A Trip-based Multicasting Model in Wormhole-routed Networks with Virtual Channels*

Yu-Chee Tseng, Dhabaleswar K. Panda, and Ten-Hwang Lai

Department of Computer and Information Science
The Ohio State University
Columbus, Ohio 43210
Tel: (614)292-5813, Fax: (614)292-2911
Email: {tseng, panda, lai}@cis.ohio-state.edu

Abstract

This paper focuses on efficient multicasting in wormhole-routed networks. A *trip-based model* is proposed to support adaptive, distributed, and deadlock-free *multiple multicast* on any network with arbitrary topology using at most two virtual channels per physical channel. This model significantly generalizes the path-based model proposed earlier [21, 22], which works only for Hamiltonian networks and can not be applicable to networks with arbitrary topology resulted due to system faults. Fundamentals of the trip-based model, including the necessary and sufficient condition to be deadlock-free, and the use of appropriate number of virtual channels to avoid deadlock are investigated. The potential of this model is illustrated by applying it to hypercubes with faulty nodes. Simulation results indicate that the proposed model can implement multiple multicast on faulty hypercubes with negligible performance degradation.

Key Words: Routing algorithm, Interprocessor communication, Multicast, Virtual channel, Wormhole-routing, Path-based routing, Collective Communication, and Fault tolerance.

*A preliminary version of this paper has been presented in *Int'l Parallel Processing Symposium.*, 1993.

Contents

1 Introduction

In a multicomputer network, processors often need to communicate with each other by message-passing for various reasons, such as job migration, load balance, and event synchronization. The problem of message routing in a multicomputer system can be characterized, among others, by network topology (e.g., mesh, hypercube, torus, or tree), communication pattern (e.g., one-to-one, permutation, broadcast, or multicast), and switching technique (e.g., wormhole, circuit, or packet switching). Packet and circuit switching have been used in the first- and second-generation parallel computers and much work about routing can be found in the literature [3, 4, 16, 17, 29]. However, the more recent parallel computers (such as Intel Touchstone DELTA, Intel Paragon, MIT J-machine, Caltech MOSAIC, and Cray T3D) have adopted wormhole routing, which is known to be quite insensitive to routing distance if the network is contention-free.

In this paper, we study the *multiple multicast problem* in wormhole-routed networks, where each source wants to send a message to some destination nodes. Multicast represents the most complex communication pattern. Numerous applications, such as parallel search algorithms, graph algorithms, and barrier synchronization, can benefit from multicast services [21]. Multicasting is also necessary to support *cache-coherency* in distributed-shared-memory systems [20]. Multicasting/broadcasting has been defined in the MPI (Message Passing Interface) standard [1] as an instrumental operation in implementing *collective communication operations* (barrier synchronization, allgather, and global reduction) in distributed-memory systems. These collective communication operations are defined over process groups and the associated broadcasting operations result into multicast (from the perspective of a system) when (a) only a subset of processes of a task participate in the operation or (b) all processes of a task participate and they are allocated into discontinuous locations in the system. Hence, a good solution for single-source and multiple-source multicast is critical to the development of high performance parallel systems supporting distributed-memory or distributed-shared-memory paradigms.

Solutions for multicasting can be classified into 3 categories: *unicast-based*, *tree-based* and *path-based*. The unicast-based solutions make use of 1-to-1 communication and add additional software to support multicast. Examples include [25] (for one-port architectures) and [15, 24] (for all-port architectures). Disadvantages of this approach include necessary involvement in message propagation by intermediate nodes and required latency to start-up messages (which is typically in the order of 1.0 to 20.0 microseconds in current generation system) in each intermediate node.

The tree-based solutions rely on finding a spanning tree rooted at the source node. A multicast message is then propagated along the tree (e.g., multicast in hypercubes [19, 28], faulty hypercubes [18], and meshes [21]). Such solutions are adequate for packet-switched networks, but incur long latency in wormhole-routed networks [22]. More seriously, as shown in [22], such approach leads to deadlock if multiple-source multicasts are to be performed in a system simultaneously.

To prevent these problems, an elegant path-based model has been proposed by Lin *et al.*

[21, 22] for wormhole-routed networks. This scheme is based on finding a Hamiltonian path in the network which is used to establish the dependency relation between resources (e.g., buffers and channels). This allows multiple-source multicasts to be performed in a system without deadlock. Duato [12] further studies the use of virtual channels in this scheme in a 2-D mesh to exploit more adaptivity. The limitation of these results is that the network must be Hamiltonian. For irregular networks or networks with faults, the existence of a Hamiltonian path is not guaranteed. Fault tolerance is an important issue in massively parallel processing. Although most popular networks are Hamiltonian, a single fault may destroy their Hamiltonian property. Hence the above solution can not be adopted to systems with faults. Furthermore, finding a Hamiltonian path in any given graph is NP-complete [13].

In this paper, we generalize the path-based solutions and propose a new *trip-based model* to support multiple multicasts. This model can be applied to any network of *arbitrary* topology as long as it remains *connected.* Thus the new model eliminates the limitation associated with the solutions proposed in [12, 21, 22]. The underlying concept used in this model is a node sequence called *trip.* The resource-dependency relation is established based on this sequence and the possibility of *cyclic dependency* [10] and thus deadlock is avoided. The model is supported by adding a certain number of virtual channels to each physical channel and this number gets derived depending on the trip. It is shown that the path-based solutions in [21, 22] are a special case of ours when the trip is a Hamiltonian path. In this case, our model requires only one virtual channel per physical channel (i.e., no need the support of virtual channels). We develop the necessary and sufficient condition on trips such that only one virtual channel is needed. For arbitrary networks we show how to construct a special *skirt-based trip* such that at most two virtual channels per physical channel are needed. With at most 2 virtual channels, our result significantly generalizes the path-based model to be applied to arbitrary network topologies.

Based on this trip model, we present a low-cost, adaptive, and distributed routing algorithm to implement multicasting. To demonstrate the potential of this model, we apply it to faulty hypercubes. A novel scheme is proposed for constructing a skirt-based trip in a faulty n-cube with at most $n-1$ faulty nodes. It is shown that the trip length can be very limited ($\leq 2^n$) and it is verified by simulations that with such a construction, the performance degradation in the cube is almost negligible in the presence of faults.

Background of wormhole routing can be found in [9] and that of virtual channels in [10]. Using virtual channels is common in the literature and has been proved to have the following merits: to avoid deadlock [5, 10, 23], to increase physical channel utilization [8], and to exploit routing adaptivity [11, 14, 23]. A survey of various routing algorithms can be found in [26].

The rest of this paper is organized as follows: The trip-based model and its fundamental properties are investigated in Section 2. A multicast algorithm is presented in Section 3. Section 4 shows how to apply our model to faulty hypercubes and presents our simulation results. Conclusions are drawn in Section 5.

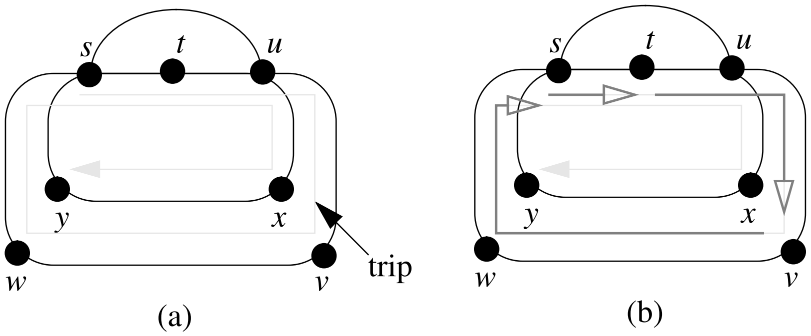


Figure 1: (a) a system graph G with an example trip $\alpha = [s, t, u, v, w, s, t, u, x, y]$ on it and (b) a deadlock scenario with three worms ($s \to t, t \to v, v \to s$) if virtual channels are not supported.

2 Fundamental of the Trip-based Multicast Model

2.1 Basic Idea

A multicomputer network is represented by an undirected *system graph* $G = (\mathcal{V}, \mathcal{E})$ with vertex set $\mathcal{V}$ corresponding to processors (nodes) and edge set $\mathcal{E}$ to communication links. Each (undirected) communication link $(u, v) \in \mathcal{E}$ has two directed physical channels, denoted as $pch\langle u, v\rangle$ and $pch\langle v, u\rangle$, going from u to v and v to u, respectively. Each physical channel can support a certain number of virtual channels. In our model, this number is a factor to be derived. Without loss of generality, G is assumed to be connected.

Definition 1 Given a system graph $G = (\mathcal{V}, \mathcal{E})$, a *trip* on G is any path which visits each node of $\mathcal{V}$ at least once. A trip is denoted as a sequence $\alpha = [\alpha_1, \ldots, \alpha_n], \alpha_i \in \mathcal{V}, i = 1..n$. Given an α, for each $i = 1..n$, $\hat{\alpha}_i$ is defined to be the two-tuple (α_i, i), which is regarded as a *virtual node* of α_i. The $\hat{\alpha}$ corresponding to α is defined to be $[\hat{\alpha}_1, \ldots, \hat{\alpha}_n]$.

For instance, Fig. 1(a) shows a system graph and an example trip of length $n = 10$. Clearly, in a connected graph, there can be an infinite number of trips. When α is a Hamiltonian path, $n = |\mathcal{V}|$. Note that for any i and j such that $i \neq j$, it indicates $\hat{\alpha}_i \neq \hat{\alpha}_j$. But, it does not necessarily indicate $\alpha_i \neq \alpha_j$. The purpose of $\hat{\alpha}_i$ is to distinguish the multiple occurrences of a node and to indicate its position in α. As a convention, for any alphabet, say x, symbol $\hat{x}$ or $\hat{x}_i$ is used appropriately to denote a virtual node of $x \in \mathcal{V}$, where the latter is used when the location (i) is important. We define $\hat{x}_i$ to be on the left (right) of $\hat{y}_j$ if $i < j$ ($i > j$). A function $leftmost(x)$ is defined to return the leftmost virtual node $\hat{x}$ corresponding to $x \in \mathcal{V}$ and another function $rightmost(x)$ returns the rightmost one.

