Adaptive Fault-Tolerant Deadlock-Free Routing in Meshes and Hypercubes

Chien-Chun Su and Kang G. Shin

Real-Time Computing Laboratory
Department of Electrical Engineering and Computer Science
The University of Michigan
Ann Arbor, MI 48109–2122
{ccsu,kgshin}@eecs.umich.edu

ABSTRACT

We present an adaptive deadlock-free routing algorithm which decomposes a given network into two virtual interconnection networks, VIN_1 and VIN_2. VIN_1 supports deterministic deadlock-free routing, and VIN_2 supports fully-adaptive routing. Whenever a channel in VIN_1 or VIN_2 is available, it can be used to route a message.

Each node is identified to be in one of three states: safe, unsafe, and faulty. The unsafe state is used for deadlock-free routing, and an unsafe node can still send and receive messages. When nodes become faulty/unsafe, some channels in VIN_2 around the faulty/unsafe nodes are used as the detours of those channels in VIN_1 passing through the faulty/unsafe nodes, i.e., the adaptability in VIN_2 is transformed to support fault-tolerant deadlock-free routing. Using information on the state of each node's neighbors, we have developed an adaptive fault-tolerant deadlock-free routing scheme for n-dimensional meshes and hypercubes with only two virtual channels.

In an n-dimensional hypercube, any pattern of faulty nodes can be tolerated as long as the number of faulty nodes is no more than $\lceil n/2 \rceil$. The maximum number of faulty nodes that can be tolerated is 2^{n-1}, which occurs when all faulty nodes can be encompassed in an $(n-1)$-cube. In an n-dimensional mesh, we use a more general fault model, called a *disconnected rectangular block*. Any arbitrary pattern of faulty nodes can be modeled as a rectangular block after finding both unsafe and disabled nodes (which are then treated as faulty nodes). This concept can also be applied to k-ary n-cubes with four virtual channels, two in VIN_1 and the other two in VIN_2. Finally, we present simulation results for both hypercubes and 2-dimensional meshes by using a variety of workloads and fault patterns.

Index Terms — Wormhole routing, adaptive deadlock-free routing, fault-tolerant routing, hypercubes and meshes, k-ary n-cubes.

The work reported in this paper was supported in part by the National Science Foundation under Grant MIP–9203895 and the Office of Naval Research under Grants N00014-J–92-1080 and N00014-J-91-1115. The opinions, findings, and recommendations expressed in this paper are those of the authors, and do not necessarily reflect the views of the funding agencies.

1 Introduction

Distributed-memory MIMD (multiple-instruction-multiple-data) multicomputers are usually composed of a large number of nodes, each with its own processor and local memory. These nodes use an interconnection network to exchange data and synchronize with one another. Thus, the performance of a multicomputer depends strongly on network latency and throughput.

There are two types of message routing: (1) *deterministic routing* that uses only a single path from the source to destination, and (2) *adaptive routing* that allows more freedom in selecting message paths. Most commercial multicomputers use deterministic routing because of its deadlock freedom and ease of implementation. However, adaptive routing can reduce network latency and increase network throughput [4]. It can also tolerate more faults than deterministic routing. But, the flexibility of adaptive routing may cause deadlock and/or livelock problems. A deadlock occurs when a message waits for an event that will never happen. In contrast, a livelock keeps a message moving indefinitely without reaching the destination.

Four types of switching methods for sending or receiving messages are used in distributed-memory multicomputer systems: store-and-forward [18], circuit switching, virtual cut-through [19], and wormhole routing [7]. The store-and-forward and circuit switching methods were used in the first-generation multicomputer systems, such as the Intel iPSC [27]. Contemporary multicomputers use wormhole routing due mainly to its high transmission efficiency and reduced buffer size requirements [26, 33]. All the algorithms described in this paper are thus based on wormhole routing. In wormhole-routed multicomputers, a message contains one or more flow-control digits (flits). The first flit of a message (head flit) builds the transmission path between the source and destination. Each flit of a message following the head flit advances as soon as the preceding flit arrives at a node (i.e., flit pipelining) and gets blocked when the required channel resources are not available.

A routing algorithm is said to be *minimal* if the message goes through a shortest path to its destination. A minimal, fully-adaptive routing algorithm allows the message to be routed via any one of the shortest paths between its source and destination. In this paper, we first present an adaptive deadlock-free routing algorithm by using only one extra virtual channel as compared to deterministic routing. This proposed algorithm may use minimal or non-minimal routing for a given specific topology, such as hypercubes and k-ary n-cubes. Second, for n-dimensional hypercubes and meshes, we present adaptive fault-tolerant deadlock-free routing schemes based on the proposed adaptive routing algorithm, which require only two virtual channels. A node fault is assumed to be the basic fault element in order to deal with fault-tolerance. The advance of VLSI technology has enabled the computation and communication parts in a node to be integrated into a single VLSI chip, as was done in iWarp [28], Transputer [29], Ncube and MDP [6]. For a simpler processor node such as CM-2 [31], even more processors can be integrated into a single chip. This situation makes the node fault model reasonable. In our fault-tolerance scheme, each node needs to be marked as being in one of three states: safe, unsafe, and faulty. The unsafe state is used to facilitate deadlock-free routing. An unsafe node can still transmit and receive messages. For n-dimensional meshes, we use a *disconnected rectangular block* as the fault model, which is composed of unsafe and faulty nodes. The basic concept of the proposed adaptive fault-tolerant routing is to find a detour in VIN_2 whenever the requested channel in VIN_1 is connected to an unsafe or faulty node. We also discuss a fault-tolerant routing scheme for k-ary n-cubes.

This paper is organized as follows. Section 2 gives a brief review of the related work. Section 3 describes our adaptive deadlock-free routing algorithm and its application to n-dimensional meshes. Section 4 presents adaptive fault-tolerant deadlock-free routing algorithms for n-dimensional hypercubes and meshes. Section 5 presents the simulation results for both hypercubes and 2-dimensional meshes with and without faulty nodes. The paper concludes with Section 6.

2 Related Work

Numerous routing algorithms in multicomputers have been proposed in recent years. Some of them consider adaptability and fault-tolerance in routing messages without considering deadlock freedom [1, 2]. However, deadlock freedom is an important design issue for multicomputer systems, so we will focus on deadlock-free routing algorithms.

Dally and Seitz [8] proposed the concept of virtual channel to develop deadlock-free routing algorithms. Virtual channels are time-multiplexed over a single physical link, so one separate queue must be maintained in a node for each virtual channel. Virtual channels are also used to remove the cycles in the channel-dependency graph (CDG), providing deadlock-freedom during message transmission. But the algorithms in [8] are deterministic and hence cannot handle component failures in the selected path.

Linder and Harden [24] extended the concept of virtual channel to multiple, virtual interconnection networks that provide adaptability, deadlock-freedom, and fault-tolerance. Each link is shared by many virtual channels, and the number of virtual channels used depends on how many virtual networks are needed. These virtual channels can be divided into several groups or virtual networks. Message passing inside a virtual network is deadlock-free and messages are constrained to travel through virtual networks in a certain predefined order. When the message is blocked in a virtual network due to congested or faulty nodes/links, it can keep moving forward via another virtual network, thereby increasing routing adaptability. The chief disadvantage of this method is the requirement of many virtual channels,[1] e.g., a k-ary n-cube needs $2^{n-1}(n+1)$ virtual channels.

[1] In general, its number is an exponential function of the network dimension.

Glass and Ni [15, 16] proposed partially-adaptive routing algorithms for n-dimensional meshes and k-ary n-cubes without adding any physical or virtual channels. They first investigated the possible deadlock cycles on 2D- and 3D- meshes, then proposed some prohibited turns of these cycles to prevent deadlocks. However, if minimal routing is required, then at least one half of source-destination pairs will have only a single routing path from the source to destination. Because this algorithm cannot always route messages along shortest paths in the network, it is called *partially-adaptive* routing. The authors of [17] modified the routing algorithm of this turn model to make it $(n-1)$ fault-tolerant for n-dimensional meshes. However, for 4 or higher dimensional meshes, it is only an assertion or conjecture, i.e., its validity remains to be proved.

Chien and Kim [3] proposed a planar-adaptive routing algorithm which limits the routing freedom to two dimensions at a time. The reduced freedom makes it possible to prevent deadlocks with only a fixed number of virtual channels, independent of network dimension. This algorithm needs three virtual channels for n-dimensional meshes and only supports partially-adaptive routing.

Duato [9–12] developed a general theorem of adaptive deadlock-free routing, defining a criterion for deadlock-freedom and then using the theorem to propose a fully-adaptive algorithm. The theorem states that by separating virtual channels on a physical link into restricted and unrestricted partitions, fully-adaptive deadlock-free routing can be achieved. Dally and Aoki [5] proposed a dynamic deadlock-free routing algorithm which also divides the virtual channels of each physical channel into two non-empty classes: adaptive and deterministic. Messages originate from adaptive channels. Once a message is on a deterministic channel, the message must be routed in dimension order and cannot re-enter any adaptive channel. Actually, their method relies on a stronger constraint than Duato's algorithm. Lin *et al.* [23] proposed a message flow model in wormhole-routed networks, and applied it to develop adaptive routing algorithms. Any routing algorithm under this message flow model can be guaranteed to have deadlock-freedom.

In [13, 30], Gaughan and Yalamanchili proposed a misrouting backtracking protocol under packet circuit switching (PCS), a variant of wormhole routing. This protocol is deadlock-free and can tolerate up to $2n-1$ physical link failures in k-ary n-cubes. Like traditional circuit switching, PCS needs to set up a path before transmitting messages, and hence, the path setup overhead is relatively high for small-size messages.

In [22], Lee and Hayes proposed a fault-tolerant deadlock-free routing scheme for n-dimensional hypercubes, but it requires $(n+1)$ virtual channels. It is basically derived from the hop-count method [18] used in packet-switched networks. The authors of [20] proposed a deadlock-free fault-tolerant routing scheme for hypercubes. This scheme does not require additional virtual channel, nor supports adaptive routing.

The adaptive deadlock-free routing we propose in this paper also decomposes the network into two virtual interconnection networks. We use a novel numbering method to develop fully-adaptive routing algorithms, which turn out to yield a similar result as Duato's algorithm. However, we will primarily focus on fault-tolerant routing by using this numbering method.

3 Adaptive Deadlock-Free Routing Algorithm

We will first develop an adaptive deadlock-free routing algorithm under the following assumptions:

A1. The underlying interconnection network can be divided into two virtual interconnection networks, VIN_1 and VIN_2. The virtual channels in VIN_1 and VIN_2 share the bandwidth of each physical link.

A2. A message reaching its destination will eventually be consumed.

A3. The channel allocation algorithm used is free from starvation using, for example, first-come first-serve or round-robin scheduling.

A4. Each virtual channel in VIN_1 is assigned a channel number. When a message traverses the nodes between the source and destination, the head flit can always find a channel in VIN_1 to use for each hop in an increasing order of channel numbers. (We can also use a decreasing order of channel numbers, but for convenience of presentation, we will henceforth use an increasing order of channel numbers.)

Algorithm 3.1: (Adaptive Routing) *The head flit first checks to see if there is any channel available in VIN_1 or VIN_2. Whenever a channel is available in VIN_1 or VIN_2, route the message via this channel. Note that the requested channel in VIN_1 should satisfy **A4**.*

The chief advantage of this algorithm comes from the channels in VIN_2 that can be treated as *free* channels. A free channel can be used whenever it is released by a message without any dimension-order restriction.

Theorem 1: The proposed adaptive routing algorithm 3.1 is deadlock-free.

Proof: See the Appendix.

Algorithm 3.1 can use minimal or non-minimal routing as long as **A1**—**A4** are satisfied, irrespective of the interconnection network topology used. But it's possible that the head flit cannot get a higher-number channel in VIN_1 if the routing algorithm arbitrarily chooses an outgoing channel. For some popular topologies, such as hypercubes, n-dimensional meshes, and k-ary n-cubes, both deadlock and livelock freedom can be guaranteed with minimal routing. However, livelock freedom is not guaranteed if non-minimal routing is used. At the end of this section, we will mention how to develop a non-minimal deadlock-free routing algorithm for n-dimensional meshes. Although we don't consider the store-and-forward switching scheme, Algorithm 3.1 can also be applied to packet-switched networks. In packet-switched networks [18], the resources in the channel-dependency graph are replaced by packet buffers, and thus, if **A1**—**A4** are satisfied, we need two packet buffers on each node to support adaptive routing.

Now, we would like to apply Algorithm 3.1 to n-dimensional meshes. An n-dimensional mesh consists of $k_{n-1} \times k_{n-2} \times \cdots \times k_1 \times k_0$ nodes, where $k_i \geq 2$ is the number of nodes along dimension i. A node X is represented by n coordinates, $(x_{n-1}, x_{n-2}, \ldots, x_0)$, $0 \leq x_i \leq k_i - 1$, $0 \leq i \leq n-1$. Two nodes X and Y are neighbors if and only if $x_i = y_i$ for all i, $0 \leq i \leq n-1$, except for one coordinate, say j, such that $y_j = x_j \pm 1$. Thus, each node has n to $2n$ neighbors, depending on its location in the mesh. If X and Y are neighbors, then the channel of dimension i at node X is in the positive direction relative to node Y when $x_i = y_i - 1$, or in the negative direction when $x_i = y_i + 1$.

