A New Approach to Fault-Tolerant Wormhole Routing for Mesh-Connected Parallel Computers

Revised, September 13, 2003

Ching-Tien Ho
IBM Almaden Research Center
San Jose, CA 95120
ho@almaden.ibm.com

Larry Stockmeyer
IBM Almaden Research Center
San Jose, CA 95120
stock@acm.org

Abstract

A new method for fault-tolerant routing in arbitrary dimensional meshes is introduced. The method was motivated by certain routing requirements of an initial design of the Blue Gene supercomputer in IBM Research. The machine is organized as a 3-dimensional mesh containing many thousands of nodes. Among the requirements were to provide deterministic deadlock-free wormhole routing in a 3-dimensional mesh, in the presence of many faults (up to a few percent of the number of nodes in the machine), while using two virtual channels. It was also desired to minimize the number of "turns" in each route, *i.e.*, the number of times that the route changes direction. There has been much work on routing methods for meshes that route messages around faults or regions of faults. The new method is to declare certain nonfaulty nodes to be "lambs"; a lamb is used for routing but not processing, so a lamb is neither the source nor the destination of a message. The lambs are chosen so that every "survivor node", a node that is neither faulty nor a lamb, can reach every survivor node by at most two rounds of dimension-ordered (such as e-cube) routing. An algorithm for finding a set of lambs is presented. The results of simulations on 2D and 3D meshes of various sizes with various numbers of random node faults are given. For example, on a $32 \times 32 \times 32$ 3D mesh with 3% random faults, and using at most two rounds of e-cube routing for each message, the average number of lambs is less than 68, which is less than 7% of the number 983 of faults and less than 0.21% of the number 32768 of nodes. The computational complexity of finding the minimum number of lambs for a given fault set is also explored, and this problem is shown to be NP-hard for 3-dimensional meshes with two rounds of e-cube routing.

An abridged version of this paper will appear in the IEEE Transactions on Computers.

1 Introduction

A new method for fault-tolerant routing in arbitrary dimensional meshes is introduced. The method was motivated by certain routing requirements of an initial design of the Blue Gene supercomputer in IBM Research [1, 2]. The machine is planned to be organized as a 3-dimensional mesh containing many thousands of nodes. Among the requirements were to provide: (i) deterministic deadlock-free wormhole routing in a 3-dimensional mesh, and (ii) the ability to tolerate a fairly large number of faults (up to a few percent of the number of nodes in the machine). Some hardware, software and performance restrictions were: (iii) for the purpose of avoiding deadlock, at most two virtual channels can be used, and (iv) it is advantageous to minimize the number of "turns" in each route, *i.e.*, the number of times that the route changes direction. In wormhole switching [11], each message is divided into *flow control units*, or *flits*. The flits of a message follow the same path through the network in a pipelined manner. When the head flit of a message is at a certain node, it and all the other flits of the message can proceed if the needed channel is not occupied; otherwise, all flits wait at their current nodes until the channel becomes free. If the routing is not done with some care, messages can wait for one other, causing a deadlock.

A useful way to define routes in a mesh is by fixing an ordering of the dimensions of the mesh. For example, in a 3D mesh with dimensions named X, Y and Z, the XYZ-route from node (x, y, z) to node (x', y', z') first moves along the X dimension to (x', y, z), then along the Y dimension to (x', y', z), and finally along the Z dimension to (x', y', z'). The ordering that follows the ascending order of dimensions is called the *e-cube* routing. In the presence of faults, certain nodes will not be able to reach other nodes by one round of XYZ routing. In this case, more than one round can be used. In the case of two rounds of XYZ routing, for example, a route follows the order of dimensions $XYZXYZ$. It is assumed that the entire message is not stored at the node reached after the first round (XYZ) of routing, but continues in a pipelined fashion throughout both rounds. Any number k of rounds of dimension-ordered routing (possibly using a different ordering in different rounds, but always using the same ordering during the same round) is guaranteed to be deadlock free, assuming that a different virtual channel is used during each round. And with fewer than k virtual channels, deadlock freedom cannot be guaranteed in general. So with two virtual channels we can use $k = 2$ rounds. A machine with k virtual channels can route k different messages concurrently over the same physical link, but the cost of the machine increases as k increases.

There has been much work on methods for for fault-tolerant wormhole routing, for example, [4, 5, 6, 7, 8, 10, 13, 17, 18]. One popular method is to route messages around fault regions. Boppana and Chalasani [4] give a method that uses only two virtual channels, provided that the fault regions are rectangular and the fault rings (nonfaulty boundaries around fault regions) do not overlap. Chalasani and Boppana [6] extend the fault model to include regions, called *solid faults*, such as crosses, L's and T's; their method uses four virtual channels, again under the assumption that fault rings do not overlap. This was improved by Chen and Chiu [7] by reducing the number of virtual channels to three and allowing fault rings to overlap. For all of these methods, routes can involve extra turns when going around fault regions. For example, there is a fault set on a 2D $n \times n$ mesh that causes some routes to use a constant times n turns. Although these methods are attractive if more than two virtual channels can be used, the faults belong to certain patterns, and the number of turns is not an issue, other alternatives are needed when these assumptions do not hold.

The new method is to declare certain nonfaulty nodes to be "sacrificial lambs"; a lamb is used for routing but not processing, so a lamb is neither the source nor the destination of a message. The lambs are chosen so that every "survivor node", a node that is neither faulty nor a lamb, can reach every survivor node by at most two rounds of dimension-ordered routing.

An algorithm that finds a lamb set for a given fault set is presented. The algorithm is designed to handle arbitrary d-dimensional meshes of arbitrary size and any number k of rounds of dimension-ordered routing (possibly a different ordering in each round). Not counting the time to output the lamb set, the algorithm runs in time polynomial in d, k, and the number f of faults; however, the time is independent of the size of the mesh. If d and k are constants, the worst-case running time is proportional to f^3 plus the size of the lamb set found. The algorithm is not guaranteed to find a lamb set of minimum size for the given fault set, although we prove that it always finds a lamb set whose size is within a factor of two of the minimum size (it is a "2-approximation" algorithm, cf. [12, 16]). We also give a variation of the algorithm that finds an optimally small lamb set and has running time exponential in df; this would be useful only for very small d and f, but again the time does not depend on the size of the mesh.

Some justification for using an approximation algorithm is provided by another result showing that minimizing the number of lambs is an NP-hard problem if $d \geq 3$ and $k \geq 2$. It is also shown that the best ratio of approximation by a polynomial-time algorithm for the lamb problem is essentially the same as that for the vertex cover problem. Even though there has been a considerable amount of work on polynomial-time approximation algorithms for the vertex cover problem (see, for example, [16, §3]), 2 is the best constant factor of approximation currently known.

Of course, the usefulness of this method depends on the number of good nodes that must be "sacrificed". The results of simulation studies for randomly chosen faults show that the number is reasonably small if the number of faults does not exceed the bisection width of the mesh. For example, on a $32 \times 32 \times 32$ 3D mesh with $k = 2$ rounds of e-cube routing and $32^2 = 1024$ random faults (so 3.125% of the mesh is faulty), the average number of lambs is less than 79, which is less than 0.25% of the number 32768 of nodes and less than 8% of the number 1024 of faults.

Some researchers have argued that it is impractical to assume a fault model of static faults (no new faults can occur) and global knowledge of the fault distribution. However, in some modern parallel computers, a system diagnostic program will be invoked when new faults are detected. This will roll back to a previous checkpoint of the application, redefine the new set of faults, and reconfigure the machine assuming static faults and global knowledge. Our approach and algorithm would be part of the reconfiguration step. Thus, our approach is applicable to dynamic faults within the roll-back/reconfigure framework.

It is unlikely that any one solution to a problem will be the best in all respects, and there are usually trade-offs to be considered. On the one hand, some number of nodes must be sacrificed in our approach: they cannot be used for processing that needs to communicate with other nodes. On the other hand, all surviving nodes can reach each other by a "simple route", in our case, 2 rounds of dimension-ordered routing. This translates into a simpler machine design with lower cost per node when compared to designs that use more complicated routing strategies. (Although we concentrate on mesh topologies in this paper, it should be noted that this general approach could be used for other topologies and other notions of "simple route".)

An algorithm that is designed to work only when fault regions have certain shapes (such as the algorithms of [4, 6, 7]) can be made to work for arbitrarily located faults by first deactivating certain nodes in such a way that the regions of faulty/deactivated nodes have the required shapes. The deactivated nodes are similar to lambs, except that an deactivated node is not used for either processing or routing. A question (to which we do not have an answer) is how many nodes need to be deactivated as compared to the number of lambs in our approach.

The rest of the paper is now outlined. The lamb problem is defined in Section 2. In Section 3 it is shown that using only one round of dimension-ordered routing leads to unnacceptably large lamb sets, thus justifying the use of two rounds. Section 4 describes the main method used to obtain a running time that does not depend on the size of the mesh. The idea is to partition the nodes into groups such that all nodes in the same group have the same reachability property with respect to all the nodes. There are actually two partitions, one where nodes are grouped according to their reachability property as a source, and another according to their reachability property as a destination. As shown by the algorithm given later, for both the source and destination cases, a partition can always be found where the number of groups is at most $(2d-1)f+1$. Before giving the algorithm in full generality and detail, its operation is illustrated in Section 5 for a particular 12×12 mesh with three faults. The algorithm is described in detail in Section 6. Some straightforward extensions of the algorithm are described in Section 7. Section 8 gives the results of the simulations. Results on the computational complexity of the lamb minimization problem are given in Section 9. Section 10 is the conclusion.

2 Formulation of the Problem

2.1 Background on meshes and routings

Although we are primarily interested in 2- and 3-dimensional meshes, we give the definitions in general for d-dimensional meshes because the results apply for arbitrary d. Throughout the paper, d denotes the dimension of a mesh.

Definition 2.1 Let $d \geq 1$ and $n_1, \ldots, n_d \geq 2$ be integers. The *d-dimensional mesh with widths* $n_1, \ldots, n_d$, denoted $M_d(n_1, \ldots, n_d)$, is the graph with nodes

$$\{ (v_1, v_2, \ldots, v_d) \mid 0 \leq v_i < n_i \text{ for } i = 1, 2, \ldots, d \}.$$

For each pair v and w of nodes whose L_1 distance equals 1 (*i.e.*, $\sum_{i=1}^{d} |v_i - w_i| = 1$) there is an edge directed from v to w and an edge directed from w to v. We refer to an edge as a *link*.[1] We let $\langle v, w \rangle$ denote the link from node v to node w. We use $N = n_1 n_2 \cdots n_d$ to denote the total number

[1] The two unidirectional links can be viewed as a single bidirectional link. Our definition has the flexibility of allowing a link to fail only in one direction.

of nodes. Let $M_d(n)$ denote the d-dimensional mesh where all widths are n; here, $N = n^d$. For a mesh M we let *nodes*(M) (resp., *links*(M)) denote the set of nodes (resp., links) of M.

Definition 2.2 A *dimension-ordered routing* on an d-dimensional mesh is a static routing in which every message travels according to the same predetermined order of dimensions. An ordering is specified by a permutation π of $\{1, 2, \ldots, d\}$, called a *π-ordering*, and the resulting dimension-ordered routing is called a *π-routing*.

For convenience, we will call the two dimensions of a 2D mesh as X and Y, and the three dimensions of a 3D mesh as X, Y, and Z. In particular, an *XY-routing* on a 2D mesh is a dimension-ordered routing in which the order of dimensions is (X, Y), and an *XYZ-routing* on a 3D mesh is a dimension-ordered routing in which the order of dimensions is (X, Y, Z). For example, the XYZ-routing from node (x, y, z) to node (x', y', z') first moves from (x, y, z) to (x', y, z), then to (x', y', z), and finally to (x', y', z'). On a d-dimensional binary hypercube, i.e., a d-dimensional mesh with all widths equal to 2, a dimension-ordered routing following the increasing order of dimensions is sometimes called the *ascending routing* and commonly known as the *e-cube routing*. The XY-routing and XYZ-routing are 2D-mesh and 3D-mesh versions of the ascending routing, by viewing X as the lowest dimension.

For every pair v and w of nodes of a mesh and every ordering π, node w is reachable from node v by a π-routing, assuming that all nodes and links of the mesh are good (*i.e.*, nonfaulty). If nodes or links can be faulty, however, then paths through faulty nodes and links are not possible and there can be v, w, π such that w is not reachable from v by a π-routing. For example in a 2D mesh, node $(3, 2)$ is not reachable from node $(0, 0)$ by an XY-routing if any one of the nodes $(1, 0), (2, 0), (3, 0), (3, 1)$ is faulty; however, node $(0, 0)$ may be reachable from node $(3, 2)$ even if all of $(1, 0), (2, 0), (3, 0), (3, 1)$ are faulty, because the XY-routing in this case passes through $(2, 2), (1, 2), (0, 2), (0, 1)$. Thus, when there are faults, one round of a fixed dimension-ordered routing may not be enough for every good node to reach every good node. In this case, we can consider multiple rounds of routing. In general, each round can use a different dimension-ordered routing.

Definition 2.3 Let $k \geq 1$ and let $\pi_1, \ldots, \pi_k$ be permutations of $\{1, 2, \ldots, d\}$. A *$(\pi_1, \ldots, \pi_k)$-ordered k-round routing* is one where the i-th round of routing is a π_i-ordered routing for $1 \leq i \leq k$. A *π-ordered k-round routing* is a $(\pi, \pi, \ldots, \pi)$-ordered k-round routing.

A $(\pi_1, \ldots, \pi_k)$-ordered k-round routing from a node v to a node w first uses a π_1-routing in the first round to reach some intermediate node u_1, then uses a π_2-routing in the second round to reach some intermediate node u_2, and so on until a π_k-routing is used in the last round from the final intermediate node u_{k-1} to w. The entire message is not stored at the intermediate nodes, but it continues in a pipelined manner through all k rounds. To avoid deadlock, each round uses a separate virtual channel. It is possible that two or more consecutive nodes in the path $v, u_1, u_2, \ldots, u_{k-1}, w$ are identical, *i.e.*, that fewer than k rounds are actually sufficient. This is allowed because by convention there is a (degenerate) π-routing from each node to itself, for every permutation π.

Note that when $k \geq 2$, the specification $(\pi_1, \ldots, \pi_k)$ of the ordering for each round does not determine a fixed route from one node to another, because there can be many choices for the (at most) $k-1$ intermediate nodes. This choice can affect message congestion, although the problem of how to choose the intermediate nodes is not a subject of this paper. (One heuristic is to choose routes of shortest length, breaking ties randomly.) However, $(\pi_1, \ldots, \pi_k)$ and the positions of the faulty nodes and links do determine, for each pair v and w of nodes, whether w is reachable from v by a $(\pi_1, \ldots, \pi_k)$-ordered routing that passes through no faulty node or link. We continue this development in the next section where faults are formally introduced.

2.2 Definitions related to faults

In the definitions and algorithms we allow both node faults and link faults. However, in examples and in the simulation results, we consider only node faults for simplicity. One way to handle link faults is to consider either endpoint of each faulty link to be a faulty node. But because this introduces unnecessary additional faults, we consider link faults separately.

Definition 2.4 A *fault set* for a mesh M has the form $F = (F_N, F_L)$ where $F_N \subseteq \mathit{nodes}(M)$ and $F_L \subseteq \mathit{links}(M)$. We let $f = |F_N| + |F_L|$ denote the total number of faults.

A nonfaulty node or link, *i.e.*, a node or link not in F_N or F_L, is said to be *good*. If $v \in F_N$ then all links incident to v are faulty in the sense that these links cannot be used for routing, although these links do not appear explicitly in F_L. For example, Figure 1 shows the 12 by 12 mesh $M_2(12)$. We will assume the origin of the mesh, node $(0,0)$, is located at the upper left corner and node $(11,0)$ is located at the upper right corner. Figure 2 shows an example of the same mesh with 3 node faults ($F_N = \{(9,1), (11,6), (10,10)\}$) and no link faults ($F_L = \emptyset$).

We now give definitions related to the reachability of one node from another in the presence of faults.

Definition 2.5 Let $F = (F_N, F_L)$ be a fault set of a mesh M and let $v, w \in \mathit{nodes}(M)$.

1. Let π be a 1-round ordering. Then w is (F,π)*-reachable* from v, and v can (F,π)*-reach* w, if the (unique) π-ordered routing from v to w does not visit any node in F_N nor any link in F_L; in particular, if v can (F,π)-reach w then neither v nor w belongs to F_N.
2. Let $\vec{\pi} = (\pi_1, \ldots, \pi_k)$ be a k-round ordering. Then w is $(k, F, \vec{\pi})$*-reachable* from v, and v can $(k, F, \vec{\pi})$*-reach* w, if there exist nodes $u_0, u_1, \ldots, u_k$ such that $u_0 = v$, $u_k = w$, and u_{i-1} can (F, π_i)-reach u_i for $1 \le i \le k$.

For simplicity in the examples and the simulations, they are restricted to 2- and 3-dimensional meshes, only node faults, and π-ordered 2-round routings where the order π in each round is XY (for 2D) or XYZ (for 3D).

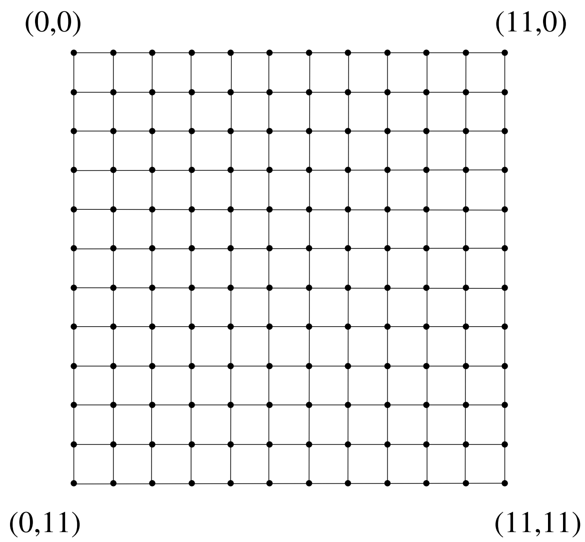

Figure 1: An example of a 12×12 mesh.