Consider a node s trying to multicast a message m to a set $D \subseteq \mathcal{V}$ of destination nodes. Assuming that $\alpha = [\alpha_1, \ldots, \alpha_n]$ is the trip, to illustrate our main idea, we present a simple multicast algorithm as follows:

Algorithm *Basic*()

1) Find any index $x, 1 \leq x \leq n$, such that $\alpha_x = s$.
2) Send a worm which carries m, starting from α_x and sequentially traversing nodes $\alpha_{x+1}, \alpha_{x+2}, \ldots, \alpha_n$.
3) Send a worm which carries m, starting from α_x and sequentially traversing nodes $\alpha_{x-1}, \alpha_{x-2}, \ldots, \alpha_1$. □

Let's call the first worm a *forward-worm* (or *f-worm*) and the second a *backward-worm* (or *b-worm*). The underlying idea of the trip-based multicast model is to send f-worms and b-worms in the forward and backward directions of the trip, respectively. However, *Basic*() may not be efficient enough because the routing is not adaptive. In the next subsection, we address this issue and introduce the idea of "short-cut".

The following conventions regarding sequences are used throughout the paper: a^k represents k occurrences of a's and a^+ a positive number of a's. The first element of a sequence S is denoted as $head(S)$. Given a trip α and two nodes $x, y \in \mathcal{V}$, $\alpha(x, y)$ is the subsequence of α containing all occurrences of x and y. Similarly, $\hat{\alpha}(x, y)$ is the subsequence of $\hat{\alpha}$ containing all occurrences of $\hat{x}$ and $\hat{y}$ corresponding to x and y, respectively. For instance, in Fig. 1, $\alpha(s, u) = [s, u, s, u]$ and $\hat{\alpha}(s, u) = [\hat{s}_1, \hat{u}_3, \hat{s}_6, \hat{u}_8]$.

2.2 Making Use of Short-cuts

In *Basic*(), it would be more efficient if worms can take some "short-cuts". For example, if $(\alpha_i, \alpha_j) \in \mathcal{E}$ and $\hat{\alpha} = [\ldots, \hat{\alpha}_i, \ldots, \hat{\alpha}_j, \ldots]$, then an f-worm (resp., b-worm) arriving at $\hat{\alpha}_i$ (resp., $\hat{\alpha}_j$) can be directed to $\hat{\alpha}_j$ along $pch\langle \alpha_i, \alpha_j \rangle$ (resp., to $\hat{\alpha}_i$ along $pch\langle \alpha_j, \alpha_i \rangle$) if there is no destination node between $\hat{\alpha}_i$ and $\hat{\alpha}_j$. The essential rule in taking such short-cuts can be informally described as follows: a f-worm (b-worm) always visits $\hat{\alpha}_i$ before $\hat{\alpha}_j$ such that $i < j$ ($i > j$). Below we formalize this idea by extending G to two directed graphs. These graphs specify all possible routes that can be taken by an f-worm and a b-worm, respectively.

Definition 2 Given a system graph G and a trip $\alpha = [\alpha_1, \alpha_2, \ldots, \alpha_n]$ on G, the *forward-graph* or *f-graph*, $G_f(\alpha)$ and the *backward-graph* or *b-graph*, $G_b(\alpha)$ of G are directed graphs such that:

$$\begin{aligned}
V(G_f(\alpha)) &= V(G_b(\alpha)) = \{\hat{\alpha}_1, \hat{\alpha}_2, \ldots, \hat{\alpha}_n\} \\
E(G_f(\alpha)) &= \{\langle \hat{\alpha}_i, \hat{\alpha}_j \rangle | (\alpha_i, \alpha_j) \in \mathcal{E} \wedge i < j\} \\
E(G_b(\alpha)) &= \{\langle \hat{\alpha}_i, \hat{\alpha}_j \rangle | (\alpha_i, \alpha_j) \in \mathcal{E} \wedge i > j\}.
\end{aligned}$$

For example, Fig. 2 shows the $G_f(\alpha)$ and $G_b(\alpha)$ corresponding to the system graph G and the trip α as shown in Fig. 1. For each pair of $\hat{\alpha}_i$ and $\hat{\alpha}_j$, if $(\alpha_i, \alpha_j) \in \mathcal{E}$ and $\hat{\alpha}_i$ is on the left (right) of $\hat{\alpha}_j$, we define an edge leading from $\hat{\alpha}_i$ to $\hat{\alpha}_j$ in $G_f(\alpha)$ ($G_b(\alpha)$). To make the presentation simpler, here onwards, we will omit indicating α whenever it is clear.

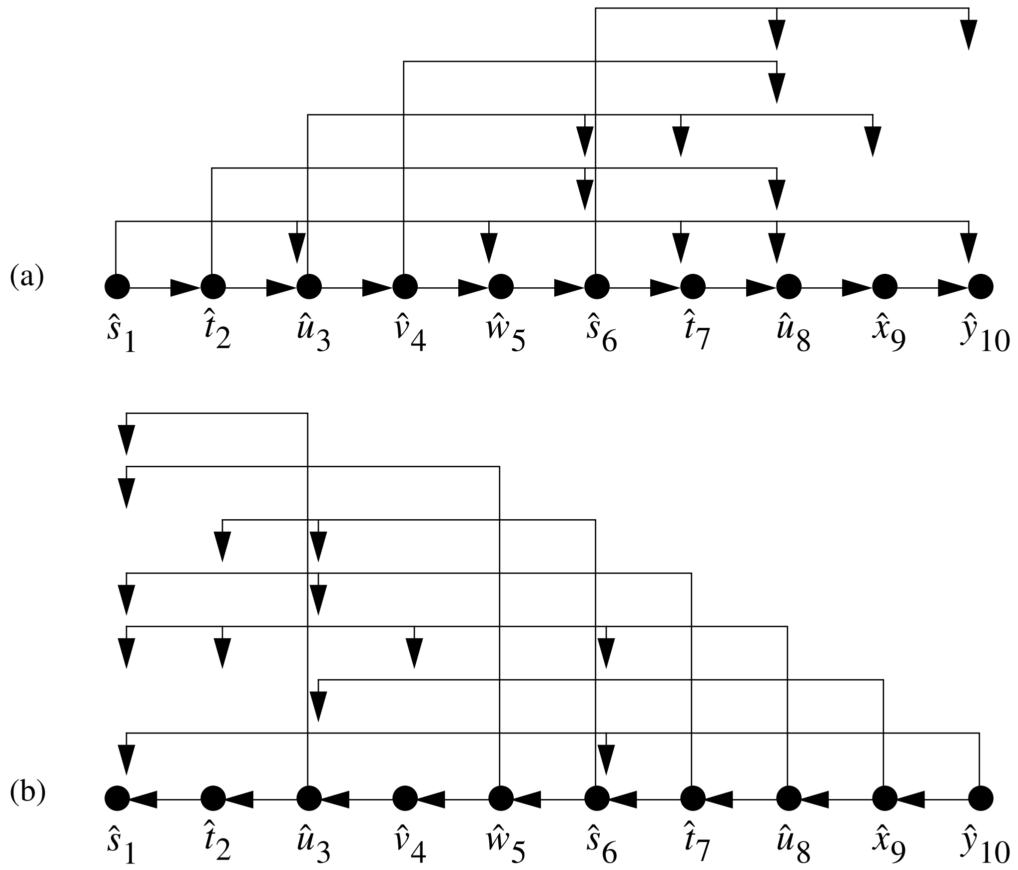


Figure 2: Forward and backward virtual graphs corresponding to the system graph G and trip α as shown in Fig. 1: (a) the f-graph $G_f(\alpha)$ and (b) the b-graph $G_b(\alpha)$.

We regard G_f and G_b as virtual graphs of G with each $\hat{\alpha}_i$ a virtual node of α_i and each edge $\langle \hat{\alpha}_i, \hat{\alpha}_j \rangle$ as a virtual edge of $pch\langle \alpha_i, \alpha_j \rangle$. Although G_f and G_b are virtual graphs, we will later use them to interpret the message routing of f-worms and b-worms. If a message is routed on a virtual edge $\langle \hat{\alpha}_i, \hat{\alpha}_j \rangle$, it is actually routed on the real link $pch\langle \alpha_i, \alpha_j \rangle$. The word "routing" is thus interchangeably referred to G and to G_f and G_b. To facilitate our presentation, we will denote: i) $\hat{\alpha}_i \approx \hat{\alpha}_j$ iff $\alpha_i = \alpha_j$, and ii) $\langle \hat{\alpha}_i, \hat{\alpha}_j \rangle \approx \langle \hat{\alpha}_x, \hat{\alpha}_y \rangle$ iff $\alpha_i = \alpha_x$ and $\alpha_j = \alpha_y$.

2.3 Channel Dependency Analysis

In wormhole routing, it is important to know whether a given routing algorithm is deadlock-free or not. In fact, *Basic*() is susceptible to deadlock, as shown in the scenario in Fig. 1(b), where three f-worms (one from s arriving at t, one from t arriving at v, and one from v arriving at s) wait for each other to proceed, which may never happen. With the introduction of short-cuts, the possibility of deadlock may be higher.

The deadlock problem can be reasoned by deriving the *channel dependency relation* [10] in a wormhole-routed network resulted from worms proceeding in a pipelined manner. If a message in channel c can be propagated to channel c', there is a resource dependency from c to c', denoted as $c \mapsto c'$. A routing algorithm is deadlock-free if there does not exist a sequence of dependencies $c_1 \mapsto c_2 \mapsto \cdots \mapsto c_k$ such that $c_1 = c_k$.

Recall that we will be using G_f and G_b to interpret the message routing of f-worms and b-worms. Hence, we derive channel dependencies on G_f and G_b as follows:

$$\begin{aligned} D(G_f(\alpha)) &= \{pch\langle\alpha_i,\alpha_j\rangle \mapsto pch\langle\alpha_j,\alpha_k\rangle | \langle\hat{\alpha}_i,\hat{\alpha}_j\rangle, \langle\hat{\alpha}_j,\hat{\alpha}_k\rangle \in E(G_f(\alpha))\} \\ D(G_b(\alpha)) &= \{pch\langle\alpha_i,\alpha_j\rangle \mapsto pch\langle\alpha_j,\alpha_k\rangle | \langle\hat{\alpha}_i,\hat{\alpha}_j\rangle, \langle\hat{\alpha}_j,\hat{\alpha}_k\rangle \in E(G_b(\alpha))\} \end{aligned}$$

That is, for any two adjacent virtual edges in G_f or G_b, there is a channel dependency in $D(G_f)$ or $D(G_b)$, respectively. Given any trip α, it is desirable to know whether there exists a cycle in $D(G_f) \cup D(G_b)$ or not. The following theorem answers this question.

Theorem 1 *Given any trip α, $D(G_f(\alpha)) \cup D(G_b(\alpha))$ is acyclic iff*

(a) for any node $x \in \mathcal{V}$, x appears in α at most twice; and

(b) for any two nodes $x, y \in \mathcal{V}$ which appear twice in α, the sequence $\alpha(x,y)$ is equal to $[x,y,y,x]$, or $[y,x,x,y]$.

