Broadcasting and routing in faulty mesh networks

Milos Stojmenović, Amiya Nayak
SITE, University of Ottawa, Ottawa, Ontario, Canada K1N 6N5
{mstoj075, anayak }@site.uottawa.ca

Abstract— **Broadcasting is a data communication task in which one processor sends the same message to all other processors. Routing is a task where a source processor sends a message to a destination processor. A faulty node is in an error state and cannot participate in the activities or the communication in a given network. In this paper, we consider the family of mesh networks, which include the mesh connected computer (MCC), *k*-dimensional mesh, torus, and *k*-ary *n*-cube. Our goal is to design routing and broadcasting algorithms which will use local knowledge of faults, no additional resources, will work for an arbitrary number and structure of faults, will guarantee delivery to all nodes connected to the source, and will remain optimal in a fault free mesh. We did not find any solution in literature to satisfy these desirable properties. Our routing and broadcasting schemes for MCCs and tori, and our broadcasting algorithm for the all-port model on any faulty mesh network satisfy all of these properties. For routing and broadcasting in a one-port model in higher dimensions, a condition on fault structure needs to be met. We propose a new broadcasting algorithm which guarantees delivery to all processors connected to the source in the all-port model of faulty meshes. We then describe a routing algorithm that guarantees delivery in faulty MCCs and tori, the connectivity of the source and destination being the only obvious requirement. The algorithm can be extended to faulty *k*-D meshes and *k*-ary *n*-cubes, where the delivery will be guaranteed if healthy nodes in every 2-D submesh (sub-tori) remain connected. We then describe broadcasting algorithms for the one-port model, which again guarantee delivery to all connected processors in two-dimensional cases, and guarantee delivery in *k*-dimensional cases if healthy processors in every 2-D submesh (sub-tori) remain connected.**

Keywords: Mesh networks, fault tolerance, routing, broadcasting

I. INTRODUCTION

A mesh connected computer (MCC) is a set of processors (nodes) arranged in a square grid. Each node in the grid has up to 4 neighbors, which are nodes to the left, right, bottom and top (if they exist, and are not faulty). A MCC of size k has k rows and k columns, and therefore has k^2 processors. A torus is a mesh with additional edges such that all nodes have exactly 4 neighbors. This is achieved by connecting all nodes on the border of the mesh (which have less than 4 neighbors) to their corresponding node on the opposite border. A torus of size k has k rows and k columns. An n-D mesh network (a generalization of MCC) of size k has k nodes along each dimension, and therefore has k^n processors overall. Each node has an address of the form $(a_1, a_2, \dots, a_n)$, where $0 \le a_i \le k-1$. Each such node has $2n$ neighbors, that is, two neighbors along each dimension (if such a neighbor exists). The neighbors along the i^{th} coordinate share the $n-1$ coordinates, while the i^{th} coordinates are $a_i + 1$ and $a_i - 1$, respectively. That is, the neighbors of node $(a_1, a_2, \dots, a_n)$ are $(a_1, \dots, a_{i-1}, a_i+1, a_{i+1}, \dots, a_n)$) and $(a_1,\dots, a_{i-1}, a_i - 1, a_{i+1}, \dots, a_n)$).

A k-ary n-cube is a generalization of a torus network and an extension of an n-D mesh by adding some edges. Each node has an address of the form $(a_1, a_2, \dots, a_n)$, where $0 \le a_i \le k-1$. Each such node has $2n$ neighbors, that is, two neighbors along each dimension. The neighbors along the i^{th} dimension share the $n-1$ coordinates, while the i^{th} coordinates are $a_i + 1 \pmod k$ and $a_i - 1 \pmod k$, respectively. That is, the neighbors of node $(a_1, a_2, \dots, a_n)$ are $(a_1, \dots, a_{i-1}, a_i+1 \bmod k, a_{i+1}, \dots, a_n)$) and $(a_1,\dots, a_{i-1}, a_i - 1 \bmod k, a_{i+1}, \dots, a_n)$). Therefore this is the n-D mesh model with added wrap around edges. Note that a 2-ary n-cube is an n-dimensional hypercube. We are interested in mesh type networks, where $k>n$ (that is, with low dimensions but large size along each dimension).

In accordance with literature, we assume that each processor can correctly receive messages from several neighboring processors simultaneously. For broadcasting, we consider both the one-port and all-port models. For routing, we only consider the one-port model since the all-port model is irrelevant on a path anyway. In a one-port model, a message sent from one processor is received by only one of its neighbors, while in an all-port model, it is simultaneously received by all of its neighbors. In the rest of this text, processors will also be known as nodes, and the links between them as edges in the considered network.

Routing is a task where a source node $A(u, v)$ sends a message to a destination node $B(w, z)$. Routing on a MCC can be done by sending the message horizontally to node $C(w,v)$. If $u<w$, the message is sent right. If $u>v$, the message is sent left. The second step of the simple algorithm involves sending the message vertically along the mesh from $C(w, v)$ to $B(w, z)$. This well known routing algorithm can be generalized for n-D meshes and k-ary n-cubes in a straightforward manner. At each step, the message can advance along any dimension where the coordinates of the current node and destination node differ. Broadcasting is a task of transmitting a message from one node to all other nodes in the mesh network. Assume node $X(u,v)$ wants to broadcast a message. In a well known solution, the message is first broadcasted horizontally across the mesh. This means that all nodes with coordinates (w,v) receive it. The second phase involves vertical transmission by all of the nodes which currently have the message, in parallel. By forwarding the message in both the up and down directions from node (w,v), all nodes (w,z) in that column receive the message.

