New Fault-Tolerant Routing Algorithms for *k*-Ary *n*-Cube Networks

Jehad Al-Sadi

Dissertation Submitted for the Degree of Doctor of Philosophy
to the Faculty of Computing Science, Mathematics and Statistics
University of Glasgow

Abstract

The interconnection network is one of the most crucial components in a multicomputer as it greatly influences the overall system performance. Networks belonging to the family of k-ary n-cubes (e.g., tori and hypercubes) have been widely adopted in practical machines due to their desirable properties, including a low diameter, symmetry, regularity, and ability to exploit communication locality found in many real-world parallel applications.

A routing algorithm specifies how a message selects a path to cross from source to destination, and has great impact on network performance. Routing in fault-free networks has been extensively studied in the past. As the network size scales up the probability of processor and link failure also increases. It is therefore essential to design fault-tolerant routing algorithms that allow messages to reach their destinations even in the presence of faulty components (links and nodes). Although many fault-tolerant routing algorithms have been proposed for common multicomputer networks, e.g. hypercubes and meshes, little research has been devoted to developing fault-tolerant routing for well-known versions of k-ary n-cubes, such as 2 and 3-dimensional tori.

Previous work on fault-tolerant routing has focused on designing algorithms with strict conditions imposed on the number of faulty components (nodes and links) or their locations in the network. Most existing fault-tolerant routing algorithms have assumed that a node knows either only the status of its neighbours (such a model is called local-information-based) or the status of all nodes (global-information-based). The main challenge is to devise a simple and efficient way of representing limited global fault information that allows optimal or near-optimal fault-tolerant routing.

This thesis proposes two new limited-global-information-based fault-tolerant routing algorithms for k-ary n-cubes, namely the *unsafety vectors* and *probability vectors* algorithms.

While the first algorithm uses a deterministic approach, which has been widely employed by other existing algorithms, the second algorithm is the first that uses probability-based fault-tolerant routing. These two algorithms have two important advantages over those already existing in the relevant literature. Both algorithms ensure fault-tolerance under relaxed assumptions, regarding the number of faulty components and their locations in the network. Furthermore, the new algorithms are more general in that they can easily be adapted to different topologies, including those that belong to the family of k-ary n-cubes (e.g. tori and hypercubes) and those that do not (e.g., generalised hypercubes and meshes).

Since very little work has considered fault-tolerant routing in k-ary n-cubes, this study compares the relative performance merits of the two proposed algorithms, the unsafety and probability vectors, on these networks. The results reveal that for practical number of faulty nodes, both algorithms achieve good performance levels. However, the probability vectors algorithm has the advantage of being simpler to implement. Since previous research has focused mostly on the hypercube, this study adapts the new algorithms to the hypercube in order to conduct a comparative study against the recently proposed safety vectors algorithm. Results from extensive simulation experiments demonstrate that our algorithms exhibit superior performance in terms of reachability (chances of a message reaching its destination), deviation from optimality (average difference between minimum distance and actual routing distance), and looping (chances of a message continuously looping in the network without reaching destination) to the safety vectors.

Acknowledgement

I would like to express my grateful thanks to God for giving me the strength to complete this thesis. Also, I would like to express my deep gratitude to Dr. M. Ould-Khaoua for his great help throughout the course of this work. My great thanks are also due to Dr. K. Day. They both provided me with continuous guidance, encouragement, and valuable feedback from the early stages of this thesis.

I am particularly indebted to my mother, wife, and my kids, for their unconditional love and moral support.

Finally, I am indebted to everyone who helped or encouraged me during my study for this postgraduate degree.

Contents

List of Figures

List of Tables

List of Symbols

$A^{(i)}$	neighbour node of node A along dimension i
$A^{(i+)}$, $A^{(i-)}$	the two neighbours of node A along the i^{th} dimension in a k-ary n-cube network
$A^{(i\pm)}$	$A^{(i+)}$ or $A^{(i-)}$
$\overline{D}$	average routing distance in the k-ary n-cube
$\overline{D}_k$	average routing distance assuming message is not discarded when routing between two nodes at Hamming distance k in the hypercube
d_L	Lee distance which is the shortest path between nodes A and B
$\overline{D}_l$	average routing distance from the source node A to destinations at Lee distance l
$\overline{D}_{l_1,l_2}$	average routing distance from the source node A to destinations with Lee distance component (l_1, l_2) in k-ary 2-cube
F_A	faulty set, which comprises those nodes which are either faulty or unreachable from node A
$H(A, B)$	*Hamming distance* between two nodes A and B
$i\pm$	positive or negative direction along dimension i
l_1	Lee distance across the first dimension in a k-ary 2-cube network
l_2	Lee distance across the second dimension in a k-ary 2-cube network
N_l	number of nodes at Lee distance l from the source node A
P^A	probability vector of node A