Let's define the following notation that will be used throughout the rest of this paper.

Source Node $S = (s_{n-1}, s_{n-2}, \ldots, s_0)$
Destination Node $D = (d_{n-1}, d_{n-2}, \ldots, d_0)$
Current Node $C = (c_{n-1}, c_{n-2}, \ldots, c_0)$
Routing Tag $R = (r_{n-1}, r_{n-2}, \ldots, r_0)$
$= (d_{n-1} - s_{n-1}, d_{n-2} - s_{n-2}, \ldots, d_0 - s_0)$; /* for n-dimensional meshes */
$= (d_{n-1} \oplus s_{n-1}, d_{n-2} \oplus s_{n-2}, \ldots, d_0 \oplus s_0)$; /* for n-dimensional hypercubes */
$f(R)$: number of nonzero elements in R;
$f(R, i)$: the directed dimension with nonzero i-th element in R;
$VC_{i,j}$: the virtual channel in the i-th dimension of the j-th virtual interconnection network
$VC_{i,*}$: the virtual channel in the i-th dimension of VIN_1 or VIN_2
$num(VC_{i,j})$: the channel number assigned to $VC_{i,j}$.
$detour\_group_{i,j}$: the detour of channel $VC_{i,1}$ in dimension j.

Note that the subscript i in $VC_{i,j}$ is a signed integer for n-dimensional meshes. If i is positive, then $VC_{i,j}$ is in the positive direction; otherwise, $VC_{i,j}$ is in the negative direction. Using the above notation, the traditional dimension-order routing algorithm for n-dimensional meshes can be described as follows.

Algorithm 3.2: Dimension-order routing for n-dimensional meshes.

S1. *Update R; /*$r_i := r_i \pm 1$ */*
S2. *If ($R == 0$), route the message to the local processor and exit;*
S3. *Route the message via $VC_{f(R,1),1}$ and exit; /* only one virtual interconnection network is required */*

For example, consider source node $S = (1, 3, 4, 2)$ and destination node $D = (3, 3, 1, 3)$ in a $4 \times 4 \times 5 \times 4$ mesh. The routing path between S and D is determined by the routing tag $R =$

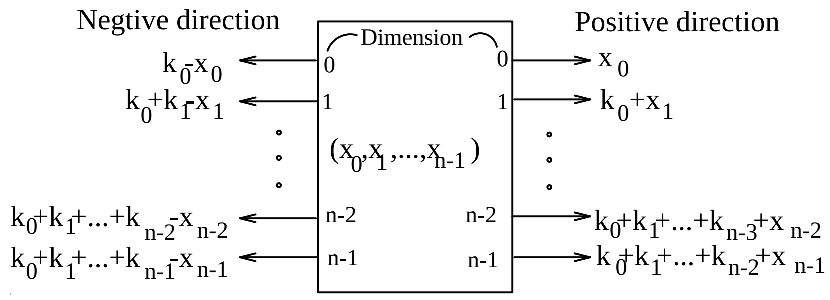

Figure 1: Numbering the output channels of a node in an n-dimensional mesh.

$(r_3, r_2, r_1, r_0) = (3-1, 3-3, 1-4, 3-2) = (2, 0, -3, 1)$, where $|r_i|$ represents the number of hops in dimension i and the sign of r_i indicates the routing direction. $f(R) = 3$, $f(R, 1) = +0$, $f(R, 2) = -1$, and $f(R, 3) = +3$. If minimal dimension-order routing is used, then the routing path in this example is $(1, 3, 4, 2) \stackrel{(2,0,-3,1)}{\longrightarrow} (1, 3, 4, 3) \stackrel{(2,0,-3,0)}{\longrightarrow} (1, 3, 3, 3) \stackrel{(2,0,-2,0)}{\longrightarrow} (1, 3, 2, 3) \stackrel{(2,0,-1,0)}{\longrightarrow} (1, 3, 1, 3) \stackrel{(2,0,0,0)}{\longrightarrow} (2, 3, 1, 3) \stackrel{(1,0,0,0)}{\longrightarrow} (3, 3, 1, 3)$. The symbols above the right arrows are the corresponding routing tags. The dimension-order routing for n-dimensional meshes is deadlock-free, minimal and deterministic.

Presented below is an adaptive minimal routing algorithm for n-dimensional meshes based on Algorithm 3.2. Assume there are two virtual channels running over each physical link, one in VIN_1 and the other in VIN_2. The hypercube can be treated as a special case of n-dimensional meshes.

Algorithm 3.3: Adaptive minimal routing for n-dimensional meshes.

S1'. *Update R; /*$r_i := r_i \pm 1$ */*
S2'. *If $(R == 0)$, route the message to the local processor and exit;*
S3'. *Parallel_Request $VC_{f(R,1),*}, VC_{f(R,2),2}, \ldots, VC_{f(R,f(R)),2}$;*
S4'. *Route the message via the first available channel among those requested and exit.*

S1' and ***S2'*** are the same as the dimension-order routing mentioned above. In ***S3'***, the algorithm requests simultaneously all the channels in VIN_2 corresponding to nonzero elements of R. It also requests the channel $(VC_{f(R,1),1})$ in VIN_1 according to the dimension-ordering rule. Whenever any of the requested channels becomes available, the message will be routed via that channel. Obviously, this is minimal routing because $|r_i|$ will be decremented by one for each hop the message takes.

Theorem 2: Algorithm 3.3 for n-dimensional meshes is both deadlock-free and livelock-free.

Proof: Since this is minimal routing, it guarantees livelock-freedom. From Theorem 1, we know the virtual channels in VIN_2 will not affect deadlock-freedom if **A1**—**A4** can be satisfied. **A1** is satisfied since two virtual interconnection networks are used. **A2** and **A3** are related to the

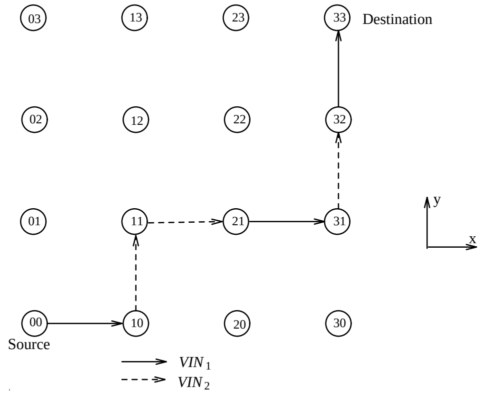

Figure 2: A message holding virtual channels of VIN_1 and VIN_2 simultaneously.

underlying scheduling method; one can design a scheduling method to satisfy **A2** and **A3**, e.g., first-come-first-serve and round-robin scheduling. Thus, we only have to consider how to satisfy **A4**, i.e., for each hop the head flit can always find a channel (with an increasing order of channel numbers) in VIN_1 to use.

For a $k_{n-1} \times k_{n-2} \times \ldots \times k_0$ mesh, Fig. 1 shows how to number the output channels in VIN_1 for node X $=(x_{n-1}, x_{n-2}, \ldots, x_0)$. The number of positive-direction channels of dimension j is $\sum_{i=0}^{j-1} k_i + x_j$, where $\sum_{i=0}^{-1} k_i = 0$, and that of negative-direction channels of dimension j is $\sum_{i=0}^{j} k_i - x_j$. The number of higher dimensions is always larger than that of lower dimensions. The numbers of positive- or negative- direction channels increase along the corresponding coordinate.

In Algorithm 3.3, a message may hold the channels in VIN_1 and VIN_2 as shown in Fig. 2. If the message takes one or more hops via the channel in VIN_2, **A4** requires that there always exist an outgoing channel in VIN_1 with a higher channel number than those channels already held by the message. Assume the message holds a channel $VC_{f(R,1),1}$ at node X and takes one or more hops via the channels in VIN_2, then the message arrives at node Y and requests the channel $VC_{f(R,1),1}$. There are two cases to consider: (C1) ($f(R,1)$ in node X) = ($f(R,1)$ in node Y), and (C2) ($|f(R,1)|$ in node X) < ($|f(R,1)|$ in node Y). Note that ($|f(R,1)|$ in node X) $\leq$ ($|f(R,1)|$ in node Y), because Algorithm 3.3 requests the channels of VIN_1 in an increasing dimension order. For C1, $VC_{f(R,1),1}$'s in both X and Y use the same dimension and the same direction. If the message is routed along the positive direction, then $\sum_{i=0}^{s-1} k_i + y_s > \sum_{i=0}^{s-1} k_i + x_s$, where $y_s > x_s$, $s \in \{1, 2, \ldots, n-1\}$ and $x_s, y_s \in \{0, 1, \ldots, k_s\}$. If the message is routed along the negative direction, then $\sum_{i=0}^{s} k_i - y_s > \sum_{i=0}^{s} k_i - x_s$, where $y_s < x_s$, $s \in \{1, 2, \ldots, n-1\}$ and $x_s, y_s \in \{0, 1, \ldots, k_s\}$. Therefore, ($num(VC_{f(R,1),1})$ in node Y) > ($num(VC_{f(R,1),1})$ in node X). For C2, one can easily check if the following inequality holds: ($\sum_{i=0}^{s-1} k_i + y_s$ or $\sum_{i=0}^{s} k_i - y_s$) > ($\sum_{i=0}^{t-1} k_i + x_t$ or $\sum_{i=0}^{t} k_i - x_t$), where $s > t$ and $s \in \{1, 2, \ldots, n-1\}$. Therefore, the increasing channel order in requesting the channels in VIN_1 can be guaranteed, and hence, **A4** is satisfied. □

Can we develop an adaptive non-minimal deadlock-free routing scheme for n-dimensional meshes? One key point in ensuring deadlock-freedom is to make sure that, after taking each hop, a message can always find a channel in VIN_1 in an increasing order of channel numbers. In n-dimensional meshes, the message routed via the channel in VIN_2 with the dimension larger than $f(R,1)$ will not violate the dimension-ordering rule because the channel numbers in higher dimensions always have larger (using the numbering method in Theorem 2) than that in dimension $f(R,1)$. For example, if $R = \{4, 0, -1, 0\}$, then, because $f(R,1) = -1$, the message can be misrouted via the channels in VIN_2 in dimensions 2 and 3. That is, after misrouting the message for one hop, R becomes $\{4, 1, -1, 0\}$ or $\{4, -1, -1, 0\}$, or $\{5, 0, -1, 0\}$. However, it may result in a livelock. One way to avoid the livelock is that the number of misrouting hops along each dimension cannot be greater than a fixed number, but the head flit should use more bits to carry this information.

Dally [32] and Glass and Ni [14] mentioned the fully-adaptive routing in 2D meshes by using only one extra channel in either dimension. In our algorithm, if we don't use any free channel in dimension 0, the minimal fully-adaptive routing can still be supported. This is because the dimension-order routing in VIN_1 can let messages go through the lowest dimension (i.e., $f(R,1)$) in a finite time. Dimension 0 is always the lowest dimension for any source-destination pair, so free channels in dimension 0 are not necessary for supporting minimal fully-adaptive routing. The number of virtual channels required for 2D meshes is therefore the same as the algorithms in [14, 32]. However, for notational convenience, we still use two virtual channels over a physical link in each dimension. Note that the current VLSI technology allows for more than two virtual channels per channel in such lower dimension topologies as 2-dimensional meshes. The iWarp is an example of using 4 virtual channels. Using more virtual channels improve not only performance but also fault-tolerance. In Section 5, we will compare the performance of 2 and 4 virtual-channel schemes in 2-dimensional meshes.

4 Adaptive Fault-Tolerant Deadlock-Free Routing

Link and/or node faults are usually used to model faulty/injured multicomputers. As mentioned before, we use a node fault as a basic faulty element because both communication and computation units can be integrated into a single chip in contemporary multicomputer systems. The fault on each outgoing link can be treated as a corresponding node fault. There are two fault types for supporting fault-tolerant routing: dynamic and static faults [17]. For a system to tolerate dynamic faults, nodes may become faulty or nonfaulty at any time. By contrast, the static case lets faults occur only when the network is shut down. Tolerating dynamic faults can enhance the run-time life of a multicomputer, thus increasing reliability. On the other hand, tolerating static faults can enhance system availability. When a system is running and the number of faulty nodes exceeds the tolerable limit, this system should be shut down to run a diagnostic program and decide node's state. Then, the system can be run again in a degraded mode; for example, unsafe binary subcubes in hypercubes and disconnected rectangular blocks in meshes (to be discussed later). Furthermore, if we need to tolerate dynamic faults, each node needs to be equipped with on-line detection mechanisms.

First, let's consider the problem that the proposed adaptive routing algorithm 3.3 will encounter when a link/node becomes faulty. Assume VIN_1 uses deterministic routing. If no channel in VIN_1 or VIN_2 is available, then the message must wait for only one channel of VIN_1 due to VIN_1's use of deterministic routing. However, if this channel is connected to a faulty node or runs over a faulty link, then it won't be available, which may in turn lead to a deadlock. One can solve this problem by using adaptive routing in VIN_1, such as planar routing [3], enabling each message to request at least two channels in different dimensions so that it can avoid the faulty link/node. However, this method requires more virtual channels than necessary for fully-adaptive routing.