In order to solve the problem of broadcasting and routing in faulty mesh networks, we studied various properties of existing solutions, and made a list of several desirable ones. We would like to avoid placing restrictions on the number of faults or the location of faults in the mesh. Certain algorithms only work in cases where the number and/or positions of the faulty nodes conform to certain specific standards. We will search for a solution that does not rely on fault position or the occurrence of too few faults. Such a solution is obviously more flexible and applicable to any mesh. We managed to find such solutions for 2-D cases, while for k-D cases, we had to add a reasonable condition which covers a wide range of scenarios.

1-4244-0054-6/06/$20.00

We also assume that there are no additional resources being used such as virtual channels. The next goal is to have protocols that remain optimal if there are no faults in the network. The final goal in our research is to design routing and broadcasting protocols that guarantee delivery of the message to all non-faulty nodes connected to the source. We will search for a solution in which processors only have local knowledge of faults. Local knowledge of faults means that each node only knows the fault status of its neighbors. Algorithms that require global knowledge of faulty nodes (that is, full network information at each node) require pre-processing steps to spread such information throughout the network whenever a fault occurs, and whenever a fault is repaired. Some protocols even require grouping healthy nodes into rectangular boxes, or enclosing faulty nodes into rectangular or convex boxes. The latter boxes could include some healthy nodes as well, which are disabled, and cannot participate in communication tasks. If global knowledge is available, the source node can apply a shortest paths scheme to find the route to each destination, and this computation can be repeated at each intermediate node, or the information could be included in the message itself. In mesh networks with many nodes, faults can occur dynamically, frequently and nodes may even be recovered or replaced, and messages sent to the whole network informing it of its status may cause considerable overhead. Moreover, these messages may not reach all connected and healthy nodes if additional faults occur dynamically in the network while the information about the previous fault is still circulating. That is, this broadcasting task itself, and the assumption of having global knowledge may be impossible to provide when failures are dynamic, frequent and occur almost simultaneously at several nodes. This means that our goal of guaranteed delivery to all connected nodes can only be achieved using local knowledge.

We are not aware of any existing method for either of the two problems, while at the same time satisfying all of the listed criteria. The papers that proved most relevant were [WCW] for broadcasting and [ZK] for both routing and broadcasting in faulty meshes. Out of these two, only [WCW] describes an algorithm in which nodes have local knowledge of faults. However, this algorithm does not always guarantee delivery. [ZK] guarantees delivery, but involves global knowledge of faults (as discussed above, the occurrence of dynamic faults while routing or broadcasting is in progress may endanger the guaranteed delivery property). Some of the remaining papers require additional resources such as virtual channels: [TKL,PSY,LH,SS,SW]. Other papers that require global knowledge about faults include [AB,AC,CA,W1,W2]. There is another set of papers which impose restrictions on the location and number of faults in the mesh: [BC,GN,CC,W3].

This paper is organized in the following way. Section 2 gives a literature review on routing and broadcasting in faulty meshes. In Section 3, we propose a new broadcasting algorithm which has guaranteed delivery (to all processors connected to the source) in the all-port model of faulty meshes. In Section 4, we describe a routing algorithm that guarantees delivery in faulty MCCs and tori, the connectivity of the source and destination being the only obvious requirement. The algorithm is extended in Section 5 to faulty *k*-D meshes and *k*-ary *n*-cubes, where the delivery is guaranteed if healthy nodes in every 2-D submesh (sub-tori) remain connected. In Sections 6, 7, we describe broadcasting algorithms for the one-port model, which again guarantee delivery to all connected nodes in 2-dimensional cases, and also in *k*-dimensional cases if healthy nodes in every 2-D submesh (sub-tori) remain connected. The conclusion section completes this article.

II. Literature Review

A. Broadcasting in faulty meshes

1) Local knowledge and no additional resources

Wu, Chen and Wu [WCW] describe a broadcasting algorithm for MCC that does not require global knowledge in order to broadcast messages. They assume an all-port model of communication. They claim that the problem is very difficult and only give a solution for a special case of an m-subMCC. The algorithm takes a kxk mesh as input. The mesh is partitioned into mxm submeshes where m<k. Each of these submeshes acts as a node in the sense that it is assumed to be non faulty and can receive and transmit messages to its neighboring 'nodes'. These pseudo nodes are deemed more likely to transmit messages to their neighboring nodes since they have more connecting lines for neighbor-to-neighbor communication. When less than m/2 nodes on each border of each submesh are faulty, communication between two pseudo-nodes would be successful. The following two steps alternate. The first step involves broadcasting the message locally through the pseudo node. This is done using a breadth first search administered locally within the pseudo node. Breadth first search uses only local submesh knowledge to find a routing path to each node within the pseudo node. Once every node in the pseudo-node has received the message, it is passed on to neighboring pseudo-nodes in the second step, that is equivalent to broadcasting in a network of pseudo-nodes (submeshes). This process is repeated until the message is successfully broadcasted throughout the mesh. Note that a synchronization step may be needed for communication between submeshes. The authors assume that each of the m-submeshes remains connected internally, after removing the faulty nodes. They claim that their algorithm guarantees delivery 99% of the time if faulty nodes occur less than 12% of the time. The protocol however does not guarantee delivery of messages to all non-faulty nodes if the percentage of node failures is high enough. Faults within a submesh may disconnect it internally, but the nodes may remain connected via other nodes in the mesh. The protocol can be generalized to n-D meshes and k-ary n-cubes with similar properties.

2) Broadcasting with global knowledge or additional resources

Jiang and Wu [JW] described a broadcasting algorithm which has two phases. In the pre-broadcasting phase, faulty nodes are grouped into rectangular blocks, which may include some healthy nodes which are disabled. The rest of the mesh is then divided into rectilinearly convex polygonal regions (that is, convex in both horizontal and vertical directions). In the broadcasting phase, the message is sent from the source to an 'eye' node in each convex region reachable from it (faulty blocks being excluded for communication). These 'eye' nodes then broadcast the message within their own region using a recursive algorithm. In this recursive algorithm, the current region is subdivided into smaller ones, and message is sent to one 'eye' in each sub region. The process continues until each

reachable node receives the message. The algorithm [JW] therefore requires global faulty information, and also does not guarantee delivery to each node reachable from the source. Disabled nodes by the design do not receive the message. Also, rectangular faulty blocks, which are expanded by disabling some nodes, may disconnect the mesh, with one region being unreachable from the other, although in reality they could be linked via disabled, but healthy nodes.