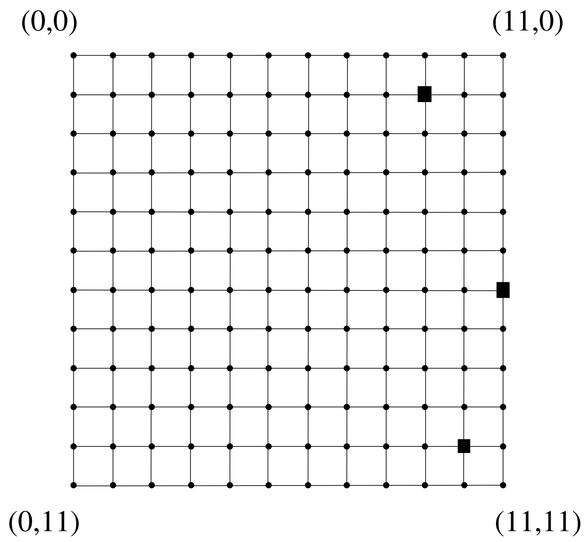

Figure 2: An example of a 12×12 mesh with 3 faults.

2.3 The lamb problem

A set Λ of good nodes is a lamb set if all good nodes not in Λ can communicate with one another in k rounds, where nodes in Λ can be used for routing but faulty nodes and links cannot.

Definition 2.6 A subset $V \subseteq \mathit{nodes}(M)$ is a $(k, F, \vec{\pi})$-*survivor set* if, for all $v, w \in V$, node v can $(k, F, \vec{\pi})$-reach node w; in particular, $V \cap F_N = \emptyset$. A subset $\Lambda \subseteq \mathit{nodes}(M)$ is a $(k, F, \vec{\pi})$-*lamb set* if $\Lambda \cap F_N = \emptyset$ and $\mathit{nodes}(M) - (\Lambda \cup F_N)$ is a $(k, F, \vec{\pi})$-survivor set.

In other words, by disallowing nodes in a lamb set Λ to send or receive messages, but allowing them to be on the routing paths of messages, all good nodes not in Λ can communicate with one another in k rounds. In many cases the fault set F, the ordering $\vec{\pi}$, and the number k of rounds will be clear from context. In such cases we simplify the terminology by saying, for example, w is reachable from v, Λ is a lamb set, etc.

In this paper, we are interested in finding a survivor set of maximum size for a given mesh, fault set, and routing ordering. This is the same as minimizing the size of a lamb set.

Definition 2.7 The *lamb problem* is the following. Given parameters $d, n_1, \ldots, n_d$ of a mesh $M = M_d(n_1, \ldots, n_d)$, a number k of rounds, a fault set F for M, and a k-round ordering $\vec{\pi}$ for M, find a $(k, F, \vec{\pi})$-lamb set of minimum size. Let $\lambda(M, k, F, \vec{\pi})$ denote the minimum size of a lamb set.

Note that the minimum-size lamb set need not be unique, although its size is. Obviously, for fixed M, F and $\vec{\pi}$, as k increases the minimum size $\lambda(M, k, F, \vec{\pi})$ can decrease but never increase. One conclusion of this paper is that for random faults there is a significant decrease in the size of the lamb set when k increases from 1 to 2. In Section 3, we prove a lower bound on the expected size of the lamb set, as a function of the number of random node faults, when $d = 3$ and $k = 1$.

In the version of the lamb problem above, the k-round ordering $\vec{\pi}$ is an input to the problem. A more general problem is to *choose* a k-round ordering to minimize the number of lambs. An obvious solution is to try all possible orderings at each round (except for degenerate ones such as XYZ followed by ZXY). We have not investigated whether a more efficient algorithm is possible.

2.4 Approximation algorithms

For certain optimization problems, finding a solution that is close to optimal can be considerably easier than finding an optimal solution. In general, a combinatorial minimization problem consists of (i) a set of *inputs*, (ii) for each input I a finite set $S(I)$ of *solutions for* I, and (iii) for each solution s a *cost*, denoted $\mathit{cost}(s)$. For the lamb problem, for example, an input has the form $I = (d, n_1, \ldots, n_d, F, k, \vec{\pi})$, the set $S(I)$ of solutions is the set of $(k, F, \vec{\pi})$-lamb sets Λ for $M_d(n_1, \ldots, n_d)$, and the cost of Λ is $|\Lambda|$.

An algorithm $\mathcal{A}$ is an *approximation algorithm* for a minimization problem if, when given an arbitrary input I, it returns some member of $S(I)$. Let $\mathcal{A}(I)$ be the cost of the solution returned by $\mathcal{A}$ on input I. Let $\mathit{opt}(I)$ be the cost of an optimal solution for I, that is, $\mathit{opt}(I)$ is the minimum of $\mathit{cost}(s)$ over all $s \in S(I)$.

Definition 2.8 Let r be a real number. The approximation algorithm $\mathcal{A}$ is an r-approximation algorithm if $\mathcal{A}(I) \leq r \cdot opt(I)$ for all inputs I.

See [16] for more information on the subject of approximation algorithms.

3 One Round of Routing

The purpose of this section is to show that allowing only one round of routing leads to unacceptably large lamb sets. Because our motivating example is a 3-dimensional mesh with all widths roughly the same, we concentrate on the mesh $M_3(n)$ in this section to simplify the presentation. It is shown that if the number of random node faults is $f \leq n$, then the expected minimum size of a lamb set is proportional to fn^2. Thus, if the number of faults is only $n = N^{1/3}$, a constant fraction of the N nodes must be sacrificed. As f increases from 1 to n, the lower bound of Theorem 3.1 on the expected size of the minimum-size lamb set increases from roughly $n^2/4$ to $n^3/12$.

Theorem 3.1 *Let F_1 be a set of f nodes chosen at random from $M_3(n)$ where $f \leq n$, and let π be a one-round ordering for $M_3(n)$. Then the expected value of the size $\lambda(M_3(n), 1, F_1, \pi)$ of a minimum-size lamb set is at least*

$$\frac{fn^2}{4} - \frac{f^2 n}{4} + \frac{f^3}{12} - f.$$

Proof. The proof is given in the Appendix. □

For example, if $n = f = 32$, the lower bound of Theorem 3.1 is 2698. A result of simulation for this case gives a lower bound of 5750. To compare one-round with two-round, simulations were done on $M_3(32)$ for 10,000 trials, using $k = 2$ rounds of XYZ routing and 32 random node faults at each trial. One lamb was needed in 5 of the trials, and none were needed in the other 9995.

4 Source Equivalent Sets and Destination Equivalent Sets

Fix a mesh M, a fault set F, and a k-round ordering $\vec{\pi} = (\pi_1, \ldots, \pi_k)$. In order to choose a lamb set it is important to know, for each pair v and w of nodes, whether v can $(k, F, \vec{\pi})$-reach w. A straightforward way to do this is to construct a breadth-first spanning tree [9] from each good node while restricting each tree to paths that use at most k rounds, are $\vec{\pi}$-ordered, and avoid faults. Viewing d and k as constants for the moment, this takes time at least N for each tree, so the total time to construct the N trees is at least N^2.

This section introduces concepts that can reduce the time to find a nontrivial lamb set Λ to $O(f^3 + |\Lambda|)$ where f is the total number of node and link faults. This time is independent of N, and it is superior to the time of the spanning tree approach in the typical case that f is small compared to N. The idea is to group nodes into sets such that all nodes in the same set have the same reachability properties with respect to all the nodes.

Definition 4.1 Let F be a fault set and π be a 1-round ordering for a mesh M.

1. A *source equivalent set* (SES) is a set S of good nodes such that, for all $v, v' \in S$ and all $w \in nodes(M)$, w is (F,π)-reachable from v iff w is (F,π)-reachable from v'. A *destination equivalent set* (DES) is a set D of good nodes such that, for all $w, w' \in D$ and all $v \in nodes(M)$, w is (F,π)-reachable from v iff w' is (F,π)-reachable from v.

2. A collection $\Sigma = \{S_1, \ldots, S_m\}$ of sets of nodes is an *SES partition* of M if the sets partition the good nodes (*i.e.*, they are pairwise disjoint and their union is $nodes(M) - F_N$), and each $S \in \Sigma$ is an SES. The *size* of the SES partition Σ, denoted $|\Sigma|$, is m. A *DES partition* is defined analogously, replacing "SES" everywhere by "DES".

For example, Figure 3 shows an SES partition of $M_2(12)$ with respect to the XY-ordering and the three faults shown. It contains nine SES's, labeled $S_1, \ldots, S_9$. Similarly, Figure 4 shows a DES partition with respect to the same ordering and three faults.

Remark 4.1 Define a *source equivalence class (SEC)* (resp., a *destination equivalence class (DEC)*) X to be an SES (resp., a DES) such that X is maximal, that is, for each node $u \in nodes(M) - X$, the set $X \cup \{u\}$ is not an SES (resp., a DES). An *SEC partition* and a *DEC partition* are defined analogously to an SES partition and a DES partition above. Each SEC is an equivalence class of the equivalence relation[2] $\equiv^S_{F,\pi}$ on the good nodes defined by: $u \equiv^S_{F,\pi} u'$ if for all $w \in nodes(M)$, node u can (F,π)-reach node w iff node u' can (F,π)-reach node w. It follows that for each M, F, and π, there is a unique SEC partition (up to renaming of the SEC's in the partition) and that this partition is the SES partition of minimum size. Analogous statements apply to DEC's and the equivalence relation $\equiv^D_{F,\pi}$ on the good nodes defined by: $w \equiv^D_{F,\pi} w'$ if for all $u \in nodes(M)$, node u can (F,π)-reach node w iff node u can (F,π)-reach node w'. In fact, Figures 3 and 4 show the SEC partition and the DEC partition, respectively, of this faulty mesh. The reason to consider SES and DES partitions is that the rest of the algorithm works for any SES partition and any DES partition; but decreasing the number of SES's and DES's improves the algorithm's efficiency. Finding a partition into a reasonably small number of SES's and DES's is easier than finding the (unique) SEC partition and DEC partition.

An immediate consequence of the definitions is that, given a particular SES partition and DES partition, to determine the reachability properties of the DES's from the SES's, it suffices to choose an (arbitrary) representative member of each SES and DES. This is stated next as a lemma.

Lemma 4.1 *Fix M, F and a 1-round ordering π. Let S be an SES and D be a DES. Let $v_0 \in S$ and $w_0 \in D$ be arbitrary. Then v_0 can reach w_0 iff v can reach w for all $v \in S$ and all $w \in D$.*

For example, Figure 5 (resp., 6) shows the SEC's (resp., DEC's) with one representative from each set.

[2] A binary relation is an equivalence relation if it is reflexive, symmetric, and transitive. The equivalence classes of an equivalence relation on a set S form a partition of S. See, e.g., [9, §5.2].

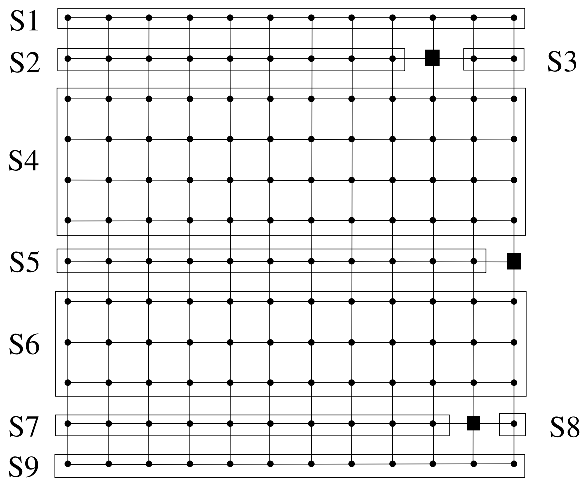

Figure 3: An SES partition of size nine for the faulty mesh in Figure 2.

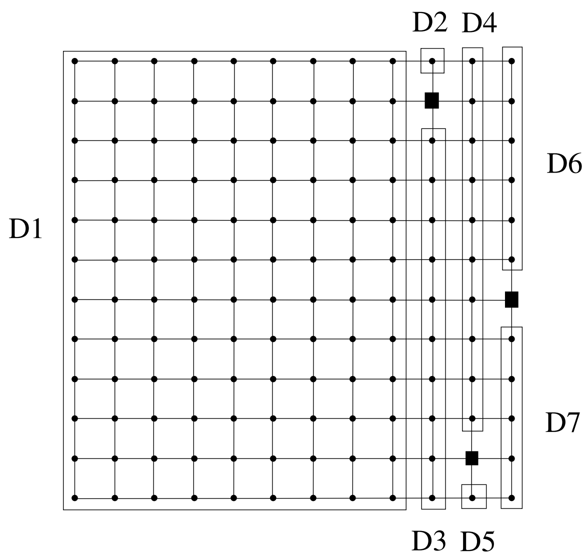

Figure 4: A DES partition of size seven for the faulty mesh in Figure 2.

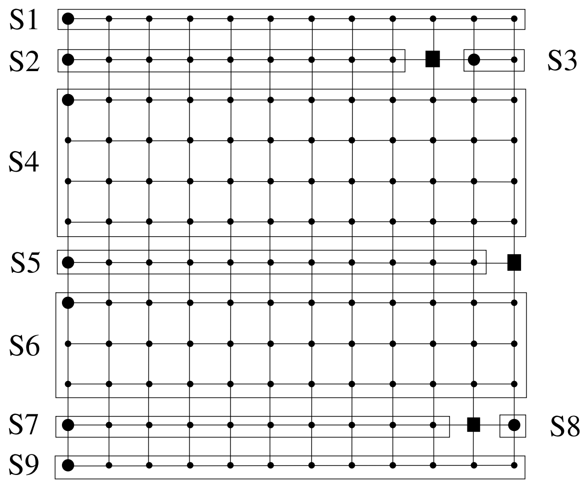

Figure 5: The SES partition of Figure 3 with one representative from each SES.

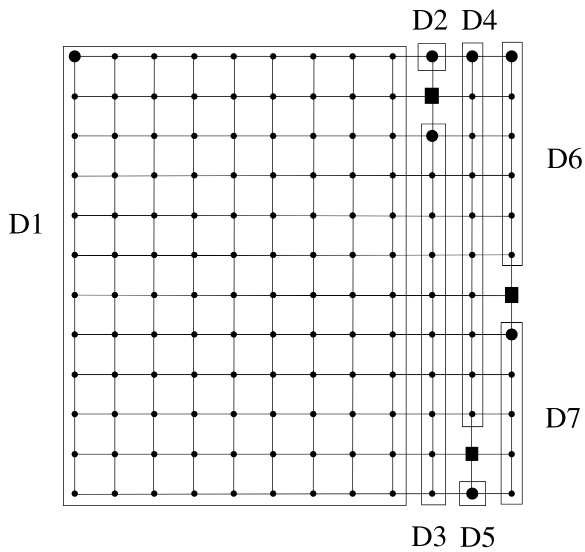

Figure 6: The DES partition of Figure 4 with one representative from each DES.

	D_1	D_2	D_3	D_4	D_5	D_6	D_7
S_1	1	1	0	1	0	1	0
S_2	1	0	0	0	0	0	0
S_3	0	0	0	1	0	1	0
S_4	1	0	1	1	0	1	0
S_5	1	0	1	1	0	0	0
S_6	1	0	1	1	0	0	1
S_7	1	0	1	0	0	0	0
S_8	0	0	0	0	0	0	1
S_9	1	0	1	0	1	0	1

Table 1: The matrix R for the SES's and DES's of Figures 3 and 4.

5 An Illustrative 2D Example With $k = 2$

Before proceeding with the details of the algorithm, we will go over an illustrative 2D example with $k = 2$ rounds and XY-routing in each round. Suppose that the faulty mesh is the one shown in Figure 2 and that we have found the SES partition $\Sigma = \{S_1, \ldots, S_9\}$ and DES partition $\Delta = \{D_1, \ldots, D_7\}$ shown in Figures 3 and 4.

The next step is to determine for each SES S_i and each DES D_j whether all nodes in S_i can reach all nodes in D_j in two rounds. For example, Figure 7 shows a two-XY-round spanning tree from S_8, which implies that DES's D_2 and D_6 are not reachable from S_8 in two rounds. Similarly, Figure 8 shows a two-XY-round spanning tree from S_3, which implies that DES D_5 is not reachable from S_3 in two rounds. However, constructing even one spanning tree takes time $\sim N$.

To get a running time that depends on the number of SES's and DES's but not on N, we use a different approach. Let R be the 9×7 matrix such that $R(i, j) = 1$ if every node of S_i can reach every node of D_j in one XY-routing round; otherwise $R(i, j) = 0$. Suppose that we have also found the representatives shown in Figures 5 and 6. Let $rep(S_i)$ and $rep(D_j)$ denote the representative of S_i and D_j, respectively. By Lemma 4.1, $R(i, j) = 1$ iff $rep(S_i)$ can reach $rep(D_j)$ in one round. For this example, R is shown in Table 1.

We now compute $R^{(2)}$ where $R^{(2)}(i, j) = 1$ iff every node of S_i can reach every node of D_j in two XY-routing rounds. Define the 7×9 intersection matrix I by $I(j', i') = 1$ iff $D_{j'} \cap S_{i'} \neq \emptyset$.[3] (The next two lemmas, 5.1 and 5.2, are stated and proved for the case $k = 2$, where $|\Sigma|$ and $|\Delta|$ can be arbitrary. They are fully generalized in the next section.)