$P_{i,s}$	probability that s spare moves are made on dimension i
P_k^A	probability that a destination at Hamming distance k from A is not minimally reachable from A in the hypercube
$P_{k,f+1}$	probability of making more than f spare moves (i.e., probability of discarding a message) when routing between two nodes at Hamming distance k in the hypercube
$P_{k,s}$	probability of making exactly s spare moves when routing between two nodes at Hamming distance k in the hypercube
P_l^A	probability that a destination node at Lee distance l from A cannot be reached from A using a minimal path due to faulty nodes and links
$P_{l_1,l_2,s}$	probability of making exactly s spare moves when routing between the source node A and a destination with Lee distance components (l_1, l_2)
P_r	the least expected routing distance if node A can route through a preferred neighbour
PRA	Probabilistic Routing Algorithm
Q_n	n-dimensional hypercube which is an undirected graph with 2^n vertices
Q_n^k	k-ary n-cube, an undirected graph with k^n vertices (nodes)
$R_k^{A^{(i)}}$	probability that a destination at Hamming distance k from A is minimally reachable via its neighbour $A^{(i)}$ in the hypercube
$R_l^{A^{(i\pm)}}$	probability that a destination at Lee distance l from A is minimally reachable via its neighbour $A^{(i\pm)}$ in the k-ary n-cube
S_k^A	set of all nodes at Hamming distance k from A which are faulty or unreachable from A in a hypercube network
S_l^A	set of all nodes at distance l from A which are faulty or unreachable from A in a k-ary n-cube network
S_p	the least expected routing distance if node A can route through a spare neighbour

u_k	routing capability of node u to k-Hamming distance destinations in a hypercube network
$u_k^{A,D}$	number of faulty or unreachable (A, D)-preferred transit nodes at Hamming distance k from A in the hypercube
$u_l^{A,D}$	number of faulty or unreachable (A, D)-preferred transit nodes at Lee distance l from A
W_{l_1,l_2}	ratio of the number of nodes with Lee distance components (l_1, l_2) to the number of nodes at lee distance $l = l_1 + l_2$ from the source node A

Chapter 1

Introduction

Large-scale parallel systems are generally considered to be the most feasible way of achieving the enormous computational power required by many real-world applications in science, engineering, and a number of other fields [25, 27, 61, 76]. In such systems, parallel tasks inherent in an application are distributed over a set of processors to run simultaneously. The processors are physically interconnected by an *interconnection network*, and co-ordinate their activities to solve a common problem by exchanging information. Depending on the way the communication is achieved between processors, two types of parallel systems can be distinguished: *multiprocessors*, where processors share a common memory through which they communicate [33, 40, 61, 74], and *multicomputers*, where each processor has its own local memory, communicating with the other processors by message passing [54, 58, 60]. Multicomputers have experienced rapid development during the past decade and have gained more popularity over multiprocessors due to their superior scalability [9, 48, 73].

The interconnection network is one of the most crucial components in a multicomputer as it greatly influences the overall system performance. It is desirable for a network to be able to accommodate a large number of processors while maintaining low communication overhead.

Furthermore, it should be able to deliver messages reliably to their destinations through using alternative paths when some faults are detected. In order to allow processors to concentrate on computational tasks and permit the overlapping of communication with computation, a *router*, is used for handling message communication among processors, and is usually associated with each processor; the assembly of processor and router is called a *node*.

