On Deadlock-free Multicasting in Hypercubes with Faulty Links

Abhijit Sengupta
Department of Computer Science
University of South Carolina
Columbia, SC 29208

and

Arunabha Sen
Department of Computer Science
Arizona State University
Tempe, AZ 85287

Abstract : This paper presents a deadlock-free algorithm for multicast routing in a faulty hypercube when some of the links have failed. Wormhole routing is assumed for message communication and the number of faulty links is assumed to be fewer than the dimension of the hypercube to ensure the faulty network is not segmented. The performance of the algorithm is analyzed through simulation experiments which shows a marginal degradation in communication delay because of faults in the network.

I. Introduction

In a distributed memory multiprocessor system, the processor nodes communicate by passing messages over the communication links. A message has to be routed through other nodes if the communicating nodes lack a direct link between them. The communication mechanism and the routing algorithm determine the communication time between two processors and hence plays an important role in the performance of the paralel system. In a multiprocessor system, the multicast routing problem (where the same message originating from one processor node has to be delivered to several other processor nodes) arises in numerous applications. It is known [8] that treating a multicast routing problem as a multiple one-to-one routing problem creates too much traffic with unacceptable delay and congestion. The communication delay in a multiprocessor system is also significantly influenced by the communication mechanism and the routing algorithm. Message communication between processors is usually accomplished by one of the three routing strategies, namely, store-and-forward [17], virtual cut-through [7], and wormhole routing [2]. In store-and-forward routing, the message is stored in its entirety at each intermediate node of the routing path leading to the destination node. In wormhole routing, the message is never stored at intermediate nodes and the routing is done by the router chip in each node which buffers only a few flow control digits or information (usually referred to as flits) of the head of a message packet and are used by the router to determine the link to be used to forward the packet. Once the link is available, the flits and the remaining part of the message flow as a pipeline. Hence, at each intermediate node, a packet gets forwarded directly from an incoming link to an outgoing link by the router in the forwarding node. However, the message might get blocked at an intermediate node if the forwarding router cannot get the required link because it is not available. This might happen if the link is currently being used or the node to whom the message is to be forwarded has its buffer full or the link is faulty. In wormhole routing, in case a packet is blocked, it is not stored at the intermediate node, it remains in the network. In virtual cut-through routing, on the other hand, the message forwarding occurs the same way as the wormhole routing excepting that a message gets stored at the intermediate node when it cannot be forwarded.

Among these different routing schemes [17], [7], [2]. wormhole routing is used in most of the recent machines. The wormhole routing strategy has several advantages over the others. It avoids the buffering time and the communication start-up time at intermediate nodes as well as the buffering space, Also the time needed to communicate the message from the source node to the destination node is virtually independent of the number of intermediate nodes between them, when there is no channel contention [10] and no message blocking. Parallel computers that adopt the wormhole routing strategy include the Symult 2010 [14], the Intel Paragon [6] and the nCUBE-2 [11].

Most parallel machines support some form of broadcast whereby, the same message is sent to all the processors in the network. Multicast operations, on the other hand, require the message to be sent to a subset (referred to as the multicast set) of the nodes. Thus broadcast is a special case of multicast. In the store and forward approach, since a message is buffered in every intermediate node, a tree type communication to implement the multicasting is natural. However, a tree type multicasting is difficult to use with wormhole routing strategy. The routers in each node handle only message forwarding from an incoming link to an outgoing link and saving messages destined for the current node. The routers usually donot copy messages nor can originate messages. In most machines, the startup time for message origination is expensive and hence a message is originated by the processor nodes only. Clearly, in a tree type multicast, the non-leaf nodes of the multicast tree will need message copying and origination to its children. In short, the wormhole routing loses its advantage for a tree type multicasting. It has also been shown [8] blocking probability increases significantly when tree type multicast is used. As suggested in [8], messages can move faster if a path like routing is used with an attempt to use short-cuts where possible. A deadlock-free routing algorithm has been presented in [8] using this type of routing in fault-free hypercube. In this paper, we consider the same principle of routing when some of the links have failed. It is simple to see that the algorithm presented in [8] does not work when some links are faulty.

In a multiprocessor interconnection network, deadlock can occur during message passing when none of the messages can progress towards its destination. This can occur when several processors have their queues of messages to be forwarded full [17] and are in a cyclic wait condition with each of these processors waiting for the next processor in the cycle to forward at least one message from its queue. Deadlock-free routing for store-and-forward routing uses the concept of structured buffer pool [4], [9], [18], [3] where message buffers (partitioned into buffer classes) in each processor are partially ordered, the ordering being given by some mapping of the messages to the buffers. Deadlock-free routing has been achieved for wormhole routing using the idea of virtual channels, where each physical link is considered to have several parallel virtual channels and some partial ordering relation is defined on these channels depending on the destination nodes.