Lemma 5.1 $R^{(2)} = RIR$.

Proof. We show for each i, j that the entry $R^{(2)}(i, j)$ is computed correctly by $R^{(2)} = RIR$. Let $v \in S_i$ and $w \in D_j$. Now v can 2-reach w iff there is a node v' such that v can 1-reach v' and v'

[3] It happens that if $d = 2$ and the partitions are an SEC and DEC partition, as in this example, then $R = I^T$; however, it is not true that $R = I^T$ in general.

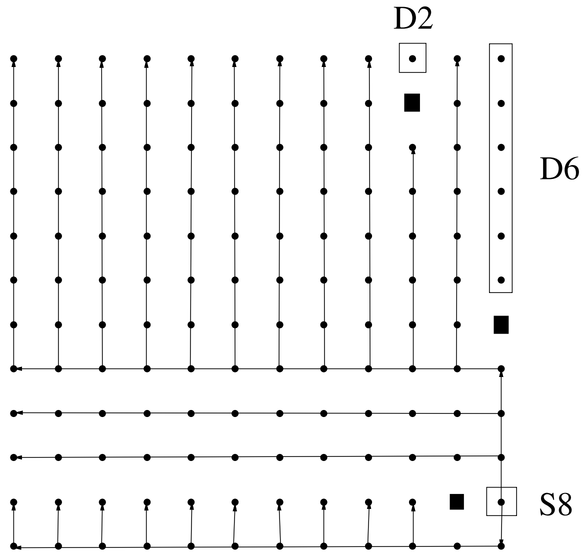

Figure 7: An example of a two-round spanning tree from SES S_8 showing DES's D_2 and D_6 are not reachable from S_8 in two rounds.

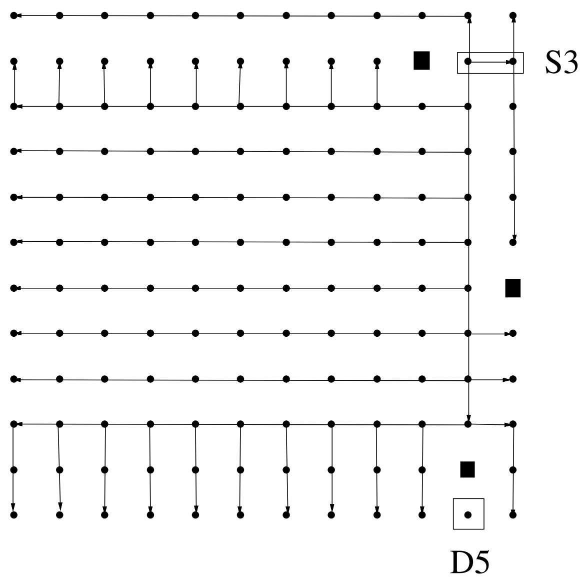

Figure 8: An example of a two-round spanning tree from SES S_3 showing DES D_5 is not reachable from S_3 in two rounds.

	D_1	D_2	D_3	D_4	D_5	D_6	D_7
S_1	1	1	1	1	1	1	1
S_2	1	1	1	1	1	1	1
S_3	1	1	1	1	0	1	1
S_4	1	1	1	1	1	1	1
S_5	1	1	1	1	1	1	1
S_6	1	1	1	1	1	1	1
S_7	1	1	1	1	1	1	1
S_8	1	0	1	1	1	0	1
S_9	1	1	1	1	1	1	1

Table 2: The matrix $R^{(2)}$ for the SES's and DES's of Figures 3 and 4.

can 1-reach w. By definition of R, v can 1-reach v' iff $R(i, j') = 1$ where $D_{j'}$ is the DES to which v' belongs; and v' can 1-reach w iff $R(i', j) = 1$ where $S_{i'}$ is the SES to which v' belongs. It follows that v can 2-reach w iff there are an SES $S_{i'}$ and a DES $D_{j'}$ such that $D_{j'} \cap S_{i'} \neq \emptyset$, $R(i, j') = 1$ and $R(i', j) = 1$. It is then easy to see by the definition of Boolean matrix multiplication that $R^{(2)} = RIR$ computes $R^{(2)}(i, j)$ correctly. $\square$

For this example, $R^{(2)}$ is shown in Table 2.

We state the next observation as a lemma because it will be used later when proving correctness of the general algorithm.

Lemma 5.2 *Λ is a lamb set if and only if, for all (i, j), if $R^{(2)}(i, j) = 0$ then either $S_i \subseteq \Lambda$ or $D_j \subseteq \Lambda$.*

Proof. (*if*) Suppose that Λ is not a lamb set. Then there would be $v, w \in V = \mathit{nodes}(M) - (\Lambda \cup F_N)$ such that v cannot 2-reach w. Let (i, j) be such that $v \in S_i$ and $w \in D_j$, so $R^{(2)}(i, j) = 0$. Now $R^{(2)}(i, j) = 0$ implies that either $S_i \subseteq \Lambda$ or $D_j \subseteq \Lambda$. So either $v \in \Lambda$ or $w \in \Lambda$, and this contradicts either $v \in V$ or $w \in V$, respectively.

(*only if*) Suppose that there exists (i, j) such that $R^{(2)}(i, j) = 0$ and neither $S_i \subseteq \Lambda$ nor $D_j \subseteq \Lambda$. Then there would be a $v \in S_i$ and $w \in D_j$ with $v, w \notin \Lambda$, so $v, w \in V$. This contradicts the definition of a survivor set V because v cannot 2-reach w. $\square$

Figure 9 shows all SES's S_i such that $\exists j\, R^{(2)}(i, j) = 0$ and all DES's D_j such that $\exists i\, R^{(2)}(i, j) = 0$. By Lemma 5.2, these are candidates for inclusion in a lamb set.

Our goal now is to put certain SES's and DES's into Λ to ensure that if $R^{(2)}(i, j) = 0$ then either S_i or D_j is put into Λ, and attempt to minimize the size of Λ. We reduce this problem to solving the weighted vertex cover (WVC) problem on a bipartite graph.[4] For an undirected graph, we let (u, u') denote the edge between vertices u and u'.

[4] A graph is *bipartite* if its vertex set can be partitioned into two subsets such that there are no edges between vertices in the same subset.

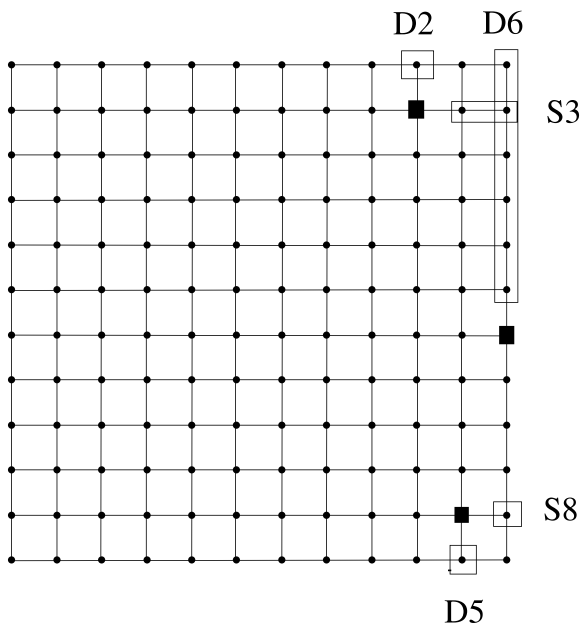

Figure 9: The SES's and DES's that are candidates for inclusion in a lamb set.

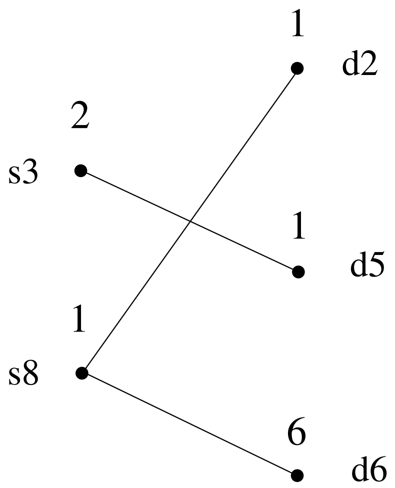

Figure 10: The weighted bipartite graph with edges indicating that S_3 cannot 2-reach D_5 and that S_8 cannot 2-reach D_2 and D_6. Weights are shown above the nodes.

Definition 5.1 Let G be an undirected graph with vertices U, where each vertex u is assigned a positive numerical weight $\omega(u)$. A *vertex cover* of G is a set $C \subseteq U$ such that, for every edge (u, u') of G, either $u \in C$ or $u' \in C$. The weight of C, denoted $\omega(C)$, is the sum of the weights of the vertices in C. The *weighted vertex cover* (WVC) *problem* is: Given a vertex-weighted graph G, find a vertex cover of minimum weight. The *vertex cover* (VC) *problem* is to find a vertex cover whose size $|C|$ is minimized (equivalently, VC is WVC where all weights equal 1).

In our example, the relevant SES's are S_3 and S_8 (the rows in which $R^{(2)}$ has a zero) and the relevant DES's are D_2, D_5, and D_6 (the columns in which $R^{(2)}$ has a zero). The vertices of the bipartite graph are s_3, s_8, d_2, d_5, d_6 corresponding to the relevant SES's and DES's S_3, S_8, D_2, D_5, D_6, respectively. The graph has an edge (s_i, d_j) for each (i, j) such that $R^{(2)}(i, j) = 0$. The weight of each vertex is the size of its corresponding SES or DES. The graph so obtained is shown in Figure 10.

In general, to reduce the lamb problem to the WVC problem on a weighted bipartite graph G: for every SES S_i there is a vertex s_i with $\omega(s_i) = |S_i|$, and for every DES D_j there is a vertex d_j with $\omega(d_j) = |D_j|$. There is an edge between s_i and d_j iff $R^{(2)}(i, j) = 0$. We now solve the WVC problem for G. It is known that there is a polynomial-time algorithm for solving WVC optimally on bipartite graphs [14]. (However, solving VC optimally, and therefore WVC, is an NP-hard problem for general graphs [12].) Let C be be the vertex cover found and define Λ to be the union of the SES's S_i such that $s_i \in C$ and the DES's D_j such that $d_j \in C$. By definition of a vertex cover, for every (i, j) such that $R^{(2)}(i, j) = 0$, either $S_i \subseteq \Lambda$ or $D_j \subseteq \Lambda$. So by Lemma 5.2(if), Λ is a lamb set. In the example, the minimum weight vertex cover is $\{s_8, d_5\}$ and it has weight 2. Therefore, $\Lambda = S_8 \cup D_5 = \{(11, 10), (10, 11)\}$.

It remains to consider the issue of whether this method minimizes $|\Lambda|$. In the example it is easy to see that it does. One could suspect that it might not in general because we have minimized the sum of the sizes of the SES's and DES's that are included in Λ; this is different than minimizing the size of Λ because the same node can be in both a chosen SES and a chosen DES. After describing the algorithm in more detail and generality, we prove that the size of the lamb set found is at most twice optimal; that is, it is a 2-approximation algorithm. We also give an adversarial example where the algorithm finds a Λ whose size is almost twice optimal. Also given is a reduction of the lamb problem to the WVC problem on a general graph; here a minimum weight vertex cover yields a lamb set of optimally minimum size, but solving WVC optimally on general graphs is NP-hard.

6 Algorithms

Algorithms for general d and k are now given. As shown by the illustrative example, an algorithm for the lamb minimization problem has the following main steps, given a mesh M, a fault set F, a number k of rounds, and a k-round ordering $\vec{\pi} = (\pi_1, \dots, \pi_k)$.

1. For each 1-round ordering π_t that appears in $(\pi_1, \dots, \pi_k)$, find an SES partition Σ_t and a DES partition Δ_t of small size, and find a representative node for each SES and DES in the partitions.

2. For each SES representative node v in Σ_1 and each DES representative node w in Δ_k, determine whether v can $(k, F, \vec{\pi})$-reach w.

3. Reduce the lamb problem to a WVC problem and solve the WVC problem.

In the following subsections, each of these steps is considered in detail.

6.1 Finding small SES and DES partitions

We use the following notation to describe a "rectangular" subset of the nodes of a mesh $M = M_d(n_1, \ldots, n_d)$. Each of the coordinates j can be one of the following three types of objects: (1) $*$, meaning that any value v_j with $0 \le v_j < n_j$ can appear in that coordinate; (2) an *interval* $[\ell_j, r_j]$ where $0 \le \ell_j \le r_j < n_j$, meaning that any value v_j with $\ell_j \le v_j \le r_j$ can appear; (3) a *constant* c_j where $0 \le c_j < n_j$, meaning that only the value c_j can appear. For example, $(*, [\ell_2, r_2], c_3)$ is the set of nodes (v_1, v_2, v_3) such that $0 \le v_1 < n_1$, $\ell_2 \le v_2 \le r_2$, and $v_3 = c_3$. Of course, a constant and a $*$ are special cases of an interval, but it will be useful to distinguish among the three.

By renaming the dimensions, we assume without loss of generality that the 1-round ordering π is the ascending ordering $(1, 2, \ldots, d)$. In this case, it happens that the algorithm finds SES's of the form $S = (*, \ldots, *, [\ell_j, r_j], c_{j+1}, \ldots, c_d)$ and DES's of the form $D = (c_1, \ldots, c_{j-1}, [\ell_j, r_j], *, \ldots, *)$. For simplicity, we take the representatives to be

$$\textit{rep}(S) = (0, \ldots, 0, \ell_j, c_{j+1}, \ldots, c_d) \quad \text{and} \quad \textit{rep}(D) = (c_1, \ldots, c_{j-1}, \ell_j, 0, \ldots, 0).$$

Let $M = M_d(n_1, \ldots, n_{d-1}, n_d)$ and $M' = M_{d-1}(n_1, \ldots, n_{d-1})$. For $S' \subseteq \textit{nodes}(M')$, define

$$S' \cdot c = \{\, (v_1, \ldots, v_{d-1}, c) \mid (v_1, \ldots, v_{d-1}) \in S' \,\}.$$

For $S \subseteq \textit{nodes}(M)$, define

$$S/c = \{\, (v_1, \ldots, v_{d-1}) \mid (v_1, \ldots, v_{d-1}, c) \in S \,\}.$$

Similarly, for $F_L \subseteq \textit{links}(M)$, F_L/c is the set of links $\langle u, v\rangle$ of M' such that $\langle u \cdot c, v \cdot c\rangle$ belongs to F_L. If $F_L = (F_N, F_L)$ is a fault set, then $F/c = (F_N/c, F_L/c)$. These notations are defined for the rectangular abbreviations in the obvious way, for example, $(*, \ldots, *, [\ell_j, r_j], c_{j+1}, \ldots, c_{d-1}) \cdot c = (*, \ldots, *, [\ell_j, r_j], c_{j+1}, \ldots, c_{d-1}, c)$.

We first describe an algorithm for finding an SES partition of size at most $(2d-1)f + 1$, given a d-dimensional mesh M, a fault set F, and a 1-round ordering π. We will then state the minor differences needed to obtain an algorithm for finding a DES partition with the same upper bound on size. It is convenient to describe the algorithm recursively, although this may not be the most efficient implementation in practice. Because we assume that the ordering is $(1, 2, \ldots, d)$, the recursive algorithm can consider the coordinates in decreasing order to simplify the presentation. For a general ordering $\pi = (j_1, \ldots, j_d)$, it will be obvious how to modify the algorithm to consider the coordinates in the reverse order given by π.

The next lemma says that if we have an SES S' for a $(d-1)$-dimensional mesh M', then $S' \cdot c$ is an SES for every d-dimensional extension of M'.

Lemma 6.1 *Let $M' = M_{d-1}(n_1, \ldots, n_{d-1})$ and $M = M_d(n_1, \ldots, n_{d-1}, n_d)$. Let F be a fault set for M, let $0 \le c < n_d$, and let $F' = F/c$. Let S' be an SES for M' for the fault set F' and the ordering $(1, 2, \ldots, d-1)$. Then $S = S' \cdot c$ is an SES for M for the fault set F and the ordering $(1, 2, \ldots, d)$.*

Proof. We must show that, for all $v, v' \in S = S' \cdot c$ and all $w \in \mathit{nodes}(M)$, if v can 1-reach w then v' can 1-reach w. If the last coordinate of w is c then we are done, because S' is an SES for M', so it also an SES for $\mathit{nodes}(M') \cdot c$. So assume that the last coordinate of w is not c. Let $v = u_0, u_1, \ldots, u_{d-1}, u_d = w$ be a (fault-free) 1-round path from v to w, where the path from u_{j-1} to u_j is along the j-th coordinate. Because c is the last coordinate of each node $u_0, \ldots, u_{d-1}$ and because v/c and v'/c belong to the same SES S', there must be a path $u'_0, u'_1, \ldots, u'_{d-2}, u_{d-1}$ where $v' = u'_0$. Therefore $v' = u'_0, u'_1, \ldots, u'_{d-2}, u_{d-1}, u_d = w$ is a 1-round path from v' to w. □

The analogue of Lemma 6.1 for DES's is stated next. It can be viewed as a simple corollary of Lemma 6.1, because a set of nodes is a DES for the ordering $(1, 2, \ldots, d)$ iff it is an SES for the ordering $(d, \ldots, 2, 1)$. Define the notations $c \cdot S$ and c/S similarly to $S \cdot c$ and S/c, respectively, except that the coordinate fixed to c is the first.