Proof. We prove the "if" part first. Given an α satisfying conditions (a) and (b), let us construct a sequence α' such that (i) α is a subsequence of α', (ii) each node in $\mathcal{V}$ appears exactly twice in α', and (iii) α' satisfies the condition (b). (It is not hard to find such an α' — for any node $x \in V$ which appears only once in α, simply insert another x into α in a proper position.) Note that α' may not be a trip. Also note that ensured by conditions (ii) and (iii), α' is symmetric in the sense that it is equal to its reverse.

Although α' may not be a trip, we can apply Definition 2 on α' to construct an f-graph $G_f(\alpha')$ and a b-graph $G_b(\alpha')$. We can conclude two facts from this construction. First, because α is a subsequence of α', any channel dependency incurred by $G_f(\alpha)$ must be a dependency in $G_f(\alpha')$. Similarly, this is true for $G_b(\alpha)$ and $G_b(\alpha')$. Hence, $D(G_f(\alpha)) \cup D(G_b(\alpha)) \subseteq D(G_f(\alpha')) \cup D(G_b(\alpha'))$. Second, because α' is symmetric, any dependency incurred by $G_f(\alpha')$ must be a dependency in $G_b(\alpha')$, and vice versa. Hence, $D(G_f(\alpha')) = D(G_b(\alpha'))$. It follows that the "if" part is proved if $D(G_f(\alpha'))$ can be proven as acyclic.

In order to prove it, we use the following property that can be derived from condition (b) (note that this property may not be true for any trip):

P: Let $pch\langle x,y\rangle \mapsto pch\langle y,z\rangle$ be in $D(G_f(\alpha'))$. Given any two virtual nodes $\hat{x}$ and $\hat{y}$ such that $\hat{y}$ is the first virtual node of y that is on $\hat{x}$'s right-hand side, there must exist a node $\hat{z}$ on $\hat{y}$'s right-hand side.

As a contradiction assume that in $D(G_f(\alpha'))$ there is a sequence of dependencies $pch\langle\alpha_{r_1},\alpha_{r_2}\rangle \mapsto pch\langle\alpha_{r_2},\alpha_{r_3}\rangle \mapsto \cdots \mapsto pch\langle\alpha_{r_{m-1}},\alpha_{r_m}\rangle$ such that $pch\langle\alpha_{r_1},\alpha_{r_2}\rangle = pch\langle\alpha_{r_{m-1}},\alpha_{r_m}\rangle$. Construct a sequence of virtual nodes $\hat{\alpha}_{s_i}, i = 1..m$ as follows. Let $\hat{\alpha}_{s_1} = leftmost(\alpha_{r_1})$. Let $\hat{\alpha}_{s_2}$ be the *first* virtual node of α_{r_2} that is on $\hat{\alpha}_{s_1}$'s right-hand side ($\hat{\alpha}_{s_2}$ must exist, by the existence of $pch\langle\alpha_{r_1}, \alpha_{r_2}\rangle$). Because of the dependency $pch\langle\alpha_{r_1}, \alpha_{r_2}\rangle \mapsto pch\langle\alpha_{r_2}, \alpha_{r_3}\rangle$, using property **P**, we can define $\hat{\alpha}_{s_3}$ to be the first virtual node of α_{r_3} that is on $\hat{\alpha}_{s_2}$'s right-hand side. We can repeat the above argument using the sequence of dependencies given above to define all $\hat{\alpha}_{s_i}, i = 4..m$. Unfortunately, because $s_1 < \cdots < s_m$, the success of this construction implies that $[\alpha_{r_1}, \alpha_{r_2}, \alpha_{r_{m-1}}, \alpha_{r_m}]$ is a subsequence of α, which violates condition (b).

Now we prove the "only if" part. First, we disprove the possibility that a node, say $x \in \mathcal{V}$, can appear more than twice in α. Suppose $\hat{\alpha} = [\ldots, \hat{x}_i, \hat{y}_{i+1}, \ldots, \hat{x}_j, \hat{z}_{j+1}, \ldots, \hat{x}_k, \ldots]$, where y and z are the nodes directly following x in the trip. Apparently, there is a path $\hat{x}_i \rightarrow \hat{y}_{i+1} \rightarrow \hat{x}_j \rightarrow \hat{z}_{j+1}$ in G_f. Also there is a path $\hat{x}_k \rightarrow \hat{z}_{j+1} \rightarrow \hat{x}_j \rightarrow \hat{y}_{i+1}$ in G_b. This indicates that there is a dependency from $pch\langle x, y\rangle$ to itself, which is a contradiction.

Now consider any $x, y \in \mathcal{V}$ which appear twice in α. Without loss of generality, we need to disprove the possibilities of $\alpha(x, y)$ being $[x, x, y, y]$ or $[x, y, x, y]$. To disprove the first possibility, let $\hat{\alpha} = [\ldots, \hat{x}_i, \hat{w}_{i+1}, \ldots, \hat{x}_j, \ldots, \hat{y}_k, \hat{z}_{k+1}, \ldots, \hat{y}_l, \ldots]$, where w and z are the nodes directly following the first x and y in α, respectively. There is a path $\hat{x}_i \rightarrow \hat{w}_{i+1} \rightarrow \cdots \rightarrow \hat{y}_k \rightarrow \hat{z}_{k+1}$ in G_f, and another $\hat{y}_l \rightarrow \hat{z}_{k+1} \rightarrow \cdots \rightarrow \hat{x}_j \rightarrow \hat{w}_{i+1}$ in G_b. Thus, $pch\langle x, w\rangle$ depends on itself, which is a contradiction. The second possibility can be disproved similarly. $\Box$

The above results indicate that if we describe message routing of some f-worms and b-worms using G_f and G_b, the real routing in the network G must be deadlock-free as long as the trip α satisfies the conditions in Theorem 1.

2.4 Using Virtual Channels to Avoid Deadlock

Theorem 1 has indicated the kind of trips that can be used safely for multicast routing without deadlock. However, given any network, it might be difficult to find such a trip (if not impossible). Below we show how to solve this problem using virtual channels. Using virtual channels is a common technique to separate the resource dependency and thus avoid possible deadlock [5, 10, 23]. Conceptually, a physical channel is separated into multiple virtual channels which share the physical channel in time-multiplexing or demand-multiplexing manner. Messages are propagated in the network by grabbing one virtual channel after another. A dependency $c \mapsto c'$ is thus established when a message is sent from one virtual channel c to another c'. Similarly, a routing algorithm is deadlock-free if the dependency is acyclic. More details about virtual channels can be found in [10, 26].

Let α be any trip. Consider associating a virtual channel with each edge $\langle\hat{x}, \hat{y}\rangle$ in G_f and G_b (note that any two edges may not necessarily be associated with distinct virtual channels). Whenever a message is sent on an edge in G_f or G_b, we regard it as being sent along the corresponding virtual channel. Consider any physical channel $pch\langle x, y\rangle$. Let $\alpha(x, y) = [y^{d_0}, x^{c_1}, y^{d_1}, x^{c_2}, y^{d_2}, \ldots, x^{c_k}, y^{d_k}]$, where integer $k > 0$ and all integers $c_i, d_i > 0$ except that integers $d_0, d_k \geq 0$. Note that k must be uniquely defined. Let's write $\hat{\alpha}(x, y)$, the sequence corresponding to $\alpha(x, y)$, as:

$$
\begin{array}{ll}
[& \hat{y}_{0,1}, \hat{y}_{0,2}, \ldots, \hat{y}_{0,d_0}, \\
& \hat{x}_{1,1}, \hat{x}_{1,2}, \ldots, \hat{x}_{1,c_1}, \hat{y}_{1,1}, \hat{y}_{1,2}, \ldots, \hat{y}_{1,d_1}, \\
& \hat{x}_{2,1}, \hat{x}_{2,2}, \ldots, \hat{x}_{2,c_2}, \hat{y}_{2,1}, \hat{y}_{2,2}, \ldots, \hat{y}_{2,d_2}, \\
& \ldots \\
& \hat{x}_{k,1}, \hat{x}_{k,2}, \ldots, \hat{x}_{k,c_k}, \hat{y}_{k,1}, \hat{y}_{k,2}, \ldots, \hat{y}_{k,d_k} \quad]
\end{array}
\tag{1}
$$

where each $\hat{x}_{i,j}$ and $\hat{y}_{i,j}$ is a virtual node corresponding to x and y, respectively. We define the following virtual channels in $pch\langle x, y\rangle$, where $*$ means any possible integer:

$\mathbf{VC}_f$: For each edge $c = \langle \hat{x}_{i,*}, \hat{y}_{*,*}\rangle \in G_f$, associate a virtual channel $vch_f\langle x, y, i\rangle$ with c.

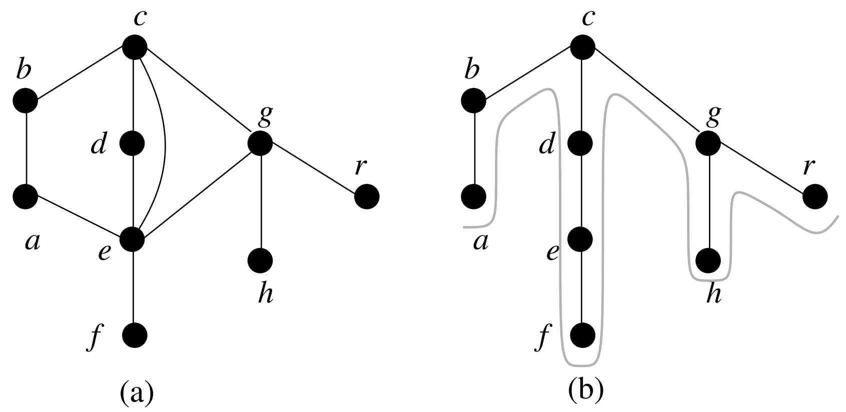
$\mathbf{VC}_b$: For each edge $c = \langle \hat{x}_{i,*}, \hat{y}_{*,*}\rangle \in G_b$, associate a virtual channel $vch_b\langle x, y, i\rangle$ with c.