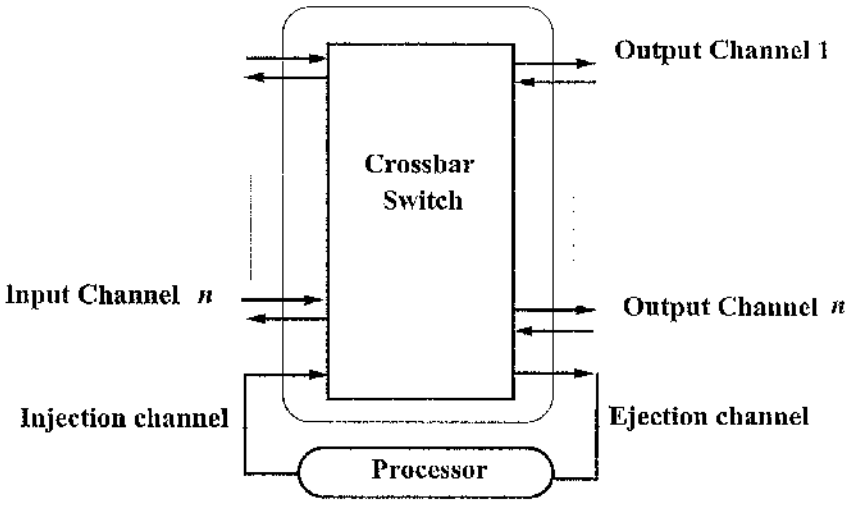


Fig. 1.1: The node structure in an interconnection network.

Fig 1.1 shows a typical node structure in an interconnection network. Each node consists of a processor and router. A node is connected to its neighbouring nodes through input and output channels. The injection/ejection channel is used by the processor to inject/eject messages to/from the network. A crossbar switch directs messages from any input channel to any output channel.

Network topology describes the way system nodes are connected, and is often characterised by its degree, diameter, and regularity. An interconnection topology can be modelled as an undirected graph where the vertices (nodes) represent the processors and the edges represent the

communication links between the processors. The *node degree* is the number of channels connecting the node to its neighbours, and the degree of a topology is the maximum degree of any node. The *diameter* of a topology is the maximum value of the shortest distance over all pairs of nodes. Finally, a topology is said to be *regular* if all its nodes have the same degree. Ideally, a network topology should have a small number of edges, a small degree, a low diameter, and regular structure. Needless to say, a large variety of topologies have been suggested in the hope of approaching these goals [1, 14, 24, 27, 66, 77].

Most practical multicomputers [8, 38, 63, 68, 80] employ *direct* networks where each node has a point-to-point or direct connection to some of the other nodes (known as its *neighbours*) allowing for direct communication between processors. An *indirect* network is another major class of interconnection networks where nodes are connected to other nodes (or memory banks in a shared-memory architecture) through multiple intermediate stages of switches. Because of their ability to exploit communication locality found in many parallel applications and better scalability, direct networks have been very popular in practical parallel machines. In particular, k-ary n-cubes have been widely used in current multicomputers [8, 75], due to their ease of implementation, regularity, and ability to exploit communication locality to reduce message delays.

The k-ary n-cube, where k is referred to as the *radix* and n as the *dimension*, has $N=k^n$ nodes, arranged in n dimensions, with k nodes per dimension. Links in the k-ary n-cube can be either uni- or bi-directional. In this thesis, we will focus on k-ary n-cubes with bi-directional links as they have been more popular in multicomputers [13, 38, 54, 56, 58, 68]. Each node can be identified by an n-digit radix k address $(a_1, a_2, \ldots, a_n)$. The i^{th} digit of the address vector, a_i, represents the node position in the i^{th} dimension. Nodes with address $(a_1, a_2, \ldots, a_n)$ and $(b_1, b_2$

,...,b_n) are connected if and only if there exists i, $(1 \le i \le n)$, such that $a_i = (b_i \pm 1) \bmod k$ and $a_j = b_j$ for $1 \le j \le n$; $i \ne j$. Typical properties of k-ary n-cubes include

- The number of nodes $= k^n$
- The diameter $= \begin{cases} 0 & \text{if } k = 1 \\ n\lfloor k/2 \rfloor & \text{if } k \ge 2 \end{cases}$
- The degree $= \begin{cases} 0 & \text{if } k = 1 \\ n & \text{if } k = 2 \\ 2n & \text{if } k > 2 \end{cases}$
- The number of links $= \begin{cases} nk^n & \text{if } k > 2 \\ \frac{1}{2}nk^n & \text{if } k = 2 \end{cases}$

Fig 1.2 gives examples of the 2 and 3-dimensional torus and binary hypercube (or hypercube for short), which are the most common instances of k-ary n-cubes. While the hypercube has been used in early multicomputers such as the Cosmic Cube [68], iPSC/2 [58], and NCube [54], the torus has become more common in the last generation of multicomputers, such as in the case of the iWarp [13], J-machine [56], CRAY T3D [38], and Cray T3E [8].

