Adaptive routing in $k$-ary $n$-cubes using incomplete diagnostic information

C.P. Ravikumar$^a$, C.S. Panda$^b$,

$^a$Department of Electrical Engineering, Indian Institute of Technology, Hauz Khas, New Delhi 110016, India
$^b$Global Telecom Services, C-47, South Extension, New Delhi, 110049, India

Received 8 March 1996; revised 25 July 1996; accepted 16 September 1996

Abstract

In this paper, we present a fault-tolerant routing algorithm for $k$-ary $n$-cube interconnection networks which have become increasingly popular for the construction of massively parallel computers. The $k$-ary $n$-cube is a generalization of 2-ary hypercube network, and can model several interesting topologies such as the 2-d torus, 3-d torus, and the binary $n$-cube. In our routing algorithm, we assume that each node has static knowledge of the fault status of its immediate neighbors. Using this information, the nodes of the network execute a distributed diagnosis procedure whereby each node learns the fault status of all other nodes that are reachable from it within $k$ hops, $k > 1$. We refer to $k$ as the diagnostic radius. Our simulation results indicate that a diagnostic radius larger than 1 can improve the performance of the routing algorithm. Our routing algorithm is a significant improvement over a similar algorithm due to Blough and Najand in that our algorithm does not place overheads on each message. The Blough–Najand algorithm requires each message to store the entire path from the source to destination, which can be quite large for a massively parallel multiprocessor. We compare the relative merits and demerits of our algorithm with those in the literature.

Keywords: Fault-tolerant routing; Interconnection networks; Fault Diagnosis; Massively Parallel Computers

1. Introduction

Applications such as weather forecasting, computational fluid dynamics, seismic data processing, scientific visualization, and medical imaging need instructional execution speed of the order of tera flops. It is now accepted that such high computing performance can only be achieved through the use of massively parallel supercomputers [1]. Presently, a number of commercially available massively parallel processors are constructed as distributed memory multiprocessor machines, e.g. Intel Paragon, Cray T3D, and NCUBE/10 (see [1]). The main challenge in building massively parallel multiprocessors is the construction of a suitable interprocessor interconnection network (ICN); by and large, it is the ICN which influences the cost as well as the performance of the multiprocessor. With some notable exceptions like the IBM RP3 [1], most massively parallel processors today employ direct networks, where direct communication links are used to carry data between two processors.

Multistage interconnection networks, which employ several stages of switches to realize permutations of data amongst processors, are still too expensive for massively parallel multiprocessors. In this paper, we focus only on direct networks. It is desirable that the ICN provide small communication diameter, small average latency for point-to-point communication, support efficient multicasting and broadcasting. At the same time, the topology of the ICN must be regular and require a small number of communication ports per node. The interconnection strategy is important because it determines the average latency of a message transmitted from a source to a destination.

Some of the interconnection networks which have been used in commercially available parallel processors include the linear array (Intel i-WARP), two-dimensional mesh (Goodyear MPP), hypercubes (Intel iPSC and NCUBE/10), and two-dimensional torus (Intel Paragon). Daily has proposed a generalization of meshes and hypercubes, which he termed $k$-ary $n$-cubes [2]. A $k$-ary $n$-cube is a direct interconnection network defined on $k^n$-nodes. The nodes in a $k$-ary $n$-cube are identified using $n$ digit radix-$k$ addresses $a_0, a_1, \ldots, a_{n-1}$. The $i$-th digit of the address, $a_i$, represents
The node's position in the $i$-th dimension, $0 \leq i \leq n - 1$. A node $a_0, a_1, \ldots, a_{n-1}$ is connected to another node $b_0, b_1, \ldots, b_{n-1}$ in the $k$-ary $n$-cube if and only if $b_i = (a_i \pm 1) \mod k$ for some dimension $i$, and $a_i = b_i$ for all $i \neq i$. The salient properties of the $k$-ary $n$-cube are summarized in Table 1. It is easy to see that an $n$-dimensional hypercube is a 2-ary $n$-cube, and an $n \times n$ two-dimensional torus in an $n$-ary 2-cube. $k$-ary $n$-cubes with small $n$ are preferred due to ease of physical design.