In $\mathbf{VC}_f$, integer i ranges from 1 to $k-1$ if $d_k = 0$ and from 1 to k otherwise. In $\mathbf{VC}_b$, integer i ranges from 2 to k if $d_0 = 0$, and 1 to k otherwise. Hence, we need $2(k-1)$ to $2k$ virtual channels for $pch\langle x, y\rangle$. We call those virtual channels used by G_f as *forward-channels* or *f-channels*, and those used by G_b as *backward-channels* or *b-channels*. For instance, in Fig. 1, $pch\langle s, u\rangle$ will have 3 virtual channels: $vch_f\langle s, u, 1\rangle, vch\langle s, u, 2\rangle$, and $vch_b\langle s, u, 2\rangle$. Similarly, $pch\langle u, s\rangle$ will have 3 virtual channels: $vch_f\langle u, s, 1\rangle, vch_b\langle u, s, 1\rangle$, and $vch_b\langle u, s, 2\rangle$. Note that it is a result of our definition that the number of virtual channels required by any physical channel $pch\langle x, y\rangle$ is equal to that required by $pch\langle y, x\rangle$. Let's redefine $D(G_f)$ and $D(G_b)$ as follows:

$$
\begin{aligned}
D(G_f(\alpha)) &= \{vch_f\langle x, y, i\rangle \mapsto vch_f\langle y, z, j\rangle | \langle \hat{x}_{i,*}, \hat{y}_{j,*}\rangle, \langle \hat{y}_{j,*}, \hat{z}_{*,*}\rangle \in E(G_f(\alpha))\} \\
D(G_b(\alpha)) &= \{vch_b\langle x, y, i\rangle \mapsto vch_b\langle y, z, j\rangle | \langle \hat{x}_{i,*}, \hat{y}_{j,*}\rangle, \langle \hat{y}_{j,*}, \hat{z}_{*,*}\rangle \in E(G_b(\alpha))\}
\end{aligned}
$$

Theorem 2 *Given any trip α, $D(G_f(\alpha)) \cup D(G_b(\alpha))$ is acyclic if edges in $G_f(\alpha)$ and $G_b(\alpha)$ are associated with virtual channels according to $\mathbf{VC}_f$ and $\mathbf{VC}_b$.*

Proof. Since there is no dependency between f-channels and b-channels, it suffices to show the part for G_f. Suppose for contradiction that there is a sequence of dependencies $vch_f\langle \alpha_{r_1}, \alpha_{r_2}, i_1\rangle \mapsto vch_f\langle \alpha_{r_2}, \alpha_{r_3}, i_2\rangle \mapsto \cdots \mapsto vch_f\langle \alpha_{r_{m-1}}, \alpha_{r_m}, i_{m-1}\rangle$ such that $vch_f\langle \alpha_{r_1}, \alpha_{r_2}, i_1\rangle = vch_f\langle \alpha_{r_{m-1}}, \alpha_{r_m}, i_{m-1}\rangle$. Let us construct a sequence of virtual nodes $\hat{\alpha}_{s_i}, i = 1..m$ as follows. Let $\hat{\alpha}_{s_1}$ be the leftmost virtual node that is incident by an edge associated with $vch_f\langle \alpha_{r_1}, \alpha_{r_2}, i_1\rangle$. The dependency $vch_f\langle \alpha_{r_1}, \alpha_{r_2}, i_1\rangle \mapsto vch_f\langle \alpha_{r_2}, \alpha_{r_3}, i_2\rangle$ implies that there is a virtual node $\hat{\alpha}_{s_2}$ which is to the right of $\hat{\alpha}_{s_1}$ and is incident by $vch_f\langle \alpha_{r_2}, \alpha_{r_3}, i_2\rangle$. The argument can be easily repeated for the remaining sequence of the dependencies. At the end, we will create two edges $\langle \alpha_{s_1}, \alpha_{s_2}\rangle$ and $\langle \alpha_{s_{m-1}}, \alpha_{s_m}\rangle$ which are associated with the same virtual channel. However, because $s_1 < s_2 < s_{m-1} < s_m$, according to $\mathbf{VC}_f$, these edges can not be associated with the same virtual channel. $\Box$



$$\alpha_T = [a, b, c, d, e, f, e, d, c, g, h, g, r]$$

(c)

Figure 3: (a) A system graph G with arbitrary topology, (b) a spanning tree T of G and its skirt, and (c) the associated skirt-based trip α_T.

2.5 Skirt-based Trip for Networks with Arbitrary Topology

Given any trip α, the above theorem indicates the required number of virtual channels to avoid deadlock. One immediate question is: what kind of α will need fewer virtual channels. The following discussion shows that in any network, there always exists an α which needs at most 2 virtual channels per physical channel.

Definition 3 Let T be a spanning tree of the system graph G rooted at a node r. A *skirt-based trip* with respect to T, denoted as α_T, is defined to be the path that starts from r's leftmost descendant, ends with r's rightmost descendant, and wraps around the boundary of T by visiting each node one by one.

An example skirt-based trip is shown in Fig. 3. It is not hard to prove the following.

Theorem 3 *Let T be any spanning tree of G. Then α_T has the following properties:*
(a) The length of α_T is between (including) $|\mathcal{V}|$ and $2|\mathcal{V}| - 1$.
(b) For any two distinct nodes $x, y \in \mathcal{V}$, $\alpha_T(x, y) = [x^+, y^+], [y^+, x^+], [x^+, y^+, x^+]$, or $[y^+, x^+, y^+]$.

We remark that the trip length will reflect the maximum number of hops a worm may travel in the network in our yet-to-be-presented multicast algorithm. The above trip length is no more than two times the minimum, $|\mathcal{V}|$. Note that finding a trip of a minimum length in any graph is no easier than deciding whether the graph is Hamiltonian, which is NP-complete [13].

Corollary 1 *Let T be any spanning tree of G, and $pch\langle x, y\rangle$ any edge in G. Using the rules* $\mathbf{VC}_f$ *and* $\mathbf{VC}_b$*, for every physical channel $pch\langle x, y\rangle$, there is a requirement of* *(a) one f-channel if* $\alpha(x,y) = [x^+, y^+]$,

(b) one b-channel if $\alpha(x,y) = [y^+, x^+]$, *and*

(c) one f-channel and one b-channel if $\alpha(x,y) = [x^+, y^+, x^+]$ *or* $[y^+, x^+, y^+]$.

Hence, given a network G, if one can find a Hamiltonian path or a trip satisfying Theorem 1 then no virtual channel is needed to avoid deadlock. However, if this is not possible (for reasons such as irregular networks due to link/node faults), then one can always go for a skirt-based trip, which requires at most 2 (forward plus backward) virtual channels per physical channel as shown in Corollary 1. As a result, our model has very broad applicability with low hardware requirement. In the next section, we will present our multicast algorithm based on these trips.

3 The Multicast Algorithm

In this section, we present our multicast algorithm. We first reduce virtual graphs G_f and G_b by eliminating some edges from them. Then we present the algorithm in Section 3.2. Complexity analysis and the significance of this model is given in Section 3.3.

The trip α, discussed in the previous section, is assumed to satisfy either Theorem 1 or Theorem 3. If α satisfies Theorem 1, there is no need of virtual channels. If α satisfies Theorem 3, the edges in G_f and G_b are associated with virtual channels according to $\mathbf{VC}_f$ and $\mathbf{VC}_b$. However, in the following presentation, we will use the term $vch_f\langle x,y\rangle$ (or $vch_b\langle x,y\rangle$) for convenience and for the following reasons: i) if Theorem 1 is applicable then $vch_f\langle x,y\rangle$ (or $vch_b\langle x,y\rangle$) can be used to indicate $pch\langle x,y\rangle$ without loss of generality and ii) if Theorem 3 is the case then there is at most one f-channel (b-channel) in $pch\langle x,y\rangle$ and hence, the third parameter in $vch_f\langle x,y,i\rangle$ (or $vch_b\langle x,y,i\rangle$) can be omitted without loss of generality.

3.1 Reduced F-graphs and B-graphs

It is to be noted that a node can appear in a trip for a number of times. This leads to multiple edges between two nodes in the virtual graph. Below, we show how to eliminate some of these multiple edges from G_f and G_b. Such elimination reduces the complexity of G_f and G_b without diminishing the routing capability of the original construction.

Definition 4 Let $\alpha = [\alpha_1, \alpha_2, \ldots, \alpha_n]$ be the trip. The *reduced f-graph* $G'_f(\alpha_T)$ and the *reduced b-graph* $G'_b(\alpha_T)$ are directed graphs such that

$$\begin{aligned}
V(G'_f(\alpha)) &= V(G'_b(\alpha)) = \{\hat{\alpha}_1, \hat{\alpha}_2, \ldots, \hat{\alpha}_n\} \\
E(G'_f(\alpha)) &= \{\langle \hat{\alpha}_i, \hat{\alpha}_j\rangle | \langle \hat{\alpha}_i, \hat{\alpha}_j\rangle \in E(G_f(\alpha)) \wedge closest_f(\hat{\alpha}_i, \hat{\alpha}_j)\} \\
E(G'_b(\alpha)) &= \{\langle \hat{\alpha}_i, \hat{\alpha}_j\rangle | \langle \hat{\alpha}_i, \hat{\alpha}_j\rangle \in E(G_b(\alpha)) \wedge closest_b(\hat{\alpha}_j, \hat{\alpha}_i)\}
\end{aligned}$$

where

$$\begin{aligned}
closest_f(\hat{\alpha}_i, \hat{\alpha}_j) &\stackrel{def}{=} (i < j) \wedge \neg(\exists \hat{\alpha}_{j'} \in V(G_f(\alpha)) : \hat{\alpha}_{j'} \approx \hat{\alpha}_j \wedge j' < j) \\
closest_b(\hat{\alpha}_j, \hat{\alpha}_i) &\stackrel{def}{=} (i > j) \wedge \neg(\exists \hat{\alpha}_{j'} \in V(G_b(\alpha)) : \hat{\alpha}_{j'} \approx \hat{\alpha}_j \wedge j' > j).
\end{aligned}$$