To reduce the hardware overhead in supporting fault-tolerance, we propose a new scheme that treats some of the channels in VIN_2 around the faulty or unsafe nodes[2] as detours of the channels in VIN_1 connected to the faulty/unsafe nodes. Each message waiting for a channel in VIN_1 connected to a faulty/unsafe node can then be routed via the corresponding detour in VIN_2. In effect, this scheme trades adaptability around the faulty nodes for fault-tolerance and deadlock-freedom in the entire system. However, the adaptability thus achieved affects only a part of the system and depends on the number and pattern of faulty nodes.

In order to ensure that fault-tolerant routing is still deadlock-free, each possible path on VIN_1 going through the faulty/unsafe node should find a detour. The overhead of fault-tolerant routing depends on the interconnection topology used. To develop a fault-tolerant routing algorithm, we need two more assumptions in addition to **A1**—**A4** in Section 3:

A5. There is no message from or to the faulty node.

A6. Each node knows its neighbors' state (i.e., faulty, unsafe or safe).

The following two subsections describe the proposed adaptive fault-tolerant deadlock-free routing algorithms for hypercubes and n-dimensional meshes. Due to the limit of space, we discuss briefly the adaptive fault-tolerant deadlock-free routing for k-ary n-cubes.

[2]To be defined later in Definition 1.

4.1 Hypercubes

Fault model for hypercubes: We use unsafe binary subcubes to model node faults in hypercubes [22]. Each node is in one of three states: safe, unsafe, and faulty. The unsafe nodes [22] are used to facilitate deadlock-free routing. A node marked as safe or unsafe can still transmit or receive messages.

Definition 1: A "good" node connected to two or more faulty/unsafe nodes is called an "unsafe" node; otherwise, it is called a "safe" node.

Before running user programs on the machine, the operating system on each node should determine the state of its neighbors. The authors of [21] developed a distributed algorithm of communication complexity $O(n^3)$ to identify all unsafe nodes in an n-cube. Each faulty/unsafe node belongs to a corresponding binary subcube, called an *unsafe binary subcube*, which is formed by faulty and/or unsafe nodes. If the number of faulty nodes is no more than $\lceil n/2 \rceil$, then the message can be routed to its destination via a path of length no greater than the minimal path length plus two[22]. However, if we consider deadlock-freedom, this algorithm needs $n+1$ virtual channels. By contrast, our proposed algorithm requires only two virtual channels, and moreover, it provides partial adaptability when nodes became faulty. The number of faulty nodes that can be tolerated and the maximum message path needed are the same as those of the algorithm in [21, 22].

Detour channels: One way to assign detour channels around an unsafe subcube is that each channel (say, $VC_{m,1}$, $0 \le m \le n-2$) connected to the unsafe subcube is replaced by detour channels: $VC_{m+1,2}, VC_{m+2,2}, \ldots, VC_{n-1,2}$. Note that the subscript i in $VC_{i,j}$ is an unsigned number because in a hypercube a node has only one neighbor in each dimension. Also, there is no need to assign a detour channel for channel $VC_{n-1,1}$ because one of its neighbor nodes is the destination.

Renumbering method: Each VIN_1 channel in dimension m $(0 \le m \le n-2)$ is assigned a channel number m. Fig. 3 shows a 4-dimensional hypercube with the assigned channel numbers. When some nodes become faulty, the channels connected to an unsafe node must be renumbered. The channels in VIN_1 are assigned integer numbers, and the detour channels are assigned non-integer numbers. Specifically, we use the following renumbering method.

1. If the original channel number of $VC_{m,1}$ is m $(0 \le m \le n-2)$[3], then all the detour channels $VC_{m+1,2}, VC_{m+2,2}, \ldots, VC_{n-1,2}$ are assigned non-integer channel numbers such that $m-1 < num(VC_{m+1,2}) < num(VC_{m+2,2}) < \ldots < num(VC_{n-1,2}) < m$.

[3] Note that the maximum of m is equal to $n-2$ rather than $n-1$, because it would be the faulty last node if $m = n-1$

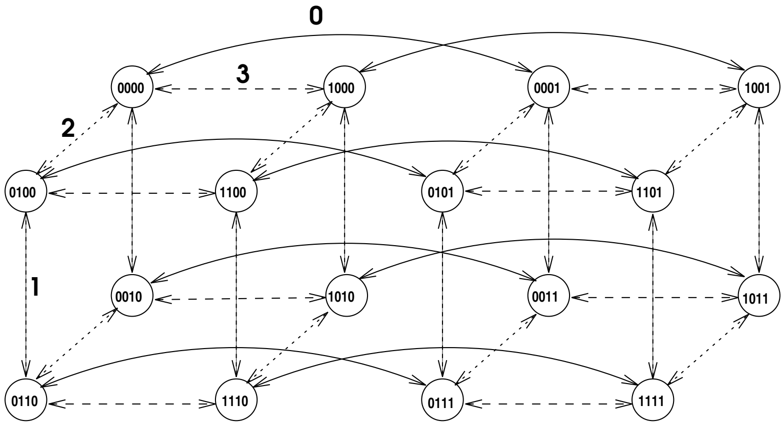

Figure 3: Numbering the output channels in a 4-dimensional hypercube.

2. Each channel in a safe node which is connected to an unsafe node is re-assigned the largest channel number, n.

3. Each channel in an unsafe node which is connected to a safe node is re-assigned the smallest channel number, -1.

Example 1: Consider a 4-dimensional hypercube as shown in Fig. 4. If nodes 0000 and 1010 are faulty, then nodes 1000 and 0010 are unsafe nodes. Thus, nodes 0000, 1000, 0010, and 1010 form an unsafe 2-cube. The safe nodes around this unsafe subcube must build detours. For example, the channel $VC_{3,2}$ in node 0100 is treated as the detour of channel $VC_{2,1}$ in node 0100 (channel number = 2). This detour can be assigned channel number 1.5. Actually, in this case, we can assign any non-integer number between 1 and 2. The channels $VC_{1,2}, VC_{2,2}$, and $VC_{3,2}$ in node 0001 are treated as the detours of channel $VC_{0,1}$ in node 0001, and their channel numbers are assigned to be $-0.9, -0.6$ and -0.3, respectively. Likewise, we can assign any non-integer number such that $-1 < num(VC_{1,2}) < num(VC_{2,2}) < num(VC_{3,2}) < 0$. Following the same rule, each virtual channel in VIN_1 connected to an unsafe subcube can find the detours in VIN_2, and assign proper channel numbers. Fig. 4 shows all the detours around the unsafe subcube. □

Now, we present a fault-tolerant deadlock-free routing algorithm for faulty/injured hypercubes.

Algorithm 4.1: Fault-tolerant adaptive routing for hypercubes.

1. *Update R; /* $r_i := r_i \oplus 1$ */*
2. *If ($R == 0$), route the message to the local processor and exit;*
3.1. */* only one more hop required to reach the destination */*
If ($f(R) == 1$) then Parallel_Request $VC_{f(R,1),1}, VC_{f(R,1),2}$;
3.2. */* messages from an unsafe node to a safe node and misrouting may be used*/*

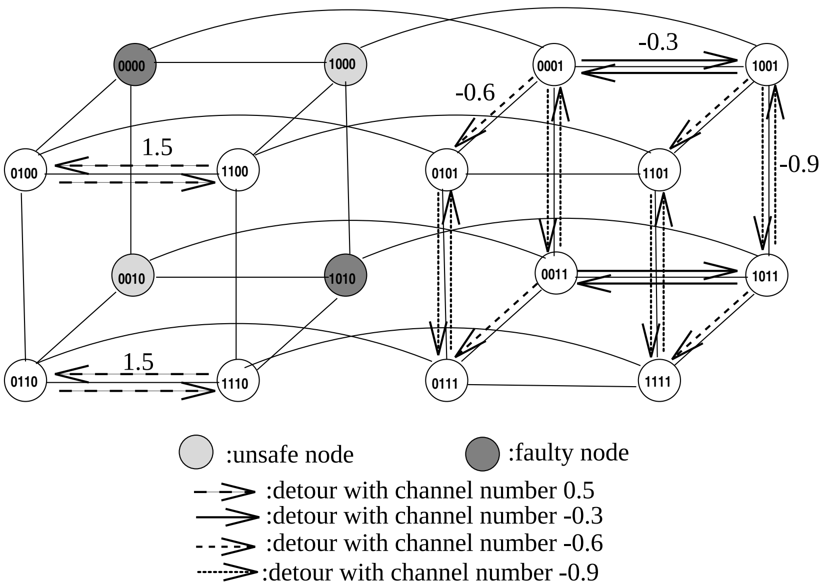

Figure 4: Building detours in a faulty/injured hypercube.

else if (C==unsafe node) then Parallel_Request any channel connected to a safe node;
3.3. */*messages from a safe node to a safe node*/*
else if $VC_{f(R,1),1}$ is connected to a safe node
then Parallel_Request $VC_{f(R,1),1}$ and all useful channels in VIN_2;
3.4. *else Request $VC_{f(R,2),2}$; /* route via detour */*
4. *Route the message via the first available channel requested and exit;*

In Step 3.3, a "useful" channel means that it is not a detour channel and the corresponding element in the routing tag R is nonzero. Basically, Algorithm 4.1 uses minimal routing except for the message whose source and destination nodes belong to the same unsafe subcube and are more than one hop apart. In case the source is a safe node and $VC_{f(R,1),1}$ is connected to a safe node, the message can request simultaneously $VC_{f(R,1),1}$ and any non-detour channel in VIN_2 with the minimal routing distance. If $VC_{f(R,1),1}$ is connected to an unsafe node and more than one hop is required to reach the destination, then the message requests the detour $VC_{f(R,2),2}$ instead. Based on the concept of unsafe binary subcube, if $VC_{f(R,1),1}$ in a safe node is connected to a faulty/unsafe node, then $VC_{f(R,2),2}$ would be connected to a safe node because a safe node is connected to at most one unsafe/faulty node [22]. If only one more hop is required to reach the destination, then the message requests $VC_{f(R,1),1}$ or $VC_{f(R,2),2}$, regardless whether they are connected to an unsafe node or not. So, the message from a safe source node can always reach the destination, because, whenever $VC_{f(R,1),1}$ meets an unsafe/faulty node (except for the final hop), the message can always find a detour $VC_{f(R,2),2}$.

If the source node is unsafe, the message should first leave this unsafe subcube unless its destination is a neighbor of the source. Using the node fault model, a safe node can be found to route the message. After taking this hop, the message will follow the same procedure for the message from a safe node as discussed above.

Example 2: We use the same faulty hypercube as shown in Fig. 4 for this example with the source = 0001 and the destination = 1110. At the source node, because $VC_{f(R,1),1}$ is connected to a faulty node 0000 (Step 3.4), the message is routed via $VC_{f(R,2),2} = VC_{1,2}$ (channel number $= -0.9$). Note that in node 0001, there are three channels defined as detours, i.e., $VC_{1,2}, VC_{2,2}$ and $VC_{3,2}$, and they cannot be used unless the next lowest dimension to be traversed is across a faulty node. In this case, the message is routed via $VC_{1,2}$. After making one hop (i.e., arriving at node 0011), because $VC_{f(R,1),1}$ is connected to an unsafe node 0010 (Step 3.4), the message is routed via $VC_{f(R,2),2} = VC_{2,2}$ (channel number $= -0.6$). Then, in node 0111, because $VC_{f(R,1),1}$ is connected to a safe node 0110 (Step 3.3), the message can request simultaneously the channels $VC_{0,1}, VC_{0,2}$, and $VC_{3,2}$. Assume the message is routed via $VC_{0,1}$ (channel number =0) and arrives at node 0110. Then, $f(R) = 1$, and hence, the last hop is made via $VC_{3,1}$ or $VC_{3,2}$ (Step 3.1).

Let's consider another example with an unsafe source node and an unsafe destination node. Assume source node = 1000 and destination node = 0010. For the first hop, because node 1000 is an unsafe node (Step 3.2), the message is routed via $VC_{0,1}, VC_{0,2}, VC_{2,1}$, or $VC_{2,2}$ (channel number $= -1$). Assume the message is routed via $VC_{2,2}$ and arrives at node 1100. Because $VC_{f(R,1),1}$ is connected to a safe node 1110, the message is routed via $VC_{1,2}$ or $VC_{1,1}$ (Step 3.3). Note that the channel $VC_{3,2}$ is a detour, so it is not eligible for the message to use. Assume the message is routed via $VC_{1,2}$ and arrives at node 1110. Because the $VC_{f(R,1),1}$ is connected to a faulty node, the message is routed via the detour $VC_{f(R,2),2} = VC_{3,2}$ (Step 3.3). Then, the final hop is made via $VC_{2,1}$ or $VC_{2,2}$ (Step 3.1). □

Theorem 3: Algorithm 4.1 is free from both deadlock and livelock if the dimension/size of each unsafe binary subcube is no more than $\lceil n/2 \rceil$.