1.1. Fault tolerant routing algorithms

$k$-ary $n$-cubes have a simple deadlock-free routing algorithm (called cube algorithm in the literature) for point-to-point shortest-path routing [2,3]. The cube algorithm examines the source and destination labels from left to right and selects the leftmost dimension $i$ such that $S_i \neq D_i$, and forwards the packet to a neighbour of $S$ in dimension $i$. As an example, consider routing from a node 2320 to node 2123 in a 4-ary 4-cube. The cube algorithm will select dimension 1 and forward the packet from source 2320 to its neighbour 1 = 2020 along dimension 1. The complete path of the packet from $S$ to $D$ is 2320 $\rightarrow$ 2020 $\rightarrow$ 2120 $\rightarrow$ 2123. In case of $n$-ary 2-cubes, the cube routing algorithm is popularly known as XY routing, since the packet is first routed along the $X$ dimension until $S_0 = D_0$ and then routed along the $Y$ dimension until $S_1 = D_1$. While the cube algorithm is simple to implement and is deadlock free, it suffers from non-adaptivity to network conditions. If an intermediate node along the path from $S$ to $D$ is faulty, the cube algorithm fails to route the packet. Similarly, if a link along the path is faulty or congested, the routing algorithm cannot use a quicker alternate path to route the packet.

The $k$-ary $n$-cube exhibits good node fault-tolerance (see Table 1). Further, there are $N_p$ shortest paths between a source $S$ and destination $D$ separated by distance $d$ (Eqs. (1) and (2)). For example, consider a 4-ary 2-cube and let $S = 02$ and $D = 13$. A shortest path from $S$ to $D$ is of length 2 and there are two such paths. These are $(02 \rightarrow 12 \rightarrow 13), (02 \rightarrow 03 \rightarrow 13)$. The routing algorithm in a $k$-ary $n$-cube must be tolerant to node and link faults if we are to exploit the fault-tolerant properties of the topology. Fault-tolerance is of paramount importance in massively parallel multiprocessors where the reliability of the entire system may be expected to be small due to the large number of components.

\[ N_p = \left( \delta_0, \delta_1, \ldots, \delta_{n-1} \right)^d \]

\[ d = \text{dist}(S, D) = \sum_{i=0}^{n-1} \delta_i \]

\[ \delta_i = \min(|S_i - D_i|, k - |S_i - D_i|) \]

1.2. Partial diagnosis

A routing algorithm requires to know the fault status of its neighbour nodes in order to be able to choose an alternate path for a packet. Most fault-tolerant routing algorithms proposed hitherto [4,5] assume that every node in the multiprocessor has access to diagnostic information about its immediate neighbours. This diagnostic information may be static or dynamic. With localized diagnostic information, it is possible that incorrect routing decisions are taken at an intermediate node 1, causing the packet to take a route which is longer than necessary. For example, consider a three-dimensional hypercube in which nodes 101 and 011 are faulty. If a packet originating at $S = 000$ is to be routed to $D = 111$, the cube algorithm will forward the packet to node 001. Node 001 cannot forward the packet to 101 or 011 since node 001 knows that these two nodes are faulty. The packet must therefore retrace to node 000 and an alternate path (such as $000 \rightarrow 010 \rightarrow 110 \rightarrow 111$) must be chosen to route the packet. This situation could be improved if each node has access to more diagnostic information e.g. the status of both immediate and second-level neighbours. In the above example, node 000 will not forward the packet to node 001, hence avoiding the retraction.

Distributed fault diagnosis algorithms are necessary to gather diagnostic information in a multiprocessor. Storing complete diagnostic information about all the nodes at each node of the multiprocessor can be expensive; it is also expensive in terms of CPU-time to run distributed diagnosis algorithms. Blough and Najand in [6] suggested algorithms for partial diagnosis, where each non-faulty node in the system has status information about some of the neighbouring nodes. In particular, the authors introduced the notion of $f$-reachability and $f$-neighbourhood of a node in a multiprocessor. They presented a fault-tolerant routing algorithm for the hypercube interconnection network. In this paper, we extend their work by presenting a fault-tolerant algorithm for the $k$-ary $n$-cube topology using partial
diagnostic information. Though stated for a \(k\)-ary \(n\)-cube, our algorithm is applicable for arbitrary topologies, and has several advantages over the Blough-Najand algorithm (BNA). The technique used in BNA for point-to-point routing is essentially a best-first-search starting from the source \(S\) and terminating when destination \(D\) has been visited. In a graph-search strategy, to avoid a message from looping back and forth among a subset of nodes, it is important to remember the list of nodes that a message has visited. Such a list of nodes can be very long, comparable to the total number of nodes, in a faulty multiprocessor. In a \(k\)-ary \(n\)-cube, \(n \log_2 k\) bits are required to represent the address of each node. Thus, in a 2-ary 10-cube, the message overhead can be as high as 10 KBits. We entirely avoid this overhead through the use of routing tables and randomized routing (Section 3). Unlike BNA, our algorithm is suitable both for node and link faults. BNA employs backtracking when an optimal route cannot be found to route a message. In our algorithm, we do not employ backtracking since we do not store the history of the message route along with the message; instead, we use a deflection-based routing mechanism, where a non-optimal route is chosen in the event where no optimal routes can be found. The demerit of our algorithm in comparison to BNA is the use of routing tables at each node. However, we believe that this overhead is justified by the saving in routing time which results from the absence of message overheads. We also show that storing routing tables at each node is not an impractical proposition using current-day memory technology.