Most of the deadlock-free routing algorithms use deterministic routing [3], [2], [8] although there are some adaptive and deadlock-free [12] algorithms. Most of these routing algorithms[1] (especially the deterministic algorithms), assume a fault-free network. In an earlier paper [20], one of the current authors has presented a deterministic deadlock-free, fault-tolerant multicasting algorithm in a hypercube considering node failures. This paper addresses the same problem considering link failures. Since each node failure can disable several links, it is expected that the performance degradation because of link failure should be less adverse than that in case of node failures. We assume that the number of link failures in a hypercube is less than its dimension so that the network is not segmented.

II. Definitions and Notations

[1] excepting [15] and [19], where [15] deals with one to one communication and not multicasting and [19] presents a brief discussion on multicasting in a faulty network

We assume that the interconnection between the processors is represented by a graph $G = (V, E)$ where V is the set of nodes (processors) and E is the set of edges (communication links between the processors). Standard graph theoretic terms are not defined, one might refer to [5] for them. An n-dimensional binary hypercube Q_n is a graph with 2^n nodes, where each node is denoted by an n-dimensional binary vector. Treating an n-dimensional binary vector as the binary representation of an integer, a node might be denoted by that equivalent integer. Each bit position is called a dimension of Q_n. Two nodes are adjacent iff their binary vector representations differ exactly in one bit position - if that bit position is i then the edge between the nodes is referred to as the ith dimensional edge. An n dimensional binary vector having only ith bit 1 and everything else 0 will be denoted by e_i. $\oplus$ will denote bitwise vector sum in modulo 2. For any node v_j of a Q_n, assuming v_j gives the binary vector representation of the node, the i-dimensional edge incident on v_j will be denoted by $v_j \overset{i}{\text{—}} v_j \oplus e_i$ or by the pair of nodes $(v_j, v_j \oplus e_i)$. Among the two endpoints of such an edge the node having the ith bit position 0 will be called the smaller endpoint of the edge. Let $Z = \{1, 2, \ldots, n\}$ represent the set of dimensions of a Q_n. Any string of symbols chosen from Z will be called a dimension sequence. For any permutation $\pi = x_1 x_2 \ldots x_n$ on Z, let $\sigma(\pi)$ denote the string $s_n * x_n$ ('$*$' is the concatenation operation), where $s_j, 2 \leq j \leq n$, is defined recursively as $s_j = s_{j-1} * x_j * s_{j-1}$ and $s_1 = x_1$. Clearly, $\sigma(\pi)$ is a string of length 2^n. For any node v_i, let $P(v_i, \pi)$ denote the walk traversed in Q_n as one starts from v_i and traverses the edges in the dimensions of $\sigma(\pi)$ in the order they appear in $\sigma(\pi)$. It is known [1] that for any v_i and π, $P(v_i, \pi)$ is a Hamiltonian cycle in Q_n. Assuming v_0 denote the node represented by the zero binary vector, $P(v_0, \pi)$ will be referred to as the canonical Hamiltonian cycle (HC) for the permutation π and will be denoted, in short, by $P(\pi)$. It is simple to see that the edge $v_j \text{—} v_k$ appears in $P(v_i, \pi)$ iff the edge $v_j \oplus v_i \text{—} v_k \oplus v_i$ appears in $P(\pi)$.

Throughout the paper, we will assume that F denotes the set of faulty links of the network and $|F| \leq n - 1$ to ensure that the graph obtained by removing the faulty links remains connected.

III. Deadlock-free Multicasting

In a Q_n, since the edge connectivity is n, the interconnection graph remains connected even in the presence of $n-1$ link failures. Throughout this paper, we make the following assumptions.

- Wormhole routing is used for message passing.
- Only links can fail and the number of faulty links is at most $n-1$.
- For any message, the paths used to reach the destination nodes depend only on the destination nodes and on the set of faulty links. The paths are independent of the number of messages or the distribution of the messages in the network.
- Any node can send messages of any length to other nodes at any rate.
- A message cannot be copied by the router in a forwarding node. Communication is started by the source node of the message only.
- For each channel incident at a node, there is a queue whose size gives an upper bound of the number of flits that can be buffered for that channel.
- Every node knows the identity of all the faulty links. It is conceivable that readers might question the validity of this assumption; we will explain subsequently the identity of faulty links can be known by the nodes in a distributed manner.
- Adaptive routing is not considered.

