2 for column hops on f-rings. However, no EW message uses these channels, since, around f-rings, EW messages use c_0 for column hops also. Hence, WE messages cannot wait for EW messages.

Dependencies of NS and SN messages. A message, once it becomes a NS message, stays as a NS message until it is delivered to its destination. A normal NS message can only wait for a c_0 channel in the north-to-south direction, which can be reserved by other NS messages only. A misrouted NS message can only wait for c_1 channels in any of the following directions on f-rings and f-chains: north-to-south, east-to-west, and west-to-east. On f-rings and f-chains, only misrouted WE messages use c_1 channels in the south-to-north direction. Hence, NS messages can only wait for other NS messages. A similar argument holds for SN messages.

Deadlock possibilities. Deadlocks among two different types of messages cannot occur, since NS and SN messages do not depend on any other message type. Hence, to prove deadlock-freedom, it is sufficient to show that there are no deadlocks among messages of a specific type.

Deadlocks among NS messages. First, no NS message travels in the south-to-north direction of the mesh (even the NS messages that take a u-turn can do so only along a row). Thus, there cannot be a deadlock between NS messages waiting in distinct rows. Thus, if there are cyclic dependencies among NS messages, they can only be for channels in the same row. However, for deadlocks to occur along a row, the following condition must be satisfied.

Condition 1 There must be two NS messages such that the first (respectively, second) reserved a west-to-east (respectively, east-to-west) channel and is waiting for an east-to-west (respectively, west-to-east) channel.

NS messages traveling on overlapping f-rings do not take any u-turns, since the orientation of the message alternates on successive f-rings; hence, messages traveling on overlapping f-rings cannot satisfy condition 1. A NS message can take hops in both east-to-west and west-to-east directions of a row, only if it was on an f-chain and took a u-turn at an end node of the f-chain in that row. However, condition 1 requires two such messages in the same row, which is possible only if the north boundary of the f-chain corresponds to a complete row. This, in turn, implies that a complete row of the mesh is faulty, which contradicts our assumption that the mesh is connected under faults.

Livelock freedom and correct delivery. To see that messages are correctly delivered without introducing livelocks in the faulty network, observe that (a) a message is misrouted only around an f-ring, (b) a message, once it leaves an f-ring will never revisit it, (c) there are a finite number of f-rings in the mesh, (d) a normal message progresses towards its destination with each hop, and (e) the destination node is accessible, since all non-faulty nodes are connected. Since a message is misrouted only by a finite number of hops on each f-ring and it never visits an f-ring twice, the extent of misrouting is limited. This together with the fact that each normal hop takes a message closer to the destination proves that messages are correctly delivered and livelocks do not occur. ∎

5 Concluding remarks

We have presented techniques to enhance fully-adaptive wormhole routing algorithms for fault-tolerant routing on meshes. As an example a fully-adaptive routing algorithm based on Duato's theory [11] is considered. The original version uses two virtual channels per physical channel to provide full adaptivity. With the proposed techniques multiple faulty blocks are tolerated with one additional virtual channel. Deadlock- and livelock-free routing properties are preserved with the proposed techniques. Furthermore, there is no performance degradation in the absence of faults. All virtual channels are used to provide high performance when there are no faults.

Our results are specific to 2D meshes and can be extended to 2D tori. We believe that our results can be extended to nD, $n > 2$, networks with no increase in the number of virtual channels. Further work is needed.

The concept of fault rings and fault chains can be extended to faults of arbitrary shape. In such cases, the fault rings are not rectangular. Interesting subcases include solid-region faults— fault regions such that all links and nodes in the interior of the polygon connecting the outermost points of a connected fault region are faulty. We are currently working on this problem and the results will be reported elsewhere.

References

[1] A. Agarwal et al., "The MIT Alewife machine: A large-scale distributed multiprocessor," in *Proc. of Workshop on Scalable Shared Memory Multiprocessors*, Kluwer Academic Publishers, 1991.

[2] R. V. Boppana and S. Chalasani, "Fault-tolerant routing with non-adaptive wormhole algorithms in mesh networks," in *Proc. Supercomputing '94*, Nov. 1994.

[3] S. Borkar et al., "iWarp: An integrated solution to high-speed parallel computing," in *Proc. Supercomputing '88*, pp. 330–339, 1988.

[4] S. Chalasani and R. V. Boppana, "Fault-tolerant wormhole routing in tori," in *Proc. 8th ACM Int. Conf. on Supercomputing*, July 1994.

[5] A. A. Chien and J. H. Kim, "Planar-adaptive routing: Low-cost adaptive networks for multiprocessors," in *Proc. 19th Ann. Int. Symp. on Comput. Arch.*, pp. 268–277, 1992.

[6] W. J. Dally, "Virtual-channel flow control," *IEEE Trans. on Parallel and Distributed Systems*, vol. 3, pp. 194–205, Mar. 1992.

[7] W. J. Dally and H. Aoki, "Deadlock-free adaptive routing in multicomputer networks using virtual channels," *IEEE Trans. on Parallel and Distributed Systems*, vol. 4, pp. 466–475, April 1993.

[8] W. J. Dally, L. R. Dennison, D. Harris, K. Kan, and T. Xanthopoulos, "The reliable router: A reliable and high-performance communication substrate for parallel computers," in *Proc. of Parallel Routing and Communication Workshop*, May 1994.

[9] W. J. Dally and C. L. Seitz, "Deadlock-free message routing in multiprocessor interconnection networks," *IEEE Trans. on Computers*, vol. C-36, no. 5, pp. 547–553, 1987.

[10] J. Duato, "A theory to increase the effective redundancy in wormhole networks," *Parallel Processing Letters*. To appear.

[11] J. Duato, "A new theory of deadlock-free adaptive routing in wormhole networks," *IEEE Trans. on Parallel and Distributed Systems*, vol. 4, pp. 1320–1331, Dec. 1993.

[12] S. A. Felperin, L. Gravano, G. D. Pifarré, and J. L. Sanz, "Routing techniques for massively parallel communication," *Proceedings of the IEEE*, vol. 79, no. 4, pp. 488–503, 1991.

[13] C. J. Glass and L. M. Ni, "Fault-tolerant wormhole routing in meshes," in *Twenty-Third Annual Int. Symp. on Fault-Tolerant Computing*, pp. 240–249, 1993.

[14] S. L. Lillevik, "The Touchstone 30 Gigaflop DELTA prototype," in *Sixth Distributed Memory Computing Conference*, pp. 671–677, 1991.

[15] M. D. Noakes et al., "The J-machine multicomputer: An architectural evaluation," in *Proc. 20th Ann. Int. Symp. on Comput. Arch.*, pp. 224–235, May 1993.

[16] W. Oed, "The cray research massively parallel processor system, CRAY T3D," tech. rep., Cray Research Inc., Nov. 1993.