1.3. Organization of paper

Section 2 reviews recently published literature related to adaptive routing in multiprocessors, with emphasis on partial diagnostic methods. Section 3 develops an adaptive routing algorithm for the \(k\)-ary and \(n\)-cube interconnection network. We have simulated the proposed adaptive routing algorithm, and Section 4 presents the results of simulation. Conclusions are presented in Section 5.

2. Fault-tolerant routing in multiprocessors

As mentioned in the previous section, occurrence of faults in a massively parallel multiprocessor is an event of high likelihood. Two types of faults, namely node faults and link faults, are assumed in this paper. It is assumed that a non-faulty node can diagnose its neighbour nodes and links for possible faults. A message originating at source \(S\) can be routed to its destination \(D\) as long as the two nodes are connected. The connection probability is high for \(k\)-ary \(n\)-cubes when \(n\) is large. Even for small values of \(n\), for a tolerable number of faults, it is likely that \(S\) and \(D\) are connected through a path in a faulty cube. We define the node fault tolerance of an interconnection network as the maximum number of node faults that the interconnection network can tolerate before the network becomes disconnected. The number of faults in the network is said to be tolerable when this number is smaller than the node fault tolerance of the network topology. The node fault-tolerance of a \(k\)-ary \(n\)-cube is \(2n - 1\) for \(k > 2\), and \(n - 1\) for \(k = 2\).

A routing algorithm must be able to exploit the fault-tolerance of the underlying interconnection network by adapting to fault situations and finding alternate paths for messages when faults arise. Several algorithms have been suggested for fault-tolerant routing in multiprocessors [4, 6–10]. Most of these algorithms are intended for the hypercube topology. The objective of a fault-tolerant routing algorithm is to ensure that the routing is successful in presence of faults, in addition, the algorithm attempts to reduce message latency by giving priority to shorter paths over longer ones. To be able to give any form of guarantee on the message latency, or the success of routing, or on freedom from livelock (a condition in which a message hops back and forth amongst a subset of nodes without reaching the destination) the routing scheme must resort to a systematic graph search such as breadth-first-search or depth-first-search. If routing decisions at a node \(i\) are made on knowledge of only the local neighbourhood of \(i\), then performance guarantee can only be given by restricting the number of faults (e.g. in [7], the number of faults in an \(n\)-dimensional hypercube cannot be more than \(n\)). Similarly, freedom from livelock can be achieved by storing the history of routing decisions along with the message itself. For instance, we can employ a smart form of depth first technique, where a neighbour node \(I\) which is on an optimal path from a source \(S\) to a destination node \(D\) is visited with higher probability than a neighbour node \(J\) which is on a non-optimal path from node \(S\) to \(D\) (see [4] and [6]). In such a scheme, the list of nodes visited by a message \(M\) must be stored along with the message itself to avoid livelock. As we have seen in Section 1, in massively parallel computers, such a message overhead is excessively large and hence schemes proposed in [4] and [6] are impractical.

On the other hand, it is also possible to devise routing schemes where the routing path is determined before a message leaves the source. For such a routing technique, knowledge of the fault status of the immediate neighbourhood is insufficient. In the presence of node or link faults, the regular graph topology of the interconnection network gets disrupted. If each fault-free node has the complete information about the network topology, then a shortest-path algorithm such as Dijkstra's well known algorithm can be employed to precompute the routing path for each message. The disadvantage of such a routing scheme, of course, will be the need for an expensive
procedure f_reachability(f)
// Each node determines the status of neighbours which are at distance f or less from that node
begin
    All non-faulty nodes test their neighbours and record their status.
    for i := 1 to f - 1
        begin
            send all current diagnostic information to the immediate neighbours.
            receive diagnostic information from immediate non-faulty neighbours.
        end;
end;

Fig. 1. Neighbourhood computation.

distributed diagnosis algorithm to gather fault status information. The routing algorithm will also be time consuming.