1.1. Routing Algorithms

A message usually travels across several intermediate nodes before reaching its destination. The routing algorithm specifies how a message selects a path to cross from source to destination, and has great impact on network performance. It may happen that due to some faulty network

components (either nodes or links), messages are not able to reach their destinations, even if fault-free paths exist connecting the source and destination nodes. It is therefore important to develop fault-tolerant routing algorithms that allow the network to continue to function as normally as possible, even in the presence of a large number of faults.

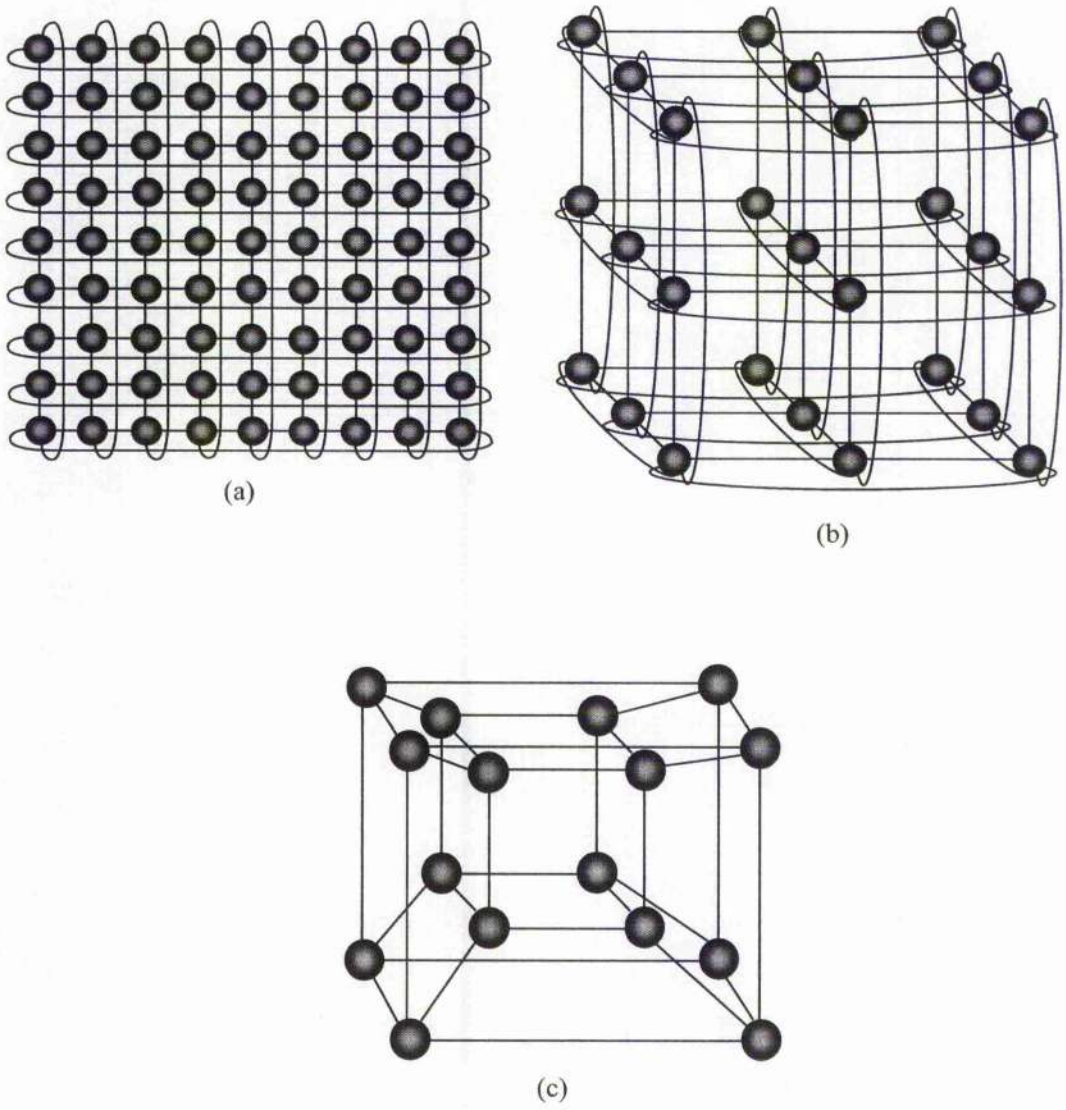


Fig. 1.2: Examples of *k*-ary *n*-cubes. (a) 9-ary 2-cube (2D-torus) (b) 3-ary 3-cube (3D-torus) (c) 2-ary 4-cube (hypercube).