Proof: Like the proof of Theorem 2, the four assumptions, **A1—A4**, in Section 3 should be satisfied. Again, only **A4** needs to be considered, i.e., after making one hop, the message always can find a higher-number channel to request. Based on Theorem 1, the free channels in VIN_2 do not affect the deadlock, so we consider the channels in VIN_1 only. Since the detour channels cannot be free channels, we can treat them as a part of VIN_1. We must consider four cases for different types of source–destination pairs, i.e., from a safe/unsafe source to a safe/unsafe destination. In each case, we must prove that the requested channel in VIN_1 or a detour channel observes the increasing-order rule to guarantee deadlock-freedom.

Case 1: From a safe source node to a safe destination node.

When a message is routed from a safe source node to a safe destination node, the intermediate nodes' $VC_{f(R,1),1}$ may be connected to unsafe/faulty nodes. If no $VC_{f(R,1),1}$ is connected to an unsafe or faulty node, then we use the dimension-order routing for requesting the channels in VIN_1. Because $num(VC_{f(R,1),1}) < num(VC_{f(R,2),1}) < \ldots < num(VC_{f(R,f(R)),1})$, the dimension ordering can be guaranteed.

If there is an intermediate node whose $VC_{f(R,1),1}$ is connected to an unsafe/faulty node, then the message will request $VC_{f(R,2),2}$ (Step 3.3). Actually, the channel $VC_{f(R,2),2}$ is the detour of channel $VC_{f(R,1),1}$ connected to a safe node. After taking this hop, $VC_{f(R,1),1}$ may meet an unsafe/faulty node again; if it does, route the message via the detour. By the renumbering method, $num(VC_{f(R,1),1}) - 1 < num(VC_{f(R,1)+1,2}) < num(VC_{f(R,1)+2,2}) < \ldots < num(VC_{n-1,2}) < num(VC_{f(R,1),1})$, thus preserving the channel-ordering rule.

Case 2: From an unsafe source node to a safe destination node.

First, the message must leave the unsafe subcube. Based on the fault model of unsafe binary subcubes, the message can always find a channel connected to a safe node if the faulty node is no more than $\lceil n/2 \rceil$ [22]. This channel is assigned the smallest channel number, -1, and is always used for the first hop. After taking this hop, the message will be routed from a safe node to a safe node as described in Case 1. Because the channel number -1 is smaller than any other channel number in Case 1, the channel ordering can be guaranteed.

Case 3: From a safe source node to an unsafe destination node.

Because a safe node is connected to at most one unsafe/faulty node, if the channel $VC_{f(R,1),1}$ is connected to an unsafe/faulty node, then the message can always find a detour $VC_{f(R,2),2}$ except for the final hop. From Case 1, we know that the channels' order can be preserved from a safe source node to a safe node next to the unsafe destination. Then, only one more hop is required to reach the destination. Since each safe channel connected to an unsafe node is assigned the largest channel number, the channel ordering can be guaranteed.

Case 4: From an unsafe source node to an unsafe destination node.

If the source node is connected directly to the destination node, then request $VC_{f(R,1),1}$ or $VC_{f(R,1),2}$. Because only one hop is required in this situation, the channel ordering is guaranteed. If two or more hops are required, the message should leave the unsafe subcube in the first hop as in Case 2. First, the message is routed from an unsafe node to a safe neighbor node, then from a safe node to a safe node (a neighbor of the destination), and finally from a safe node to an unsafe node. Because the first and final hops are made via the channel with the smallest and largest channel numbers, respectively, and intermediate hops (from safe to safe nodes) are made via the channels of increasing numbers as in Case 1, the channel ordering can be guaranteed.

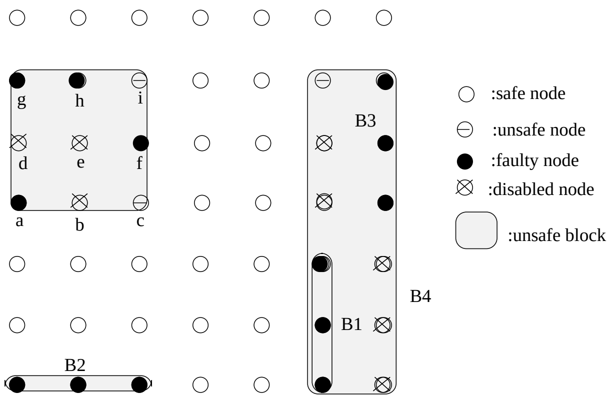

Figure 5: Disconnected rectangular blocks are used as the fault model.

Since the message in each case is routed in increasing channel order, the proposed routing is deadlock-free. Algorithm 4.1 uses minimal routing except for the first hop from the source node in Case 4, and thus, the maximum number of message hops is $n+1$. Therefore, livelock-freedom can be guaranteed. □

4.2 n-Dimensional Meshes

Fault model for n-dimensional meshes: We use disconnected rectangular blocks (or faulty blocks for short) to model node faults and to facilitate deadlock-free routing in n-dimensional meshes and k-ary n-cubes.

In addition to Definition 1 in Section 4.1, we need a rule to disable some unsafe nodes as described in the following definitions.

Definition 2: For n-dimensional meshes and k-ary n-cubes, $k > 2$, if an unsafe node is not encompassed by two or more safe nodes in each two dimensional plane, then this node is marked as faulty. □

Definition 3: Two rectangular blocks B_1 and B_2 in a 2-dimensional plane of a mesh are said to be *disconnected* if for each node $N_1 \in B_1$ and $N_2 \in B_2$, the distance between N_1 and N_2 in at least one dimension is no smaller than 3. □

For a low dimensional mesh, each physical link can be shared by more than two virtual channels. If the distance defined in Definition 3 is equal to 2 instead 3, we need $\binom{n}{2} \times 2 + 1$ virtual channels; that is, 2- and 3- dimensional meshes require 3 and 5 virtual channels, respectively. In this paper, we use two virtual channels. In order to improve performance and fault-tolerance, one can use more virtual channels for low-dimensional meshes. In Fig. 5, the states of nodes (a, f, g, h), (b, d, e), and (c, i) are faulty, disabled (marked as faulty), and unsafe, respectively. Obviously, the faulty block is composed of faulty (or disabled) and unsafe nodes, and is rectangular in each 2-dimensional plane. In Fig. 5, the rectangular faulty blocks B_1 and B_2 are disconnected, whereas B_1 and B_3 are not. Block B_4 is formed by marking some disabled and unsafe nodes between B_1 and B_3. If the faulty block is not larger than a 2×2 square and if no nodes of this faulty block lie on the border of the mesh, then no node needs to be marked as faulty; actually, this is an unsafe subcube. After determining each node's state, unsafe and faulty nodes will form a disconnected rectangular block on each 2-dimensional plane. In order for messages to bypass a rectangular block, both the width and length of the rectangular block should be smaller than the corresponding dimension size. For example, the block size in Fig. 5 must be smaller than the corresponding dimension size, 7. This fault model is referred to as a *block fault model*. Some popular fault models such as a single node/link fault are subsumed by the block fault model. Arbitrary faults can also be modeled as block faults by including some of the non-faulty nodes in the faulty block. Such nodes will be marked as faulty or unsafe.

Detour channel groups: Building a detour channel in an n-dimensional mesh is different from that in a hypercube. There are two different properties: (1) unlike the hypercube, the n-dimensional mesh is not a homogeneous topology; (2) in the n-dimensional mesh, there could be more than one hop a message must travel in *each* dimension, but in a nonfaulty hypercube minimal routing requires at most one hop in *each* dimension. The first property makes it more difficult to develop a fault-tolerant routing algorithm for n-dimensional meshes, because peripheral nodes require special care. From the second property, it is not necessary for the hypercube to find detours for the channels in the highest dimension $n-1$, because if the message requests the channel in dimension $n-1$ on VIN_1, then only one more hop is required to reach the destination. However, for an n-dimensional mesh, we still have to find detours for the channels in dimension $n-1$ because if $|r_{n-1}| \geq 2$, then the destination is not a neighbor of the current node.

Using the same method in the hypercube except for the highest-dimension channel, we can assign the detour channels around a faulty block, i.e., each channel connected to the faulty block (say, $VC_{+m,1}$ or $VC_{-m,1}$, $0 \leq m \leq n-2$) is replaced by the detour channels: $VC_{\pm(m+1),2}, VC_{\pm(m+2),2}, \ldots,$ $VC_{\pm(n-1),2}$. Note that each mesh dimension has two directions, positive and negative. In each 2-dimensional plane of the mesh, the detours around the faulty block can be divided into four detour channel groups (not including the detours of the highest-dimension channels), i.e., detour groups $a, b, c,$ and d as shown in Figs. 6 and 7. For example, the detours at nodes N_1, N_2 and N_3 in Fig. 6 form a detour group a in the positive direction and a detour group b in the negative direction. Each detour belongs to one and only one detour group. In hypercubes, each detour group contains only one detour channel. To avoid deadlocks, a message should follow the direction of detour group, and once the message leaves one detour group, it cannot enter the group again.

If $f(R) \geq 2$ and $VC_{f(R,1),1}$ is connected to a faulty block, then the message should request

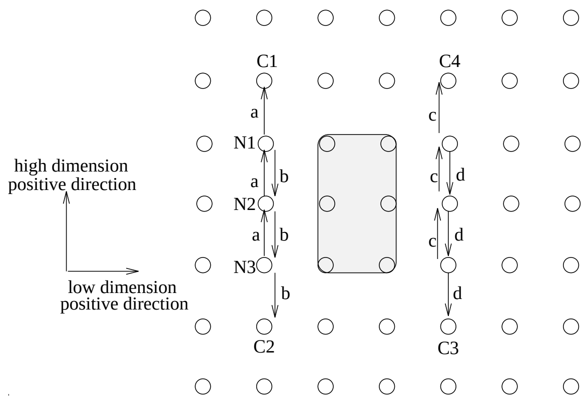

Figure 6: Building the detours in the higher dimension for the channels in VIN_1 in the lower dimension. There are four detour groups : a, b, c and d. C_1, C_2, C_3 and C_4 are the corner nodes around the faulty block.

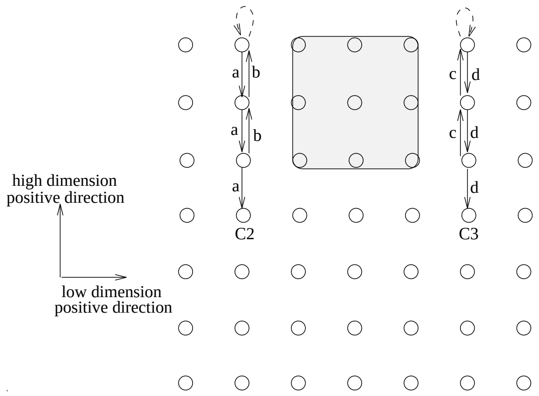

Figure 7: When the message is blocked by the border, it will be turned back along the detour in the opposite direction. There are four detour groups: a, b, c and d.

$VC_{f(R,2),2}$ instead of $VC_{f(R,1),1}$, where $|f(R,2)| > |f(R,1)|$. Fig. 6 depicts this situation. When the message from the left side of the faulty block is routed via either detour group a or b, depending on the direction of $VC_{f(R,2),2}$. After arriving at the corner node C_1 (if routed via group a) or node C_2 (if routed via group b), the message can be routed via the channel in VIN_1 according to the dimension order or routed via any useful free channel in VIN_2. When the message from the right side of the faulty block is routed via either detour group c or d depending on the direction of $VC_{f(R,2),2}$, and after arriving at the corner node C_3 or C_4, it is not routed via the detour channel. Note that when a message is routed via a detour group, only one channel can be used, i.e., in the zone around a faulty block, we use the deterministic routing to avoid the faulty block. For convenience, we use the notation $detour\_group_{f(R,1),f(R,2)}$ with different directions of $f(R,1)$ and $f(R,2)$ to indicate respectively the detour group a if $f(R,1) > 0$ and $f(R,2) > 0$, the detour group b if $f(R,1) > 0$ and $f(R,2) < 0$, the detour group c if $f(R,1) < 0$ and $f(R,2) > 0$, and the detour group d if $f(R,1) < 0$ and $f(R,2) < 0$. If the corresponding routing element in dimension $f(R,2)$ is decremented to zero before reaching the corner node, the message is routed via the new

$detour\_group_{f(R,1),f(R,2)}$.[4] The new $detour\_group_{f(R,1),f(R,2)}$ always uses higher-dimension channels instead of the original $detour\_group_{f(R,1),f(R,2)}$. We follow the same procedure until the message reaches the destination or the corner node, or $f(R)$ is decremented to one.

If $f(R) = 1$, $|f(R,1)| \neq n-1$ and $VC_{f(R,1),1}$ is connected to a faulty block, then the message can request the higher-dimension detour channel $VC_{\pm i,2}$, $|f(R,1)| < i < n$, and should follow the same detour group as shown in Fig. 6. The difference from the case mentioned above is that initially the message has more detour groups to use, but after choosing a detour group, it has only one detour channel to use until it leaves the detour group. Actually, in this case, the message will be routed away from the destination until $VC_{f(R,1),1}$ connects itself to a safe node, i.e., the corner node around the faulty block. If the message routed via detours is blocked by a mesh boundary, then the message will be routed back (also via detours) using the detour group in an opposite direction. Fig. 7 shows that if the message is first routed via detour group b (or c) and blocked by the mesh border, then it has to be turned back via detour group a (or d).