In this paper, our focus is on an intermediate solution, where each node i has knowledge of not only its immediate neighbourhood, but of all nodes that are reachable from i using f or fewer hops. When f = 1, such a routing technique degenerates to routing with local information; when f is equal to the communication diameter of the interconnection network, the technique resembles routing with global knowledge. Two other authors have presented routing algorithms with partial diagnostic information. In [8], the authors suggested fault-tolerant routing heuristics for hypercubes subjected to link failures. A node failure is modelled by associating faulty status to all the links of the injured node. In algorithms lookaside and lookahead, each node has the status of links which are within distance 2 from the node. However, their algorithm is specialized to hypercubes and cannot be generalized to k-ary n-cubes. Blough and Najand [6] suggested fault-tolerant routing algorithms for arbitrary interconnection networks in the presence of an arbitrary number of faults. They introduced two generalized types of partial diagnosis, known as f-neighbourhood and f-reachability diagnoses, which are discussed next.

In f-neighbourhood diagnosis, each non-faulty node i determines the status of every node which is at distance f or less from i. If a system becomes disconnected due to faults, then it is not always possible to achieve f-neighbourhood diagnosis for f greater than 1. Hence it is more practical to use f-reachability diagnosis. We define a node j to be f-reachable from a node i if there exists a non-faulty path of length i < f from i to a neighbour of j. In f-reachability diagnosis, each non-faulty node determines the status of every node which is f-reachable from itself. The f-reachability diagnosis is less strict than f-neighbourhood diagnosis as there may be some nodes in the f-neighbourhood of a node that are not f-reachable. The status of these unreachable nodes is not determined in an f-reachability diagnosis. Note that f-reachability diagnosis gathers maximum amount of diagnostic information when f = (N - 1), where N is the total number of nodes. At the same time, it may be noted that (N - 1)-reachability diagnosis is not the same as complete diagnosis. For if some nodes are not (N - 1)-reachable from a nonfaulty node, then they are not f-reachable for any f, and the non-faulty node cannot diagnose them. The intermediate range of f-reachability diagnosis is 1 < f < N, where N is the number of nodes in the system. The procedure for computing the f-reachability in a multiprocessor is shown in Fig. 1.

2.1. f-reachability vs f-neighbourhood diagnosis

The f-reachability diagnosis algorithm can be modified to achieve f-neighbourhood diagnosis by continuing the process of testing and exchanging information until each nonfaulty node receives the appropriate information. In case of f-reachability diagnosis, the communication of diagnostic information terminates after f steps. On the other hand, f-neighbourhood diagnosis may require up to N steps before terminating, where N is the number of nodes. Fig. 2 shows an injured 5 x 8 mesh where nodes 13, 23, 32 and 42 are faulty and node 33 is considered for 2-reachability and 2-neighbourhood diagnosis. The 2-neighbourhood nodes of node 33 are (31, 32, 22, 42, 13, 23, 43, 44, 34, 35, 36, 24). Thus the information exchange continues until node 33 gets the information about all of its 2-neighbourhood nodes. As direct information flow is blocked due to the faulty nodes, it requires more iterations to gather all the information via node 03. On the other hand,
procedure AD_Fault_Route(S, D, Packet)
// Route Packet from S to D
begin
if (S = D) then reject Packet;
if FAULTY(D) // Use reachability information
  reject Packet;
else begin
  if routing table entry of S for D is not nil
    begin
    Select an optimal neighbour I randomly with probability p_i(S, D);
    Forward Packet to I;
    end;
  else begin
    Forward Packet to any non-faulty non-optimal neighbour;
    end;
end;

Fig. 3. Adaptive fault-tolerant routing algorithm.

2-reachability information is terminated after one iteration of information exchange between nodes; the nodes that are 2-reachable from S are (23, 32, 42, 43, 44, 34, 35, 24). Apart from its computational overhead, f-neighbourhood suffers from the disadvantage that it is not always computable. In fact, it was proved in [6] that, with high probability, complete diagnosis is not possible in constant degree systems and is only possible in logarithmic degree systems for low node failure probabilities. For these reasons, we will consider only f-reachability diagnosis.

3. Adaptive routing in k-ary n-cubes

Increasing use of highly concurrent computer systems in reliability-critical applications has made fault-tolerant communication algorithms for message passing architectures important. In this section, we discuss algorithms for adaptive routing in k-ary n-cubes using incomplete diagnostic information. It is assumed that the diagnosis algorithm is run at intervals of T, where T is the mean time to failure (MTTF) of a single node or link, whichever is smaller.

3.1. An adaptive routing algorithm

The well known e-cube algorithm can be used to route messages in a non-faulty k-ary n-cube [2]. Given a source S and destination D, the algorithm identifies an intermediate node I to which the packet must be forwarded by selecting j such that S[j] ≠ D[j] and S[i] = D[i] for all i < j. Then I is a neighbour of S, as defined in Eq. (3).

\[ I[j] \in \{(S[j] + 1) \mod k, (S[j] - 1 + k) \mod k\} \]  \hspace{1cm} (3)

One of the two neighbours of S given by Eq. (3) is closer to D than the other, and the closer neighbour is chosen as the best intermediate node.