In wormhole routing, the router in each node, depending on the destination node, forwards the head of the packet from an incoming channel to the node to one of its outgoing channels. This routing can be represented by the function $\rho : V \times E \rightarrow E$ which implies that a message received from the link $(v_i, v_j) \in E$ and having a destination $v_d \in V$ forwards the message using the link$(v_j, v_k) \in E$ where all the links used are the fault-free links. On the basis of the routing function ρ, a channel dependency digraph was defined in [14] using virtual channels. Each link of G is divided into several virtual channels and the ρ function defines the message transfer, for each destination, from one virual channel to another. In the channel dependency digraph, each virtual channel is represented by a node and a directed edge exists from a virtual channel of the link (v_i, v_j) to a virtual channel of the link (v_j, v_k) iff for some destination, ρ routes from (v_i, v_j) to (v_j, v_k). It has been shown [14] that ρ is deadlock-free iff the channel dependency digraph is acyclic. Thus a simple way to achieve deadlock-free routing is to define a routing function that establishes a partial ordering relation among the links of the network. Suppose we define a one-to-one function $h : V \rightarrow N$ where $N = \{0, 1, 2, \ldots, |V| - 1\}$ and define two virtual channels for each link of G, call them *high* and *low*. A message flow along an edge (v_i, v_j) is through the *high* (*low*) virtual channel if the message flows from v_i to v_j and $h(v_i) > h(v_j)$ $(h(v_i) < h(v_j))$. Clearly if the routing function ρ is so chosen that a message flow path uses either all high virtual channels or all low virtual channels, then there cannot exist a cycle in the channel dependency digraph and as such the routing is deadlock-free.

To define the routing function in a hypercube with faulty links, we apply the above strategy. To determine the function h, we note that a Q_n with at most $n-1$ faulty links has a Hamiltonian path[2] (HP) and the h function is given by the distance of each node from one end of the path. In short, once an HP has been found, the h function renumbers the nodes. This renumbering is then used to achieve a deadlock-free multicasting using a path routing so that the problems of tree type multicasting in wormhole routing does not arise. It is conceivable that objections might be raised to the fact that routing along a hamiltonian path will cause significant delay in the routing. However, if the source node has p faultfree links then, the set of nodes destination nodes are divided into p disjoint subsets and the constructed HP is also divided into p segments such that all the nodes in each subset appear in one segment. The source node sends the message to be multicast to all its p neighbors which intheir turn forward the message in each segment concurrently. We will discuss how short-cuts can be used to avoid long paths. We compare the performance of our algorithm in the faulty link situation to that of the algorithm given in [8]. which considers fault-free Q_n.

A. Hamitonian Path in Faulty Q_n.

Since the multicast algorithm presented in [8] assumes a faultfree Q_n, the labeling of each node, in

[2] actually there are several Hamiltonian paths, and we define an algorithm to find one such Hamiltonian path and guarantee that all nodes find the same Hamiltonian path

short, the h function was assumed to be given by the binary reflected gray code and such a coding is known to provide a HC in the Q_n. In a faulty Q_n, on the other hand, we need to specify an algorithm to find the h function when the faulty link set is known. As mentioned before, we determine a HP in the faulty Q_n to define the h function. It has been shown in [16] that a Q_n has at least $2(n-2)!$ canonical HCs when the number of faulty links do not exceed $n-2$. Hence, we also need to ensure that every node determines the same h function when there are so many HCs. In the following, we first address this issue. We define an algorithm A that will be executed by each processor to find the h function and all the nodes find the same function h.

The following results were presented in [16]. Given a node v_j of a Q_n, suppose Z_j denote the set of bit positions where v_j has 1. An edge $(v_j, v_j \oplus e_i)$ in dimension i, such that $i \notin Z_j$, appears in the canonical HC $P(\pi)$ given by the permutation $\pi = x_1 x_2 ... x_n$ iff

Case I : $x_1 = i$, or

Case II: $x_1 \in Z_j$ and $x_2 = i$, or

Case III: if $x_k = i$, $2 < k < n$, then one element of Z_j must appear as x_{k-1} and all the other elements of Z_j must appear after x_k in π, or

Case IV: if $x_n = i$, then either $Z_j = \phi$ or $Z_j = \{x_{n-1}\}$.

Finding an HC in a faulty Q_n with at most $n-2$ faulty links is essentially to find a node v and a permutation π on Z such that no faulty edge appears in $P(v, \pi)$, that is, for every faulty edge $v_\alpha \_\_ v_\beta$ the edge $v_\alpha \oplus v \_\_ v_\beta \oplus v$ does not satisfy any the above cases[3]. The following algorithm A, determines a node v and a permutation π such that the HC $P(v, \pi)$ does not have any of the $n-2$ faulty links.

Algorithm A:
Given: A set of faulty edges F with $|F| = t \leq n-2$ in a Q_n
To find: A node v and a permutation π on Z such that no edge of F appears in $P(v, \pi)$
Method:
Let Z_g , Z_b respectively be the set of faultfree and faulty dimensions. Let $Z_g = \{z_1, z_2, \ldots, z_g\}$ and $Z_b = \{y_1, y_2, \ldots, y_b\}$. Hence $b \leq t$ and $g + b = n$. We will assume that Z_g and Z_b are in sorted form, sorted in ascending order, so that all the processors find the same π.