Now, we consider how to find the detour of the highest-dimension channel, i.e., the case of $f(R) = 1$ and $|f(R,1)| = n-1$. We can choose any of $VC_{\pm i,2}$'s, $i < n-1$, to be the detour of $VC_{\pm(n-1),1}$, or choose all of them. However, once a channel in VIN_2 is marked as a detour, it cannot be used as a free channel for routing adaptability. Therefore, if $VC_{\pm(n-1),1}$ is connected to a faulty block, we choose $VC_{\pm 0,2}$ to be the only corresponding detour. However, the detour of $VC_{\pm(n-1),1}$ has one special feature: if the message is routed via a detour, say, $VC_{+0,2}$, then it should be routed back via the detour, say, $VC_{-0,2}$, in the opposite side of faulty block until $f(R) = 1$ and $|f(R,1)| = n-1$ are corrected again. Thus, a detour group is composed of two opposite direction channels called *upper* and *bottom* detour groups. For example, the detour groups e ($detour\_group_{+(n-1),+0}$) and f ($detour\_group_{-(n-1),+0}$) shown in Fig. 8 are composed of the detours on the upper and bottom sides of the faulty block. Note that this situation doesn't occur when the message is routed via the detour of $VC_{\pm i,1}$, $i < n-1$, because it is routed via a higher-dimension detour channel and can be turned back via the channel in VIN_1 only after reaching a corner node around the faulty block. However, for the case with $|f(R,1)| = n-1$, once the message is sent via misrouting, it should be turned back via the detour, because there is no higher channel in VIN_1 that can be used to return the message. Because we use only two virtual channels, if the message in Fig. 8 is blocked by the faulty block, then first route it via the detour in the positive direction (i.e., $detour\_group_{\pm(n-1),+0}$) regardless whether the message is from the positive or negative direction. For example, the message from the upper (or bottom) side of the faulty block is routed via the detour group f (or e). One can also route the message via the detour in the negative direction first, and after bypassing the faulty block, then route back via the detour in the positive direction; however, all messages should

[4] The new $detour\_group_{f(R,1),f(R,2)}$ is different from the original $detour\_group_{f(R,1),f(R,2)}$ because the second routing element in R is decremented to zero. That is, the new $f(R,2)$ is the same with the original $f(R,3)$.

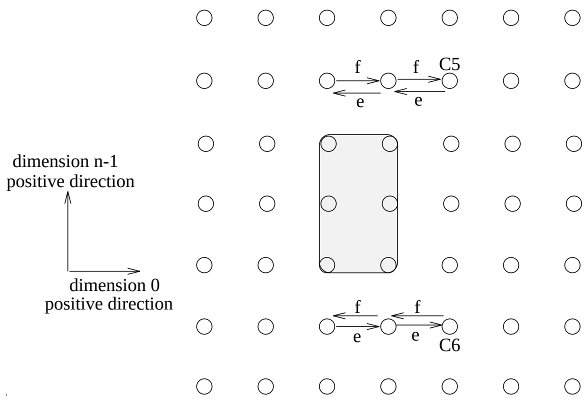

Figure 8: Building the detours in dimension 0 for the channels in VIN_1 in dimension $n-1$. Packets are routed via the right side of faulty block. There are two detour groups: e and f (or $detour\_group_{+(n-1),+0}$ and $detour\_group_{-(n-1),+0}$). C_5 and C_6 are the corner nodes around the faulty block.

follow the same procedure.

Having a message turned back when it is blocked by the mesh border may result in a deadlock as shown in Fig. 9. One way to solve this problem is to ensure that no message needs to be turned back from the mesh border, thus breaking the wait cycle. We can set up a special function in the node $X = \{x_{n-1}, x_{n-2}, \ldots, x_0\}$ with the maximum x_{n-1}, e.g., node B_1 and B_2 in Fig. 10. Once this node finds its dimension $n-1$ neighbor to be faulty or unsafe, it should transmit a message to the neighbor node in the negative direction of dimension 0 to change the direction of dimension 0 detour group, i.e., use $detour\_group_{\pm(n-1),-0}$ instead of $detour\_group_{\pm(n-1),+0}$. The receiving node of this message will also transmit the message to the next node in the negative direction of dimension 0, and so on. This procedure continues as long as the receiving node is connected to the faulty block. To synchronize this function, we need the number of message hops up to the maximum length in dimension 0 minus one. This function can be executed when the system executes a procedure for determining the state of each node. If we use two virtual channels in VIN_2 in dimension 0 (one is the low channel, the other is the high channel), then this function is not necessary. Because if a message is blocked in the highest dimension, then we can route the message via the low detour channel in VIN_2 in dimension 0, then route it back via the high detour channel if the message is blocked by the mesh border. When the message comes to the other side of faulty block, if it is from the high detour channel (i.e., blocked by the border when it is routed via the low detour channel), then route it via the high detour channel; otherwise, route it via the low detour channel. The high detour channel in the other side of faulty block can be guaranteed to change $f(R)$ to 1 again and no message routed via the high detour channel will be blocked by the border. Thus, this is an acyclic requesting chain, i.e., no deadlock will occur. However, this scheme requires three virtual channels in dimension 0, one in VIN_1 and two in VIN_2. Note that in the other dimensions, we can still use two virtual channels.

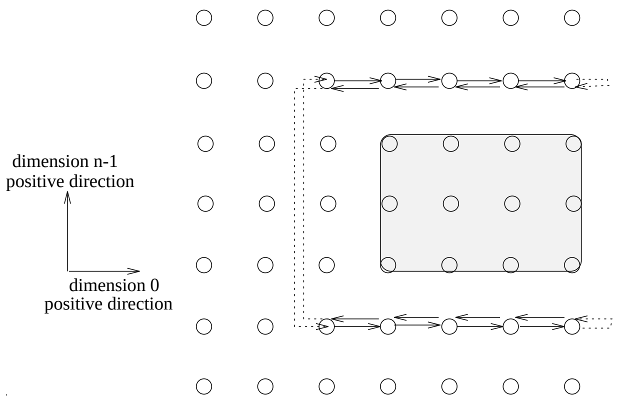

Figure 9: A deadlock may occur if the message is routed via the detour in dimension 0 and turned back after getting blocked by the mesh border.

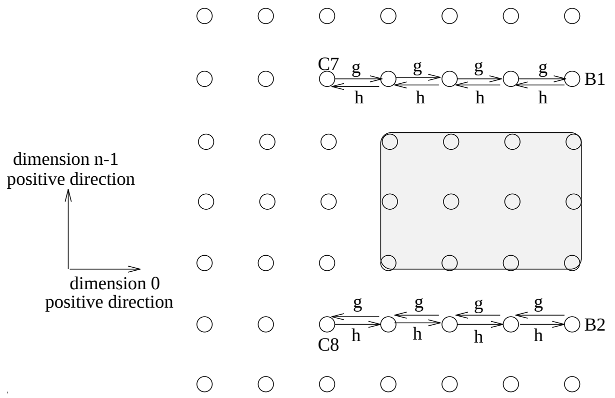

Figure 10: When the faulty block is located on the rightmost side of dimension 0, the message is first routed via the detour $VC_{-0,2}$, after encompassing the faulty block on the left side, it is turned back via $VC_{+0,2}$. There are two detour groups: g and h (or $detour\_group_{+(n-1),-0}$ and $detour\_group_{-(n-1),-0}$). C_7 and C_8 are the corner nodes around the faulty block.

Renumbering method: The channels in VIN_1 are assigned integer numbers as shown in Fig. 1, and the detour channels are assigned non-integer numbers. We use a two-level approach to assign each detour a channel number. In the higher level, we define the range of channel numbers to be assigned to each detour group. In the lower level, channels in the detour group are assigned increasing non-integer numbers along the direction of the detour group. Specifically, we use the following renumbering method.

1. If channel $VC_{+m,1}$, $0 \le m \le n-2$, is connected to a faulty block and the original channel number is k, then the channel numbers of the detours $VC_{\pm(m+1),2}, VC_{\pm(m+2),2}, \ldots, VC_{\pm(n-1),2}$ are assigned to be in the range of their corresponding detour groups as follows: $k-1 < num(detour\_group_{+m,\pm(m+1)}) < num(detour\_group_{+m,\pm(m+2)}) < \ldots < num(detour\_group_{+m,\pm(n-1)}) < k$. $detour\_group_{+m,\pm i}$ indicates $detour\_group_{+m,+i}$ or $detour\_group_{+m,-i}$, $m+1 \le i \le n-1$. The detour channels $VC_{+i,2}$ and $VC_{-i,2}$ belong, respectively, to $detour\_group_{m,+i}$ and

$detour\_group_{m,-i}$, $m+1 \leq i \leq n-1$. Detour channels in each group should be assigned increasing channel numbers along the group's direction. When the faulty block touches the mesh border, the channel in the direction away from the border will be assigned a larger number than that towards the border. For example, channels in the detour group a in Fig. 7 are assigned larger numbers than those in the detour group b.

2. Channel numbers for the detours of channel $VC_{-m,1}$, $0 \leq m \leq n-2$ can be assigned with the same method in the first item. The original channel number is k and the range of channel numbers in detour groups is defined as: $k-1 < num(detour\_group_{-m,\pm(m+1)}) < num(detour\_group_{-m,\pm(m+2)}) < \ldots < num(detour\_group_{-m,\pm(n-1)}) < k$. The detour channel $VC_{i,2}$ and channel $VC_{-i,2}$ belong, respectively, to $detour\_group_{-m,i}$ and $detour\_group_{-m,-i}$, $m+1 \leq i \leq n-1$.

3. If the highest-dimension channel $VC_{+(n-1),1}$ is connected to the upper $detour\_group_{+(n-1),+0}$ (or the bottom $detour\_group_{+(n-1),+0}$) and the original channel number is k' (or k''), then the channel numbers of the detours $VC_{+0,2}$ (next to the bottom of the faulty block) and $VC_{-0,2}$ (next to the top of the faulty block) are assigned to be in the range of their corresponding detour groups as follows: $k' < num($upper $detour\_group_{+(n-1),+0}) < (k'+1)$ and $k'' < num($ bottom $detour\_group_{+(n-1),+0}) < (k''+1)$. Based on the numbering method in Fig. 1, k'' is larger than k'. Detour channels in each group should be assigned increasing channel numbers along the group's direction. If the faulty block is on the right-side border in dimension 0, the detour will belong to $detour\_group_{+(n-1),-0}$ instead of $detour\_group_{+(n-1),+0}$, i.e., the direction will be reversed.

4. Channel numbers for the detours of channel $VC_{-(n-1),1}$ can be assigned by using the same method as in the third item. Assume the channel $VC_{-(n-1),1}$ is connected to the upper $detour\_group_{-(n-1),+0}$ (or bottom $detour\_group_{-(n-1),+0}$) and its original channel number is k' (or k''). The range of channel numbers for a detour group is assigned as: $k' < num($upper $detour\_group_{-(n-1),+0}) < (k'+1)$ and $k'' < num($bottom $detour\_group_{-(n-1),+0}) < (k''+1)$. Again, $k'' > k'$. Channels in each detour group should be assigned increasing numbers along the group's direction. If the faulty block is on the right-side border in dimension 0, the detour belongs to $detour\_group_{-(n-1),-0}$ instead of $detour\_group_{-(n-1),+0}$, i.e., the direction will be reversed.

5. Each channel in a safe node connected to an unsafe node is re-assigned the largest channel number.

6. Each channel in an unsafe node connected to a safe node is re-assigned the smallest channel number.

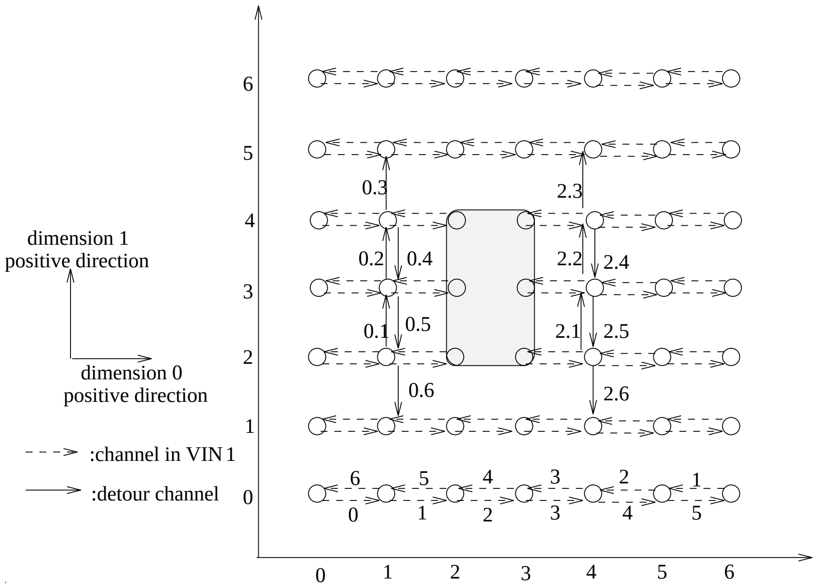

Figure 11: Numbering the detour channels for a 7×7 mesh.