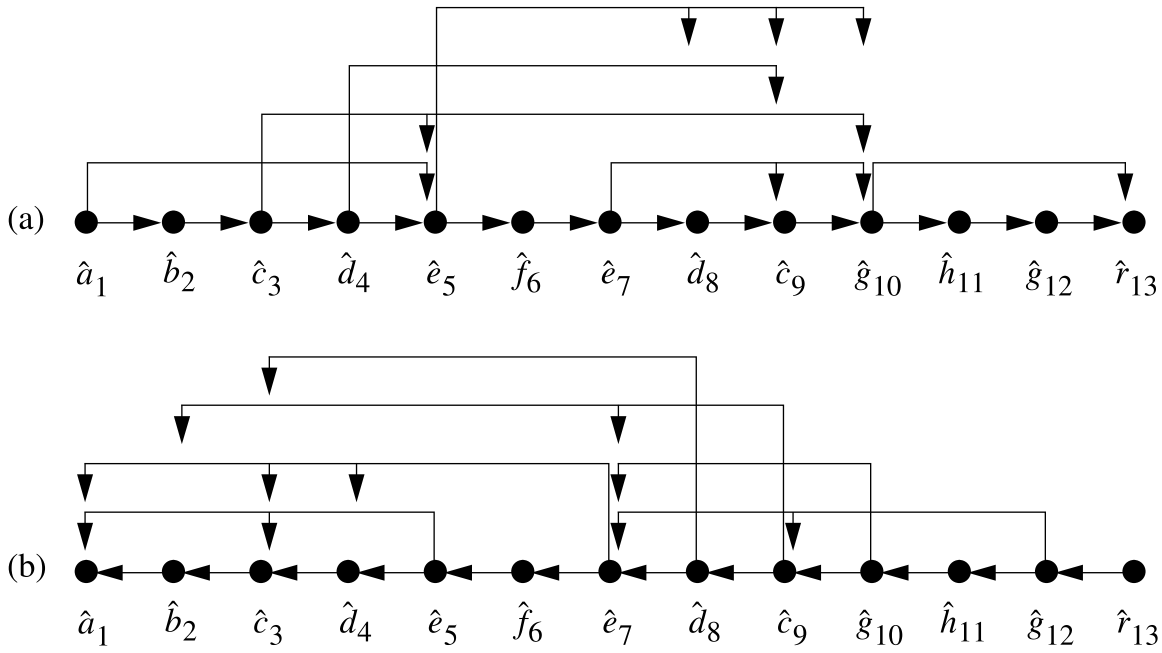


Figure 4: The reduced virtual graphs corresponding to the α_T in Fig. 3 after eliminating some edges: (a) f-graph G'_f and (b) b-graph G'_b.

That is, for each $pch\langle x, y\rangle$, an edge is introduced in G'_f from each virtual node $\hat{x}$ to the first $\hat{y}$ on $\hat{x}$'s right. Similarly, an edge is introduced in G'_b from each $\hat{x}$ to the first $\hat{y}$ on $\hat{x}$'s left. For example, the G'_f and G'_b corresponding to the α_T in Fig. 3 are shown in Fig. 4. It is to be noted that in G'_f, each of $\hat{e}_5$ and $\hat{e}_7$ has an edge leading to $\hat{g}_{10}$, but not to $\hat{g}_{12}$ because $\hat{g}_{10}$ is the first occurrence from $\hat{e}_5$ and $\hat{g}_{12}$ is the second occurrence. Similarly, in G'_b, each of $\hat{g}_{10}$ and $\hat{g}_{12}$ has an edge leading to $\hat{e}_7$, but not to $\hat{e}_5$.

Although the sets of edges in G'_f and G'_b are subsets of those of G_f and G_b, the following lemma indicates that the former are as powerful as the latter in terms of routing capability (note that this may not be true for arbitrary trips).

Lemma 1 *Let α be the trip. Given any path $p = \hat{\alpha}_{s_1} \to \hat{\alpha}_{s_2} \to \cdots \to \hat{\alpha}_{s_m}$ on $G_f(\alpha)$ (resp., $G_b(\alpha)$), there must exist another path $p' = \hat{\alpha}_{s'_1} \to \hat{\alpha}_{s'_2} \to \cdots \to \hat{\alpha}_{s'_m}$ on $G'_f(\alpha)$ (resp., $G'_b(\alpha)$) such that (i) $\alpha_{s_i} \approx \alpha_{s'_i}$ for all $i = 1..m$, and (ii) p and p' use the same sequence of f-channels (resp., b-channels).*

Proof. We prove the part for $G'_f(\alpha_T)$ only. Let $\hat{\alpha}_{s'_1}$ be the leftmost virtual node such that $\hat{\alpha}_{s'_1} \approx \hat{\alpha}_{s_1}$. Note that $s'_1 \leq s_1$. For $i = 2$ to m, let $\hat{\alpha}_{s'_i}$ be the virtual node such that $\langle \hat{\alpha}_{s'_{i-1}}, \hat{\alpha}_{s'_i}\rangle \approx \langle \hat{\alpha}_{s_{i-1}}, \hat{\alpha}_{s_i}\rangle$. Note that $s'_i \leq s_i$. The success of such construction can be proved by an induction based on the fact that before each construction of $\hat{\alpha}_{s'_i}$, the condition $s'_{i-1} \leq s_{i-1}$ holds. Furthermore, because there is only one f-channel along $pch\langle \alpha_{s_{i-1}}, \alpha_{s_i}\rangle$, $\langle \hat{\alpha}_{s'_{i-1}}, \hat{\alpha}_{s'_i}\rangle$ must be associated with the same f-channel as $\langle \hat{\alpha}_{s_{i-1}}, \hat{\alpha}_{s_i} \rangle$ is. Hence the lemma holds true. $\Box$

3.2 The Algorithm

We now make use of G'_f and G'_b to derive our multicast algorithm. Consider the problem of a node x multicasting a message m to a set of destinations D, $x \notin D$. We first convert D into two sets of virtual nodes D_f and D_b such that the following three conditions are satisfied: (i) each node $y \in D$ is converted into a virtual node $\hat{y}$ and is assigned to either set D_f or set D_b; (ii) if $\hat{y}$ is assigned to D_f, then $closest_f(leftmost(x), \hat{y})$ is true in G'_f; and (iii) if $\hat{y}$ is assigned to D_b, then $closest_b(\hat{y}, rightmost(x))$ is true in G'_n. For instance, in Fig. 4, if e is the source node and c is a destination node then c may be converted to $\hat{c}_9$ and assigned to D_f or converted to $\hat{c}_3$ and assigned to D_b. However, if e is the source node and h is the destination node then h will be converted to $\hat{h}_{11}$ and assigned to D_f uniquely. This is because node h is reachable from node e in G'_f only. For destination nodes which are reachable by both f-worm and b-worm, one can use heuristics to either balance the load in both worms or minimize path length to reduce the overall multicast latency. In our example shown later, we use a load balance strategy.

Intuitively, the above conversion translates our multicast problem (in G) into two multicast subproblems in G'_f and G'_b:

$\mathbf{P}_f$**:** Multicasting m along network G'_f from node $leftmost(x)$ to the set D_f of virtual nodes.

$\mathbf{P}_b$**:** Multicasting m along network G'_b from node $rightmost(x)$ to the set D_b of virtual nodes.

To formulate the above conversion, we define some labeling functions. Let x and y be any two nodes in $\mathcal{V}$, and $\hat{x}_i$ and $\hat{y}_j$ be any virtual nodes of x and y, respectively. Define

$$\lambda_f(x,y) = \begin{cases} \min\{i \mid \forall \hat{x}_i\} & x = y \\ j & x \neq y \land (\exists \hat{x}_i, \hat{y}_j : closest_f(\hat{x}_i, \hat{y}_j)) \\ \bot & \text{otherwise} \end{cases}$$

$$\lambda_b(x,y) = \begin{cases} \max\{i \mid \forall \hat{x}_i\} & x = y \\ j & x \neq y \land (\exists \hat{x}_i, \hat{y}_j : closest_b(\hat{y}_j, \hat{x}_i)) \\ \bot & \text{otherwise} \end{cases}$$

Functions $\lambda_f(x,x)$ and $\lambda_b(x,x)$ indicate the indices of the virtual nodes $leftmost(x)$ and *rightmost*(x), respectively. This will facilitate message initiation as discussed later. When $x \neq y$, function $\lambda_f(x,y)$ identifies the location of the first $\hat{y}_j$ that is on $leftmost(x)$'s right and function $\lambda_b(x,y)$ identifies the location of the first $\hat{y}_j$ that is on $rightmost(y)$'s left. Whenever no such $\hat{y}_j$ exists, the functions are undefined ($\bot$), which is to indicate that y is not reachable from x along network G'_f (or G'_b). The reader should notice that each point in these functions is uniquely defined because α is restricted to those satisfying Theorem 1 or Theorem 3.

For instance, some of the labeling functions for the example in Fig. 4 are listed as follows: $\lambda_f(d,d) = 4, \lambda_b(d,d) = 8, \lambda_f(d,c) = 9, \lambda_b(d,c) = 3, \lambda_f(d,g) = 10, \lambda_b(d,g) = \bot, \lambda_f(d,b) = \bot$, and $\lambda_b(d,b) = 2$. Note that it is a result of our definition that for any two nodes x and y, at most one of $\lambda_f(x,y)$ and $\lambda_b(x,y)$ is undefined.

Procedure *Message_Initiation* (D, m)
/* Assumption: node x calls this procedure. */
inputs:
D = the set of destination nodes, $x \notin D$;
m = the multicast message;
begin
1) Let $D_f = \emptyset$ and $D_b = \emptyset$;
for each $y \in D$ **do**
a) **if** $(\lambda_f(x,y) \neq \perp) \wedge (\lambda_b(x,y) = \perp)$ **then** $D_f := D_f \cup \{\lambda_f(x,y)\}$;
b) **elseif** $(\lambda_f(x,y) = \perp) \wedge (\lambda_b(x,y) \neq \perp)$ **then** $D_b := D_b \cup \{\lambda_b(x,y)\}$;
c) **else** randomly execute $D_f := D_f \cup \{\lambda_f(x,y)\}$ or $D_b := D_b \cup \{\lambda_b(x,y)\}$;
2) Sort D_f in the ascending order into a sequence S_f;
Sort D_b in the descending order into a sequence S_b;
3) **if** S_f is not empty **then**
call *Forward_Propagation*(x, x, S_f, m);
4) **if** S_b is not empty **then**
call *Backward_Propagation*(x, x, S_b, m);
end.

Figure 5: Message initiation procedure.

Now we are ready to present the routing algorithm. For a node x to multicast a message m to a set of destination nodes D, node x calls the procedure as shown in *Message_Initiation*(D, m) in Fig. 5. Step 1 converts D into two subsets D_f and D_b. Note that in step 1c, for the purpose of load balance, a node y reachable by x through both G'_f and G'_b is randomly put in D_f and D_b. In step 2, D_f and D_b are sorted from left to right and from right to left into two sequences S_f and S_b, respectively. Intuitively, this means that an f-worm is going to visit nodes in G'_f according to the sequence S_f and a b-worm is going to visit nodes in G'_b according to the sequence S_b. Steps 3 and 4 are to initiate two worms by calling the message propagation procedures shown later. Note that in both procedure calls, the first two parameters are both set to x. As to be shown later, this will bring special meanings to the procedures, making the f-worm and b-worm be initiated from $leftmost(x)$ in G'_f and from $rightmost(x)$ in G'_b, respectively, as stated in $\mathbf{P}_f$ and $\mathbf{P}_b$.