[3] recall that $v_j \_\_ v_k$ appears in $P(v_i, \pi)$ iff the edge $v_j \oplus v_i \_\_ v_k \oplus v_i$ appears in $P(\pi)$

Case 1: $b = t$. Thus all the faulty edges are in distinct dimensions. Assume that $v_i \xrightarrow{y_i} v_i \oplus e_{y_i}$ is the edge in dimension $y_i, 1 \le i \le b$.

1. Set $\pi \leftarrow z_1 z_2 \ldots z_{g-1} y_1 y_2 \ldots y_b z_g$
2. Form the vector representation of the node v as follows.
   (a) Set $v \leftarrow$ the zero vector.
   (b) for $j := 2$ to b do
   if y_{j-1}th bit of v_j is 1 then $v \leftarrow v \oplus e_{y_{j-1}}$.
   (c) if z_{g-1}th bit of v_1 is 1 then $v \leftarrow v \oplus e_{z_{g-1}}$.

Case 2: $b < t$. Thus all the faulty edges are not in distinct dimensions.

1. Choose b faulty edges such that these edges are in distinct dimensions[4].
2. Set $\pi \leftarrow z_1 z_2 \ldots z_{g-1} y_1 y_2 \ldots y_b z_g$
3. Find v as in case 1 using edges chosen in the step 1.
4. Assume that of the remaining $t - b$ edges, the ith edge[5] is in dimension $\alpha_i \in Z_b$ and is $w_i \xrightarrow{\alpha_i} w_i \oplus e_{\alpha_i}$.
   for $j := 1$ to $t - b$ do
   if z_jth bit of w_j is 0 then $v \leftarrow v \oplus e_{z_j}$.

Given a set of at most $n - 1$ faulty links F, finding an HP that does not use any edge from F is straight forward. Choose an edge e from F and set $F' = F - \{e\}$. (To insure that all processors choose the same edge e, choose among the edges of smallest dimension. If there are several edges, choose the one having the smallest smaller endpoint.) Find an HC using the algorithm A so that no edge of F' appears in the HC. If the edge e appears in the HC, delete the edge from the HC and we have an HP. Call the smaller endpoint of e the starting node of the HP. If the edge e does not appear in the HC, choose the node represented by the zero vector. Remove one edge of the HC incident on this node - we have an HP and call the node as the starting node of the HP. We define the renumbering function h as follows: for any node v in the faulty Q_n, let $h(v)$ be the distance of v from the starting node of the HP along the HP. It is important to note that every processor node needs to know only its own label and the labels of the nodes having a faultfree

[4] To insure that all processors choose the same set of b faulty edges, among all the faulty edges in same dimension, choose the edge that has the smaller endpoint smallest among all the smaller endpoints

[5] To insure that for any i, all processors choose the same edge as the ith edge, have the remaining $t - b$ edges sorted in ascending order of dimensions and among the edges in the same dimension, arrange the edges in ascending order of smaller endpoints

link with it. Also, to compute these labels, a node does not have to compute the entire HC. Since we have considered $P(v, \pi)$ only to construct the HC in presence of $n-2$ link faults, each node can construct its label or the label of any node in that HC using v and π. If the $(n-1)$th edge appears in the HC, then each label is to be corrected by the label of one endpoint of that edge.

Once a node v_j knows its own label, that is, $h(v_j)$ and also the labels of its neighbors with whom it has a fault-free link, each node will compute the *higher neighbor* set and the *lower neighbor* set as follows. For each node v_j, *higher neighbor* set $HN(v_j)$ is defined as $HN(v_j) = \{v_k \mid v_k$ has a fault-free link with v_j and $h(v_k) > h(v_j)\}$. Similarly, for each node v_j, its *lower neighbor* set $LN(v_j)$ is defined as $LN(v_j) = \{v_k \mid v_k$ has a fault-free link with v_j and $h(v_k) < h(v_j)\}$. We will assume that $LN(v_j)$ and $HN(v_j)$ are sorted in descending and ascending order respectively. In the following, we prove the correctness of the algorithm A.

Proof of correctness of Algorithm A:

Consider the cases I through IV as listed above. For the choice of π, none of the faulty dimensions appears as the first or the last dimension of π and hence neither case I nor case IV applies. Case II and Case III imply that the edge $v_j \overset{i}{\text{---}} v_j \oplus e_i$ appears in $P(\pi)$ iff assuming $\pi = x_1 x_2 \ldots x_n$, when $x_k = i$, $k \neq 1$, $k \neq n$,

- x_{k-1}th bit of v_j is 1 and
- x_1th, x_2th, $\ldots$,x_{k-2}th bits of v_j are all zeroes (applicable for $k > 2$ only).