Example 3: Consider a 7×7 mesh as shown in Fig. 11. Based on the numbering method in Fig. 1, we assign the channel numbers in VIN_1. In Fig. 11, we label the channel numbers in VIN_1 along dimension 0, but show labels only in one row. $VC_{+0,1}$ in node $X = (1, i), 0 \leq i \leq 6$, is assigned 1 and $VC_{-0,1}$ in node $Y = (4, i), 0 \leq i \leq 6$ is assigned 3. So, the detour $VC_{+1,2}$ (or $VC_{-1,2}$) in node X belonging to $detour\_group_{+0,+1}$ (or $detour\_group_{+0,-1}$) can be assigned within the range between 0 and 1. We choose 0.1, 0.2, and 0.3 (or 0.4, 0.5, and 0.6). One can assign channels any increasing numbers within this range along the group's direction. Likewise, the detour of node Y can be assigned a number within the range between 2 and 3, and in this case, we use channel numbers 2.1, 2.2, and 2.3 (or 2.4, 2.5, and 2.6). For a higher-dimensional mesh, we can first choose the range for each detour group and assign a higher detour group a higher-number range. Then, a detour channel in each group can be assigned a number along the corresponding group's direction as shown in this example. □

Algorithm 4.2: Adaptive fault-tolerant routing for n-dimensional meshes.

1. *Update R; /* $r_i := r_i \pm 1$ */*
2. *If $(R == 0)$, route the message to the local processor and exit;*
3.1. */* only one more hop required to reach destination */*
If $((f(R) == 1)$ && $(|f(R,1)| == 1))$
then Parallel_Request $VC_{f(R,1),*}$;
3.2. */* messages from an unsafe node to a safe node and misrouting may be used*/*
else if (C == unsafe node)
then Parallel_Request any channel connected to a safe node;
3.3. *else if $VC_{f(R,1),1}$ is connected to a safe node*
then Parallel_Request $VC_{f(R,1),1}$ and all useful channels in VIN_2;

3.4. *else if* $f(R) \geq 2$
then Request $VC_{f(R,2),2}$*; /* routed via the detour of* $detour\_group_{f(R,1),f(R,2)}$ **/*
3.5. *else if* $|f(R,1)| \neq n-1$
then Request $detour\_group_{f(R,1),\pm i}$ *until reaching the corner node;*
*/** $|i| > |f(R,1)|$*; If blocked by the border, then reverse the direction, i.e., routed via* $detour\_group_{f(R,1),\mp i}$*. */*
3.6. *else if the faulty block is on the rightmost border in dimension 0*
then Request $detour\_group_{f(R,1),+0}$ *until* $f(R) = 1$ *is corrected;*
3.7. *else Request* $detour\_group_{f(R,1),-0}$ *until* $f(R) = 1$ *is corrected;*
4. Route the message via the first available channel requested and exit;

Basically, Algorithm 4.2 uses minimal routing if the shortest path between the current node and the destination is not blocked (Step 3.3). If $f(R) \geq 2$ and if the channel $VC_{f(R,1),1}$ is blocked by a faulty block, then the message requests the detour $VC_{f(R,2),2}$ (Step 3.4). If $f(R) = 1$ and if the message is blocked by a faulty block, we first need to go around the faulty block by misrouting. If $|f(R,1)| \neq n-1$, then the message will request higher-dimension detours and follow the same detour group until reaching the corner node around the faulty block (Step 3.5). If the message is already in the highest dimension, then it should find a detour in dimension 0. The message is first routed via channel $VC_{+0,2}$, then via $VC_{f(R,2),1}$ (i.e., $VC_{n-1,1}$ or $VC_{-(n-1),1}$), and finally via $VC_{-0,2}$ to get to the destination, or let $f(R) = 1$ again. Note that if a message is blocked by a faulty block and $|f(R,1)| = n-1$, then it is routed via the detour in the positive direction of dimension 0 to go around the faulty block, and finally routed back via the detour in the negative direction of dimension 0, i.e., $detour\_group_{f(R,1),+0}$ (Step 3.7). However, if the faulty block is on the rightmost side of dimension 0, then first route the message via the negative direction of dimension 0 and finally route it back via the positive direction of dimension 0, $detour\_group_{f(R,1),-0}$ (Step 3.6). If $|f(R,1)| \neq n-1$, the message can be routed via either the positive or negative direction of a higher dimension.

However, when we use misrouting, the message could be blocked by a mesh border. In such a case, the message cannot go around the faulty block. In Algorithm 4.2, we just return the message via the detour in the opposite direction. Because in our fault model, the edge size of a faulty block is smaller than the corresponding dimension, the message can always go around the faulty block either in the positive or negative direction. Since the correct direction is already set up for the detours of highest-dimension channels, the message will not be blocked by the border, even when we use misrouting.

Theorem 4: Algorithm 4.2 is deadlock-free if the node fault is subjected to the block fault model.

Proof: Similar to the proof of Theorem 3, we must prove that the requested channels in VIN_1 and the detour channels in VIN_2 follow an increasing channel order, thus satisfying **A4**. Again,

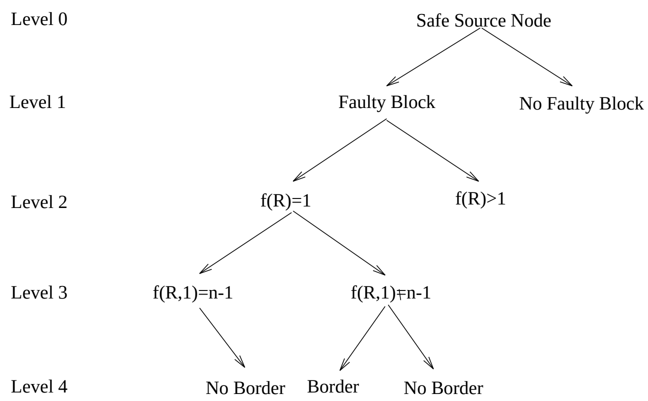

Figure 12: A tree structure to describe the situation for routing a message from a safe source to a safe destination.

we must consider four cases for different source-destination pairs. Each channel in VIN_1 and the detour channels are assigned numbers according to the proposed renumbering method.

(1). *From a safe source node to a safe destination node*:

There are several possible subcases for routing a message from a safe source to a safe destination. Fig. 12 shows a tree structure to describe the situations the message may encounter. For the first level, the channel $VC_{f(R,1),1}$ may or may not be connected to a faulty block. If $VC_{f(R,1),1}$ isn't connected to a faulty block, then it is a fault-free case which can be handled by Algorithm 3.3, thus guaranteeing the channel ordering. If $VC_{f(R,1),1}$ is connected to a faulty block, then the message should request the detour instead of $VC_{f(R,1),1}$. In level 2, if $f(R) = 2$, then the message will request the detour $VC_{f(R,2),2}$. Based on the renumbering method (1) and (2), $num(VC_{f(R,1),1}) - 1 < num(detour\_group_{f(R,1),f(R,2)}) < num(VC_{f(R,1),1})$. Also, the channel $VC_{f(R,2),2} \in detour\_group_{f(R,1),f(R,2)}$ and the channel numbers in the same detour group are assigned in an increasing order along the corresponding direction. Thus, the channel ordering can be guaranteed. However, after taking $detour\_group_{f(R,1),f(R,2)}$, the message may be blocked by a faulty block again. Then it is routed via a higher-dimension detour group (Step 3.4). Based on the renumbering method (1) and (2), the channel number of channel group in the higher dimension is higher than that in a lower dimension.

In level 2, if $f(R) = 1$ then there are two cases: $|f(R,1)| = n-1$ or $\neq n-1$. If $f(R,1) \neq n-1$, the message can request any higher-dimension detour group and will be routed by misrouting. In such a case, the message will first be routed away from the destination, so it may be blocked by a mesh border. In the latter case, the message has to be turned back. Based on the renumbering method (1) and (2), an increasing order can be guaranteed. In level 3, if $|f(R,1)| = n-1$ then the message has to request $detour\_group_{f(R,1),+0}$, or request $detour\_group_{f(R,1),-0}$. Based on the renumbering method (3) and (4), an increasing channel order can be guaranteed.

(2). *From an unsafe node to a safe node*:

The message first finds a safe neighbor (this is guaranteed by the fault model). The channel from a safe node to an unsafe node is assigned the smallest number. This channel is taken only in the first hop. After this hop, the message will be routed from a safe node to a safe node as in Case 1, thus guaranteeing the channel ordering.

(3). *From a safe node to an unsafe node*:

Whenever $VC_{f(R,1),1}$ is connected to a faulty block, the message can be routed via a detour until $f(R) = 1$. If $f(R) = 1$ and the message is still blocked by the faulty block, then route it by misrouting. Based on Definition 3, for each 2-dimensional plane of the n-dimensional mesh, an unsafe node has at least two links (in different dimensions) connected to safe nodes. When the plane uses misrouting, the misrouting path can cover three edges of a faulty block. Hence, at least one edge can reach a safe node next to the destination. Also, each channel from a safe node to an unsafe node is assigned the largest number. Thus, the channel ordering can be guaranteed.

(4). *From an unsafe node to an unsafe node*:

The message first leaves the faulty block (as Case 2), then from a safe node to the unsafe destination (Case 3). Because the channel used in the first/last hop is assigned the smallest/largest number, the channel ordering can be guaranteed.

In all cases, the requested channels will follow an increasing order, thus guaranteeing deadlock-freedom. □

Now, we would like to discuss how to develop an adaptive fault-tolerant routing algorithm for k-ary n-cubes. Since the k-ary n-cube is a homogeneous topology, special cases for its borders do not exist and hence, fault-tolerant routing for the k-ary n-cube should be simpler than for n-dimensional meshes. Similar to the n-dimensional mesh case, we first have to find the faulty block using the disconnected rectangular block model. Since deterministic routing for k-ary n-cubes needs two virtual channels [8] (one is the high channel and the other is the low channel), adaptive routing requires three virtual channels according to Algorithm 3.1. Considering fault-tolerant routing, both high and low channels in VIN_1 need to find the detour in VIN_2 to use. We need two virtual channels in VIN_2 to be assigned as the detours, i.e., one for the high channel in VIN_1 and the other for the low channel. Thus, we need a total of four virtual channels, two in VIN_1 and two in VIN_2. One can then develop an adaptive fault-tolerant deadlock-free routing for k-ary n-cubes similar to the n-dimensional mesh except for the special case of the mesh border. Due to the limit of space, we omit the details of this algorithm.

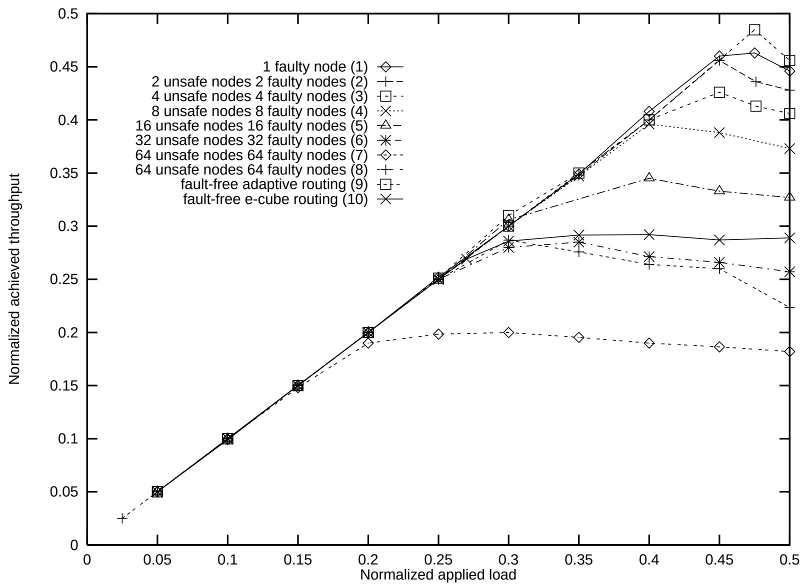

Figure 13: Normalized achieved throughput vs. normalized applied load for 8-cubes.

5 Performance Evaluation

To evaluate the performance of the proposed routing algorithms, we have carried out time-step simulations at the flit level. This simulator is written in C++ and can be used for wormhole routing in hypercubes and meshes with and without faults. In this simulator, each node consists of a processor, a crossbar switch, a router, and virtual channels (buffers). The following parameters are used for the simulation experiments.

P1. The message length is exponentially distributed with a mean of 20 flits, including head and tail flits.

P2. The destination of each message is distributed uniformly, and the inter-message interval is distributed exponentially.

P3. A faulty node is generated randomly subject to the fault model described in the last section.

P4. The crossbar switch in the router allows multiple messages to traverse a node simultaneously without interference.

P5. Each virtual channel uses one input buffer and one output buffer. Each buffer can store only single flit.

P6. Multiple messages can be sent and received simultaneously between a processor and the corresponding router.