[TKL] describe a multicasting algorithm that can broadcast in arbitrary topology but requires up to two virtual channels. [PSY] describe a fault tolerant broadcasting algorithm for torus network which uses only local knowledge of faulty nodes and can tolerate up to *k-1* faults in a *k*-ary *n*-cube. The algorithm needs two virtual channels at each physical channel if faulty nodes appear, and is too restrictive on the number of faults it can tolerate. When applied on a mesh with *k* rows and *k* columns, it can tolerate up to *k-1* faults. Virtual channels involve adding buffer space and complex control logic.

[AB] describe a fault tolerant one-to-all broadcasting algorithm for *k*-ary *n*-cubes, which requires global knowledge of faults, and can tolerate up to *2n-2* node failures provided that $k>2n-2$ and $k>3$. Therefore it does not apply to $k=2$ which is a 2-dimensional torus, and does not follow the local knowledge assumption. The solution [AC] requires more than local fault information, since certain unsafe nodes are defined recursively and propagated from a faulty node. The algorithm requires a fault free row and fault free column, and therefore tolerates up to *k* faults on a mesh of size *k*.

B. Routing in faulty meshes

1) Routing with guaranteed delivery

The paper by Zakrevski and Karpovsky [ZK] described an interesting algorithm that guarantees delivery of messages in all connected meshes, but requires global knowledge of faults to achieve this. Their algorithm involves pre-routing, and routing phases. The pre-routing phase is the pre-processing step and involves constructing the largest possible non-faulty rectangles out of the mesh. This means that all non-faulty nodes must be located in at least one non-faulty rectangle. The second step of their algorithm involves connecting these rectangles in a graph. Each rectangle is represented by a node in this new graph. Two nodes in the new graph are connected if they partially overlap each other. This means that they share at least one actual node in the mesh. The pre-routing phase is performed every time there is a new faulty node in the mesh, or every time a node becomes non faulty. The routing phase uses the rectangle graph to find the shortest path between rectangles containing the source and destination nodes. Within each rectangle, a simple routing connects common nodes with previous and next rectangles on the route. Broadcasting a message can be done as follows. The message is first routed from the source rectangle to all other rectangles using Dijkstra's shortest path algorithm. Each rectangle then distributes the message amongst its nodes using the standard broadcasting algorithm for non-faulty meshes. This local broadcasting will work because each of the rectangles only contains non-faulty nodes. Their method guarantees delivery at the cost of global knowledge of faults.

2) Routing with restricted fault structure for guaranteed delivery

Another recent algorithm by Boppana and Chalasani [BC] uses the concept of faulty rings to route around faulty nodes. Its main drawback is severe restrictions on fault locations - it is prohibited to have faulty nodes both to the *South* and *North*, or to the *East* and *West*, from a fault-free node. The main idea was then expanded in [CB] by creating faulty rings for routing around H-shaped, T-shaped, U-shaped, and +-shaped regions. Therefore the types of local knowledge faults the algorithm can handle are very restricted. It is restricted to the cases where route segments containing faulty nodes can be locally replaced by route segments with all nodes being healthy.

An example of the deadlock-free routing algorithm is NAFTA by Cunningham and Avresky [CA]. This algorithm is based on the combination of North-Last and South-Last strategies and allows deadlock-free routing without global knowledge. However, it introduces some unsafe nodes. The number of these nodes can grow as $O(k^2)$ on a mesh with *k* rows and *k* columns, for the worst configuration of faulty nodes. The solution [CA] actually requires more than local fault information and a pre-processing step, since certain unsafe nodes are defined recursively and propagated from a faulty node. It also requires fault free row or fault free column.

Glass and Ni [GN] propose a fault-tolerant routing scheme for *n*-D meshes without virtual channels, but the scheme can tolerate up to *n-1* faults. The algorithm by Chen and Chiu [CC] can only tolerate a limited number of faults, proportional to the dimension of the mesh torus. To tolerate more faults, they propose to deactivate some healthy nodes (i.e. regard them as faulty) so that faults are in rectangular shapes. However, this process can disconnect the network and routing may fail.

Algorithms described by Wu [W1, W2] require global knowledge of faults to derive optimal paths. The attempt is made to reduce the amount of fault information needed while maintaining minimal length paths. The author does not discuss whether or not any of the proposed schemes guarantee delivery. In fact, the source does not even start routing if the existence of a minimal path is not guaranteed.

The rectangular faulty model is the most commonly used fault model in designing a fault tolerant and deadlock free routing algorithm in mesh-connected computers. Although some efforts have been made either to enhance the faulty block [S] or to activate some boundary non-faulty nodes in a faulty block [BD, SS]), the major problem is that a faulty block may include many non-faulty nodes. In [W3], Wu minimized the size of faulty blocks by defining special convex polygons from a given set of rectangular faulty blocks. An additional problem in all faulty block based solutions is that routes from non-faulty nodes which are inside faulty blocks to other nodes are not given in any of solutions proposed with faulty block model, although connectivity may exist.

3) Routing that requires virtual channels

The algorithm [VUM] tolerates multiple faults but requires two virtual channels. They assume only local knowledge about faults, but the message is not allowed to backtrack. Thus, there should not be any concave faulty regions in the network where the message may get trapped [VUM]. The solution is described via a pseudo-code and is difficult to understand. The scheme [LH] requires additional virtual channels for each physical channel, and is therefore costly for implementation.