Consider the case 1 of the algorithm A. With the chosen π, if possible let us assume that $v_j \overset{y_j}{\text{---}} v_j \oplus e_{y_j}$ appears in $P(v, \pi)$. Then, $v_j \oplus v \overset{y_j}{\text{---}} v_j \oplus v \oplus e_{y_j}$ appears in $P(\pi)$. By construction, the y_{j-1}th bit of v is same as the y_{j-1}th bit of v_j. Hence the y_{j-1}th bit of $v_j \oplus v$ is 0 and so is the y_{j-1}th bit of $v_j \oplus v \oplus e_{y_j}$. But if the y_{j-1}th bit of $v_j \oplus v$ is 0 then this edge cannot appear in $P(\pi)$ by Case II and Case III. Hence we are done by contradiction.

Consider the case 2 of the algorithm A. With the chosen π, if possible let us assume (for step 4) that $\omega_j \overset{\alpha_j}{\text{---}} \omega_j \oplus e_{\alpha_j}$ appears in $P(v, \pi)$. Then, $\omega_j \oplus v \overset{\alpha_j}{\text{---}} \omega_j \oplus v \oplus e_{\alpha_j}$ appears in $P(v, \pi)$. $v_j \oplus v \overset{y_j}{\text{---}} v_j \oplus v \oplus e_{y_j}$

appears in $P(\pi)$. By construction, the z_jth bit of ω_j and v are opposite. Hence the z_jth bit of $\omega_j \oplus v$ is 1 and so is the z_jth bit of $\omega_j \oplus v \oplus e_{\alpha_j}$. But if the z_jth bit of $\omega_j \oplus v$ is 1 then this edge cannot appear in $P(\pi)$ by Case III. Hence we are done by contradiction. ■

B. Multicast Routing

Before we present the multicast routing strategy, we consider the problem of how the nodes find the identity of all the faulty links in a distributed fashion. Each node knows the fault status of the links incident on itself. Each node passes the fault status of the links incident on it to its neighbor connected by a faultfree link and each processor accepts such fault information (clearly, no information is attempted to be sent through a faulty link). It is simple to see that using such a "believe your neighbor connected by a good link" algorithm will insure that each node has the correct information about all faulty links. This is because in a Q_n, for every pair of nodes x and y, there exists a path from x to y such that all links nodes in the path is fault-free[6]

In this subsection, we present the multicast routing algorithm once an HP has been determined using the algorithm A. It should be noted that each node of the Q_n needs to compute the v and the π as given by the algorithm. This enables each node to determine the renumbering of itself and each of its neighbors. As mentioned earlier, $h(\alpha)$ will denote the renumbering or label of each node α when the v and π has been determined. However, none of the nodes need to know the complete HP that is used to do the routing. The multicast routing is divided into two phases: phase A and phase B. The phase A is the message preparation phase and starting up the communication and hence is executed only by the source of the message, whereas the phase B is message forwarding phase that is executed by each intermediate forwarding router nodes. The router in each node, during the phase B, will save the message if the node is one of the destination nodes.

[6] since a Q_n has edge connectivity n [13] and the total number of faulty links is at most $n-1$, of the n edge disjoint paths from x to y, at least one will not have any faulty link.

Phase A: Message Preparation and Startup Phase (executed by source node)

Step1. Let v_s be the source node and D is the set of destination nodes. Partition the set of destination nodes into two destination lists, HD and LD, $HD = \{v_j \mid v_j \in D \text{ and } h(v_j) > h(v_s)\}$ and $LD = \{v_j \mid v_j \in D \text{ and } h(v_j) < h(v_s)\}$. The message will be routed to the nodes in HD using *high* channels and to the nodes in LD using *low* channels. Sort the HD list in ascending order of the labels and the LD list in descending order.

Step2. Let $p = |LN(v_s)|$ and $q = |HN(v_s)|$. One of p or q is zero iff s is a starting point of the HP determined by algorithm A. Let $LN(v_s) = \{v_{l_1}, v_{l_2}, \ldots, v_{l_p}\}$ and $HN(v_s) = \{v_{h_1}, v_{h_2}, \ldots, v_{h_q}\}$ (Recall that the set LN and HN are sorted in descending and ascending order of their labels respectively). Partition the list LD into p sublists LD_i, $1 \leq i \leq p$, such that $LD_i = \{v_\alpha \mid v_\alpha \in LD \text{ and } h(v_{l_i}) \geq h(v_\alpha) > h(v_{l_{i+1}})\}$ for $1 \leq i \leq p-1$; LD_p contains all destination nodes in LD with labels at most $h(v_{l_p})$.

Step3. Partition the list HD into q sublists HD_i, $1 \leq i \leq q$, such that $HD_i = \{v_\alpha \mid v_\alpha \in HD \text{ and } h(v_{h_i}) \leq h(v_\alpha) < h(v_{h_{i+1}})\}$ for $1 \leq i \leq q-1$; HD_q contains all destination nodes in HD with labels at least $h(v_{h_q})$.

Step4. Send a copy of the message data and the sublist LD_i to the neighbor v_{l_i} along the *low* channel for $1 \leq i \leq p$.

Step5. Send a copy of the message data and the sublist HD_i to the neighbor v_{h_i} along the *high* channel for $1 \leq i \leq q$.