Lemma 6.2 *Let $M' = M_{d-1}(n_2, \ldots, n_d)$ and $M = M_d(n_1, \ldots, n_{d-1}, n_d)$. Let F be a fault set for M, let $0 \le c < n_1$, and let $F' = c/F$. Let D' be a DES for M' for the fault set F' and the ordering $(2, 3, \ldots, d)$. Then $D = c \cdot D'$ is a DES for M for the fault set F and the ordering $(1, 2, \ldots, d)$.*

Lemma 6.3 *Let $M = M_d(n_1, \ldots, n_d)$. If $S = (*, *, \ldots, *, [\ell, r])$ where $0 \le \ell \le r < n_d$ and S contains no (node or link) faults, then S is an SES for M.*

Proof. Let $v, v' \in S$ and $w \in \mathit{nodes}(M)$ be such that v can reach w. Because S is fault-free, every two nodes of S can reach each other. In particular, if $w \in S$ then v' can reach w. So assume $w \notin S$. The argument that v' can reach w is identical to the argument in Lemma 6.1. The path from v to w has the form $v = u_0, u_1, \ldots, u_{d-1}, u_d = w$ where $u_0, \ldots, u_{d-1} \in S$. There is a path from v' to w because there is a path from v' to u_{d-1}. □

Pseudocode for the algorithm is shown in Figure 11. We illustrate how this algorithm, when given the faulty 2D mesh in Figure 2, finds the SES partition shown in Figure 3. First the top level finds $H = \{1, 6, 10\}$. For $c = 1$, $F_N/1 = \{(9)\}$, and the recursive call returns $\{([0, 8]), ([10, 11])\}$, a partition of $[0, 11] - \{9\}$ into maximal intervals that do not contain 9. Thus, the top level adds $S_2 = ([0, 8], 1)$ and $S_3 = ([10, 11], 1)$ to Σ. For $c = 6$, $F_N/6 = \{(11)\}$, and the recursive call returns $\{([0, 10])\}$. Thus, the top level adds $S_5 = ([0, 10], 6)$ to Σ. For $c = 10$, $F_N/10 = \{(10)\}$, and the recursive call returns $\{([0, 9]), ([11, 11])\}$. Thus, the top level adds $S_7 = ([0, 9], 10)$ and $S_8 = ([11, 11], 10)$. This completes step 2(b). The partition $\{[0, 0], [2, 5], [7, 9], [11, 11]\}$ is found in step 2(c), and the SES's $S_1 = (*, [0, 0])$, $S_4 = (*, [2, 5])$, $S_6 = (*, [7, 9])$, and $S_9 = (*, [11, 11])$ are added in step 2(d).[5]

The algorithm Find-DES-Partition is very similar. The only difference is that the recursion considers the coordinates from left to right instead of from right to left.

[5]Thus, the order in which the algorithm produces the SES's is not the same as our numbering of the SES's.

Algorithm Find-SES-Partition
Input: $(d, n_1, \ldots, n_d)$ and a fault set $F = (F_N, F_L)$.
Output: An SES partition $\Sigma = \{S_1, \ldots, S_m\}$ for $M_d(n_1, \ldots, n_d)$, F, and the 1-round ordering $(1, 2, \ldots, d)$, where each $S \in \Sigma$ has the form $S = (*, \ldots, *, [\ell_j, r_j], c_{j+1}, \ldots, c_d)$ for some $1 \le j \le d$.

1. If $d = 1$:
   (a) Partition $[0, n_1 - 1] - F_N$ into maximal intervals $[\ell_1, r_1], \ldots, [\ell_m, r_m]$ such that each interval contains no node fault nor link fault.
   (b) Return $\Sigma = \{[\ell_1, r_1], \ldots, [\ell_m, r_m]\}$.
2. If $d > 1$:
   (a) Initialize $\Sigma = \emptyset$ and let H be the set of values c such that $(*, \ldots, *, c)$ contains either a node fault or a link fault.
   (b) For each $c \in H$:
      i. Call Find-SES-Partition on input $(d - 1, n_1, \ldots, n_{d-1})$ and F/c, and let Σ' be the SES partition returned.
      ii. For each SES $S' = (*, \ldots, *, [\ell_j, r_j], c_{j+1}, \ldots, c_{d-1})$ in Σ', add $S = S' \cdot c = (*, \ldots, *, [\ell_j, r_j], c_{j+1}, \ldots, c_{d-1}, c)$ to Σ.
   (c) Partition $[0, n_d - 1] - H$ into maximal intervals $[\ell, r]$ such that $(*, \ldots, *, [\ell, r])$ contains no node fault nor link fault.
   (d) For each such maximal interval $[\ell, r]$, add $(*, \ldots, *, [\ell, r])$ to Σ.
   (e) Return Σ.

Figure 11: Pseudocode of an algorithm that finds an SES partition.

Theorem 6.4 *Algorithm Find-SES-Partition (resp., Find-DES-Partition) returns an SES partition (resp., a DES partition) of size at most*

$$\sum_{j=2}^{d} \min\{2f, n_d n_{d-1} \cdots n_{j+1}(n_j - 1)\} + f + 1.^{6} \tag{1}$$

This bound is at most $(2d - 1)f + 1$.

Proof. We give the proof for Find-SES-Partition. The proof for Find-DES-Partition is essentially identical. First we show that the output $\Sigma = \{S_1, \ldots, S_m\}$ is an SES-partition, that is: (i) each S_i is an SES, (ii) the sets $S_1, \ldots, S_m$ are pairwise disjoint, and (iii) their union is the set of good nodes. The proof is by induction on d.

[6] By convention, if $j = d$ then $n_d n_{d-1} \cdots n_{j+1}(n_j - 1) = n_d - 1$.

If $d = 1$ it is obvious that (i), (ii), (iii) hold. Assume that these hold for $d - 1$, and let the algorithm be called with dimension input d. By the induction assumption, (i), (ii), (iii) hold for each recursive call of the algorithm. Because (i) holds for each recursive call, it follows from Lemma 6.1 that each set $S = S' \cdot c$ added in step 2(b) is an SES. It follows from Lemma 6.3 that each set added in step 2(d) is an SES.

To prove (ii), write $\Sigma = \bigcup_{c \in H} \Sigma_c \cup \Sigma'$ where Σ_c is the set of SES's added to Σ in step 2(b) for last coordinate $= c$, and Σ' is the set of SES's added in step 2(d). Let $S_1, S_2 \in \Sigma$ with $S_1 \neq S_2$. If $S_1, S_2 \in \Sigma_c$, then $S_1 \cap S_2 = \emptyset$ by the induction assumption. If $S_1 \in \Sigma_{c_1}$ and $S_2 \in \Sigma_{c_2}$ for $c_1 \neq c_2$, then $S_1 \cap S_2 = \emptyset$ because all nodes in S_i have last coordinate $= c_i$. If $S_1 \in \Sigma_c$ for some $c \in H$ and $S_2 \in \Sigma'$, then $S_1 \cap S_2 = \emptyset$ because the partition found in step 2(c) contains no member of H. If $S_1, S_2 \in \Sigma'$, then $S_1 \cap S_2 = \emptyset$ because $S_i = (*, \ldots, *, [\ell_i, r_i])$ where $[\ell_1, r_1] \cap [\ell_2, r_2] = \emptyset$.

To prove (iii), note that the SES's added to Σ in step 2(b) (resp., step 2(d)) contain all good nodes whose last coordinate belongs to (resp., does not belong to) H.

We now verify the upper bound (1) on $|\Sigma|$. Because the bound is in terms of f, the total number of node and link faults, the proof is simplified by first replacing each link fault by a node fault at either end of the faulty link, without increasing the total number of faults. So in the proof, we only have to consider node faults. Fixing $n_1, \ldots, n_d$, let $B(d, f)$ be the expression (1). Let $E(d, f)$ be the maximum number of SES's returned by the algorithm on input $(d, n_1, \ldots, n_d)$ and F containing f faults. We prove the bound $E(d, f) \leq B(d, f)$ by induction on d. For $d = 1$, obviously $E(1, f) \leq f + 1$, and this equals the bound (1) because the sum in this case is zero. Let $d > 1$ and assume that $E(d - 1, f) \leq B(d - 1, f)$ for all f.

The number of SES's returned by the algorithm when the input is $(d, n_1, \ldots, n_d)$ and there are f faults is the number m_1 added during step 2(b) plus the number m_2 added during step 2(d). Let $t = |H|$ and $H = \{h_1, \ldots, h_t\}$. Note that $t \leq f$ and $t \leq n_d$. Now m_1 is the total number of SES's returned by the recursive calls in step 2(b), and m_2 is at most $\min\{t + 1, n_d - t\}$. For $1 \leq i \leq t$, let f_i be the number of faults in $(*, *, \ldots, *, h_i)$. Clearly, $f_1 + f_2 + \cdots + f_t \leq f$. By the induction assumption, when the algorithm is called recursively on input $M_{d-1}(n_1, \ldots, n_{d-1})$ and the fault set F/h_i, it returns at most $B(d - 1, f_i)$ SES's. We therefore have

$$
\begin{aligned}
E(d, f) &\leq \sum_{i=1}^{t} B(d - 1, f_i) + \min\{t + 1, n_d - t\} \\
&= \left(\sum_{i=1}^{t} \sum_{j=2}^{d-1} \min\{2f_i, n_{d-1} \cdots n_{j+1}(n_j - 1)\} + f_i + 1 \right) + \min\{t + 1, n_d - t\} \\
&\leq \left(\sum_{j=2}^{d-1} \sum_{i=1}^{t} \min\{2f_i, n_{d-1} \cdots n_{j+1}(n_j - 1)\} \right) + f + t + \min\{t + 1, n_d - t\} \\
&\leq \left(\sum_{j=2}^{d-1} \min\{2f, t n_{d-1} \cdots n_{j+1}(n_j - 1)\} \right) + f + \min\{2t + 1, n_d\}.
\end{aligned}
$$

Because $t \le n_d$, the last expression above is at most

$$B(d, f) - (\min\{2f, n_d - 1\} + 1) + \min\{2t + 1, n_d\}.$$

Thus to complete the proof, it suffices to show that

$$\min\{2t + 1, n_d\} \le \min\{2f, n_d - 1\} + 1.$$

The proof of this inequality has two cases, depending on whether $2f$ or $n_d - 1$ achieves the min in the RHS. If $2f \le n_d - 1$, then (using $t \le f$) $\min\{2t+1, n_d\} \le 2t+1 \le 2f+1 = \min\{2f, n_d - 1\}+1$. If $n_d - 1 \le 2f$, then $\min\{2t + 1, n_d\} \le n_d = (n_d - 1) + 1$.

To prove the bound $B(d, f) \le (2d - 1)f + 1$, note that each of the $(d - 1)$ min terms is at most $2f$. Therefore $B(d, f) \le 2f(d - 1) + f + 1 = (2d - 1)f + 1$. □

On $M_d(n)$ with n odd and $f \le n^{d-1}(n - 1)/2$, the bound of Theorem 6.4 on the number of SES's returned by the algorithm is tight in the worst case, in the following sense.

Proposition 6.5 *Let $d \ge 2$, $n \ge 3$ and odd, and $f \le n^{d-1}(n - 1)/2$. There is a fault set (F_N, F_L) with $|F_N| = f$ and $F_L = \emptyset$ such that, if Find-SES-partition is given input $(d, n, \dots, n)$ and F, then it returns an SES partition of size $B(d, f) = \sum_{j=2}^{d} \min\{2f, (n - 1)n^{d-j}\} + f + 1$. The same conclusion holds for some fault set with $F_N = \emptyset$ and $|F_L| = f$.*

Proof. The proof is given in the Appendix. □

The bound $B(d, f) \le (2d - 1)f + 1$ is tight, even for SEC and DEC partitions, in the case $M = M_d(n)$, $f \le (n - 1)/2$, and n odd: there is an assignment of f node faults to $M_d(n)$ such that the number of SEC's and DEC's are both $(2d - 1)f + 1$. To achieve this, place one fault at node $(i, i, \dots, i)$ for each odd i between 1 and $2f - 1$.

We now consider the time complexity of Find-SES-Partition. Let $T(d, f)$ be the time of the algorithm for d dimensions and f faults. The straightforward recursive implementation of the algorithm gives $T(1, f) = O(f)$ and the recurrence

$$T(d, f) \le \sum_{c \in H} T(d - 1, f_c) + O(d^2 f) \quad \text{where} \quad \sum_{c \in H} f_c \le f. \tag{2}$$

The term $O(d^2 f)$ is the total (summed over all $c \in H$) time spent in steps 2(b)(i) and (ii) to prepare the input and handle the output of the recursive calls. This bound holds because there are a total of $O(df)$ SES's S' returned (by Theorem 6.4), and the rectangular abbreviation of each one has size $O(d)$. The term $O(d^2 f)$ also includes the time for steps 2(a), (c), (d) and (e). The recurrence (2) implies that $T(d, f) = O(d^3 f)$. By departing from the pure recursive implementation, there is a version of the algorithm that runs in time $O(d^2 f)$. This is an asymptotically optimal worst case bound for finding an SES or DES partition (as we have defined the problem) because in the worst case the output contains $\Omega(df)$ set abbreviations, each of length d. Because our primary interest is the case where d is constant, we do not give the details of the alternate implementation. Two hints are: (i) at the start of the algorithm sort the faults, where the significance of the coordinates increases from left to right (this can be done in time $O(df)$ by radix sort); and (ii) have calls of the algorithm below the top level add SES's to Σ directly, rather than passing them up the calling program.

Algorithm Find-Reachability
Input: A fault set F, a k-round ordering $\vec{\pi} = (\pi_1, \ldots \pi_k)$, and, for each $1 \le t \le k$, $1 \le i \le p_t$, and $1 \le j \le q_t$, a rectangular abbreviation of an SES $S_{t,i}$ for π_t, a rectangular abbreviation of a DES $D_{t,j}$ for π_t, and representatives $s_{t,i} \in S_{t,i}$ and $d_{t,j} \in D_{t,j}$.
Output: The matrix $R^{(k)}$.

1. For $1 \le t \le k$, compute the entries of the p_t by q_t (1-round reachability) matrix R_t, where $R_t(i,j) = 1$ iff $s_{t,i}$ can (F, π_t)-reach $d_{t,j}$.
2. For $1 \le t \le k-1$, compute the entries of the q_t by p_{t+1} (intersection) matrix I_t, where $I_t(j,i) = 1$ iff $D_{t,j} \cap S_{t+1,i} \ne \emptyset$.
3. $R^{(k)} = R_1 I_1 R_2 I_2 \cdots R_{k-1} I_{k-1} R_k$.

Figure 12: Pseudocode of an algorithm that computes $R^{(k)}$.

6.2 Determining reachability

Recall that we are given $M = M_d(n_1, \ldots, n_d)$, k, F, and $\vec{\pi} = (\pi_1, \ldots, \pi_k)$. For each $1 \le t \le k$, let Σ_t and Δ_t be an SES partition and DES partition for π_t, respectively. Let $p_t = |\Sigma_t|$ and $q_t = |\Delta_t|$. Let $s_{t,1}, \ldots, s_{t,p_t}$ (resp., $d_{t,1}, \ldots, d_{t,q_t}$) be representatives for the SES's (resp., DES's) in Σ_t (resp., Δ_t). The next goal is to determine, for each (i,j) with $1 \le i \le p_1$ and $1 \le j \le q_k$, whether $s_{1,i}$ can $(k, F, \vec{\pi})$-reach $d_{k,j}$; define the k-round Boolean reachability matrix $R^{(k)}$ by $R^{(k)}(i,j) = 1$ iff $s_{1,i}$ can $(k, F, \vec{\pi})$-reach $d_{k,j}$. So $R^{(k)}$ is a p_1 by q_k matrix. Pseudocode for an algorithm that computes $R^{(k)}$ is shown in Figure 12. It is a straightforward extension of the proof of Lemma 5.1 that the algorithm computes $R^{(k)}$ correctly. Of course, to save memory, the matrices $R_1, I_1, R_2, I_2, \ldots, I_{k-1}, R_k$ could be computed only when needed in the product computation (step 3), and then erased. Also, if the same ordering π and the same SES and DES partitions are used at each round, then the algorithm can be simplified because $R_1 = R_2 = \cdots = R_k$ and $I_1 = I_2 = \cdots I_{k-1}$.

The intermediate matrices $R^{(r)} = R_1 I_1 R_2 I_2 \cdots I_{r-1} R_r$, for $1 \le r < k$, are useful for choosing routes: $R^{(r)}(i,j) = 1$ means that all nodes in $S_{1,i}$ can reach all nodes in $D_{r,j}$ using only r rounds.

The time complexity of this algorithm is now bounded. We use (Theorem 6.4) that $p_t, q_t = O(df)$ for all t. Step 1 involves $O(kd^2f^2)$ 1-round node-to-node reachability computations, where each takes time $O(df)$, for a total time of $O(kd^3f^3)$. Step 2 involves $O(kd^2f^2)$ intersection tests, "$D_{t,j} \cap S_{t+1,i} = \emptyset$?". Each test can be done in time $O(d)$ by coordinatewise comparison of the rectangular abbreviations for $D_{t,j}$ and $S_{t+1,i}$, for a total time of $O(kd^3f^2)$. Step 3 involves $O(k)$ multiplications of $O(df)$ by $O(df)$ matrices, for a total time $O(kd^3f^3)$. The total time of the algorithm is therefore $O(kd^3f^3)$.[7] Viewing k and d as constants, the time is $O(f^3)$.

In our simulations involving random faults, we have observed that intersection matrices are typically sparse. Multiplying two m by m matrices, where one of them has s nonzero entries, can

[7]For f sufficiently large compared to N, it will be more efficient to compute $R^{(k)}$ by computing the k-round spanning tree from each SES representative node, using time $O(d^2fN)$ instead of $O(kd^3f^3)$.

be done in time $O(sm)$ rather than $O(m^3)$, which is an improvement if $s \ll m^2$. For example, on the $32 \times 32 \times 32$ mesh with 3% random node faults, the average density of 1's in the intersection matrix I_1 is 0.00987. Thus, the computation of $R_1 I_1$ can be made about 100 times faster by using sparsity. However, the density of $R_1 I_1$ is 0.668, and the density of R_2 (which equals R_1 in this case) is 0.175, so sparsity does not help as much in computing $R_1 I_1$ times R_2. To speed up this computation we used bitwise Boolean operation on 32-bit words, rather than single-bit operations.