Faulty blocks can be easily established and maintained through message exchanges among neighboring nodes. The convexity of each faulty block facilitates a simple fault-tolerant and deadlock free routing using relatively few virtual channels [SS, SW]). This feature is also a necessary condition for progressive routing, where the routing process never backtracks. The absence of backtracking in turn is a necessary condition for minimal routing, where the destination is reached through a minimal path from the source. Therefore the schemes [SS, SW] require convexity of each faulty block.

Boppana and Chalasani [BC] introduce the solid fault model. It is a model where any cross section of a faulty region has contiguous faulty components. The proposed method handles solid faults in meshes, which includes all convex faults and many practical nonconvex faults, for example, faults in the shape of L or T. As examples of the proposed method, adaptive and non-adaptive fault-tolerant routing algorithms using four virtual channels per physical channel are described.

C. *Routing in planar graphs*

Most of our contributions in this paper arise from the idea of routing in planar graphs. These connections will be further explained in Section 4. A planar graph is a graph where no two edges intersect. Planar graphs consist of faces. Three algorithms for routing in planar graphs will be applied here: Greedy routing, face routing and a combination of the two, called GFG routing [BMSU]. The greedy algorithm can be applied to all meshes considered in this article. The face and GFG algorithms can only be applied to 2D meshes.

In the greedy routing algorithm by Finn (see [BMSU]), each node currently holding the message forwards it to the neighbor that is physically closest to the destination node. Only nodes that are closer to the destination are considered. This is a localized optimization strategy that leads to delivery, or in many cases failure. If delivered, the message normally has a route length close to the shortest path scheme. However, in sparse or faulty graphs, the greedy approach may lead to a local maxima (a node which has no closer node to the destination than itself), and the message may not be delivered since it becomes trapped in situations where all nodes that are closer to the destination node are faulty. The message can then no longer be propagated through the graph by the greedy method.

The face routing algorithm [BMSU] is described as follows. The source draws an imaginary straight line *l* from the source *S* to the destination *D*. This line will be important in determining in which face the message is to be routed. The routing process begins with the source node sending the message to its non-faulty neighbor *B* such that angle *BSl* in minimal. Illustrations can be found in Section 4. Since we are dealing with a planar graph, each message can be sent along one face in the graph. The message is repeatedly forwarded to the next neighbor in the same face as the node currently holding the message. The message will jump to an adjacent face if during the transmission of the message from one node to another, the communications line intersects line *l*, between the previous intersection and the destination. This algorithm guarantees delivery of messages if there exists a path from source to destination. The messages contain source and destination information, and the last intersection of the imaginary intersection line. This is a memoryless procedure.

The GFG algorithm [BMSU], combines the above two approaches. Messages are routed using the greedy algorithm until it becomes stuck in a local maxima. Face routing takes over until at some point the node currently holding the message is physically closer to the destination than the node at which the local maximum occurred. At this point, the greedy algorithm takes over, and that node becomes the new source. The greedy and face modes may alternate a few times until the message is delivered. See illustration in Section 4.

III. BROADCASTING WITH GUARANTEED DELIVERY IN ALL PORT FAULTY MESHES

We will first consider the case of the all-port communication model for faulty meshes. The WCW algorithm [WCW] has some drawbacks. It does not go into details regarding the breadth first search that is used within each pseudo-node. There must also exist a synchronization step when the message is broadcasted vertically along the pseudo nodes. We will demonstrate that it does not always guarantee delivery to all nodes. Figure 1 is an example of failed broadcasting where not all healthy nodes receive the message. The three faulty nodes in the lower left pseudo node make it impossible for the message to travel vertically from that pseudo node. The effect is that no messages reach the upper left pseudo node. In fact, simple blind flooding is faster, and there is no need for a synchronization step. The WCW algorithm is designed to provide amelioration over a simple blind flooding approach by reducing the number of messages that need to be exchanged for broadcasting to be successful. It is also intended to reduce the time needed to accomplish broadcasting.

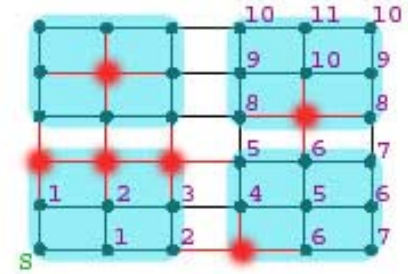

Figure 1. Broadcasting failure in WCW algorithm

We will now describe a new solution to this problem. Our solution does not have the drawbacks mentioned for the WCW algorithm. Namely, it has no need for a synchronization step, and guarantees delivery to all nodes connected to the source. The algorithm is a simple blind flooding algorithm. Blind flooding is actually equivalent to breadth first search. In the all port model, each node that receives the message for the first time transmits it to all of its healthy neighbors in the next step. Subsequent copies of the same message are ignored. The algorithm apparently has no time synchronization problems, and all nodes connected to the source will receive the message. The number of messages passed within the entire mesh is in fact greater in WCW, since it has blind flooding within each submesh, and some messages need to be sent twice by border submesh nodes; because once pseudo node blind flooding is complete, the message is sent to the next pseudo node.

The proposed solution (illustrated in Figure 2) easily generalizes to arbitrary types of faulty mesh networks described in the introduction. Guaranteed delivery and other properties are preserved on all such faulty meshes.

IV. ROUTING WITH GUARANTEED DELIVERY IN MCCS

Since we only consider routing that constructs a single path, only the one port communication model is relevant to routing solutions in this article. We observe that faulty MCCs and tori are in fact planar graphs, and therefore the face and GFG algorithms [BMSU] can be applied. The application guarantees delivery of the message, and works in all cases where there exists a path between source and destination. All of the previously reviewed papers had limitations and restrictions concerning faulty node occurrence frequency, position, etc, but our solution has no such restrictions. Therefore, our main contribution is the application of the GFG algorithm [BMSU] to meshes and tori.