Although there are multiple shortest paths between a pair of nodes S and D in a k-ary n-cube (see Section 1.1), the e-cube algorithm obliviously selects the same path for every packet originating at S and destined for D. Oblivious routing cannot deal with network conditions such as node faults, link faults, and congestion. Better performance can frequently be obtained by splitting the traffic over several paths to reduce the load on the communication links. Let \( \phi_f(S) \) denote the set of nodes f-reachable from the source S. The paths from S to D are not all of shortest length. We refer to the neighbours
procedure NonFaultyPaths (k,n,S,D)
    // Compute number of non-faulty optimal paths N_p from source S to the destination D in
    // presence of node faults using partial diagnostic information available with S . N_p is
    // number of paths, set initially to 0 outside the procedure.
    begin
        for j = 0 to n - 1 begin
            if (S[j] = D[j]) then continue;
            else begin
                Find optimal neighbour I in dimension j;
                if not Faulty(I) then begin
                    if (I = D) then
                        Increment N_p ;
                    else
                        NonFaultyPaths(k,n-1,D);
                end;
            end;
        end;
    end;

Fig. 5. Computing the number of non-faulty paths between a pair of nodes.

of $S$ which lead to shortest routes between $S$ and $D$ as
optimal nodes. In the presence of faults, forwarding the
packet to an optimal node $I$ may or may not lead to
the destination. We associate a probability $p_I$ of successful
routing with each optimal node $I$ for a given source-
destination pair $(S,D)$. This information is stored at
the source node in the form of a routing table. There
are $N$ entries in the routing table, at each node, where
$N = k^n$. Each entry corresponds to a possible destination
$D$ and is a list of tuples of the form $(I, p_I)$, where $I$ is an
optimal node w.r.t. $S$ and $D$ and $p_I$ is the associated
probability. The probability $p_I$ is computed using the
partial diagnostic information available with the source.
Let the number of non-faulty optimal paths between two
nodes $i, j$ be indicated by $\tau(i,j)$ (see Fig. 5). Then

$$P(D) = \frac{\tau(I, D)}{\sum_{J \in \Phi(S)} \tau(J, D)}$$

The optimal nodes in each routing table entry are
arranged in the descending order of routing probabilities.
When a node $S$ must forward a packet to reach a
destination $D$, it selects one of the optimal nodes randomly
from the set of optimal nodes available in the
routing table. If $X$ indicates a random variable which
denotes the selected node, then

$$P(X = I) = p_I(S,D)$$

A uniformly distributed random variable $Y$ may be
used to generate $X$, using standard statistical tech-
niques. When there is no optimal node for a given
$(S,D)$ pair due to network faults, the source chooses
a non-optimal non-faulty neighbour randomly and for-
wards the packet. This process of forwarding the packet
continues until the packet reaches its destination. The
$J$-reachability information allows a foresight into the
fault situation in the network and diverts the traffic
accordingly. The path taken by a packet to a destination
$D$ in our algorithm is independent of the choices of other
packets destined for $D$ in the past. The process of forming
the routing table is given in the next section. The com-
plete procedure for adaptive routing is shown in Fig. 3.

3.2. Routing table

The procedure MakeRoutingTable of Fig. 4 gives the
details of forming the routing table at any node $S$.
Fig. 5 shows the procedure NonFaultyPaths which is to

Table 2
Routing table for node 00 with 1-reachability diagnosis with node faults

<table>
<thead>
<tr>
<th>Destination address</th>
<th>Optimal nodes along with probabilities</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>NIL</td>
<td>Source = Destination</td>
</tr>
<tr>
<td>01</td>
<td>(01, 1.0)</td>
<td>Immediate Neighbour</td>
</tr>
<tr>
<td>02</td>
<td>(02, 1.0)</td>
<td>Immediate Neighbour</td>
</tr>
<tr>
<td>10</td>
<td>NIL</td>
<td>Node 10 is faulty</td>
</tr>
<tr>
<td>11</td>
<td>(01, 1.0)</td>
<td>Node 01 is the only non-faulty optimal node</td>
</tr>
<tr>
<td>12</td>
<td>(02, 1.0)</td>
<td>Node 02 is the only non-faulty node</td>
</tr>
<tr>
<td>20</td>
<td>(20, 1.0)</td>
<td>Immediate Neighbour</td>
</tr>
<tr>
<td>21</td>
<td>(20, 0.5) (01, 0.5)</td>
<td>Nodes 20 and 01 can both lead to destination 21</td>
</tr>
<tr>
<td>22</td>
<td>(20, 0.5) (02, 0.5)</td>
<td>Nodes 20 and 02 can both lead to destination 22</td>
</tr>
</tbody>
</table>
compute the number of nonfaulty paths from a source $S$ to a destination $D$. Eq. (4) is used in the construction of routing table for computing the probability $p_l(S, D)$ for a given source node $S$, an optimal node $l$ and a destination $D$. For example, consider a 3-ary 2-cube (Fig. 6), where node 10 and 22 are faulty. The routing table for node 00 with 1-reachability diagnostic information is shown in Table 2.