6.3 Reduction to weighted vertex cover

The final step of the lamb-finding algorithm is to reduce it to a WVC problem, and then solve the WVC problem. We describe two ways of doing this. The first reduction (Section 6.3.1) constructs a weighted bipartite graph G having at most $p_1 + q_k = O(df)$ vertices. Solving WVC optimally on a bipartite graph having b vertices is reducible to solving a max-flow problem on a flow network having $b + 2$ vertices [14], and the max-flow problem can be solved in time $O(b^3)$ [9, 14]. Thus, solving WVC on G can be done in time $O((df)^3)$. We show that the size of the lamb set found from a minimum weight vertex cover, as described in Section 5, is always within twice optimal. Also given is an adversarial example, showing that the algorithm can be nonoptimal by almost a factor of 2. The second reduction (Section 6.3.2) constructs a weighted general graph having at most $p_1 q_k = O((df)^2)$ nodes. Here the lamb set found from a minimum weight vertex cover has optimally small size. Although solving WVC optimally (and even VC) on a general graph is an NP-hard problem, there are 2-approximation algorithms that use time linear in the number of edges of the graph, for example, [3]; see also [16, §3] for more information on approximation algorithms for WVC. Because the graph has $O((df)^2)$ vertices, a worst-case upper bound on the number of edges is $O((df)^4)$. Not having tried to place a better upper bound on the number of edges, the first approach has a better theoretical time bound.

6.3.1 On a bipartite graph – an efficient 2-approximation algorithm

The only information needed from previous steps is the matrix $R^{(k)}$, rectangular abbreviations $S_1, \ldots, S_{p_1}$ for the SES's in Σ_1, and rectangular abbreviations $D_1, \ldots, D_{q_k}$ for the DES's in Δ_k. Let $p = p_1$ and $q = q_k$. The method for constructing the weighted bipartite graph and converting a vertex cover to a lamb set has been described in Section 5. The algorithm is shown in Figure 13.

The argument that Λ is a lamb set is essentially identical to the proof of the "if" direction of Lemma 5.2; the only difference is that here we are using the k-round reachability information in $R^{(k)}$ instead of the 2-round reachability information in $R^{(2)}$. (For the same reason, the "only if" direction of Lemma 5.2 also holds in general.)

As noted above, the time complexity of step 2 is $O((df)^3)$, and this dominates the time taken by step 1. The time for step 3 is $O(|\Lambda|)$, where Λ is the lamb set returned, because each node belongs to exactly one SES S_i and one DES D_j, so each node is added to Λ at most twice.

Lemma 6.6 *The lamb set Λ returned by Reduce-WVC(Bipartite) is within twice optimal, formally,* $|\Lambda| \leq 2\lambda(M_d(n_1, \ldots, n_d), k, F, \vec{\pi})$.

Algorithm Reduce-WVC(Bipartite)
Input: SES partition $\Sigma_1 = \{S_1, \ldots, S_p\}$, DES partition $\Delta_k = \{D_1, \ldots, D_q\}$, and matrix $R^{(k)}$.
Output: A lamb set Λ.

1. Let the indices of relevant SES's be $\mathcal{R}_S \stackrel{\text{def}}{=} \{\, i \mid \exists j\, R^{(k)}(i,j) = 0 \,\}$. Similarly, the indices of relevant DES's are $\mathcal{R}_D \stackrel{\text{def}}{=} \{\, j \mid \exists i\, R^{(k)}(i,j) = 0 \,\}$. Construct the weighted bipartite graph G with vertices $\{\, s_i \mid i \in \mathcal{R}_S \,\} \cup \{\, d_j \mid j \in \mathcal{R}_D \,\}$. There is an edge between s_i and d_j iff $R^{(k)}(i,j) = 0$. The weights are $\omega(s_i) = |S_i|$ and $\omega(d_j) = |D_j|$.

2. Find a minimum weight vertex cover for G. Let C be the vertex cover returned.

3. Let Λ be the union of those S_i with $s_i \in C$ and those D_j with $d_j \in C$.

Figure 13: Pseudocode for the algorithm Reduce-WVC(Bipartite).

Proof. Let Λ^* be a lamb set of minimum size. We show that there is a vertex cover C^* of G with $\omega(C^*) \leq 2|\Lambda^*|$. If C is the vertex cover found in step 2, then $\omega(C) \leq \omega(C^*)$ because C has minimum weight. This suffices to prove the lemma because $|\Lambda| \leq \omega(C) \leq \omega(C^*) \leq 2|\Lambda^*|$. Recall that $\Sigma_1 = \{S_1, \ldots, S_p\}$ and $\Delta_k = \{D_1, \ldots, D_q\}$. Let

$$C^* = \{\, s_i \mid S_i \subseteq \Lambda^* \,\} \cup \{\, d_j \mid D_j \subseteq \Lambda^* \,\}.$$

We first show that C^* is a vertex cover of G. Suppose otherwise that there were vertices $s_i, d_j \notin C^*$ that were connected by an edge, so $R^{(k)}(i,j) = 0$. The definition of C^* implies that $S_i \not\subseteq \Lambda^*$ and $D_j \not\subseteq \Lambda^*$, so there are vertices $v \in S_i - \Lambda^*$ and $w \in D_j - \Lambda^*$. This means that v and w belong to the survivor set $V = \mathit{nodes}(M) - (\Lambda \cup F_N)$. But because $R^{(k)}(i,j) = 0$, v cannot $(k, F, \vec{\pi})$-reach w, and this contradicts the definition of a survivor set.

We now show that $\omega(C^*) \leq 2|\Lambda^*|$. We show that the weight of $C_1^* = \{\, s_i \mid S_i \subseteq \Lambda^* \,\}$ is at most $|\Lambda^*|$. This is true because $\omega(s_i) = |S_i|$, and $i \neq i'$ implies $S_i \cap S_{i'} = \emptyset$. The same argument shows that the weight of $C_2^* = \{\, d_j \mid D_j \subseteq \Lambda^* \,\}$ is at most $|\Lambda^*|$. So $\omega(C^*) \leq \omega(C_1^*) + \omega(C_2^*) \leq 2|\Lambda^*|$. □

Putting all this together proves the following Theorem 6.7. Let Lamb1 be the algorithm obtained by first running Find-SES-Partition and Find-DES-Partition for each 1-round ordering π_t ($t = 1, 2, \ldots, k$) in the k-round ordering $\vec{\pi} = (\pi_1, \ldots, \pi_k)$, followed by Find-Reachability, followed by Reduce-WVC(Bipartite). The algorithm Lamb1 is shown in Figure 14; we also give for each step the figure number where pseudocode for that step is described and the time bound we have derived for that step. The algorithm is stated in this three step form for simplicity. To save memory, each SES-partition Σ_t and DES-partition Δ_t for $t = 1, 2, \ldots, k$ can be computed at the point where the reachability matrix R_t is needed in the product computation, step 3, of Find-Reachability (Figure 12). Then Σ_t, Δ_t and R_t are no longer needed and can be erased. Of course, if the same ordering is used in each round ($\pi_1 = \pi_2 = \cdots = \pi_k$) then the SES-partitions Σ_t are identical and the matrices R_t are identical for $1 \leq t \leq k$.

Algorithm Lamb1
Input: $(d, n_1, \ldots, n_d)$, a k-round ordering $\vec{\pi} = (\pi_1, \ldots, \pi_k)$, and a fault set $F = (F_N, F_L)$.
Output: A lamb set Λ.

1. For $t = 1, 2, \ldots, k$: Call Find-SES-Partition and Find-DES-Partition on input $(d, n_1, \ldots, n_d)$, the one-round ordering π_t, and the fault set F (Figure 11; time $O(d^3 f)$ for each t, for a total time $O(kd^3 f)$).
2. Call Find-Reachability (Figure 12; time $O(kd^3 f^3)$).
3. Call Reduce-WVC(Bipartite) (Figure 13; time $O(d^3 f^3 + |\Lambda|)$).

Figure 14: Pseudocode for the algorithm Lamb1.

Theorem 6.7 *Lamb1 is a 2-approximation algorithm for the lamb problem and it has time complexity $O(kd^3 f^3 + |\Lambda|)$. If k, d are constants, then the time is $O(f^3 + |\Lambda|)$.*

Our simulation results for random faults indicate that $|\Lambda| < f^3$ in the cases simulated. Therefore in these cases, the running time is $O(f^3)$. In fact, the results show $|\Lambda| < f$ for the average size of the lamb set on $M_2(32)$ and for both the average and maximum size of the lamb set on $M_3(32)$.

We now give a class of lamb problems on which Lamb1 is nonoptimal by a factor of $2 - o(1)$, showing that Lamb1 is not an r-approximation algorithm if $r < 2$. It is convenient in the examples to let the faults disconnect the mesh into three regions. There is a more complicated class of examples, with the same nonapproximation property, where every node can 3-reach every other node. However, the simpler class of examples adequately explains the reason why Lamb1 can be nonoptimal by a factor of almost 2. For each n of the form $n = 4m + 1$, where m is a positive integer, there is a lamb problem in the class for $M_2(n)$. The faults are at nodes of the form $(*, m)$ and $(*, n - m - 1)$. The example with $m = 2$ and $n = 9$ is shown in Figure 15. The faults divide M into three connected components: a component $C_1 = (*, [0, m - 1])$ of size mn, a component $C_2 = (*, [m + 1, n - m - 2])$ of size $(2m - 1)n$, and a component $C_3 = (*, [n - m, n - 1])$ of size mn. Because any lamb set must contain at least two of C_1, C_2, C_3, the minimum-size lamb set is $C_1 \cup C_3$ and it has size $2mn$. Find-SES-Partition will find $\Sigma = \{S_1, S_2, S_3\}$ where $S_i = C_i$. Find-DES-Partition will find $\Delta = \Delta_1 \cup \Delta_2 \cup \Delta_3$, where $\Delta_i = \{D_{i,1}, \ldots, D_{i,n}\}$ for $i = 1, 2, 3$. The DES's in Δ_i form a partition of S_i, and each $D_{i,j}$ has size m for $i = 1, 3$ or size $2m - 1$ for $i = 2$. The reachability computation will find that S_i can reach $D_{\ell,j}$ iff $i = \ell$. Therefore, in the bipartite graph G, there will be an edge between s_i and $d_{\ell,j}$ iff $i \neq \ell$. It is easy to see that the minimum weight vertex cover of G contains all vertices on one side of the bipartition and none on the other. In either case, $|\Lambda| = (4m - 1)n$. So the ratio of $|\Lambda|$ to the optimum size is $(4m - 1)/(2m) = 2 - \frac{1}{2m}$. This construction uses $2n$ faults. A similar construction for $M_d(n)$ uses $2n^{d-1}$ faults. Examples where f is much smaller relative to n can be constructed by placing one of the examples described above in one corner of a larger mesh.

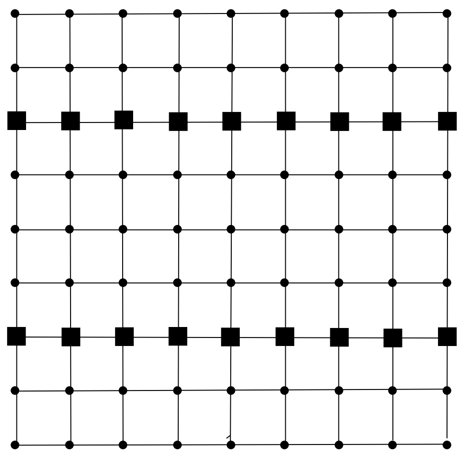

Figure 15: An example on which algorithm Lamb1 is nonoptimal.

6.3.2 On a general graph

For $1 \leq i \leq p$ and $1 \leq j \leq q$, define $I_{i,j} = S_i \cap D_j$. Let $\mathcal{I} = \{I_{i,j}\}$. Note that $\mathcal{I}$ is a partition of the good nodes of M, because both $\{S_1, \ldots, S_p\}$ and $\{D_1, \ldots, D_q\}$ are. There is a vertex of the graph for every pair (i, j) such that $I_{i,j} \neq \emptyset$. Vertices (i, j) and (i', j') have an edge between them if either S_i cannot k-reach $D_{j'}$ or $S_{i'}$ cannot k-reach D_j. The algorithm is shown in Figure 16.

Lemma 6.8 *The Λ returned by Reduce-WVC(General) is a lamb set.*

Proof. Assume for contradiction that there are nodes $v, v' \in \mathit{nodes}(M) - (\Lambda \cup F_N)$ such that v cannot k-reach v'. Let i, j, i', j' be such that $v \in S_i \cap D_j$ and $v' \in S_{i'} \cap D_{j'}$. Therefore, S_i cannot k-reach $D_{j'}$, so $R^{(k)}(i, j') = 0$. This means that there is an edge in G between $u_{i,j}$ and $u_{i',j'}$, so either $u_{i,j} \in C$ or $u_{i',j'} \in C$. In the first case $v \in S_i \cap D_j \subseteq \Lambda$, and in the second case $v' \in S_{i'} \cap D_{j'} \subseteq \Lambda$. This contradicts that $v, v' \notin \Lambda$. □

Let Lamb2 be the algorithm defined like Lamb1, except that Reduce-WVC(General) is used in the last step.

Theorem 6.9 *Assume that an r-approximation algorithm for WVC is used in step 2 of Reduce-WVC(General). Then Lamb2 is an r-approximation algorithm for the lamb problem.*

Proof. The proof is similar to that of Lemma 6.6. Let Λ^* be a lamb set of minimum size. We show that there is a vertex cover C^* of G with $\omega(C^*) \leq |\Lambda^*|$. If C is the vertex cover found in step 2,

Algorithm Reduce-WVC(General)
Input: SES partition $\Sigma_1 = \{S_1, \ldots, S_p\}$, DES partition $\Delta_k = \{D_1, \ldots, D_q\}$, and matrix $R^{(k)}$.
Output: A lamb set Λ.

1. Construct the weighted graph G with vertices $U = \{\, u_{i,j} \mid S_i \cap D_j \neq \emptyset \,\}$. There is an edge between $u_{i,j}$ and $u_{i',j'}$ iff either $R^{(k)}(i, j') = 0$ or $R^{(k)}(i', j) = 0$. The weights are $\omega(u_{i,j}) = |S_i \cap D_j|$. (As in Reduce-WVC(Bipartite), only the vertices with at least one incident edge are relevant, and the other vertices need not appear in the graph.)

2. Find a vertex cover for G. Let C be the vertex cover returned.

3. Let Λ be the union of those $S_i \cap D_j$ over all (i, j) such that $u_{i,j} \in C$.

Figure 16: Pseudocode for the algorithm Reduce-WVC(General).

then $\omega(C) \leq r \cdot \omega(C^*)$ because step 2 uses an r-approximation algorithm for WVC. This suffices to prove the lemma because $|\Lambda| \leq \omega(C) \leq r \cdot \omega(C^*) \leq r|\Lambda^*|$.

Let $C^* = \{\, u_{i,j} \mid S_i \cap D_j \subseteq \Lambda^* \,\}$. We first show that C^* is a vertex cover of G. Suppose otherwise that there were vertices $u_{i,j}, u_{i',j'} \notin C^*$ that were connected by an edge. Then $R^{(k)}(i, j') = 0$ or $R^{(k)}(i', j) = 0$. By the definition of C^*, there are nodes $v \in (S_i \cap D_j) - \Lambda^*$ and $v' \in (S_{i'} \cap D_{j'}) - \Lambda^*$, so v and v' belong to the survivor set $V = \mathit{nodes}(M) - (\Lambda^* \cup F_N)$. If $R^{(k)}(i, j') = 0$, then v cannot k-reach v', or if $R^{(k)}(i', j) = 0$, then v' cannot k-reach v. Either case contradicts the definition of a survivor set. Because $\mathcal{I} = \{S_i \cap D_j\}$ is a partition of the good nodes of M, it is clear that $\omega(C^*) \leq |\Lambda^*|$. □

Theorem 6.9 holds when $r = 1$, that is, if WVC is solved optimally, then the lamb problem is solved optimally. Because WVC can be solved optimally in time exponential in the number of vertices, this shows the following.

Corollary 6.10 *There is a constant c and a polynomial p such that the lamb minimization problem can be solved in time* $O(c^{(df)^2} + p(d, k, f) + |\Lambda|)$

7 Extensions of the Algorithm

Nodes that have not completely failed might have differing utility or value. Therefore, it is useful to associate a value $\mathit{val}(v)$ with each node v, where $\mathit{val}(v)$ is a number between 0 and 1. For example, if a node contains several processors, its value might be the fraction of processors that are good. If a node has, say, only one good processor then it has limited value so it should be a prime candidate for a lamb node, whereas if a node has mostly good processors then we lose more by choosing it as a lamb node. Values can be effortlessly incorporated into our algorithms. When constructing the graph G, the weight of a vertex u is the size of some set of nodes (S_i, D_j, or $S_i \cap D_j$). If nodes have values, the weight of a vertex u that represents the set S of nodes is the sum of the values of the nodes in S. Because the algorithm finds a minimum or approximately minimum weight vertex cover, it will choose sets of nodes having smaller total value to include in the lamb set.

Another straightforward extension permits a set P of nodes to be predetermined as lambs, for example, if it is desired that the new set of lambs be a superset of the existing set of lambs. Before constructing the bipartite graph, remove the nodes in P from every SES and DES, and continue as before, adding the nodes in P to the lamb set at the end.