Phase B: Message forwarding or receiving by node v; x is head of destination list (executed by intermediate router nodes)

Step1. If $x = v$ then, (1) buffer the message, (2) remove x from the destination list. If current destination list is not empty then set x to be the head of the current destination list and go to Step2 else done.

Step2. Let $LN(v) = \{v_{l_1}, v_{l_2}, \ldots, v_{l_p}\}$ and $HN(v) = \{v_{h_1}, v_{h_2}, \ldots, v_{h_q}\}$.
If the received message arrived via a *low* channel

then find the smallest j such that $h(v_{l_j}) > h(x)$. Forward the message data and the destination list along the low channel to v_{l_j}

else find the largest j such that $h(v_{h_j}) < h(x)$. Forward the message data and the destination list along the high channel to v_{h_j}

From the routing algorithm presented above, it is simple to observe that once a message is sent out on a low (high) channel, it gets forwarded always on the low (high) channel. From the definition of low (or high) channels, it is simple to note that freedom from deadlock is guaranteed. We used the similar principle of using multiple paths (as was used in [8]) originating from the source and reaching all the destinations. However, there are several points of significant differences worth noting.

- We divided the Hamiltonian path into several segments (depending on the number of neighbors of the source node connected by a fault-free link). Message forwarding goes on in parallel in each segment and the path used in segment is a routing path from the source. Each path is constructed based on lists of destinations. In the presented algorithm, whether two destinations will appear in the same list depends only on the source node and the set of faulty links (and not on the other destinations), whereas in [8], it does depend on the other destinations.

- If u and v are two consecutive destinations in the same list, the routing algorithm of [8], assumes that the message will be delivered to v after it has reached u. Since the message will be delivered to v traversing minimum number of edges along a hamiltonian circuit (or skipping some of the edges), while creating the list, it is important that v is put in a list such that its hamming distance to the previous node is minimum. This works fine in a fault-free Q_n. How to modify the algorithm of [8] to apply in the case of faulty Q_n is not very clear. Using the idea of finding the shortest path does not work because some of the short-cut providing links might not be usable. On the other hand, the algorithm presented here can be trivially applied to a fault-free Q_n by assuming an empty fault set. Finding the Hamiltonian path has two utilities: it provides an ordering of the channels; also it guarantees that a message will move, in the worst case, along the HP to the destinations.

- In the presented algorithm, the effect of faulty links on performance degradation is expected to be low. Because of the faulty links, some of the short-cuts might not usable. In our algorithm, we use a greedy approach of trying to reach the destination as close as possible in each short-cut. If the best such short-cut is not available because of fault, the next best is used and so on. Naturally in a random situation, it is unlikely that the best short-cuts are always faulty averaging out the contribution of extra channels used.

IV. Simulation Results

The multicast routing algorithm presented above was simulated for four different size hypercubes. To study the routing performance in contrast to a faultfree hypercube, the algorithm of [8] was also implemented.

We considered Q_n for $n = 4, 6, 8, 10$ and each case with the number of destination nodes varying from as low as 3 to near broadcast case. The simulation was carried out by selecting the source and the destinations in uniformly random way. The faulty edges were also chosen in a uniformly random way. For each such choice, the algorithm A was applied to construct the labeling of the nodes and then the multicast routing was constructed using the algorithm presented above. Considering each choice of source and multicast size with corresponding destination nodes together with the fault set as one instance of multicasting, 10,000 instances were chosen in uniformly random way. The total number of links traversed to deliver message to all the destinations was considered as the cost of the multicasting and was averaged over 10,000 instances. To multicast to p destination nodes, obviously, at least p edges will be traversed. Since we had the multicast planned along the edges of a Hamiltonian path, in the worst case, a broadcast will use exactly the same number of links as the number of nodes. The figures 1 through 4 summarizes the simulation results. The number of additional channels represent the average number of links traversed minus the number of nodes in the multicast set.

The plot for the faulty case assume $n-1$ faults and thus, gives the maximum performance degradation because of faults. As seen from the plots, both the faultfree and faulty cases use the largest number of additional channels when the size of the multicast set is about 40% of the network size. The largest increase in additional channels used seem to be quite small. For example, for a Q_{10} the largest increase was about 40 channels when there were about 400 destination nodes. For a Q_4, on the other hand, was a mere 0.8 for 7 destination nodes. Typically, in presence of faults, the largest increase in the average number of additional channel usage was about 10% of the number of destination nodes the message is being delivered to. Clearly, we can infer that our multicast algorithm works fairly well in presence of faulty links.

References

[1] Chen. M. S. and Shin. K. G., "Processor allocation in an N-cube multiprocessor using gray codes", *IEEE Trans. Computers.* vol. C-36, pp. 1396-1407, 1987.

[2] Dally. W. J. and Seitz. C. C., "Deadlock-free message routing in multiprocessor interconnection networks", *IEEE Trans. Computers.* vol. C-36, pp. 547-553, May, 1987.