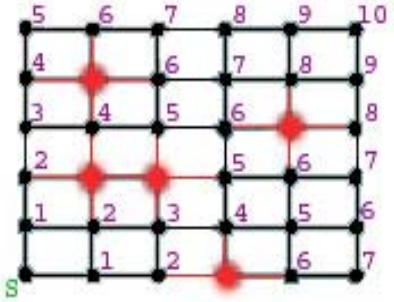

Figure 2. Blind flooding algorithm (all port model)

In order to better understand the GFG algorithm, it is worth illustrating its two components: the greedy algorithm, and the face algorithm. We begin with an example of how the greedy algorithm works in Figure 3. We see that if *S1* is the source and *D* is the destination, the algorithm fails. If however, *S2* is the source and *D* the destination, the greedy algorithm succeeds. This demonstrates that the greedy algorithm alone does not guarantee delivery.

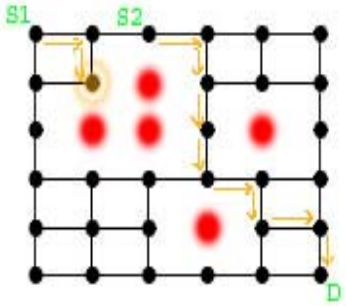

Figure 3. Greedy algorithm on a 6x6 mesh

We will now demonstrate the face algorithm on a 10x10 faulty mesh. The faulty nodes are represented in red, and their communication links are severed, as shown in Figure 4, to get a clearer picture of the algorithm. The green arrows show the progression of the face algorithm. The light blue line represents the imaginary line which contains crossover points of the face algorithm. Note that when an imaginary line passes though a healthy node and a message arrives at it, this node is treated as a new source and the face routing restarts from this node. This case is frequent in regular topologies like meshes, and is not likely in random graphs, which is the reason why it was not discussed in [BMSU].

The process of routing using the GFG algorithm was described in Section 2. It will be illustrated here using an example. In Figure 5, we see the same faulty 10x10 mesh as the one above, but this time we will apply the GFG algorithm to it in order to route a message from source *S* to destination *D*. The green arrows show the progression of the face mode of the algorithm, and the orange arrows show the progression of the greedy part of the algorithm. The light blue lines are the imaginary lines of the face algorithm. There is more than one light blue line since a new one is created every time the face algorithm is initiated. The origin of each blue line in Figure 5 is treated as a source node by the face algorithm, and routing is treated as if it had started from that node. At node *S*, the greedy mode of GFG already fails since all nodes that are closer to the source are faulty. The face algorithm is initiated, and the first light blue line is drawn. The algorithm is continued until the destination is reached.

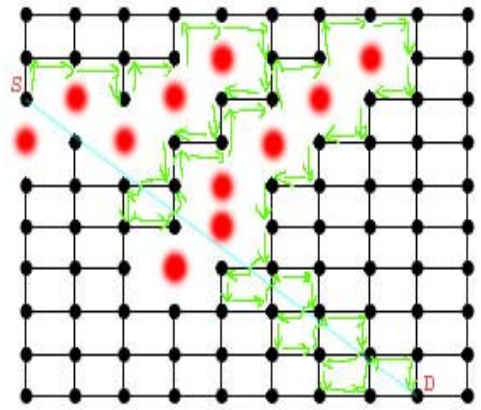

Figure 4. Face algorithm on a 10x10 faulty mesh

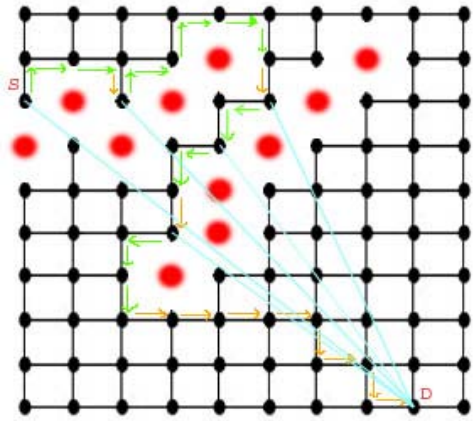

Figure 5. GFG algorithm on a fault 10x10 mesh

V. ROUTING WITH *K*-D MESHES AND *K*-ARY *N*-CUBES

In this paper, we also introduce an algorithm for routing messaged through *k*-D meshes and *k*-ary *n*-cubes. It is based on the GFG approach, with slight modifications. Namely, we have to consider the higher dimensions involved in such a routing process. In order for our algorithm to be successful, a condition must be met (other than working with planar structures). Every 2-dimensional subspace of the *k*-dimensional mesh must be connected. Our procedure for routing in *k*-dimensional meshes is based on routing in consecutive 2-D subspaces, each time coming one dimension closer to the destination, until the destination is reached. Figures 6 and 7 illustrate the algorithm on a 3-D mesh.

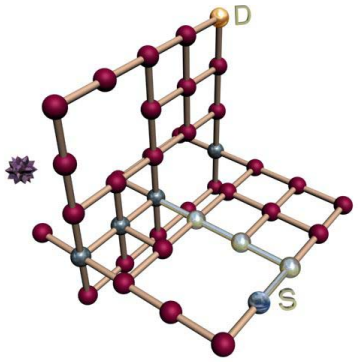

Figure 6. Route from source to a helper node

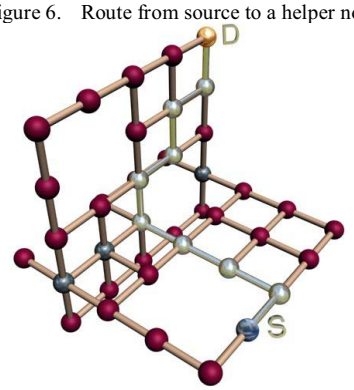

Figure 7. Route from helper node to destination

Two 2-D subspaces of the 3-D mesh are shown Figures 6 and 7. It is along these two subspaces that the message is routed. The grey nodes in both figures show the intersection of the two subspaces. Fig. 6 shows a GFG route from the source to the purple helper node, which is located in the same horizontal plane as S. The GFG route is performed in the horizontal subspace, where S is located. This node is placed behind the line of intersection with the vertical subspace, and is used to force the route path to cross this line. Once the message reaches one of the grey nodes, it switches routing dimensions, as seen in Fig. 7. Here, a simple GFG route is performed to D.