3.3. Cost of routing table

The routing table for a non-faulty node contains $N = k^n$ possible destination addresses; for each destination there are at most $(n/2)$ optimal node addresses. Each entry in the routing table needs $(n \cdot \log_2 k + \pi)$ bits where $\pi$ bits are meant for storing the probability information. The size of the routing table can be represented by Eq. (6), where $N$ is the total number of nodes in the network.

$$N \left\lceil \frac{n}{2} \left\lceil \log_2 N + \pi \right\rceil \right\rceil$$  \hspace{1cm} (6)

In a 2-ary 10-cube with 1024 nodes, each node requires at most 27 KBytes of memory to store the routing table; this is easily affordable by today’s standards.

3.4. Enhancement to $AD_{\text{Fault Route}}$

The algorithm $AD_{\text{Fault Route}}$ of Fig. 3 can be modified to adapt to both link as well as node faults with $\pi$-reachability diagnosis. The assumptions in this modification are as follows.

1. Any non-faulty node can test and determine the status of links incident on it.
2. If the link between two adjacent nodes is faulty, then neither of the nodes can directly determine the status of the neighbouring node.
3. The link between two faulty nodes can be declared unusable.
4. The links incident on a faulty node can be declared unusable for routing information, but can be used for diagnosis, if not faulty.

<table>
<thead>
<tr>
<th>Destination address</th>
<th>Optimal nodes along with probability</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>NIL</td>
<td>Source = Destination</td>
</tr>
<tr>
<td>01</td>
<td>NIL</td>
<td>Link 00-01 is faulty</td>
</tr>
<tr>
<td>02</td>
<td>(02, 1.0)</td>
<td>Immediate Neighbour</td>
</tr>
<tr>
<td>10</td>
<td>NIL</td>
<td>Node 10 is faulty</td>
</tr>
<tr>
<td>11</td>
<td>NIL</td>
<td>No optimal path</td>
</tr>
<tr>
<td>12</td>
<td>(02, 1.0)</td>
<td>02 is the only optimal non-faulty neighbour to 12</td>
</tr>
<tr>
<td>20</td>
<td>(20, 1.0)</td>
<td>Immediate Neighbour</td>
</tr>
<tr>
<td>21</td>
<td>(20, 1.0)</td>
<td>20 is the only optimal non-faulty neighbour to 21</td>
</tr>
<tr>
<td>22</td>
<td>(20, 0.5) (02, 0.5)</td>
<td>20 and 02 are both optimal nodes leading to 22</td>
</tr>
</tbody>
</table>
Table 4
Comparison of BNA and AD_Fault_Route

<table>
<thead>
<tr>
<th>Blough–Najand Algorithm</th>
<th>Our algorithm (AD_Fault_Route)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Uses best-first search approach for routing.</td>
<td>Uses routing tables at non-faulty nodes to forward the packet.</td>
</tr>
<tr>
<td>Message overhead is high, making the algorithm unsuitable for packet switching or wormhole routing.</td>
<td>The algorithm has no storage overheads per message and is well suited for packet switching and wormhole routing.</td>
</tr>
<tr>
<td>Intermediate node recomputes the path, if any node in the path is found to be faulty.</td>
<td>Forwarding node selects an optimal node randomly from the routing table entry for the given destination.</td>
</tr>
<tr>
<td>Backtracking is used in absence of any shortest path to the destination.</td>
<td>In absence of optimal path, the packet is forwarded to a non-faulty non-optimal neighbour.</td>
</tr>
<tr>
<td>Algorithm is applicable to arbitrary system topologies with arbitrary number of faults.</td>
<td>Suitable for arbitrary system topologies with arbitrary number of node faults and can be made adaptive to link faults.</td>
</tr>
<tr>
<td>Algorithm also applicable to dynamic fault environment.</td>
<td>To make it suitable for dynamic environment, the routing tables must be updated at every MTTP (mean time to failure of nodes and links).</td>
</tr>
<tr>
<td>Computational overhead is more since path for each message is to be calculated.</td>
<td>Memory overhead per node is more since routing table must be stored. However, the size of routing table is practical even for massively parallel processors.</td>
</tr>
</tbody>
</table>