The message propagation procedures are shown in Fig. 6. Procedure *Forward_Propagation*() is to be executed in two cases: (i) when a node x wants to initiate a f-worm (S_f, m) or (ii) when a node x receives a f-worm (S_f, m) from another node w along a f-channel $vch_f\langle w, x\rangle$. In either case, we interpret this as an f-worm "arriving at" the virtual node $\hat{x}_i$ in G'_f, where $i = \lambda_f(w, x)$. Observe that for case (i), because $w = x$, when *Forward_Propagation*() is first called, $\hat{x}_i$ is the virtual node $leftmost(x)$. This matches with $\mathbf{P}_f$, and explains why the first two parameters are set equal in the procedure call in message initiation part.

In step 1, the if-condition is to check whether the message is destined to node x. If so, a copy of m is sent to x, and node $\hat{x}_i$ is removed from S_f. In step 2, if S_f is not empty, then there are more destinations to cover. Assume $\hat{z}_k$ is the next destination such that $k = head(S_f)$. Node x needs to select a neighboring node y to which x can propagate the message. Recall that the message is imagined as arriving at node $\hat{x}_i$. We wish to select a virtual node $\hat{y}_j$ that is to the right of $\hat{x}_i$ but not beyond the virtual node $\hat{z}_k$ (i.e., $i < j \leq k$). The step 2.a.i specifies such a condition. Note that the index j of $\hat{y}_j$ has been interpreted using the labeling function $\lambda_f(x, y)$. Step 2.a.ii is a heuristic which favors free virtual channels. Step 2.a.iii is another heuristic which favors those $\hat{y}_j$ that are "closer" to $\hat{z}_k$. Finally, in step 2.b, a message (S_f, m) is sent to node y along $vch_f\langle x, y\rangle$. The following is an invariant ensured by our algorithm:

INV$_f$: Whenever *Forward_Propagation*() is called, the condition $\lambda_f(w, x) \leq head(S)$ holds.

This invariant ensures the existence of $\hat{y}_j$ and thus the correctness of the algorithm. Also note how our argument proceeds: when y receives a message, we interpret this as the message arriving at node $\hat{y}_j$ such that $j = \lambda_f(x, y)$.

Similar to the forward propagation procedure, procedure *Backward_Propagation*() in Fig. 6 is executed in two cases: (i) when a node x wants to initiate a b-worm (S_b, m) or (ii) when a node x receives a b-worm (S_b, m) from another node w along the b-channel $vch_b\langle w, x\rangle$. We interpret this as a node "arriving at" the virtual node $\hat{x}_i$ in G'_b, where $i = \lambda_b(w, x)$. Observe that for case (i), how this interpretation matches with what was stated in $\mathbf{P}_b$. Assuming that $\hat{z}_k$ is the next destination such that $k = head(S_b)$, we wish to select a virtual node $\hat{y}_j$ which is to the left of $\hat{x}_i$ but not beyond the virtual node $\hat{z}_k$ (i.e., $k \geq j < i$). Step 2.a.i specifies this condition. Similarly, the index j of $\hat{y}_j$ has been interpreted using the labeling function $\lambda_b(x, y)$. The following invariant is ensured for G'_b:

INV$_b$: Whenever *Backward_Propagation*() is called, the condition $\lambda_b(w, x) \geq head(S)$ holds.

We explain the routing algorithm with an example multicast pattern on Fig. 3. Suppose node d needs to multicast a message m to a destination set $D = \{b, c, g, h\}$. Set $\{c, g, h\}$ is reachable by d along G'_f and set $\{b, c\}$ reachable along G'_b. For load balance, let's choose $\{g, h\}$ to be reached by an f-worm, and $\{b, c\}$ by a b-worm. Thus, $S_f = [\lambda_f(d, g), \lambda_f(d, h)] = [10, 11]$ and $S_b = [\lambda_b(d, c), \lambda_b(d, b)] = [3, 2]$. The f-worm will start from $\hat{d}_4$ and the b-worm from $\hat{d}_8$. The real paths traversed by the f- and b-worms will dynamically depend on the traffic in the system. In Fig. 7, we show some (but all) possible routes that could be traversed by the f-worm. It clearly demonstrates the adaptive capability of the algorithm.

3.3 Complexity Analysis and Significance of the Model

In this section, we analyze the implementation details of our scheme with respect to complexity of routing table, communication start-up and propagation overheads, and architectural supports.

Procedure *Forward_Propagation* (w, x, S_f, m)
/* Assumption: called by x on receiving (S_f, m) along $vch_f\langle w, x\rangle$. */
inputs:
w = the previous node;
x = the current node, $(w, x) \in \mathcal{E}$;
S_f = a sequence of node numbers (in ascending order);
m = the multicast message;
begin
1) **if** $head(S_f) = \lambda_f(w, x)$ **then** pass m to x and remove $head(S_f)$ from S_f;
2) **if** S_f is not empty **then**
   a) Select a node y neighboring to x according to the following rules:
      i) $\lambda_f(w, x) < \lambda_f(x, y) \leq head(S_f)$;
      ii) The selection favors those y such that $vch_f\langle x, y\rangle$ is free;
      iii) The selection favors those y such that $\lambda_f(x, y)$ is larger;
   b) Propagate (S_f, m) to node y along $vch_f\langle x, y\rangle$;
end.

Procedure *Backward_Propagation* (w, x, S_b, m)
/* Assumption: called by x on receiving (S_b, m) along $vch_b\langle w, x\rangle$. */
inputs:
w = the previous node;
x = the current node, $(w, x) \in \mathcal{E}$;
S_b = a sequence of node numbers (in descending order);
m = the multicast message;
begin
1) **if** $head(S_b) = \lambda_b(w, x)$ **then** pass m to the processor and remove $head(S_b)$ from S_b;
2) **if** S_b is not empty **then**
   a) Select a node y according to the following rules:
      i) $head(S_b) \leq \lambda_b(x, y) < \lambda_b(w, x)$;
      ii) The selection favors those y such that $vch_b\langle x, y\rangle$ is free;
      iii) The selection favors those y such that $\lambda_b(x, y)$ is larger;
   b) Propagate (S_b, m) to node y along $vch_b\langle x, y\rangle$;
end.

Figure 6: Message propagation procedures.

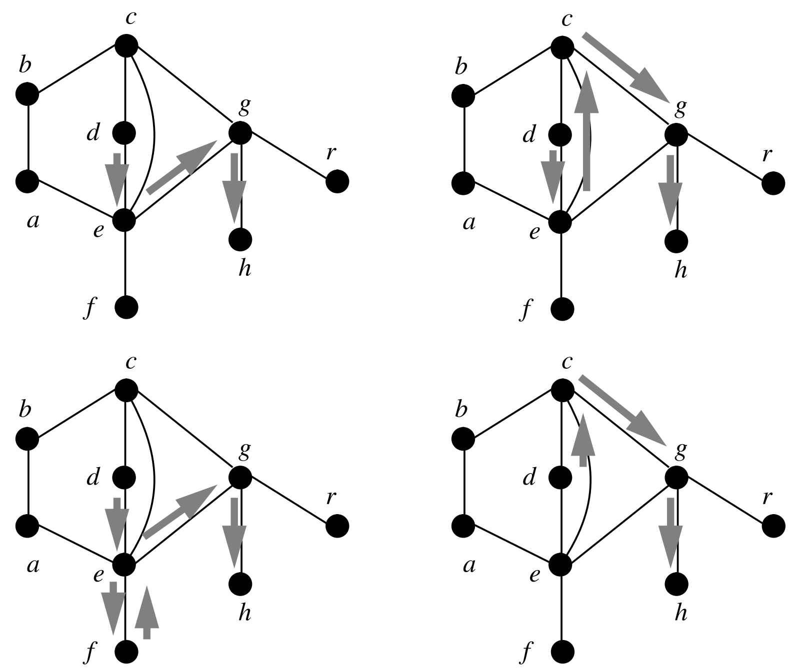


Figure 7: Possible routing alternatives of a f-worm starting from node d to destinations $\{g, h\}$ under the skirt-based trip model. Alternative routing paths reflect the adaptivity built into the algorithm.

We also emphasize on the significance of our model.

A) Complexity of Routing Table: To initiate messages, the router interface corresponding to node x needs to keep functions $\lambda_f(x,*)$ and $\lambda_b(x,*)$. Similarly, to propagate messages, the router interface needs to keep functions $\lambda_f(x,x)$, $\lambda_b(x,x)$, $\lambda_f(*_{in},x)$, $\lambda_b(*_{in},x)$, $\lambda_f(x,*_{out})$, and $\lambda_b(x,*_{out})$, where $*$ = "any node," $*_{in}$ = "any node incident into x with a link," and $*_{out}$ = "any node incident by x with a link." Thus the storage overhead for message initiation is $2|\mathcal{V}|$ while that for message propagation is $\leq 2+4d$, where d is the degree of x in G. It is to be noted that such scheme involves routing table to be stored at the router interface. It has been shown in the current Cray T3D system [7] that such routing table approach leads to flexibility in providing easy system reconfigurability and fault-tolerance.

B) Communication Overhead: The message initiation is dominated by the sorting operations and has complexity $O(|D|\log|D|)$, where D is the destination set. Routing decision during the message propagation is very simple and is dominated by the time to search through each outgoing channel. If the labeling functions $\lambda_f(x,*_{out})$ and $\lambda_b(x,*_{out})$ are pre-sorted, then a binary search will need $O(\log d)$ time, where d is the degree of the node. If a linear search is used, it will need $O(d)$ time. As the current architectures tend to adopt low- or even constant-degree networks, the routing decision overhead could be very low. This overhead is similar to that in [21, 22].