We now present the pseudo-code for our proposed algorithm. The first pseudo-code presented deals with routing in 3-D faulty meshes from source S=(s1, s2, s3) to destination D=(d1, d2, d3).

If ($d1<s1$) **then** $d'=d1-1;$
If ($d1>s1$) **then** $d'=d1+1;$
If ($d1=s1$) **then** $X=S$
Else { Follow the GFG algorithm from S to $D'=(d', s2, s3)$ until a healthy node $X=(d, s2', s3)$ is reached };
Follow the GFG routing algorithm from X to D.

Now, for the k-D case, a similar approach is taken. Routing from source $S=(s_1, s_2, \ldots, s_k)$ to destination $D=(d_1, d_2, \ldots, d_k)$ is done in one 2-D subspace at a time, and the message gets one dimension closer to the D after routing through each subspace.

For (i=1 to k-2) **do** {
If ($d_i<s_i$) **then** $d'=d_i-1;$
If ($d_i>s_i$) **then** $d'=d_i+1;$
If ($d_i \neq s_i$) **then** { Follow the GFG algorithm from $X=(d_1 \ldots d_{i-1}, x_i, x_{i+1}, s_{i+2}, \ldots, s_k)$ to $D'=(d_1 \ldots d_{i-1}, d'_i, x_{i+1}, s_{i+2}, \ldots, s_k)$ until a healthy node $Y=(d_1 \ldots d_{i-1}, d_i, y_{i+1}, s_{i+2}, \ldots, s_k)$ is reached; $X=Y$ } };

Follow the GFG algorithm from $X=(d_1, d_2 \ldots d_{k-2}, x_{k-1}, x_k)$ to $D=(d, d_2, \ldots, d_k)$.

VI. Broadcasting with Guaranteed Delivery in One-port Faulty MCCs and Tori

We have developed an algorithm that can broadcast messages through k-dimensional meshes or tori. In the case of 2D MCCs and tori, it guarantees delivery to all nodes connected to the source. Our method is a combination of the classical broadcasting algorithm described earlier, and the face routing algorithm which was also previously discussed. In case our algorithm is applied to a completely healthy MCC, it reduces to the classical solution for the one-port model. The idea from the classic algorithm stayed the same: the message is transmitted horizontally across the network, and then all receiving nodes retransmit the message vertically. This method fails if a faulty node is reached. In such a case, all healthy nodes behind the faulty one are skipped, and in essence are treated as faulty. To bypass this, we propose a scheme that literally bypasses all faulty nodes using the face routing algorithm. In Figure 8, we see a 2-dimensional MCC, where node S is broadcasting a message to all other nodes. The message is transmitted horizontally and simultaneously in both directions by node S. The following discussion will focus on broadcasting in 2D faulty MCCs, and will be generalized in the next section. The method easily extends to a 2D tori, since they are also planar graphs, which is the only requirement (in addition to connectivity), for face routing to work.

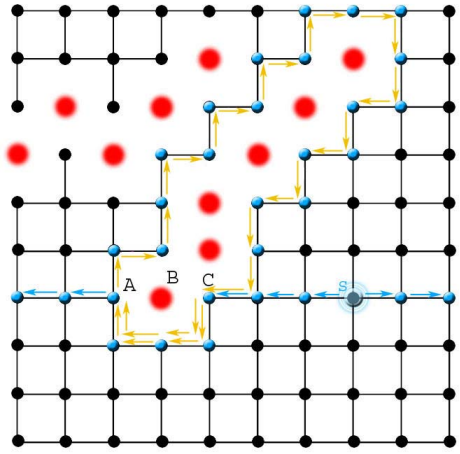

Figure 8. The next healthy node on a horizontal line

Figure 8 illustrates how the message eventually encounters a faulty node, B, when traveling left towards node A. In such situations, the face routing algorithm would be activated, and the next healthy node in the horizontal line from S would be found, if such a node exists. In this case such a node does exist, and in figure 8 it is labeled A. As seen in Figure 8, all of the nodes that were visited by the face algorithm were activated, and all of these nodes will be sending messages vertically in the next phase of the algorithm. All of the activated nodes in Figure 8 are shown in blue. If, however, all other nodes left of C were faulty, the face algorithm would return to C. In that case, the algorithm would proceed to the vertical transmission phase, seen in Figure 10. The face routing is applied here in the simple form that guarantees delivery with simple observation. It does not produce the message optimal solution and various optimizations are possible. The reason for following the full

face perimeter is to activate all nodes on the perimeter, and to find the closest healthy node in the given row, and in the given direction. It is possible that the first encountered node is not the closest, and this may complicate programming if such a node continues the horizontal advance immediately. In Figure 9, we see that the first encountered node in line with *S* is *A*. It however is not the closest healthy node to *S*. The face route encounters *B*, and finally *C*, which is the closest healthy node to *S*. The message returns to *S*, having discovered that *C* is the closest healthy node in the same row. The message is forwarded from *S* to *C* by face routing, where the horizontal advance continues.