Table 5
Performance of a non-faulty 2-ary 8-cube at different loads

<table>
<thead>
<tr>
<th>Average inter-arrival time</th>
<th>No. of packets sent</th>
<th>Average queuing delay</th>
<th>Average queue length</th>
<th>Throughput</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>20718</td>
<td>3.42</td>
<td>1.04</td>
<td>0.99</td>
</tr>
<tr>
<td>10</td>
<td>14930</td>
<td>2.69</td>
<td>0.66</td>
<td>0.99</td>
</tr>
<tr>
<td>15</td>
<td>8166</td>
<td>2.32</td>
<td>0.34</td>
<td>0.99</td>
</tr>
<tr>
<td>25</td>
<td>4946</td>
<td>2.18</td>
<td>0.20</td>
<td>0.99</td>
</tr>
<tr>
<td>30</td>
<td>4121</td>
<td>2.16</td>
<td>0.17</td>
<td>0.99</td>
</tr>
<tr>
<td>50</td>
<td>2545</td>
<td>2.10</td>
<td>0.10</td>
<td>0.99</td>
</tr>
<tr>
<td>100</td>
<td>1264</td>
<td>2.00</td>
<td>0.05</td>
<td>0.99</td>
</tr>
</tbody>
</table>

For example, let us consider the 3-ary 2-cube shown in Fig. 7, where nodes 10 and 21, and links 00-01 and 11-12 are faulty. The 1-reachable nodes of node 00 are (10,F), (02,N) and (20,N). (Symbols F and N denote faulty and non-faulty status of the node). Hence the routing table for node 00 with 1-reachability diagnosis in presence of node and link faults will be as shown in Table 3.

3.5. Discussion

The routing algorithm presented in the previous section has a major advantage over the Blough–Najand algorithm of [6] which also employs k-reachability diagnosis. If we were to employ the fault-tolerant routing algorithm of [6], each message has to carry an overhead which includes (a) the entire path from source to destination, as computed by the source node and (b) a Boolean array to remember the nodes that have already been visited. For a k-ary n-cube interconnection network, each node is represented by \( \log_2 N \) bits, and hence the message overhead can be as large as

\[
\text{Overhead} = N \log_2 N + N \text{ bits} \quad (7)
\]

For a 2-ary 10-cube interconnection network, the overhead for each message is approximately 1.1 KBytes,

Table 6
Performance of a non-faulty 16-ary 2-cube ICN at different loads

<table>
<thead>
<tr>
<th>Average inter-arrival time</th>
<th>No. of packets sent</th>
<th>Average queuing delay</th>
<th>Average queue length</th>
<th>Throughput</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>20718</td>
<td>88.34</td>
<td>38.8</td>
<td>0.14</td>
</tr>
<tr>
<td>10</td>
<td>14930</td>
<td>56.28</td>
<td>24.06</td>
<td>0.19</td>
</tr>
<tr>
<td>15</td>
<td>8166</td>
<td>28.40</td>
<td>12.84</td>
<td>0.25</td>
</tr>
<tr>
<td>25</td>
<td>4946</td>
<td>26.87</td>
<td>7.51</td>
<td>0.24</td>
</tr>
<tr>
<td>50</td>
<td>2545</td>
<td>22.53</td>
<td>3.83</td>
<td>0.25</td>
</tr>
<tr>
<td>100</td>
<td>1264</td>
<td>19.61</td>
<td>1.98</td>
<td>0.27</td>
</tr>
</tbody>
</table>
which makes the scheme in [6] impractical. This is especially so when a wormhole routing [3] scheme has to be employed. In wormhole routing, the packet is divided into small flow control digits, or flits. Each packet is decomposed into a header flt which contains the destination information and several data flits which contain the data. Routing decisions at a node are made when a header flt arrives at the node; the data flits of a packet simply follow the header flt. Thus it is possible for the header flt to reach the destination node before some of the data flits have even left the source node. If the header flt of a packet is unable to proceed further towards the destination node due to traffic congestion, the data flits of the packet also stop advancing. This is unlike store-and-forward switching techniques, where the entire packet must be assembled at every intermediate node between the source and the destination, making large buffers necessary. The situation is also unlike cut-through routing, where the entire message is assembled at an intermediate node if a packet cannot be routed further due to network congestion. Thus the main advantage of wormhole routing is that only small-sized flits must be buffered at a node. If the size of the header flt is large (as in Eq. 7), then the buffers at the intermediate nodes must also be large. Table 4 lists the relative merits and demerits of our algorithm and Blough–Najand algorithm.