Considering the architecture to support our model, we suggest that the more complex message initiation algorithm be implemented by software at the source node, while the simpler routing decision logic (which contains a small table and a search procedure) get implemented by hardware inside the routers. This is just as done in most current generation systems. Also note that our message propagation procedures do not need to read all content of a message in order to route the message. To select the next node y, *Forward_Propagation*() only needs the first number in S_f if the message is not directed to x; or the first 2 numbers in S_f otherwise. This is true for *Backward_Propagation*() too. So the message should be organized such that S_f/S_b is followed by m to exploit the pipelined transmission feature of wormhole routing. Similar suggestion has been made by the Hamiltonian path-based scheme [21, 22],

C) Significance of the Model: As a skirt-based trip always exists in any connected graph, our model can be used in any network that supports 2 virtual channels per physical channel. If the network topology may change (e.g., because nodes fail or rejoin after being fixed), we can reconfigure our model by finding another trip and updating the labeling function in each node. Reconfiguration needs to collect global topology information. Such algorithms do exist [2, 6]. As the network remaining connected is the minimum requirement to perform multicast, our algorithm has the maximum (static) fault-tolerant capability. It is also to be noted that a special case of our model is the Hamiltonian path model [21, 22], where the network is Hamiltonian. In such a case, there is no requirement of virtual channels as discussed in Section 2.4. With only 2 virtual channels per physical channel, our skirt-based-trip result is much more powerful and can be applied to network with arbitrary topology as long as the network remains connected due to fault. Hence, our result is a significant improvement over the solutions proposed in [21, 22].

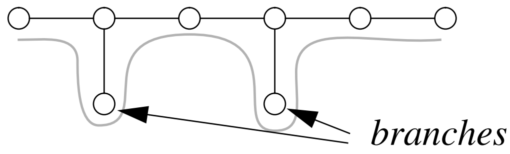


Figure 8: A caterpillar and its associated skirt-based trip.

4 Fault-Tolerant Multicast in Hypercubes

In this section, we apply our proposed model to faulty hypercubes. For an n-dimensional hypercube, or n-cube, we show how to perform trip-based multicast with at most $n-1$ faulty nodes. Simulation results for single-source and multiple-source multicast are presented with variable number of faults.

4.1 Trip Construction in Faulty Hypercubes

First we show how to construct skirt-based trips in faulty hypercubes. Before constructing such trips, we introduce a concept of *caterpillar* as follows:

Definition 5 A *caterpillar* is a linear path extended with some *branches*. Each non-terminal node in the path can be extended with at most one branch, where a branch is a node not in the path joined by an edge to a node in the path and no two branches share a node.

Thus, a caterpillar may be regarded as an "approximate" Hamiltonian path. For instance, Fig. 8 shows a caterpillar with 2 branches and a skirt-based trip on it. Given a caterpillar T, we define α_T to be the skirt-based trip induced from T such that α_T is the linear path in T extended by replacing each x with x, y, x for every occurrence of x that has a branch node y. The following lemma shows that such trips induce short trip lengths as well as low resources requirement to implement our model.

Lemma 2 *Let α_T be the trip induced from a caterpillar T.*

(a) The length of α_T is $|\mathcal{V}|+h$, where h is the number of branches in T.

(b) If x is a non-degree-3 node, then for any defined point, $\lambda_f(*, x) = \lambda_b(*, x)$, where $*$ means "any node."

(c) If x is a degree-3 node with y being the node branched out from x, then for any defined point, $\lambda_f(*', x) = \lambda_b(y, x) = leftmost(x)$ and $\lambda_f(y, x) = \lambda_b(*', x) = rightmost(x)$, where $*'$ = "any node except y."

In the following, we show that in an n-cube with at most $n-1$ faults, a caterpillar can be found, thereby enjoying the properties given in Lemma 2. We make use of a graph-embedding result developed previously by us in [30], which shows that in an n-cube with $f \le n-1$ faults,

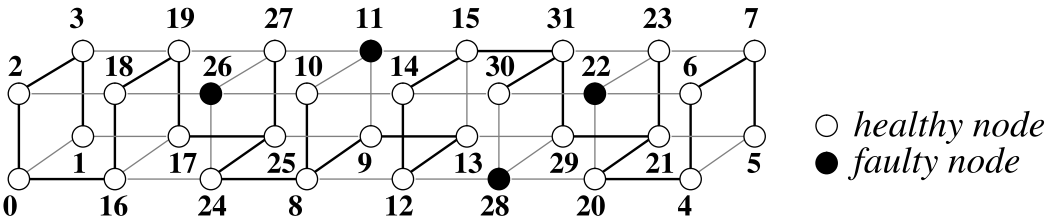


Figure 9: A caterpillar in a 5-cube with 4 faulty nodes.

there exists a subgraph of the cube such that i) the subgraph is a 3-D mesh of size $2 \times 2 \times 2^{n-2}$, ii) each 2×2 mesh in the first 2 dimensions (of the 3-D mesh) has at most 1 faulty node, and iii) these 2×2 meshes are arranged along the third dimension in a binary-reflected Gray-code manner. For instance, Fig. 9 shows a subgraph of a 5-cube with 4 faults (some edges of the cube are removed for clarity).

It is not hard to observe that the above mentioned 3-D mesh is still a connected graph even with those faulty nodes removed. In fact, it is possible to connect all healthy nodes together as a caterpillar, by simply joining nodes in one 2×2 mesh after another along the third dimension. The caterpillar will be of $(2^n - f - h)$ long with $h \leq f$ branches. For instance, the solid lines in Fig. 9 form a caterpillar with 4 branches. The reader can refer to [30] for details. This, together with Lemma 2, leads to the following theorem:

Theorem 4 *In an n-cube with $f \leq n - 1$ faulty nodes, there exists a caterpillar T whose corresponding skirt-based trip α_T is no longer than 2^n.*

4.2 Simulation Experiments and Results

We performed simulation experiments to evaluate the performance of multicast patterns on caterpillars in faulty hypercubes. We studied both single-source and multiple-source multicast patterns.

A) Average Hop Count: We simulated single-source multicast in a faulty n-cube with $f \leq n - 1$ faulty nodes. We observed the total numbers of hops being traversed by both the f-worm and the b-worm in a network free of other traffic. All sources, destinations, and faults were generated randomly. For every fault configuration, caterpillar was constructed as described earlier. Figure 10 shows the average hop count required by a single-source multicast pattern for varying number of destinations in a 6-cube with faults. Average number from 1000 simulations are reported. It can be observed that when the number of destinations is small (roughly < 40), the hop counts are indistinguishable with different faults (f). When the number of destinations is larger, interestingly, the hop counts start to bent down as f increases. This is because the worms hardly take short-cuts, giving a hop count close to the trip length. While the trip length in a fault-free cube is always 2^n, the trip length in a faulty cube could be less as shown in the previous section, but no more than 2^n.

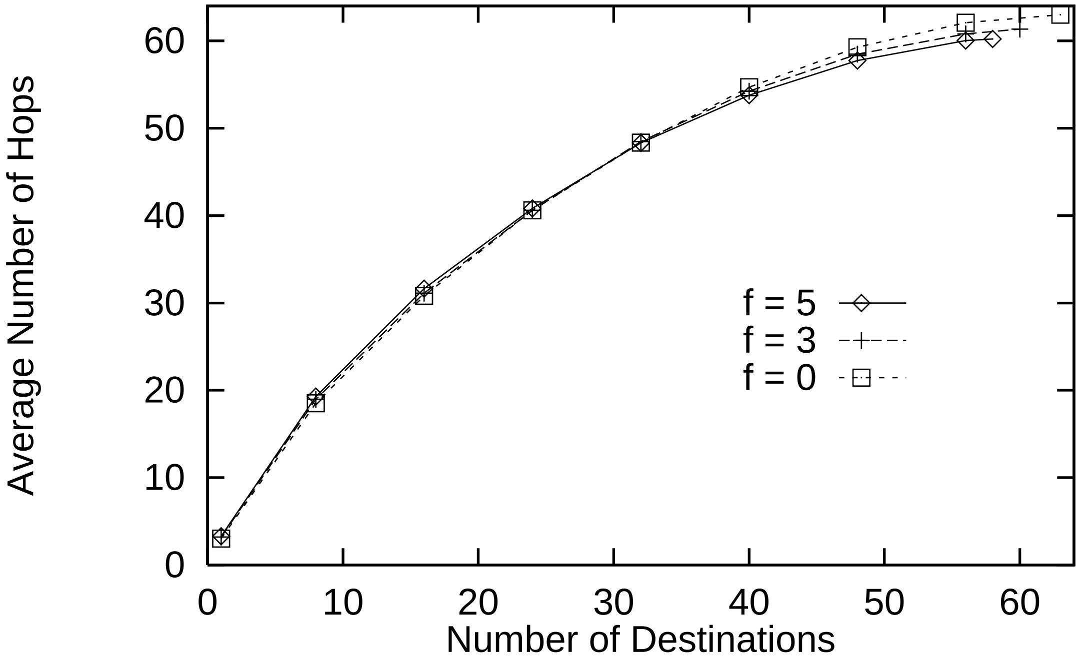
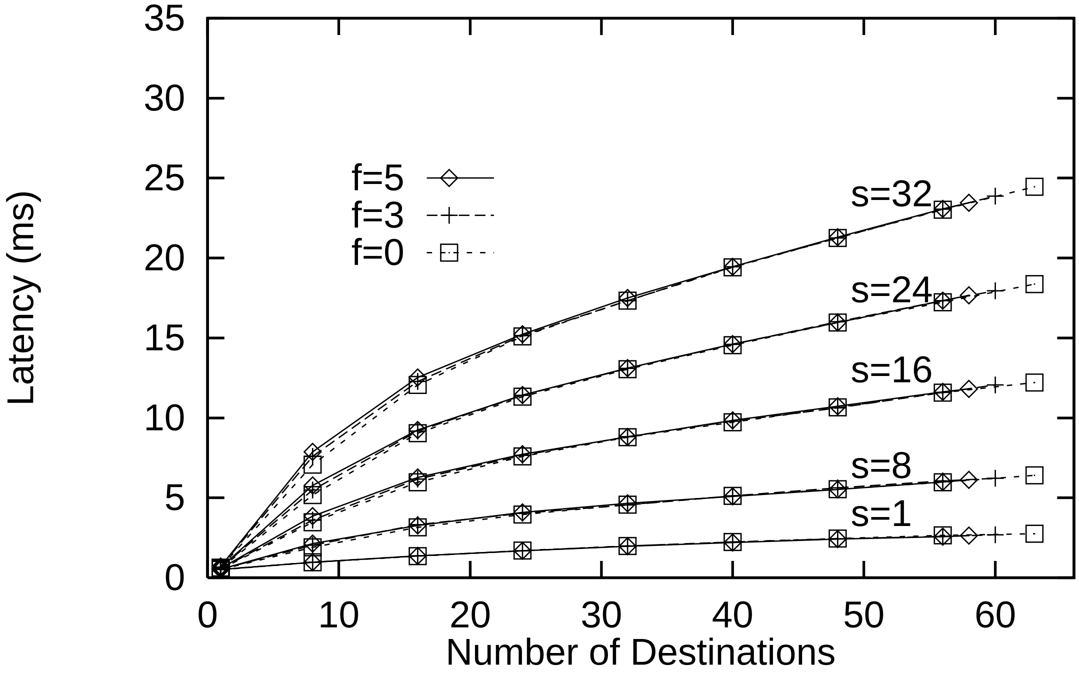