The algorithms can be applied directly to d-dimensional hypercubes, that is, meshes of the form $M_d(2)$. The methodology and some of the algorithms can be extended to other network topologies. The entire development can easily be generalized to d-dimensional torus graphs, which are similar to d-dimensional meshes, but have in addition the "wrap-around" links between $(v_1, \ldots, v_{j-1}, n_j - 1, v_{j+1}, \ldots, v_d)$ and $(v_1, \ldots, v_{j-1}, 0, v_{j+1}, \ldots, v_d)$ for all j and $v_1, \ldots, v_d$. For other network topologies, the general method for finding a lamb set is still applicable. All that is needed is a set of nodes and an efficiently computable "simple reachability" relation $R(v, w, F)$ which holds iff node v can reach node w by a "simple route" in the presence of fault set F. The algorithms Find-SES-Partition and Find-DES-Partition are designed specifically for mesh-like topologies, so they are not applicable in general. In the worst case, the SEC and DEC partition can be found by explicitly computing the reachability sets (as a source and as a destination) for each node. Then finding a lamb set can be reduced to WVC as before.

8 Simulation Results

The algorithm was implemented and experimental information was collected about how the number of lambs depends on the number of faults and the size and dimensionality of the mesh. In the first experiment, the algorithm was run on the 32×32 2D mesh and the $32 \times 32 \times 32$ 3D mesh, for various numbers of random node faults. For the 2D (3D) mesh, we repeated 1000 trials for each f equal to the percentage $0.5, 1.0, 1.5, 2.0, 2.5, 3.0$ of the number $N = 1024$ ($N = 32768$) of nodes, with new random faults chosen at each trial, and recorded the average and maximum size of the lamb set over all 1000 trials. Figures 17 and 18 show the results for the 2D and 3D mesh, respectively. Comparing the 2D and 3D cases with 3% random faulty nodes: for 2D, the average number of lambs is 9.59, which equals 0.937% of the number 1024 of nodes; for 3D, the average number of lambs is 67.6, which equals 0.206% of the number 32768 of nodes. Another way to measure the efficiency of a lamb set is to consider the "additional damage", that is, the percentage of the number of lambs over the number of faults. For example, for the 2D mesh with 3% faults, the average additional damage is $9.59/31 = 30.9\%$; for the 3D mesh with 3% faults, it is $67.6/983 = 6.88\%$. Figure 19 compares the average additional damage, 2D vs 3D, for other percentages of faults. To compare 2D with 3D on meshes having roughly the same number of nodes, the algorithm was run for 1000 trials on the 181×181 mesh (having $N = 32761$) with the same percentages of faults; the results are shown in Figure 20. As can be seen by comparing Figures 18 and 20, the number of lambs is significantly higher for the 2D mesh. A possible explanation for this is that, say for 3% faults, the number of faults (983) is about equal to the bisection width ($32^2 = 1024$) of $M_3(32)$, whereas 983 is a large multiple of the bisection width (181) of $M_2(181)$. Also, in the case of $M_2(32)$ (Figure 17),

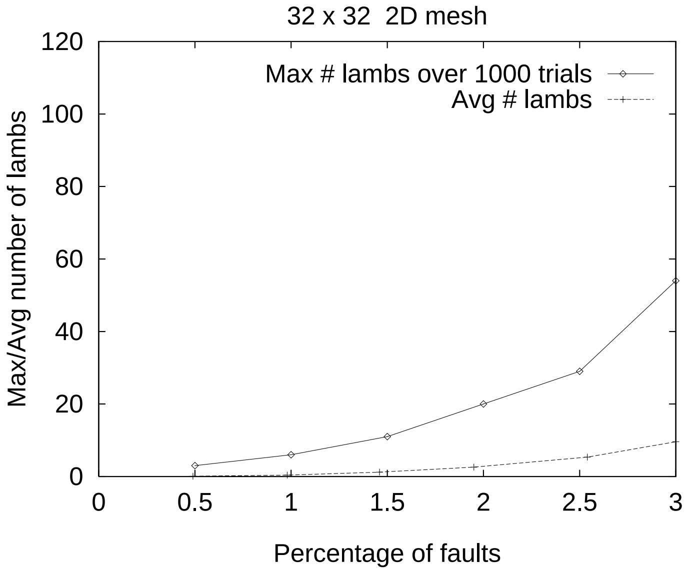

Figure 17: Maximum and average (over 1000 trials) numbers of lambs *vs* the percentage of random faults on the 32×32 2D mesh.

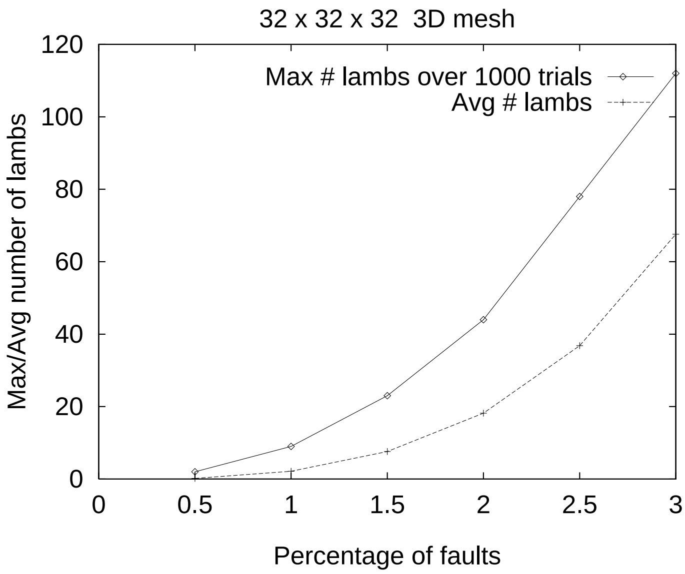

Figure 18: Maximum and average (over 1000 trials) numbers of lambs *vs* the percentage of random faults on the $32 \times 32 \times 32$ 3D mesh.

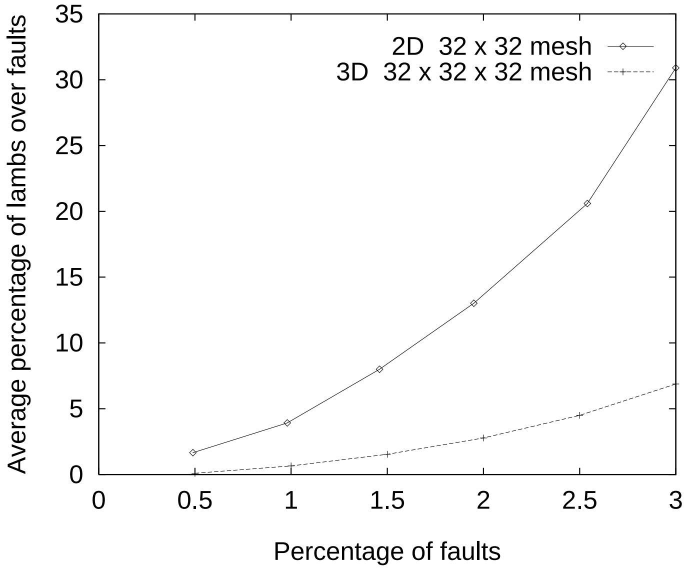

Figure 19: Comparison of 2D with 3D showing the average additional damage (percentage #lambs/#faults) *vs* the percentage of random faults.

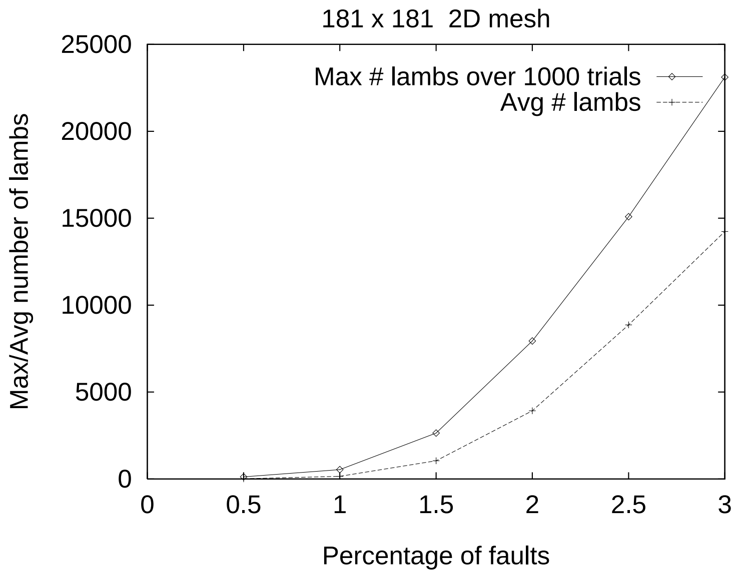

Figure 20: Maximum and average (over 1000 trials) numbers of lambs *vs* the percentage of random faults on the 181×181 2D mesh.

3% faults (31) is about equal to the bisection width (32) of $M_2(32)$. This motivates a study of how the number of lambs depends on the ratio of the number of faults to the bisection width of the mesh.

The bisection width of a graph is, roughly speaking, the number of node faults required to disconnect the graph into two pieces of roughly equal size. We take the bisection width of $M_d(n)$ to be n^{d-1}. For the 2D and 3D cases, simulations were run on square meshes of three sizes to see how the average percentage of lambs (*i.e.*, the average number of lambs as a percentage of the number N of nodes) depends on the ratio of the number of faults to the bisection width. The average was computed over 1000 trials for each mesh. Figure 21 shows the results for 2D meshes of widths $n = 32, 64, 128$, having sizes $N = 1024, 4096, 16384$, respectively. Figure 22 shows the results for 3D meshes of widths $n = 10, 16, 25$, having similar sizes $N = 1000, 4096, 15625$, respectively. In all cases, the percentage of lambs is reasonably small for a number of faults up to the bisection width, but it begins to degrade for larger numbers of faults. Also, for both 2D and 3D, the degradation is worse for smaller mesh size. A possible explanation for the second fact is that, for a fixed value of the ratio, the percentage of faults increases with decreasing mesh size. For example at ratio 3: for $M_3(25)$, the number of faults is $3 \cdot 25^2 = 1875 = 12\%$ of 25^3; for $M_3(10)$, the number of faults is $3 \cdot 10^2 = 300 = 30\%$ of 10^3.

Another experiment was done to see how the average (over 1000 trials) percentage of lambs depends on the mesh size, for a fixed percentage (3%) of faults. For the 2D (resp., 3D) case, for each integer i with $10 \leq i \leq 15$, the width n of the mesh is chosen so that the size n^2 (resp., n^3) is as close as possible to 2^i. Figures 23 and 24 show the results for 2D and 3D meshes, respectively. In both cases, the percentage of lambs increases with increasing mesh size. The likely explanation is based on the ratio of number of faults to bisection width; for a fixed fraction c of faults, this ratio increases with increasing mesh size, because the number of faults increases as cn^d while the bisection width increases as n^{d-1}.

For the $32 \times 32 \times 32$ mesh, we also recorded the average and maximum number of SES's found by the algorithm (over 1000 trials) as a function of the percentage of faults. The results are shown in Figure 25. The upper bound of Theorem 6.4 is also shown; it can be seen that this bound is considerably better than the rougher (and easier to prove) bound $(2d-1)f + 1 = 5f + 1$. The average and maximum numbers of DES's are not shown because, as would be expected for random faults, the average (resp., maximum) number of SES's is very close to the average (resp., maximum) number of DES's. For each percentage of faults considered, these number are within 0.08% (resp., 1.3%) of each other.

The average running time (over 1000 trials) of the algorithm, for the $32 \times 32 \times 32$ mesh and the 181×181 mesh, is plotted in Figure 26. The algorithm was implemented in C and run under AIX on a 133MHz IBM 7248 workstation with 128MB of memory.

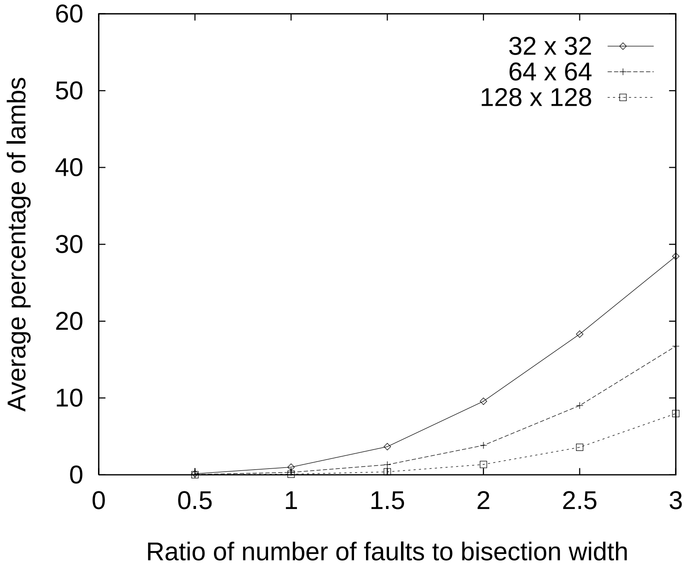

Figure 21: Average (over 1000 trials) percentage of lambs *vs* the ratio of the number of random faults to the bisection width of the mesh, for 2D meshes of three sizes.

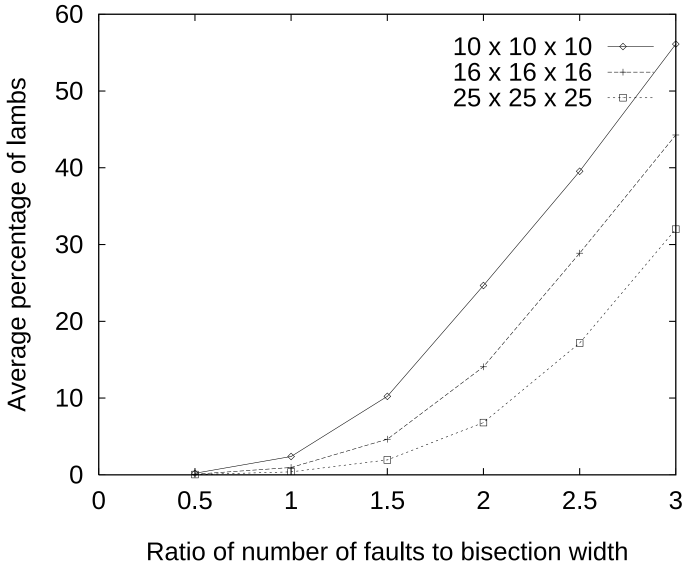

Figure 22: Average (over 1000 trials) percentage of lambs *vs* the ratio of the number of random faults to the bisection width of the mesh, for 3D meshes of three sizes.

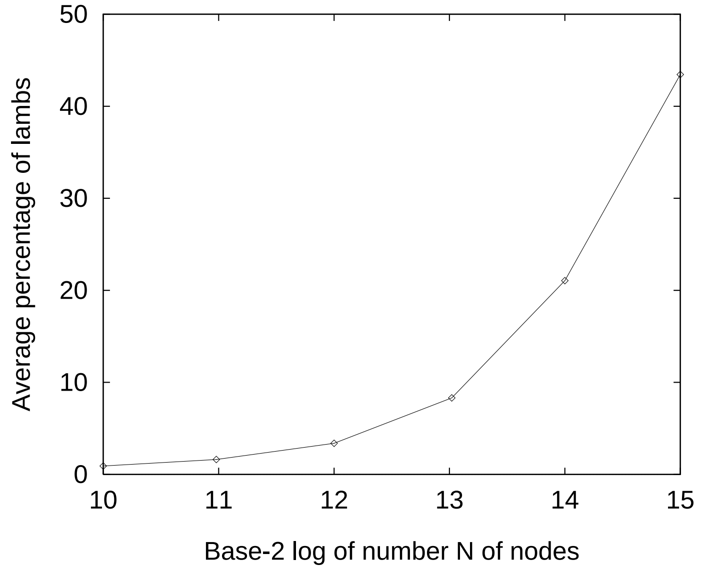

Figure 23: Average (over 1000 trials) percentage of lambs *vs* the size $N = n^2$ of the mesh, for $n \times n$ 2D meshes of various sizes and 3% random faults.

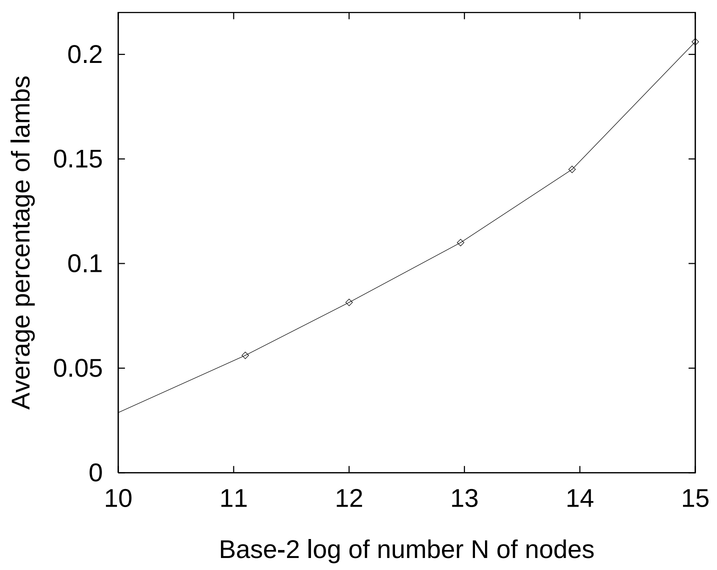

Figure 24: Average (over 1000 trials) percentage of lambs *vs* the size $N = n^3$ of the mesh, for $n \times n \times n$ 3D meshes of various sizes and 3% random faults.