Node *E* was chosen to illustrate the vertical transmission phase in Figure 10. It transmits the message vertically and simultaneously, both up and down. Once the message reaches node *F*, it must perform another face route to get to node *G*. At this point it continues on to node *H*. Note again that the vertical phase of the algorithm performs the same type of vertical transmission of the message from every activated node. Using this methodology, all nodes connected to the source will receive the message. Note also that the procedure for vertical transmission is the same as for horizontal transmission.

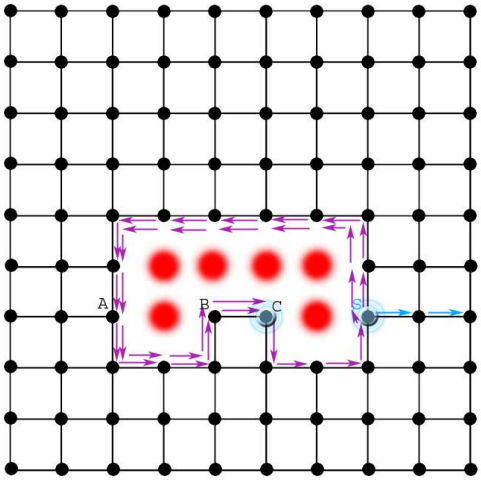

Figure 9. Search for the closest healthy node in the same row

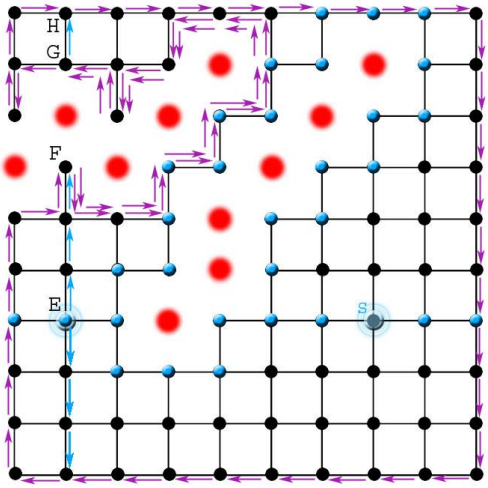

Figure 10. The vertical transmission phase from node *E*

The following section will present and explain the pseudo-code for the proposed algorithm. The first procedure in the algorithm is called send_to_row. It has the following arguments: *X*: the source node; *dim_to_send*: dimension the message is transmitted; and *dim_to_help*: determines the dimension in which face routing will operate in case a faulty node is detected. In this section, for 2D MCCs, one can fix *dim_to_send*=1 and *dim_to_help*=2. This procedure implements the first phase of the proposed algorithm: the horizontal transmission of the message. The direction variable is set to either 1 or -1, and specifies whether the message is traveling left or right along dimension *dim_to_send*. Advance is a Boolean variable, and specifies whether a node has reached the farthest healthy node in dimension *dim_to_send*.

Procedure *send_to_row(X, dim_to_send, dim_to_help)* {
XL=X; XR=X;
Do the following two while loops in parallel {
While ((*XL* is not the leftmost node in row *dim_to_send*) or (*advance*=false)){ *Advance*=true; *Direction*= -1;
Route (XL, direction, dim_to_send, dim_to_help, advance);}
While ((*XR* is not the rightmost in row *dim_to_send*) or (*advance*=false)){*Direction*= 1; *Advance*=true;
Route(XR, direction, dim_to_send, dim_to_help, advance);}}}

The two while loops in procedure *dim_to_send* call procedure *route*. It has the following arguments *X*: the source node; *direction*: integer which specifies the direction of the transmission; *dim_to_send*: dimension the message is transmitted; *dim_to_help*: determines the dimension in which face routing will operate in case a faulty node is detected; and advance: the Boolean variable responsible for determining the last healthy node in *dim_to_send*. The route procedure routes messages from healthy neighbour to healthy neighbour in *dim_to_send*. It is the procedure that calls the face routing algorithm if the next node in dimension *dim_to_send* is faulty.

Procedure *route(X, direction, dim_to_send, dim_to_help, advance)* {*D= X + direction* (in *dim_to_send*); /* **if** $X=(X_1, X_2)$ and *direction = -1*, **then** $D=(X_1\text{-}1, X_2)$ */
If (*D* is healthy) **then** { Route message to *D; X=D;* } **else** {
Face route the message to *D* along *dim_to_help;*
Memorize the closest node *D'* along the desired direction in the desired row until the message returns to *X;*
If (any such node *D'* found) **then** { Face route the message to *D'; X=D';* }**else** { *Advance*=false;} }}

To broadcast in a 2D mesh, procedure Broadcast_2D is called. It invokes the two previously discussed procedures.

Procedure Broadcasting_2D (*X*) {
Send_to_row(*X,* 1, 2);
For (each healthy node *Y* that received the message from *X*) { *Send_to_row*(*Y,* 2, 1); } }.

The algorithm guarantees delivery to all nodes connected to the source since it activates at least one in every column in the first phase, and these nodes then send the message to all other nodes in their column. Note that some redundancy may occur, but it can be reduced by some simple programming extensions.

VII. Broadcasting in One-Port Faulty k-D Meshes and *k*-ary *n*-cubes

To broadcast in a k-D MCC, a series of successive broadcasts are done in the same 2-D subspaces of the k-D MCC. The following pseudo code underlines this. The condition is that each 2-D subspace of the k-D MCC must be

connected. Otherwise, one such plane with disconnected healthy nodes may not forward the message from one component to another, even if they could be connected via other nodes in other dimensions.