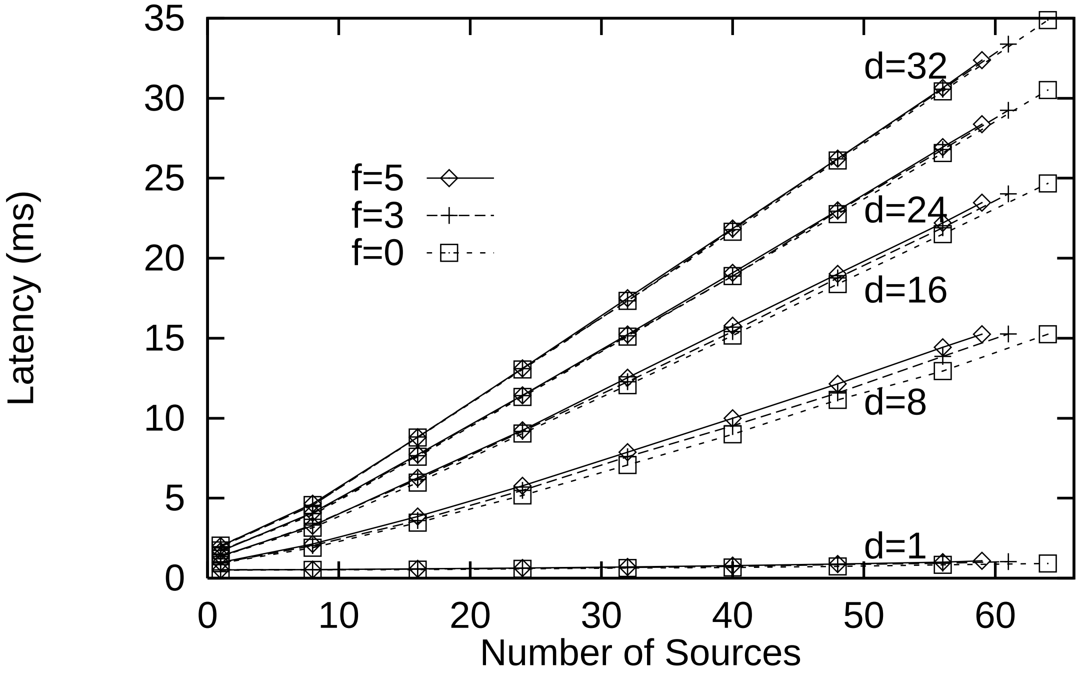
Figure 10: The average hop counts to perform single-source multicast in a 6-cube with f faulty nodes.

B) Multiple Multicast Latency: Next we investigated the performance of multiple multicast by modeling hypercubes with 2 virtual channels per physical channel. We used a CSIM-based [27] flit-level simulator with demand multiplexing of virtual channels. Virtual channels were arranged according to $\mathbf{VC}_f$ and $\mathbf{VC}_b$ allocation discussed in Section 2.4. However, for fairness, if $\mathbf{VC}_f$ did not require a f-channel then both virtual channels were used by $\mathbf{VC}_b$ as b-channels, and vice versa. We assumed the following system parameters representing the current trend in technology: 0.01 microsecond to transmit a flit across a physical channel, an overhead of 0.01 microsecond inside a router for making routing decision, and 0.01 microsecond for routing (e.g., buffering, switching, etc.). Each multiple multicast pattern started synchronously and we measured the time for the entire pattern to get finished. A message length of 20 flits (excluding the header) was assumed. We observed the communication latency with varying number of sources (s), destinations (d), and faults (f).

The simulation results for a 6-cube are shown in Fig. 11, where each point has been derived from 100 simulations. It can be observed that with trips constructed from caterpillars, the performance degradation due to faults is almost negligible. Also, the performance of trip-based routing, similar to the Hamiltonian-path-based routing, is more sensitive to the number of sources (s) than the number of destinations per source (d). With a fixed d, the latency increases about linearly with in s. With a fixed s, the latency increases sub-linearly with d.



(a)



(b)

Figure 11: The average multicast latency to perform multiple multicast in a 6-cube with faults f = 0, 3, and 5, when (a) number of sources s = 1, 8, 16, 24, and 32 with various destinations (d), and (b) d = 1, 8, 16, 24, and 32 with various s's.

5 Conclusions

We have presented a trip-based multicast model for wormhole-routed multicomputer networks with virtual channels. Fundamental properties of this model have been investigated. A skirt-based trip algorithm for routing single/multiple multicast has been proposed that can be applied to any connected network. This solution requires at most 2 virtual channels per physical channel on networks with arbitrary topologies (regular as well as irregular resulted due to faults). This result is a significant improvement over the Hamiltonian-path solution proposed earlier for Hamiltonian networks only.

Using skirt-based trips, we have shown that simple, distributed, adaptive, and deadlock-free multicast routing can be implemented with low communication overhead. The model has been applied to faulty hypercubes and it has been shown that multicast operations can be achieved with negligible performance degradation. The skirt-based-trip presented in this paper is based on any existing spanning tree of the network. It is an open problem how to select a best trip (spanning tree) for a given network in order to optimize message routing and maximize performance. The skirt-based trip requires at most 2 virtual channels. If a network supports more than 2 virtual channels, it is quite interesting to see how multiple trips can coexist in the system to reduce contention and increase system performance.

References

[1] MPI: A message-passing interface standard. In *Message Passing Interface Forum*, March 1994.

[2] H. Abu-Amara. A fast topology maintainance algorithm for high-bandwidth networks. *IEEE/ACM Trans. on Networks*, 1(3):386–394, June 1993.

[3] D. P. Bertsekas, C. Ozveren, G. D. Stamoulis, P. Tseng, and J. N. Tsitsiklis. Optimal communication algorithms for hypercubes. *J. of Parallel and Distrib. Comput.*, 11:263–275, 1991.

[4] G. I. Chen and T. H. Lai. Constructing parallel paths between two subcubes. *IEEE Trans. on Comput.*, 41(1):118–123, Jan. 1992.

[5] A. A. Chien and J. H. Kim. Planar-adaptive routing: Low-cost adaptive networks for multiprocessors. In *Int'l Symp. on Computer Architecture*, pages 268–277, 1992.

[6] I. Cidon, I. Gopal, and S. Kutten. New models and algorithms for future networks. In *Symp. on Principle of Distributed Computing*, pages 75–89, 1988.

[7] Cray Research, Inc. *Cray T3D System Architecture Overview*, 1993.

[8] W. J. Dally. Virtual-channel flow control. *IEEE Trans. on Paral. and Distrib. Sys.*, 3(2):194–205, March 1992.

[9] W. J. Dally and C. L. Seitz. The torus routing chip. *J. of Parallel and Distrib. Comput.*, 1(3):187–196, 1986.

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

[11] J. T. Draper and J. Ghosh. Multipath e-cube algorithm (MECA) for adaptive wormhole routing and broadcasting in k-ary n-cubes. In *Int'l Parallel Processing Symp.*, pages 407–410, 1992.

[12] J. Duato. A new theory of deadlock-free adaptive multicast routing in wormhole networks. In *Symp. on Parallel and Distributed Processing*, pages 64–71, 1993.

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

[14] C. J. Glass and L. M. Ni. The turn model for adaptive routing. In *Int'l Symp. on Computer Architecture*, pages 278–287, 1992.

[15] C.-T. Ho and M.-Y. Kao. Optimal broadcast on hypercubes with wormhole and ecube routings. In *Int'l Conf. on Parallel and Distributed Systems*, pages 694–697, 1993.

[16] J. M. Jaffe. Distributed multi-destination routing: the constraints of local information. *SIAM J. Comput.*, 14(4):875–888, Nov. 1985.

[17] S. L. Johnsson and C. T. Ho. Optimal broadcasting and personalized communication in hypercubes. *IEEE Trans. on Comput.*, 38(9):1249–68, Sep. 1989.

[18] Y. Lan. Multicast in faulty hypercubes. In *Int'l Conf. on Parallel Processing*, pages I–58–61, 1992.

[19] Y. Lan, A.-H. Esfahanian, and L. M. Ni. Multicast in hypercube multiprocessors. *J. of Parallel and Distrib. Comput.*, 8:30–41, 1990.

[20] D. Lenoski et al. The Directory-Based Cache Coherence Protocol for the DASH Multiprocessor. In *Proceedings of the 17th Annual Symposium on Computer Architecture*, pages 148–159, May 1990.

[21] X. Lin, P. K. McKinley, and L. M. Ni. Performance evaluation of multicast wormhole routing in 2D-mesh multicomputers. In *Int'l Conf. on Parallel Processing*, pages I–435–442, 1991.

[22] X. Lin and L. M. Ni. Deadlock-free multicast wormhole routing in multicomputer networks. In *Int'l Symp. on Computer Architecture*, pages 116–124, 1991. (to appear in *IEEE Trans. on Paral. and Dist. Sys.*).

[23] D. H. Linder and J. C. Harden. An adaptive and fault tolerant wormhole routing strategy for k-ary n-cubes. *IEEE Trans. on Comput.*, 40(1):2–12, Jan. 1991.

[24] P. K. McKinley and C. Trefftz. Efficient broadcast in all-port wormhole-routed hypercubes. In *Int'l Conf. on Parallel Processing*, pages II–288–291, 1993.

[25] P. K. McKinley, H. Xu, A.-H. Esfahanian, and L. M. Ni. Unicast-based multicast communication in wormhole-routed networks. In *Int'l Conf. on Parallel Processing*, pages II–10–19, 1992.

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

[27] H. Schwetman. CSIM users' guide. Technical Report Tech. Rpt.: ACT-126-90, MCC.

[28] J.-P. Sheu and M.-Y. Su. A multicast algorithm for hypercube multiprocessors. In *Int'l Conf. on Parallel Processing*, pages III–18–22, 1992.

[29] G. D. Stamoulis and J. N. Tsitsiklis. Efficient routing schemes for multiple broadcasts in hypercubes. *IEEE Trans. on Paral. and Distrib. Sys.*, 4(7):725–739, July 1993.

[30] Y.-C. Tseng and T.-H. Lai. Ring embedding in an injured hypercube. In *Int'l Conf. on Parallel Processing*, pages III–149–152, 1993.