P7. A flit can be transferred from an input buffer in node A to an input buffer in node B during one flit cycle, where nodes A and B are neighbors.

P8. A round-robin scheduling policy is used to arbitrate the multiple requests in virtual channels and physical links, except for the messages from (or to) processors that possess the highest priority.

P9. We ran each simulation for 20,000 flit times and discarded information obtained during the first 2,000 flit times (i.e., a warmup period).

Different parameters may provide different results. For example, an efficient mechanism is necessary to transfer messages between processors and routers; otherwise, many messages will be queued in source nodes even under light traffic. To evaluate the efficiency of a routing algorithm, we are more interested in network queueing delay than source queueing delay. Therefore, in **P6** we assume that multiple messages can be sent and received simultaneously between a processor and a router.

The most important performance measures are message latency and network throughput. Many different units have been used to express performance measures, including flits injected/cycle, flits/ns, bits/cycle, normalized bandwidth, etc. As a result, one has to convert units if he wants to compare one result with others. So, we use normalized bandwidth; this representation of simulation results was proposed by Synder [25]. Normalized bandwidth simply expresses the load or throughput as a fraction of the bisection bandwidth limited by the maximum bandwidth of the network under uniform random traffic. Essentially, this limit corresponds to normalized throughput = 1.0 and it is derived by considering the fact that 50% of uniform random traffic crosses the bisection line/plane of the network. Thus, if a network has bisection bandwidth B flits/cycle, each node in an N-node network can inject a maximum of 2B/N flits/cycle. An optimal routing algorithm could handle such a load without saturation. The normalized throughput is equal to the number of messages received over the number of messages that can be transmitted at the maximum load, 2B/N. The normalized applied load is equal to the traffic density[5] over the maximum load 2B/N. The bisection bandwidth B is equal to 2^n and $2n$ for n-cubes and $n \times n$ meshes, respectively. In what follows, we present the simulation results for 8-cubes and 16×16 meshes for a variety of workloads and fault patterns. Duato [12] has shown that the adaptive routing algorithm for hypercubes scales well with network size. Due to the limit of space, we don't consider the scalability issue.

5.1 Simulation results for hypercubes

Figs. 14 and 13 plot the latency and throughput, respectively, for different workloads and fault patterns for 8-cubes. For fault-free hypercubes, λ_s [6] for e-cube routing (without any virtual channel) and adaptive routing are 0.25 and 0.475, respectively. The message latency is from 24 flit cycles to 75 flit cycles before saturation. We found the curve 10 in Fig. 14 to have a similar shape to the simulation results in [12] for the adaptive routing algorithm with 2 virtual channels, confirming the

[5]traffic density = (average message length)/(average message inter-arrival time $\times$ flit number that can be transmitted per cycle)

[6]λ_s is the value of λ at which the network becomes saturated.

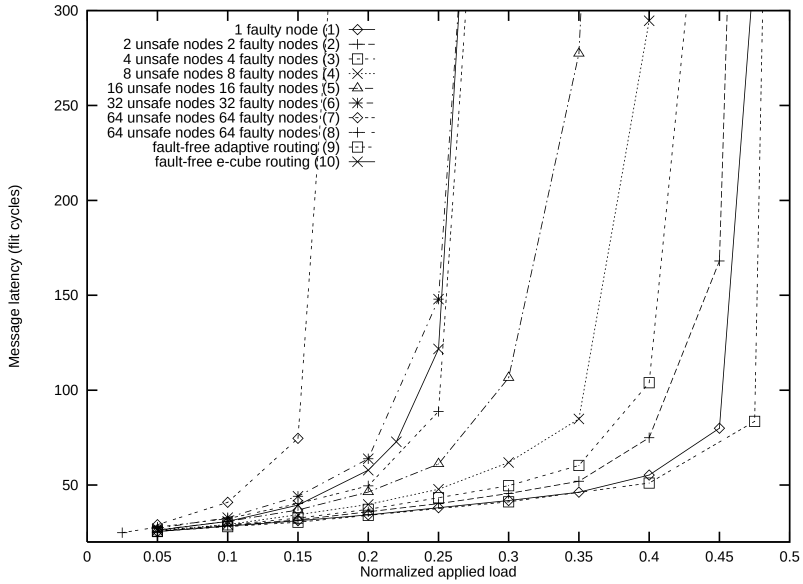

Figure 14: Messages latency vs. normalized applied load for 8-cubes.

correctness of our simulation results. However, it is difficult to compare our results with others' because of the differences in the underlying simulation parameters, such as the number of virtual channels, topology, flit scheduling, and the length of simulation time.

To evaluate the performance degradation caused by the occurrence of faults, we ran simulations with 8 different fault patterns, i.e., a single node fault and k-dimensional unsafe subcubes, $2 \leq k \leq 7$. One half of the nodes in an unsafe subcube are assumed to be faulty. The reason for this assumption is to get average numbers. Under the proposed routing algorithm, the number of free channels assigned as detours depends on the underlying fault patterns. If a faulty/unsafe node is connected along dimension k_1, all free channels along dimension $k_2 > k_1$ (except for the highest dimension) are assigned to be detours. So, when every faulty/unsafe node is connected to a safe node in dimension 0, one can get the worst-case performance degradation caused by faults. On the other hand, if the faulty/unsafe node is connected along the highest dimension, there is no need for assigning detours. Curves 7 and 8 in Figs. 14 and 13 show these two extreme cases for $(n-1)$-dimensional unsafe subcubes with 64 faulty nodes. The value of λ_s is reduced from 0.25 to 0.15. For all fault patterns of an unsafe $(n-1)$-cube with 64 faulty nodes, λ_s ranges anywhere between 0.25 and 0.15. For curves $1-6$ and 8, we consider only the worst case, i.e., every faulty node is chosen from only even-numbered (or odd-numbered) nodes without mixing the two types. Any 8-cube with an unsafe subcube of the same size as in curves $1-6$ and 8 in which a half of the nodes are faulty has performance better than, or equal to, those shown in curves $1-6$ and 8. From this series of simulations, the message latency of adaptive routing in a network with an unsafe subcube smaller than a 4-cube is found to be smaller than that of e-cube routing without virtual channels.

5.2 Simulation results for 2-dimensional meshes

Figs. 16 and 15 plot the message latency and throughput, respectively, for different workloads and fault patterns within 8×8 meshes. For e-cube routing without virtual channels, the network becomes saturated after $\lambda = 0.35$. For adaptive routing with 2 virtual channels, $\lambda_s = 0.45$. We can use more virtual channels for low-dimensional topologies like 2-dimensional meshes. For adaptive routing with four virtual channels, $\lambda_s = 0.7$. Using more virtual channels can not only improve the throughput and latency, but also enhance the fault-tolerance because a less-constrained fault model can be used, i.e., the minimum distance between faulty blocks can be two. In this paper, due to the limit of space we show only the performance of a network with 2 virtual channels. Each faulty block consists of a single faulty node and the total number of faulty nodes ranges from 4 to 24. The reason for this assumption is that we need a fault pattern that yields a small variation in evaluating performance measures. Fig. 15 plots the performance measures of fault-tolerant routing with 4 faulty nodes which provides lower message latency and higher network throughput as compared with the fault-free e-cube routing. The saturation points λ_s for fault-tolerant routing with 8, 16, and 24 faulty nodes are 0.3, 0.25, and 0.25, respectively. Obviously, they are worse than e-cube routing (the performance with 8 faulty nodes is very close to that of e-cube routing). When faults occur, the network gets saturated quickly. One solution is to use more virtual channels. We indeed found that the performance can be improved greatly with more virtual channels. When 4 virtual channels were used, the performance of the proposed fault-tolerant routing with the fault patterns described above is better than, or similar to, that of e-cube routing.

6 Conclusion

Wormhole routing has become popular due mainly to its reduced need of buffers and support for efficient communication. When the head flit is blocked by a busy buffer, it will busy-wait for the buffer to be available, thereby introducing a possible deadlock. To avoid deadlocks, most multicomputers use deterministic routing schemes, such as dimension-order routing. Under deterministic routing, there is only one path between the source and destination. Adaptive routing, on the other hand, can choose one of the multiple paths between the source and destination based on a local traffic condition[7] to avoid the busy link, but it makes deadlock-freedom more difficult.

We proposed an adaptive routing algorithm which decomposes the network into two virtual

[7]It is impractical to use a global condition for all but simple multicomputer systems.

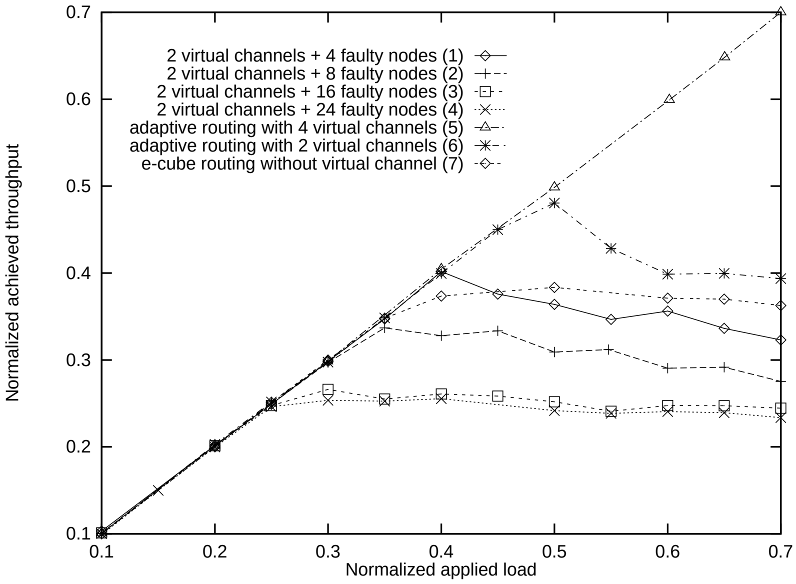

Figure 15: Normalized achieved throughput vs. normalized applied load for 8×8 meshes.

interconnection networks, VIN_1 and VIN_2. VIN_1 supports deadlock-free routing and VIN_2 is used as the source of free channels to enhance adaptability. Whenever a channel in VIN_1 or VIN_2 is available, a message can be routed via this channel. We applied this algorithm to n-dimensional meshes, requiring two virtual channels except for the highest dimension which needs only one virtual channel for supporting fully-adaptive routing. It is applicable to hypercubes, since an n-dimensional hypercube can be treated as a special case of n-dimensional mesh. This algorithm can also be applied to k-ary n-cubes with three virtual channels, two in VIN_1 and one in VIN_2.

When a multicomputer is built with thousands of processing nodes, the probability of one or more nodes becoming faulty is high. Moreover, a faulty node may cause a communication deadlock. It is therefore important to design a deadlock-free fault-tolerant routing algorithm. The number of required virtual channels should not be too large to implement an efficient crossbar switch in the communication router. The numbers of virtual channels required to support our adaptive fault-tolerant routing scheme are two, two, and four, respectively, for hypercubes, n-dimensional meshes, and k-ary n-cubes.

A node is in one of three states: safe, unsafe, and faulty. Each node is assumed to know the state of its neighbors. This state information can help messages avoid faulty nodes and communication deadlocks. In case of an n-dimensional hypercube, our scheme can tolerate any fault pattern composed of up to $\lceil n/2 \rceil$ faulty nodes. If the faulty nodes form an $(n-1)$-cube, then it can tolerate up to 2^{n-1} nodes. For n-dimensional meshes and k-ary n-cubes, we use a disconnected rectangular block as the fault model. Any arbitrary combination of faults can be modeled as a faulty rectangular block.

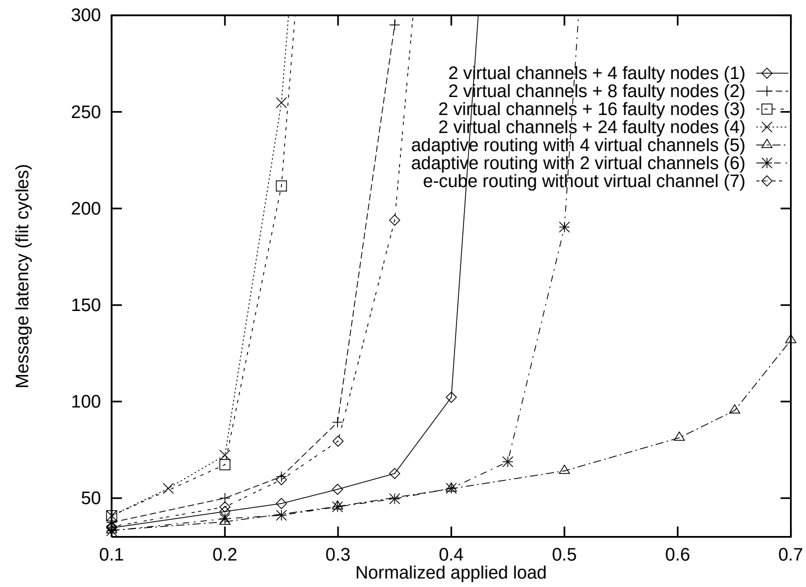

Figure 16: Messages latency vs. normalized applied load for 8×8 meshes.