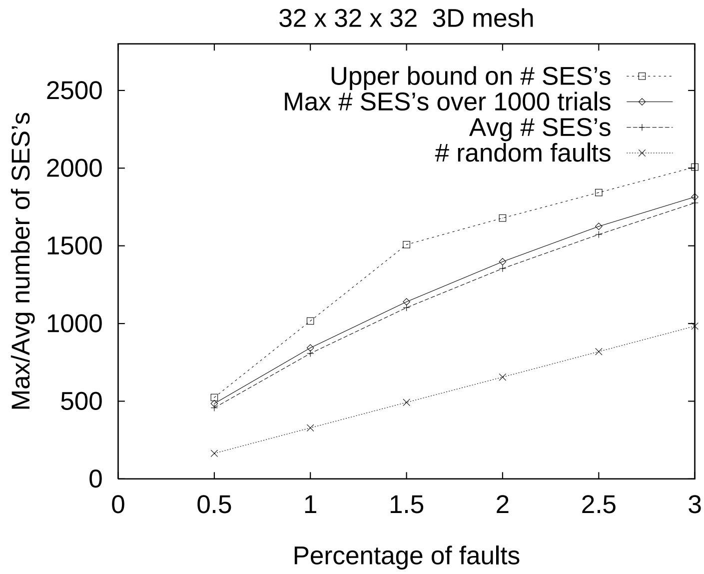

Figure 25: Maximum and average (over 1000 trials) number of SES's *vs* the percentage of random faults on the $32 \times 32 \times 32$ mesh. Also shown is the upper bound of Theorem 6.4.

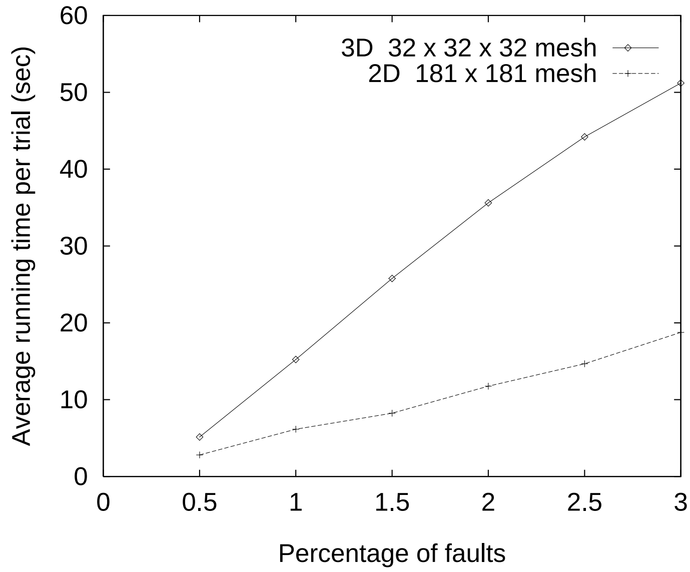

Figure 26: Average (over 1000 trials) running time *vs* the percentage of random faults on the $32 \times 32 \times 32$ mesh and the 181×181 mesh.

9 Computational Complexity of Lamb Minimization

In this section it is shown that the lamb minimization problem is NP-hard if $d \geq 3$ and $k \geq 2$. This justifies our use of an approximation algorithm, because if the lamb problem could be solved optimally in polynomial time then P = NP. More strongly, it is shown that if there is a polynomial-time r-approximation algorithm for the lamb problem (again for $d \geq 3$ and $k \geq 2$) then for any $\varepsilon > 0$ there is a polynomial-time $(r+\varepsilon)$-approximation algorithm for the vertex cover (VC) problem. Even though there has been a considerable amount of work on polynomial-time approximation algorithms for the vertex cover problem (see, for example, [16, §3]), 2 is the best constant factor of approximation currently known. Moreover, Håstad [15] has proved that if P $\neq$ NP and $\varepsilon > 0$ then there is no polynomial-time $(\frac{7}{6} - \varepsilon)$-approximation algorithm for VC; so the same conclusion holds for the lamb problem.

Definition 9.1 Fix $d, k \geq 1$. The (d,k)*-lamb problem* is the lamb problem restricted to instances where the mesh is $M_d(n)$ for some n, the number of rounds is k, and the ordering in each round is the ascending ordering $\alpha = (1, 2, \ldots, d)$.

Theorem 9.1 *Fix $d \geq 3$, $k \geq 2$, $r \geq 1$, and $\varepsilon > 0$. If there is a polynomial-time r-approximation algorithm for the (d,k)-lamb problem then there is a polynomial-time $(r + \varepsilon)$-approximation algorithm for the vertex cover problem.*

Proof. The detailed proof is given for the case $d = 3$ and $k = 2$. The proof for larger d and k then follows by minor modifications.

We describe an approximation algorithm $\mathcal{A}$ for VC. Let the graph G be an instance of the VC problem. It is convenient for the proof to add to G a vertex u_0 that has no incident edges; let G' be the new graph. Clearly, C is a vertex cover of G' iff $C - \{u_0\}$ is a vertex cover of G. Let V (resp., E) denote the set of vertices (resp., edges) of G', and let $V = \{u_0, \ldots, u_{|V|-1}\}$. A set F_N of node faults is described for $M = M_3(n)$ where n will be chosen later. The three dimensions are denoted X, Y, and Z, and the ordering in each round is (X, Y, Z). The rough idea is that a "column" of M along the Y dimension is associated with each vertex of G. If γ_1 and γ_2 are columns associated with vertices u_1 and u_2, respectively, then most nodes in γ_1 can 2-reach most nodes in γ_2, and vice versa, if and only if there is *no* edge between u_1 and u_2. Then, again very roughly, a vertex cover $C \subseteq V$ corresponds to a lamb set containing the columns associated with the vertices in C.

One constraint on n is that $n \geq 2|V|$. All of the faults belong to the $([0, 2|V|-1], *, [0, 2|V|-1])$ submesh. Nodes in (resp., outside) this submesh are called *internal nodes* (resp., *external nodes*). Imagine that M is divided into "planes" $(*, y, *)$ for $0 \leq y < n$, where y is called the *level* of the plane. There are two types of planes. One type is shown in Figure 27 and is called a *column plane*. The purpose is to create a column $(2i, *, 2i)$ for each vertex u_i. Letting y_0 be the level of the plane, F_N contains all nodes (x, y_0, z) with $0 \leq x, z < 2|V|$ *except* the nodes $(2i, y_0, 2i)$ for $i = 0, 1, \ldots, |V| - 1$. Figure 27 shows a column plane for $|V| = 5$ columns. Each node of a column in a column plane is called an *r-column node* for *restricted column node.* Note that no path from an r-column node along the X or Z dimension is possible.

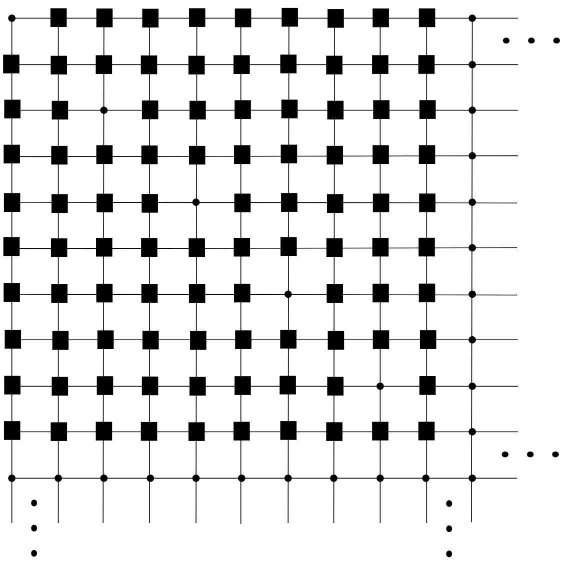

Figure 27: A column plane for five vertices $u_0, \ldots, u_4$. The r-column node in column-i is node $(2i, 2i)$.

For each i, j with $0 \leq i < j \leq |V| - 1$ such that (u_i, u_j) is *not* an edge in E, there is a *non-edge plane* depicted in Figure 28. Again let y_0 be the level of this plane. It is similar to a column plane except that there is an XZ path from the node $(2i, y_0, 2i)$ in column i to the node $(2j, y_0, 2j)$ in column j and vice versa, and an X path and a Z path from each of these nodes to external nodes. The nodes $(2i, y_0, 2i)$ and $(2j, y_0, 2j)$ are called *outlet nodes*. The internal nodes on the X and Z paths, other than the two outlets, are called *path nodes*. In the final construction, there will be a column plane in the two adjacent levels, $y_0 - 1$ and $y_0 + 1$, so there is no path from a path node along the Y dimension.

The entire construction consists of a non-edge plane for each non-edge of G' and enough column planes so that (i) there is an column plane between each pair of non-edge planes, and (ii) the total number of planes is $n \stackrel{\text{def}}{=} |V| + \lceil r(|V|^2 + 8|V|^3)/\varepsilon \rceil$. The set of nodes $(2i, *, 2i)$ is called *column-i*. Each node in column i is either an r-column node or an outlet. No node in a column is faulty. Recall that there is no edge between u_0 and the other nodes. So each column contains at least one outlet, but no more than $|V| - 1$ outlets.

Below we argue that the following reachability properties hold. Let $0 \leq i < j \leq |V| - 1$.

1. If v is a node in column-i, w is a node in column-j, and (u_i, u_j) is not an edge of G', then v can 2-reach w.

2. If v is a node in column-i, w is a node in column-j, neither v nor w is an outlet, and (u_i, u_j) is an edge of G', then v cannot 2-reach w.

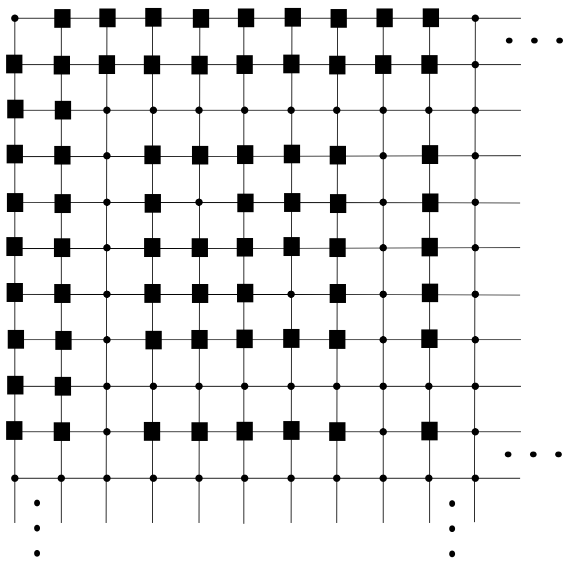

Figure 28: A non-edge plane for the absence of an edge between u_1 and u_4. Nodes $(2,2)$ and $(8,8)$ are outlets.

3. If v and w belong to the union of the nodes in column-i (where i is fixed) and the external nodes, then v can 2-reach w.

To prove property 1, we describe an $XYZXYZ$ route from v to w. Using Y the route moves (if necessary) to the outlet that is in column-i and in the non-edge plane for u_i and u_j. Using ZX the route moves along path nodes to the outlet in column-j in this plane. Using Y the route can now move to any node in column-j.

To prove property 2, consider an $XYZXYZ$ path from v. The first X route cannot be used because v is not an outlet. The first Y route must go to an outlet in column-i, for the next Z and X routes to be useful. Then the Z and X route can move through the XZ-plane to either (i) an outlet of column-ℓ where (u_i, u_ℓ) is *not* an edge of G, and thus $\ell \neq j$; or (ii) an external or path node in this XZ-plane. In either case, the next Y route cannot reach any non-outlet in column-j, so neither can the following Z route.

To prove property 3, we first note that if (x,y,z) and (x',y',z') are any two external nodes with either $z \geq 2|V|$ or $x' \geq 2|V|$, then (x,y,z) can XYZ-reach (x',y',z'). This is true because the intermediate nodes in $([x,x'],y,z)$ (or $([x',x],y,z)$), in $(x',[y,y'],z)$) (or $(x',[y',y],z)$), and in $(x',y',[z,z']))$ (or $(x',y',[z',z]))$ are external nodes, and all external nodes are good. (If $z < 2|V|$ and $x' < 2|V|$ then (x',y',z) might be an internal node.) We now describe an $XYZXYZ$ route from v to w. If v,w both belong to the same column-i or they are both external nodes, then it is obvious that v can 2-reach w. Say that v is in column-i and w is external. A YZ route from v can reach an external node (x,y,z) with $z \geq 2|V|$. By the observation above, there is now an XYZ route from (x,y,z) to the external node w. Say now that v is external and w is in column-i. Let (x',y',z') be an external node such that there is an X route to some outlet of column-i and $x' \geq 2|V|$. Therefore, there is an XYZ route from v to (x', y', z'), then an X route to an outlet in column-i, and then a Y route to any node in column-i.

Give $M_3(n)$ and F_N as input to the assumed r-approximation algorithm for the lamb problem, and let Λ be the lamb set returned. Define the set $C \subseteq V$ by $u_i \in C$ iff all non-outlets in column-i belong to Λ. Note that

$$|C| \leq \frac{|\Lambda|}{n - |V|} \tag{3}$$

because each column contains at least $n - |V|$ non-outlets. We show that C is a vertex cover of G. Suppose otherwise that there are $u_i, u_j \in V - C$ with $(u_i, u_j) \in E$. Because $u_i, u_j \notin C$, there is a non-outlet v in column-i and a non-outlet w in column-j such that $v, w \notin \Lambda$. But by reachability property 2, v cannot 2-reach w, which contradicts that Λ is a lamb set. The approximation algorithm $\mathcal{A}$ for VC outputs $C - \{u_0\}$.

It remains to show that $\mathcal{A}(G) \leq (r + \varepsilon) \cdot opt(G)$ for every input G, where $\mathcal{A}(G) \leq |C|$ and $opt(G)$ = the minimum size of a vertex cover for G. Let K^* denote this minimum size, and let C^* be a vertex cover of G with $|C^*| = K^*$. We may assume that G contains at least one edge, so $K^* \geq 1$. We define a lamb set Λ^* from C^* as follows. Let Λ^* contain all nodes in column-i for each $u_i \in C^*$ as well as all path nodes. We first show that Λ^* is a lamb set. If v and w are in the survivor set, then $v, w \notin \Lambda^* \cup F_N$. Therefore, each of v and w is either an external node or a node in column-i for some $u_i \notin C^*$. If v is in column-i and w is in column-j where $i \neq j$, then $u_i, u_j \notin C^*$, so there is no edge (u_i, u_j), so v can 2-reach w by reachability property 1 above. The only other possibility is that there is an i such that v and w belong to the union of the nodes in column-i and the external nodes; then v can 2-reach w by reachability property 3.

The number of path nodes in each non-edge plane is at most $8|V|$, and there are a total of at most $|V|^2$ non-edge planes. Therefore $|\Lambda^*| \leq nK^* + 8|V|^3$. Because Λ was found using an r-approximation algorithm for the lamb problem, $|\Lambda| \leq r|\Lambda^*| \leq r(nK^* + 8|V|^3)$. Abbreviate $B = \lceil r(|V|^2 + 8|V|^3)/\varepsilon \rceil$, and recall that $n = |V| + B$ by definition of n. Using (3),

$$\begin{aligned}
|C| &\leq \frac{|\Lambda|}{n - |V|} \\
&\leq \frac{r(nK^* + 8|V|^3)}{n - |V|} \\
&= \frac{r(K^*(|V| + B) + 8|V|^3)}{B} \\
&= rK^* + \frac{r(K^*|V| + 8|V|^3)}{B} \\
&\leq rK^* + \varepsilon \leq (r + \varepsilon)K^*.
\end{aligned}$$

This completes the proof for $d = 3$ and $k = 2$. If $k > 2$, add $2(k - 2)$ additional "turns" to the paths in each non-edge plane, so that from an outlet it takes $k - 1$ rounds of ZYX routing to reach another outlet or an external node (recall that no route can leave a path node along the Y

dimension). An easy way to modify the proof for $d > 3$ is to let the construction for the $d = 3$ and $k \geq 2$ case occupy the submesh $(*, *, *, 0, 0, \ldots, 0)$ of $M_d(n)$. If $(v_4, v_5, \ldots, v_d) \neq (0, 0, \ldots, 0)$, then $(x, y, z, v_4, v_5, \ldots, v_d)$ is faulty iff (x, y, z) is an internal node. Thus the external nodes (all good) of the new construction are those in the union of $(x, y, z, *, *, \ldots, *)$ over all external nodes (x, y, z). □

Using the result of Håstad [15] mentioned above gives the following.

Corollary 9.2 *Fix $d \geq 3$ and $k \geq 2$. If there is a polynomial-time 1.166-approximation algorithm for the (d, k)-lamb problem then* P = NP.

The following "converse" to Theorem 9.1 is immediate from Theorem 6.9, using in algorithm Lamb2 the SES (DES) partition where each node is a separate SES (DES). Then the graph produced by Reduce-WVC(General) has all vertex weights equal to 1.

Theorem 9.3 *Fix $r \geq 1$. If there is a polynomial-time r-approximation algorithm for the vertex cover problem then there is a polynomial-time r-approximation algorithm for the lamb problem.*

By Theorems 9.1 and 9.3, the lamb problem with $d \geq 3$ and $k \geq 2$ and the vertex cover problem have identical best factors of approximation by polynomial-time algorithms, to within an arbitrarily small $\varepsilon > 0$.

Finally, it is a corollary of the proof of Theorem 9.1 in the case $r = 1$ that the decision version of the lamb problem is NP-complete. The decision problem (d, k)-LAMB is defined in the style of [12].

(d, k)-LAMB
Instance: Numbers n, L and a node fault set $F_N \subseteq nodes(M_d(n))$.
Question: Does there exist a $(k, F_N, (\alpha, \ldots, \alpha))$-lamb set Λ with $|\Lambda| \leq L$?