### Table 8
Performance of an 8-ary 2-cube with 10 faults, for varying diagnostic radius. Packet arrival rate = 0.05

<table>
<thead>
<tr>
<th>Diagnostic radius</th>
<th>No. of hops travelled by undelivered packets</th>
<th>Average queuing delay</th>
<th>Throughput</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>192</td>
<td>6.08</td>
<td>0.78</td>
</tr>
<tr>
<td>3</td>
<td>145</td>
<td>2.73</td>
<td>0.80</td>
</tr>
<tr>
<td>4</td>
<td>47</td>
<td>2.62</td>
<td>0.80</td>
</tr>
<tr>
<td>5</td>
<td>17</td>
<td>2.62</td>
<td>0.80</td>
</tr>
<tr>
<td>6</td>
<td>3</td>
<td>2.62</td>
<td>0.80</td>
</tr>
<tr>
<td>7</td>
<td>1</td>
<td>2.60</td>
<td>0.80</td>
</tr>
<tr>
<td>8</td>
<td>0</td>
<td>2.58</td>
<td>0.80</td>
</tr>
</tbody>
</table>

### Table 9
Performance of an 8-ary 2-cube with 20 faults, for varying diagnostic radius. Packet arrival rate = 0.1

<table>
<thead>
<tr>
<th>Diagnostic radius</th>
<th>No. of hops travelled by undelivered packets</th>
<th>Average queuing delay</th>
<th>Throughput</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>1631</td>
<td>9.12</td>
<td>0.34</td>
</tr>
<tr>
<td>3</td>
<td>1112</td>
<td>4.44</td>
<td>0.35</td>
</tr>
<tr>
<td>4</td>
<td>630</td>
<td>2.36</td>
<td>0.37</td>
</tr>
<tr>
<td>5</td>
<td>398</td>
<td>2.11</td>
<td>0.37</td>
</tr>
<tr>
<td>6</td>
<td>98</td>
<td>1.95</td>
<td>0.38</td>
</tr>
<tr>
<td>7</td>
<td>14</td>
<td>1.95</td>
<td>0.38</td>
</tr>
<tr>
<td>8</td>
<td>0</td>
<td>1.95</td>
<td>0.38</td>
</tr>
</tbody>
</table>

## 4. Experimental results

The algorithms suggested in the previous section have been coded and simulated on both faulty and non-faulty k-ary n-cube interconnection networks. A SUN SPARC workstation was used as the host computer. The simulator software was developed in C and requires about 6000 lines of source code. We studied the dynamic performance of several k-ary n-cube networks. In particular, we were interested in performance metrics such as the average delay per packet, the average queue length at each node, and the message throughput. In our experimentation, we assume that each node generates messages randomly, the interval between two messages being an exponentially distributed random variable with mean $\alpha$. We further assume that a node can generate any of the $k^n$ addresses as destination using a uniform distribution.

Table 5 shows the performance of a non-faulty 2-ary 8-cube (256 nodes) when the procedure e-cube algorithm is used for routing. We simulated the performance of the network for different traffic conditions. The interarrival time represents the network traffic: smaller the interarrival time, the higher the traffic and vice versa. It can be seen that the throughput of the network remains close to unit with increasing traffic. The increase in queuing delay is modest, but the increase in queue length is significant.

## 5. Conclusions

In this paper, we have presented a routing algorithm for k-ary n-cubes which can adapt to network situations such as node faults and link faults. Routing decisions at each non-faulty node are made using a routing table which is constructed from f-reachability diagnostic information. A controlled amount of randomization is used in our algorithm to avoid backtracking and the associated message overheads. Simulation results indicate that the dynamic performance of our algorithm is good in the presence of node and link faults. We found
that increasing the diagnostic radius improves the dynamic performance of the interconnection network. With a smaller value of $\lambda$, the presence of a faulty node or link on the path is not known until the message gets nearer to the faulty component. This may force an intermediate node to use a spare dimension for routing messages around the faulty component, thus increasing the length of the actual path. We also considered the possibility of making our algorithm adaptive to traffic congestion. If congestion information is made available to nodes, the probability entries in the routing table can be dynamically updated, hence improving the network performance. But we feel that it is impractical to include congestion information (queue sizes) as part of diagnostic information. Even if the overhead for doing so were not prohibitive, the delays involved in transferring the congestion information to the nodes could make this information obsolete by the time the information reaches the other nodes. We feel the best alternative is to use $\lambda$-reachability diagnosis for faults and 1-reachability diagnosis for congestion information.

References