Using the state information of a node's neighbors, some channels in VIN_2 are chosen as detours of the channels in VIN_1 connected to an unsafe/faulty node. We presented adaptive fault-tolerant routing algorithms for n-dimensional hypercubes and meshes, both of which use only two virtual channels. Another advantage of our fault-tolerant routing scheme is that it is still adaptive if the message does not encounter any unsafe/faulty node on its way to the destination. For k-ary n-cubes, this adaptive fault-tolerant routing scheme needs four virtual channels, two in VIN_1 and two in VIN_2.

To evaluate the proposed algorithms, we have carried out time-step simulations of the network operation at the flit level. 8-cubes and 16×16 meshes are simulated. From the simulation results, networks with up to a 5-dimensional unsafe subcube in which a half of the nodes are faulty) can get smaller latency and higher throughput as compared to that of fault-free e-cube routing. In case of 6-dimensional unsafe subcube, the performance measures are similar to those of e-cube routing. For 16×16 meshes, we assume each faulty block to consist of a single faulty node and the number of random faulty nodes are 4, 8, 16 and 24. The simulation results have shown the network to get saturated after the applied normalized load is 0.4, 0.3, 0.25, 0.25, when the number of faulty nodes is 4, 8, 16, and 24, respectively. By reflecting the reality of low-dimensional topologies, we can use more virtual channels to improve the performance.

References

[1] M.-S. Chen and K. G. Shin, "Adaptive fault-tolerant routing in hypercube multicomputers," *IEEE Trans. on Computers*, vol. 39, no. 12, pp. 1406–1416, Dec. 1990.

[2] M.-S. Chen and K. G. Shin, "Depth-first search approach for fault-tolerant routing in hypercube multicomputers," *IEEE Trans. on Computers*, pp. 152–159, Apr. 1990.

[3] A. A. Chien and J. H. Kim, "Planar-adaptive routing: Low-cost adaptive networks for multiprocessors," *in 19th Annual International Symposium on Computer Architecture*, pp. 268–277, 1992.

[4] W. Chou, A. W. Bragg, and A. A. Nilsson, "The need for adaptive routing in the chaotic and unbalanced traffic environment," *IEEE Trans. on Communications*, pp. 481–490, Apr. 1981.

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

[6] W. J. Dally, "The message-driven processor: a multicomputer processing node with efficient mechanisms," *IEEE Micro*, pp. 23–39, April 1992.

[7] W. J. Dally, "Virtual-channel flow control," *IEEE Trans. on Computers*, vol. 3, no. 2, pp. 194–205, March 1992.

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

[9] J. Duato, "On the design of deadlock-free adaptive routing algorithms for multicomputers: Design methodologies," *in Proc. Parallel Architectures and Languages Europe*, pp. 391–405, June 1991.

[10] J. Duato, "On the design of deadlock-free adaptive routing algorithms for multicomputers: Theoretical aspects," *in Proc. 2nd European Distributed Memory Computing Conf.*, pp. 234–243, April 1991.

[11] J. Duato, "Improving the efficiency of virtual channels with time-dependent selection functions," *in Proc. Parallel Architectures and Languages Europe*, pp. 635–650, June 1992.

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

[13] P. T. Gaughan and S. Yalamanchili, "Pipelined circuit-switching: A fault-tolerant variant of wormhole routing," *in Proc. of the IEEE Symp. on Parallel and Distributed Processing*, Dec. 1992.

[14] C. Glass and L. Ni, "Maximally fully adaptive routing in 2d meshes," *in Proc. of the 1992 International Conference on Parallel Processing*, pp. I101–104, August 1992.

[15] C. J. Glass and L. M. Ni, "Adaptive routing in mesh-connected networks," *in Proc. of the 1992 International Conference on Distributed Computing Systems*, pp. 12–19, 1992.

[16] C. J. Glass and L. M. Ni, "The turn model for adaptive routing," *in Proc of the 1992 International Symposium on Computer Architecture*, pp. 278–287, 1992.

[17] C. J. Glass and L. M. Ni, "Fault-tolerant wormhole routing in meshes," *in Proc. of IEEE 23th Int'l Symposium on Fault-Tolerant Computing*, pp. 240–249, 1993.

[18] I. S. Gopal, "Prevention of store-and-forward deadlock in computer networks," *IEEE Trans. on Communications*, pp. 1258–1264, December 1985.

[19] P. Kermani and L. Kleinrock, "Virtual cut–through: A new computer communication switching technique," *Computer Networks*, vol. 3, pp. 267–286, Sep. 1979.

[20] J. Kim and K. G. Shin, "Deadlock-free fault-tolerant routing in injured hypercubes," *IEEE Trans. on Computers*, vol. 42, no. 9, pp. 1078–1088, Sep. 1993.

[21] T. C. Lee and J. P. Hayes, "Routing and broadcasting in faulty hypercube computers," *in Proc. 3rd Conf. Hypercube Concurrent Comput. and Appl.*, pp. 346–354, Jan. 1988.

[22] T. C. Lee and J. P. Hayes, "A fault-tolerant communication scheme for hypercube computers," *IEEE Trans. on Computers*, pp. 1242–1256, Oct. 1992.

[23] X. Lin, P. K. McKinley, and L. M. Li, "The message flow model for routing in wormhole-routed networks," *in Proc. of Int. Conf. on Parallel Processing*, vol. 1, pp. I294–I297, Aug. 1993.

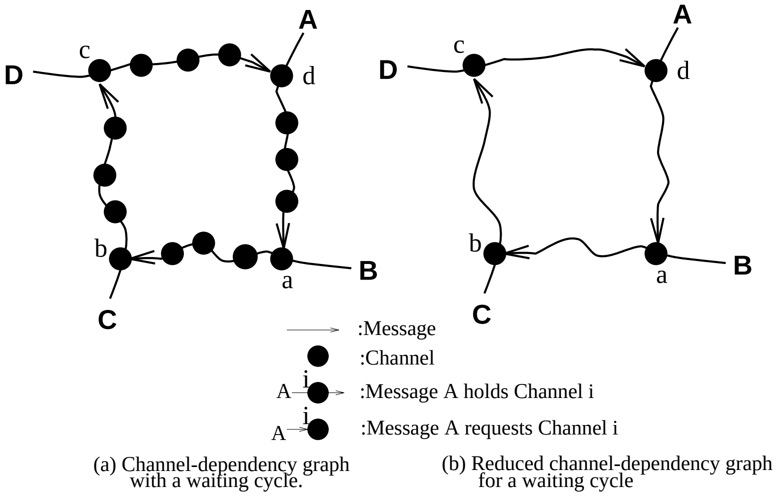

Figure 17: A waiting cycle of four messages.

[24] D. H. Linder and J. C. Harden, "An adaptive and fault tolerant wormhole routing strategy for *k-ary n*-cubes," *IEEE Trans. on Computers*, vol. 40, no. 1, pp. 2–12, Jan. 1991.

[25] T. D. Nguyen and L. Snyder, "Performance of a minimal adaptive router," *in Proc. of Parallel Computer Routing and Communication Workshop, Seattle, Washington*, pp. 31–44, May 1994.

[26] L. M. Ni and P. K. McKinley, "A survey of wormhole routing techniques in direct networks," *IEEE Computer*, pp. 62–76, Feb. 1993.

[27] D. Reed and R. Fujimoto, editors, *Multicomputer Network: Message-Based Parallel Processing*, MIT Press, Cambridge, Mass, 1987.

[28] J. Sutton, P. Wiley, and C. Peterson, "iWarp: a 100-MOPS, LIW microprocessor for multicomputers," *IEEE Micro*, pp. 26–29, June 1991.

[29] D. Talia, "Message-routing systems for transputer-based multicomputers," *IEEE Micro*, pp. 62–72, June 1993.

[30] P. T.Gaughan and S. Yalamanchili, "A family of fault-tolerant routing protocols for direct multiprocessor networks," *to appear in IEEE Trans. on Parallel and Distributed Systems*, 1994.

[31] W.D.Hillis, *The Connection Machine*, MIT Press, Cambridge, Mass, 1985.

[32] W.J.Dally, "Fine-grain message passing concurrent computers," *in the Proc. of the Third Conference on Hypercube Concurrent Computers*, pp. 2–12, January 1988.

[33] X. Zhang, "System effects of interprocessor communication latency in multicomputers," *IEEE Micro*, pp. 12–15 and 52–55, Apr. 1991.

Theorem 1: The proposed adaptive routing algorithm 3.1 is deadlock-free.

Proof: Since the wormhole routing is used, if the head flit advances one hop closer to the destination, then the subsequent flits will progress, without getting blocked, one hop on the same channel path; otherwise, the subsequent flits will be blocked. In the latter case, we first check the waiting cycle in the channel-dependency graph, as shown in Fig. 17(a), where the arrowed lines represent messages and the black circles indicate the channels requested or held by the messages. A channel with blocked messages is called a *blocked channel*, e.g., channels a, b, c, and d in Fig. 17(a). Using the

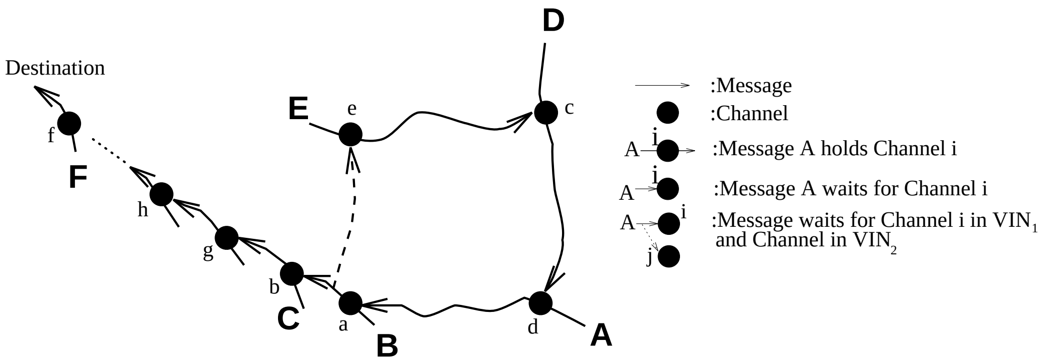

Figure 18: A waiting cycle in the channel-dependency graph is not a real deadlock.

property of wormhole routing, the channels between two blocked channels can be removed or added without affecting the deadlock itself. So, the channel-dependency graph (CDG) can be reduced to the one consisting of blocked channels only, as shown in Fig. 17(b).

We can then decompose the proof into two parts: (1) at least one blocked channel in the waiting cycle of CDG does not belong to VIN_1, and (2) the waiting cycle in CDG is not a real deadlock.

Proof of Part 1: We prove this part by contradiction. Assume there exists a deadlock in which all of the deadlocked channels belong to VIN_1, then we can get a reduced CDG with a circular wait among the channel resources as shown in Fig. 17(b). This is a waiting cycle of four messages; one can easily extend this proof to those cases with more messages involved.

Channels a, b, c, and d are therefore the virtual channels of VIN_1. According to the proposed routing algorithm, the virtual channels of VIN_1 should obey a strict (say, increasing) order of the requested channel numbers, and the relationships among channels a, b, c, and d are summarized as: a < b for message B, b < c for message C, c < d for message D, and d < a for message A. Thus, a<b<c<d<a, a contradiction, and hence, there is at least one virtual channel belonging to VIN_2.

Proof of Part 2: Since Algorithm 3.1 is an adaptive routing algorithm, a message may simultaneously request more than one channel. From Part 1, we know that at least one blocked channel belongs to VIN_2. Fig. 18 shows a waiting cycle (a reduced CDG) consisting of four messages: the blocked channel e belongs to VIN_2 and the blocked channels a, c, and d belong to VIN_1. Since the channel e in Fig. 18 belongs to VIN_2, the message B in Fig. 18 can find a channel b in VIN_1 to request, while satisfying the increasing order restriction, **A4**. If channel b is allowed to be held only for a finite time, then the waiting cycle consisting of channels a, c, d, and e is *not* a real deadlock, because once message B holds channel b, the channel e will no longer be requested by message B, and hence, the waiting cycle no longer exists. Now, we check to see if channel b is held for a finite time. At a particular time, if channel b cannot be held by message B, there must be another message C holding channel b and requesting a higher-number channel. Thus, the number of higher-number channels needed increases with the number of requested channels involved, i.e., f $>\cdots>$ h $>$ g $>$ b $>$ a $>$ d $>$ c. Since only a finite number of channels are used, the message holding the last channel f must reach its destination or keep moving closer to its destination via the channels in VIN_2. According to **A2**, a message reaching its destination will eventually be consumed, and hence, channel f will be released. Since a fair channel allocation scheme is assumed (**A3**), the message requesting channel f will hold channel f only for a finite time. Packet B is thus guaranteed to acquire channel b in finite time.

For a waiting cycle with more blocked channels involved and with more than one blocked channel belonging to VIN_2, we can follow the same logic as above and guarantee the message requesting a channel in VIN_2 to capture it in a finite time. Thus, under the proposed routing algorithm, the waiting cycle in the channel-dependency graph is not a real deadlock, i.e., the proposed routing algorithm is deadlock-free. □