Broadcasting_in_a_*k*-D mesh(*X*) {
  Send_to_row(*X*, 1, 2) // with other coordinates same as in *X*
  For (dimensions *j=2* **to** *k*) {
    For (each healthy node *Y* that received the message from *X*) { Send_to_row(*Y, j, j-1*); } } }

VIII. Conclusions

In this paper, we proposed some routing and broadcasting algorithms for faulty meshes which use local knowledge of faults, no additional resources, work for an arbitrary number and structure of faults, guarantee delivery to all nodes connected to the source, and remain optimal in a fault free mesh. These are the first known solutions with such properties. We propose a new broadcasting algorithm which guarantees the delivery (to all processors connected to the source) in the all port model of faulty meshes. We then describe routing and broadcasting algorithms that guarantee delivery in faulty MCCs and tori, the connectivity of the source and destination(s) being the only obvious requirement. The algorithms are then extended to faulty *k*-D meshes and *k*-ary *n*-cubes, where the delivery will be guaranteed if healthy nodes in every 2-D submesh (sub-tori) remain connected.

The routing and broadcasting problems for the one-port model of higher dimensional faulty meshes and tori was solved only with the mentioned condition. The open problem that remains for further study is the design of algorithms with weaker or hopefully no conditions attached.

This paper did not study deadlock-free, livelock-free and some other additional important properties for protocols in interconnection networks. It is possible to construct examples where our algorithms cause deadlocks. Therefore, it is an open problem to modify proposed algorithms to become deadlock-free. Next, our algorithms are proposed with variants that can be easily expressed and understood, and a number of optimizations are possible. It is also of interest to study the upper bounds on the number of unicast steps, and to experimentally compare existing and new algorithms.

References

[AB] B.F.A. AlMohammad, B. Bose, Fault-tolerant communication algorithms in toroidal networks, *IEEE Trans. Par. Distrib. Systems*, 10, 10, 1999, 976-983.

[AC] D.R. Avresky, C.M. Cunningham, Single-source fault-tolerant broadcasting for two-dimensional meshes without virtual channels, *Proc. European Dependable Computing Conference*, LNCS 1150, 1996, 178-189.

[BC] R. V. Boppana and S. Chalasani. Fault-tolerant wormhole routing algorithms for mesh networks. *IEEE Trans. on Computers*, 44(7):848--864, July 1995.

[BD] Y. M. Boura and C. R. Das. Fault-tolerant routing in mesh networks. *In Proc. 1995 Int. Conf. on Parallel Processing*, I.106--109, Aug. 1995.

[BMSU] P. Bose, P. Morin. I. Stojmenovic, J. Urrutia, Routing with guaranteed delivery in ad hoc wireless networks, *Wireless Networks,* 7, 6, 2001, 609-616.

[CA] C. Cunningham and D. Avresky, Fault-Tolerant Adaptive Routing for Two-Dimensional Meshes, *Int. Symp. High Performance Computing Architecture*, Raleigh, 1995.

[CB] S. Chalasani and R V. Boppana, Communication in Multicomputers with Nonconvex Faults. *IEEE Trans. on Computers*, vol. 46, pp. 616-622, May 1997.

[CC] K.-H. Chen and G.-M. Chiu. Fault-tolerant routing algorithm for meshes without using virtual channels. *Technical report, Nat'l Taiwan Inst. of Tech*., 1997.

[GN] C. Glass and L. Ni. Fault-Tolerant Wormhole Routing in Meshes without Virtual Channels. *IEEE Trans. Par. Distributed Systems* vol. 7, no. 6, 620-636, June 1996.

[JW] Z. Jiang and J. Wu, A fault-tolerant broadcasting in 2-D wormhole-routed meshes, *Int. J. Computers and Their Applications*, to appear.

[LH] D.H. Linder and J,.C. Harden, An adaptive and fault-tolerant wormhole routing strategy for *k*-ary *n*-cubes, *IEEE Trans. Computers*, 40, 2-12, Jan. 1991.

[PSY] S. Park, S. Seidel, J.H. Youn, Fault-tolerance broadcasting in wormwhole-routed torus networks*, IEEE Int. Parallel Processing Symp. IPDPS*, 2002.

[S] J. D. Shih. Adaptive fault-tolerant wormhole routing algorithms for hypercube and mesh interconnection networks. *Proc. of the 11th International Parallel Processing Symposium*. April 1997, 333-340.

[SS] C. C. Su and K. G. Shin. Adaptive fault-tolerant deadlock free routing in meshes and hypercubes. *IEEE Transactions on Computers*. 45, (6), 1996, 672-683.

[SW] P.H. Sui and S.D. Wang, An improved algorithm for fault-tolerant wormhole routing in meshes*, IEEE Trans. Computers*, 46, 9, 1997, 1040-1042.

[TKL] Y.C. Tseng, D. Kpanda, T.H. Lai, A trip-based multicasting model in wormwhole-routed networks with virtual channels, *IEEE TPDS*, 7, 2, 1996, 138-150.

[VUM] V. Varavithya, J. Upadhyay, P. Mohapatra, An efficient fault-tolerant routing scheme for two-dimensional meshes, *Int. Conf. High-Perf. Comp.*, 1995, 773-778.

[W1] J. Wu. A fault-tolerant adaptive and minimal routing approach in 3-D meshes. *Proc. of the 7th Int'l Conf. on Parallel and Distributed Systems (ICPADS)*. July 2000.

[W2] J. Wu. Fault-tolerant adaptive and minimal routing in mesh-connected multicomputers using extended safety levels, *IEEE Transactions on Parallel and Distributed Systems*, 11, (2), Feb. 2000, 149-159.

[W3] J. Wu, A distributed formation of orthogonal convex polygons in mesh-connected multicomputers, *Proc. IEEE Int. Parallel Processing Symp*. IPPS, 2001.

[WCW] G.C. Wang, J. Chen, G.J. Wang, A probabilistic approach to fault tolerant broadcast routing algorithms on mesh networks, *IEEE IPDPS,* 2003.

[ZK] L. Zakrevski, M. Karpovsky, Fault-tolerant message routing for multiprocessors, *IEEE IPPS/SPDP Work.*, 1998.