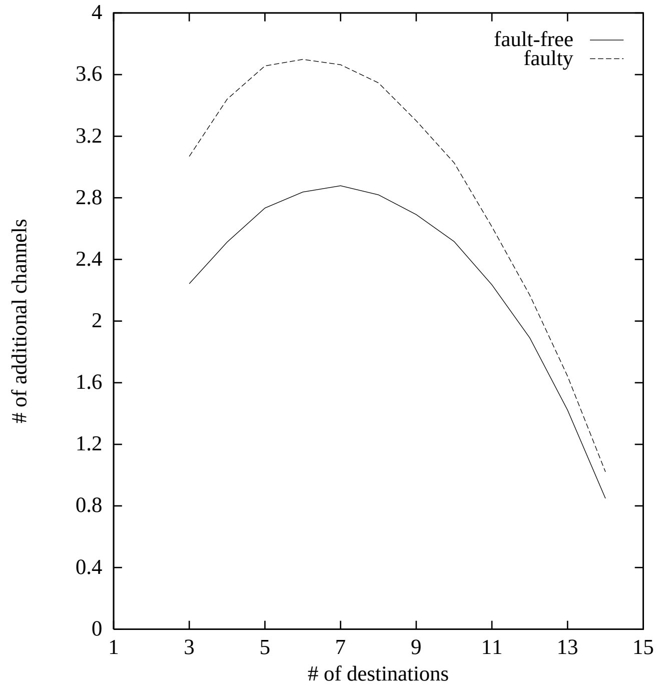

Figure 1: Routing performance (n=4)

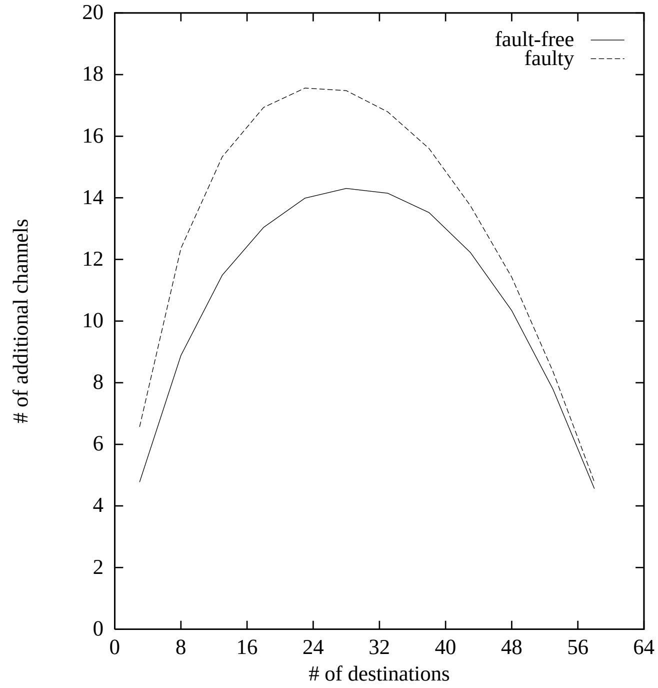

Figure 2: Routing performance (n=6)

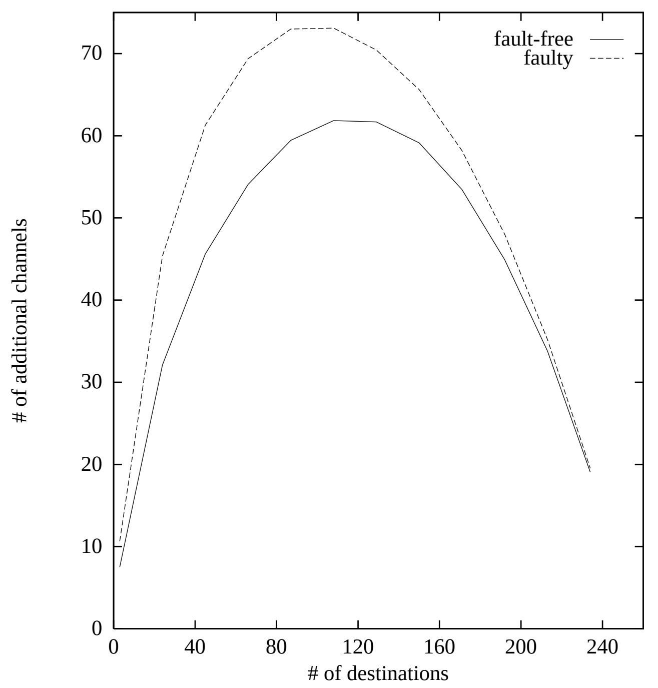

Figure 3: Routing performance (n=8)