Theorem 9.4 *For each $d \geq 3$ and $k \geq 2$, (d, k)-LAMB is NP-complete.*

In 2 dimensions, it is an open question whether $(2, k)$-LAMB is NP-complete, or whether it can be solved in polynomial time.

10 Conclusion

This paper has introduced a new method for fault-tolerant routing in meshes in the case that the number of virtual channels is only two. Rather than attempting to route around faults, the method sacrifices certain good nodes, called lambs, in order that the other good nodes can reach one another in at most two rounds of dimension-ordered routing. We have shown that allowing only one round of routing leads to an unacceptably large number of lambs. But with two rounds, the number of lambs that must be sacrificed is quite small when the number of faults is not too much larger than the bisection width of the mesh. For example, on a $32 \times 32 \times 32$ 3D mesh with $k = 2$ rounds of XYZ-routing and $32^2 = 1024$ random faults (so 3.125% of the mesh is faulty), the average number of lambs is less than 79, which is less than 0.25% of the number 32768 of nodes and less than 8% of the number 1024 of faults.

For the general case of a d-dimensional mesh with f faults and k rounds of dimension-ordered routing, we have given algorithms for finding a lamb set, whose running times depend on d, k, and f, but not on the size of the mesh. One of these algorithms uses time polynomial in d, k, f and linear in $|\Lambda|$ where Λ is the lamb set found (if d, k are constants, the time is $O(f^3 + |\Lambda|)$), and it is guaranteed to find a lamb set whose size is at most twice the optimal size.

We have shown that finding the minimum number of lambs is an NP-hard problem if the number of dimensions is at least 3 and the number of routing rounds is at least 2. Therefore, it will be difficult, or perhaps impossible, to invent a polynomial-time algorithm that always finds the minimum number of lambs in these cases. We have also shown that the best ratio of polynomial-time approximation for the lamb problem is essentially the same as that for the vertex cover problem.

Acknowledgment. We thank José Moreira and Marc Snir for their helpful comments.

References

[1] F. Allen and et al. Blue Gene: A vision for protein science using a petaflop supercomputer. *IBM Systems J.*, 40:310–327, 2001.

[2] G. S. Almasi, C. Caşcaval, J. G. Castaños, M. Denneau, W. Donath, M. Eleftheriou, M. Giampapa, H. Ho, D. Lieber, J. E. Moreira, D. Newns, M. Snir, and H. S. Warren Jr. Demonstrating the scalability of a molecular dynamics application on a petaflop computer. In *Proc. 15th ACM Int'l Conf. on Supercomputing (ICS)*, 2001.

[3] R. Bar-Yehuda and S. Even. A linear-time approximation algorithm for the weighted vertex cover problem. *J. Algorithms*, 2:198–203, 1981.

[4] R. V. Boppana and S. Chalasani. Fault-tolerant wormhole routing algorithms for mesh networks. *IEEE Trans. Comput.*, 44:848–864, 1995.

[5] Y. M. Boura and C. R. Das. Fault-tolerant routing in mesh networks. In *Proc. 1995 Int'l Conf. Parallel Processing, Vol. I*, pages 106–109, 1995.

[6] S. Chalasani and R. V. Boppana. Communication in multicomputers with nonconvex faults. *IEEE Trans. Comput.*, 46:616–622, 1997.

[7] C.-L. Chen and G.-M. Chiu. A fault-tolerant routing scheme for meshes with nonconvex faults. *IEEE Trans. Parallel Distrib. Syst.*, 12:467–475, 2001.

[8] A. A. Chien and J. H. Kim. Planar-adpative routing: Low-cost adaptive networks for multiprocessors. *J. ACM*, 42:91–123, 1995.

[9] T. H. Cormen, C. E. Leiserson, and R. L. Rivest. *Introduction to Algorithms.* MIT Press, Cambridge, MA, 1990.

[10] W. J. Dally and H. Aoki. Deadlock-free adaptive routing in multicomputer networks using virtual channels. *IEEE Trans. Parallel Distrib. Syst.*, 4:466–475, 1993.

[11] W. J. Dally and C. L. Seitz. Deadlock-free message routing in multiprocessor interconnection networks. *IEEE Trans. Comput.*, 36:547–553, 1987.

[12] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness.* W. H. Freeman, New York, 1979.

[13] C. J. Glass and L. M. Ni. Fault-tolerant wormhole routing in meshes. In *Proc. 23rd Int'l Symp. Fault-Tolerant Computing*, pages 240–249, 1993.

[14] D. Gusfield. Design (with analysis) of efficient algorithms. In *Handbooks in Operations Research and Management Science, Vol. 3: Computing*, chapter 8. North-Holland, New York, 1992.

[15] J. Håstad. Some optimal inapproximability results. *J. Assoc. Comput. Mach.*, 48:798–859, 2001.

[16] D. Hochbaum. *Approximation Algorithms for NP-Hard Problems.* PWS Publishing, Boston, 1997.

[17] Y.-J. Suh, B. V. Dao, J. Duato, and S. Yalamanchili. Software based fault-tolerant oblivious routing in pipelined networks. In *Proc. 1995 Int'l Conf. Parallel Processing, Vol. I*, pages 101–105, 1995.

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

Appendix

Proof of Theorem 3.1

By permuting the coordinates if necessary, we may assume that the 1-round routing π is the XYZ-routing. For each $u = (x_0, y_0, z_0) \in nodes(M_3(n))$, define the sets

$$\begin{aligned} A(u) &= \{ (x, y, z_0) \mid 0 \le x < n,\ y \le y_0, \text{ and } y < (n-1)/2 \} \\ B(u) &= \{ (x_0, y, z) \mid 0 \le z < n,\ y \ge y_0, \text{ and } y > (n-1)/2 \}. \end{aligned}$$

For each fixed $F_N \subseteq nodes(M_3(n))$, we show that these sets have the following properties:

1. If $y_0 < (n-1)/2$ then $|B(u)| \ge |A(u)| \ge (y_0+1)n$.
   If $y_0 > (n-1)/2$ then $|A(u)| \ge |B(u)| \ge (n-y_0)n$.
2. Let $u = (x_0, y_0, z_0)$ and $u' = (x'_0, y'_0, z'_0)$ where $x_0 \ne x'_0$ and $z_0 \ne z'_0$. Then $A(u)$, $B(u)$, $A(u')$, $B(u')$ are pairwise disjoint.
3. If Λ is a lamb set and $u \in F_N$, then either $(A(u) - F_N) \subseteq \Lambda$ or $(B(u) - F_N) \subseteq \Lambda$.

Property 1 is immediate from the definition of $A(u)$ and $B(u)$.

The proof of property 2 has three cases: (i) $A(u) \cap A(u') = \emptyset$ because $z_0 \ne z'_0$; (ii) $B(u) \cap B(u') = \emptyset$ because $x_0 \ne x'_0$; (iii) if (p, p') is one of (u, u), (u, u'), (u', u), or (u', u'), then $A(p) \cap B(p') = \emptyset$ because $y < (n-1)/2$ for every $(x, y, z) \in A(p)$ and $y' > (n-1)/2$ for every $(x', y', z') \in B(p')$.

To prove property 3, suppose that there are good $v \in A(u) - \Lambda$ and $w \in B(u) - \Lambda$. Then v, w belong to the survivor set $V = nodes(M_3(n)) - (\Lambda \cup F_N)$. So v can (F_N, π)-reach w. We obtain a contradiction by showing that the unique XYZ route from v to w must pass through a node in F_N. Let $u = (x_0, y_0, z_0) \in F_N$. Consider the XYZ route from $v = (x, y, z) \in A(u)$ to $w = (x', y', z') \in B(u)$. First note by definition of $A(u)$ and $B(u)$ that $z = z_0$, $x' = x_0$, and $y \le y_0 \le y'$. The XYZ route first moves from $v = (x, y, z_0)$ to (x_0, y, z_0), then to (x_0, y', z_0), and finally to $(x_0, y', z') = w$. But the path from (x_0, y, z_0) to (x_0, y', z_0) touches the faulty node (x_0, y_0, z_0). This proves property 3.

Define the random sets of nodes, F_2 and S, by the following random process. The notes in parentheses follow from the definition of $A(u)$ and $B(u)$ and property 1.

1. For $i = 1, 2, \ldots, f$:
   (a) Choose $u_i = (x_i, y_i, z_i)$ at random from $[0, n-1]^3$.
   (b) If $y_i < (n-1)/2$ then set $C_i = A(u_i)$ (note $|B(u_i)| \ge |A(u_i)| \ge (y_i+1)n$);
   if $y_i > (n-1)/2$ then set $C_i = B(u_i)$ (note $|A(u_i)| \ge |B(u_i)| \ge (n-y_i)n$);
   if $y_i = (n-1)/2$ then set $C_i = A((x_i, y_i - 1, z_i))$ (note $|B(u_i)| = |A(u_i)| \ge |C_i| = \frac{n-1}{2} n \ge \frac{n^2}{4}$).
2. Initialize sets $\mathcal{F} = \mathcal{S} = \mathcal{X} = \mathcal{Z} = \emptyset$ and $I_i = 0$ for $1 \le i \le f$.

3. For $i = 1, 2, \ldots, f$:
   If $x_i \notin \mathcal{X}$ and $z_i \notin \mathcal{Z}$, then $I_i = 1$, $\mathcal{F} \leftarrow \mathcal{F} \cup \{(x_i, y_i, z_i)\}$, $\mathcal{S} \leftarrow \mathcal{S} \cup C_i$, $\mathcal{X} \leftarrow \mathcal{X} \cup \{x_i\}$, and $\mathcal{Z} \leftarrow \mathcal{Z} \cup \{z_i\}$.

4. Let $F_2 = \mathcal{F}$ and $S = \mathcal{S}$.

Let the random variable F_1 be (as in the lemma) a random subset of f nodes chosen from $nodes(M_3(n))$ (without replacement). For $k = 1, 2$, define the r.v. $L_k = \lambda(M_3(n), 1, F_k, \pi)$. Let $\mathbf{E}$ denote expected value. The random process above is equivalent to choosing f random faults (with replacement) (x_i, y_i, z_i) for $1 \le i \le f$, and then changing a fault (x_i, y_i, z_i) to a good node if there is some fault (x_i, y, z) or (x, y, z_i) that was chosen earlier. Removing nodes from F_1 cannot increase the value of $\lambda(M_3(n), 1, F_1, \pi)$. Therefore, $\mathbf{E}(L_1) \ge \mathbf{E}(L_2)$.

Note four facts about the random bit I_i and the random set C_i. (i) If $I_i = 1$ then $u_i = (x_i, y_i, z_i) \in F_2$ so, by property 3, a set of size at least $|C_i|$ (either $A(u_i)$ or $B(u_i)$) must be included in the lamb set Λ. (ii) By property 2, if $i \ne \ell$, then $C_i \cap C_\ell = \emptyset$ with probability 1, given that $I_i = I_\ell = 1$. (iii) For each fixed i, I_i and C_i are independent because I_i depends only on the choice of x_i and z_i, and C_i depends only on the choice of y_i. (iv) For each i, $\mathbf{E}(|C_i|) \ge n^2/4$. This follows from property 1 and

$$\mathbf{E}(\,(y_i + 1)n \mid y_i < (n-1)/2\,) = \mathbf{E}(\,(n - y_i)n \mid y_i > (n-1)2\,) \ge n^2/4.$$

It follows from these four facts and $\mathbf{E}(F_2) \le f$ that

$$\begin{aligned}
\mathbf{E}(L_2) \ge \mathbf{E}(|S - F_2|) \ge \mathbf{E}\left(\sum_{i=1}^{f} I_i \cdot |C_i|\right) - f = \sum_{i=1}^{f} \mathbf{E}(I_i \cdot |C_i|) - f &= \sum_{i=1}^{f} \mathbf{E}(I_i) \cdot \mathbf{E}(|C_i|) - f \\
&\ge \frac{n^2}{4} \sum_{i=1}^{f} \mathbf{E}(I_i) - f.
\end{aligned}$$

To complete the proof, we derive a lower bound on $\sum_{i=1}^{f} \mathbf{E}(I_i)$. At the start of the i-th iteration, $|\mathcal{X}| \le i - 1$ and $|\mathcal{Z}| \le i - 1$. So $\mathbf{E}(I_i) = \Pr[\,I_i = 1\,] \ge (n - i + 1)^2/n^2$. Using the known formula $\sum_{j=1}^{m} j^2 = m(m+1)(2m+1)/6$,

$$\begin{aligned}
\sum_{i=1}^{f} \mathbf{E}(I_i) &\ge n^{-2} \sum_{j=n-f+1}^{n} j^2 \qquad \text{(setting } j = n - i + 1\text{)} \\
&= n^{-2} \left(\sum_{j=1}^{n} j^2 - \sum_{j=1}^{n-f} j^2 \right) \\
&\ge f - \frac{f^2}{n} + \frac{f^3}{3n^2}.
\end{aligned}$$

So $\mathbf{E}(L_1) \ge \mathbf{E}(L_2) \ge (n^2/4) \sum_{i=1}^{f} \mathbf{E}(I_i) - f \ge fn^2/4 - f^2 n/4 + f^3/12 - f$. $\square$

Proof of Proposition 6.5

The goal is to construct, for every odd $n \geq 3$ and every d, f with with $f \leq n^{d-1}(n-1)/2$, a fault set such that Find-SES-Partition will return an SES partition of size $B(d, f)$. The proof is by induction on d. The proof for link faults and the proof for node faults can be given in parallel, because they differ only in the basis $d = 1$.

Let $d = 1$, so $f \leq (n-1)/2$ and $B(d, f) = f + 1$. Letting F_N be the set of nodes $2i - 1$ for $i = 1, 2, \ldots, f$, or letting F_L be the set of links whose left endpoint is in F_N, it is clear that the algorithm returns an SES partition of size $f + 1$.

Let $d > 1$ and assume the proposition is true for $d - 1$. There are two cases. In each case, we fix the number of faults that lie in each of the $(d-1)$-dimensional submeshes $(*, \ldots, *, c)$ for $0 \leq c \leq n-1$. Given these numbers, the positions of the faults in these submeshes are found using the induction assumption.

The first case is $2f \leq n - 1$, so $B(d, f) = 2f(d-1) + f + 1 = 2fd - f + 1$. Each submesh $(*, \ldots, *, 2i-1)$ contains one fault for $i = 1, 2, \ldots, f$. The size of the SES partition is $(f + 1)$ plus the sum of the sizes of the SES partitions returned by the recursive calls on $(*, \ldots, *, 2i-1)$ for $i = 1, 2, \ldots, f$. By the induction assumption, this total is

$$(f + 1) + f \cdot B(d-1, 1) = (f + 1) + f \cdot (2(d-2) + 1 + 1) = 2fd - f + 1 = B(d, f).$$

The second case is $n - 1 < 2f \leq n^{d-1}(n-1)$. Write $f = qn + r$ where q and r are integers with $q \geq 0$ and $0 \leq r < n$. Each submesh $(*, \ldots, *, 2i-1)$ for $i = 1, 2, \ldots, (n-1)/2$ contains at least one fault, r submeshes contain $q + 1$ faults each, and $n - r$ submeshes contain q faults each (other than these requirements, it does not matter how the faults are partitioned to the submeshes). By the induction assumption, the size of the SES partition is $r \cdot B(d-1, q+1) + (n-r) \cdot B(d-1, q)$.

Let j_0 be the minimum j such that $2f \geq n^{d-j}(n-1)$. By definition of $B(d, f)$,

$$B(d, f) = \sum_{j=j_0}^{d} n^{d-j}(n-1) + \sum_{j=2}^{j_0-1} 2f + (f + 1). \tag{4}$$

The next goal is to obtain expressions for $B(d-1, q)$ and $B(d-1, q+1)$ in terms of j_0. Let j be arbitrary with $2 \leq j \leq d-1$. (i) If $j \geq j_0$ then $f \geq n^{d-j}(n-1)/2$ by definition of j_0. Using $f = qn+r$, this implies that $q \geq n^{d-1-j}(n-1)/2 - r/n$. Because q and $n^{d-1-j}(n-1)/2$ are both integers, and $r/n < 1$, this implies that $q \geq n^{d-1-j}(n-1)/2$. So $2(q+1) > 2q \geq n^{d-1-j}(n-1)$. (ii) If $j < j_0$ then $f < n^{d-j}(n-1)/2$ by definition of j_0. Using $nq \leq f$ this gives $q < n^{d-1-j}(n-1)/2$. Because q and $n^{d-1-j}(n-1)/2$ are integers, we also have $q + 1 \leq n^{d-1-j}(n-1)/2$, so $2(q+1) \leq n^{d-1-j}(n-1)$.

So the size of the SES partition is

$$\begin{aligned}
r \cdot B(d-1, q+1) + (n-r) \cdot B(d-1, q) &= r\left(\sum_{j=j_0}^{d-1} n^{d-1-j}(n-1) + \sum_{j=2}^{j_0-1} 2(q+1) + (q+1) + 1\right) \\
&\quad + (n-r)\left(\sum_{j=j_0}^{d-1} n^{d-1-j}(n-1) + \sum_{j=2}^{j_0-1} 2q + q + 1\right) \\
&= n \cdot \sum_{j=j_0}^{d-1} n^{d-1-j}(n-1) + \sum_{j=2}^{j_0-1} (2qn + 2r) + (qn + r + n) \\
&= \sum_{j=j_0}^{d-1} n^{d-j}(n-1) + \sum_{j=2}^{j_0-1} 2f + (f+1) + (n-1) \\
&= \sum_{j=j_0}^{d} n^{d-j}(n-1) + \sum_{j=2}^{j_0-1} 2f + (f+1),
\end{aligned}$$

which equals the expression for $B(d, f)$ in (4). $\square$