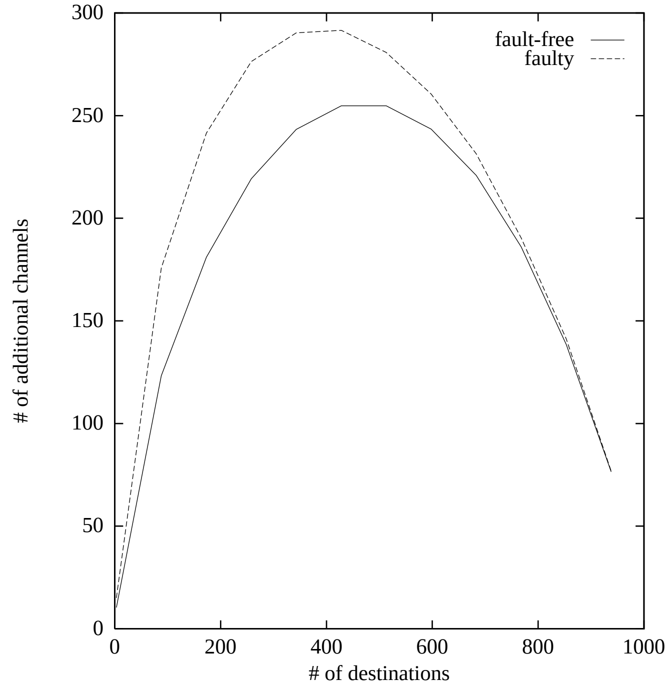

Figure 4: Routing performance (n=10)

[3] Gelernter. D., “A DAG-based algorithm for prevention of store-and-forward deadlock in packet networks”, *IEEE Trans. Computers.* vol. C-30, pp. 709-715, Oct, 1981.

[4] Gunther,K. D. ”Prevention of deadlocks in packet switched data transport systems”, IEEE Trans. Communications, vol. COM-29, pp. 512-524, April, 1981.

[5] Harary. F., Graph Theory, Addison-Wesley, 1969.

[6] Intel Corp., *Paragon Overview*, 1993.

[7] Kermani. P. and Kleinrock. L., “Virtual cut-through: A new computer communication switching technique”, *Comput. Networks*, vol. 3, pp. 267-286, 1979.

[8] Lin. Xiaola. and Ni. L. M., “Deadlock-free multicast wormhole routing in multicomputer networks”, *Proc. 18th Annual Symposium on Computer Architecture* , 1991.

[9] P.M.Merlin and P.J.Schweitzer, ”Deadlock avoidance in store-and-forward networks - I : Store-and-forward deadlock”, IEEE Trans. Communications, vol. COM-28, pp. 345-354, March, 1980.

[10] Ni. L. M. and McKinley. P.K., “A survey of wormhole routing techniques in direct networks”, *IEEE Comput.,* vol. 26, pp. 62-76, Feb., 1993.

[11] NCUBE Company, *NCUBE 6400 Processor Manual*, 1990.

[12] Pifarre. G.D., Gravano. L., Denicolay. G. and Danz. L.C., “Adaptive deadlock- and livelock-free routing in the hypercube network”, *IEEE Trans. Parallel Distrib. Syst.*, vol. 5, No. 11, pp. 1121-1138, Nov, 1994.

[13] Saad. Y. and Schultz. M.H., “Topological properties of hypercubes”, *IEEE Trans. Computers.* vol. C-37, pp. 867-872, July, 1988.

[14] Seitz. C. L., Athas. W. C., Flaig. C. M., Martin. A. J., Seizovic. J., Steels. C. S. and Su. W.-K., “The architecture and programming of the Ametek Series 2010 multicomputer”, *Proc. Third Conf. Hypercube Concurrent Comput. Applicat.*, vol. I, pp. 33-36, Pasedena, CA, ACM, Jan. 1988.

[15] Sengupta. A. and Bandyopadhyay. S., “Deadlock free routing in k-ary hypercube network in presence of processor failures”, *Information Processing Letters*, vol. 34, no. 6, pp. 323-328, May, 1990.

[16] Sen. A., Sengupta. A. and Bandyopadhyay. S., “On some topological properties of hypercube, incomplete hypercube and supercube”, *Proc. International Parallel Processing Symposium*, pp. 636-642, Newport Beach, April, 1993.

[17] Tannenbaum. A.S., Computer Networks, Prentice Hall, Englewood Cliffs, 1981.

[18] S.Toueg and J.D.Ullman, ”Deadlock-free packet switching networks”, Proc. 11th ACM Symp. Theory of Computing, p p. 89-98, 1979.

[19] Tseng. Y-C. and Panda. D. K., “A trip based multicasting model for wormhole routed networks with virtual channels”, *Proc. 7th International Parallel Processing Symposium,* pp. 276-283, April, 1993.

[20] Viswanathan. S. and Sengupta. A. “Deadlock-free Fault-tolerant Multicast Routing in Hypercubes”, Submitted to IEEE Trans. Parallel and Distributed Systems.