Routing algorithms can be broadly classified as *deterministic* or *adaptive* [27]. The former offers a predetermined path to each message fixed by its source and destination addresses [37, 41, 42, 72]. The latter gives flexibility to messages in choosing their paths to avoid congested or faulty regions [16, 26, 31, 35, 44, 47, 64].

In a non-faulty network, a simple idea to route messages is to use a deterministic approach. A typical way of implementing this is seen in the classic deterministic routing, widely known as the *dimension-ordered* routing algorithm (also known as *e-cube* [22, 69]) used in the k-ary n-cube. In this form of routing, messages are restricted to change dimensions in a pre-defined (increasing or decreasing) order to ensure message reachability toward their destinations. Several practical multicomputers, such as the J-machine [56], Cray T3D [38], and N-Cube [54], have employed deterministic routing [37, 41].

Adaptive routing enables messages to explore all alternative paths to avoid congested regions or faulty components inside the network. Adaptive routing algorithms can be classified as progressive or backtracking [16, 17, 18, 20]. Progressive routing always moves forward, reserving a new node at each routing operation. Backtracking routing allows a message to backtrack to previously reserved nodes in cases where faulty nodes block the message to move forward. Backtracking is not often used for fault-tolerant routing because it is complex and costly to implement. The Cray T3E [8] and Reliable Router [23] are examples of recent practical systems that use adaptive routing.

Communication among nodes can be viewed as a hierarchy of services, starting from the physical layer that synchronises the transfer of bit streams to higher-level protocol layers that perform functions, such as packetisation. There are three major layers in the operation of the interconnection network, the physical layer, switching layer, and routing layer [27]. The physical

layer refers to link-level protocols for transferring messages across links interconnecting two adjacent routers. The switching layer utilises the physical layer protocols to implement mechanisms for forwarding messages from one router to the next. Finally, the routing layer makes routing decisions to determine a candidate intermediate node to route to and thereby establish the path through the network. The design of switching techniques (e.g., wormhole [26, 29], circuit [27, 51, 78], and store-and-forward switching [21, 28]) and their properties, (e.g. deadlock avoidance, live lock detection), are determined by the services provided by the switching layer [27, 28]. While our results are general enough and can be discussed in the context of, for example, wormhole and store-and forward switching this thesis deals primarily with issues related to the routing layer. More specifically, this thesis will show how the topological properties of a given interconnection, the k-ary n-cube in our case, can be exploited to provide efficient message routing that exhibits good fault-tolerance.

1.2. Fault-Tolerant Routing

Multicomputer systems are more susceptible to failure than conventional uniprocessor machines. This is because as the system size scales up, the probability of a component failure (node or link) also increases. There are two classes of faults. Either the entire node or any channel may fail. The former is referred to as *faulty node* and the latter as a *faulty link*. An *unreachable node* is a node that cannot be reached from the current node due to faulty links. Messages usually travel across several intermediate nodes before reaching their destinations. However, it may happen that some messages are not able to reach their destinations, even if fault-free paths exist connecting the source and destination nodes, due to failures of the routing decisions. In this work, we are interested in routing algorithms that can route a message from source to destination, if a path exists between such two nodes, in the presence of faulty components.

Fault-tolerant routing algorithms provide techniques to guarantee message delivery in the presence of faulty components in the network. The objective of a fault-tolerant routing algorithm is to ensure that the routing is successful in the presence of faults. In addition, the algorithm attempts to reduce latency by giving priority to a shorter alternative path over the longer ones when the optimal path is faulty.

Routing in faulty networks has been extensively studied in the past [15, 18, 81, 46, 83, 27]. As the network size scales up the probability of processor and link failure also increases. It is therefore essential to design fault-tolerant routing algorithms that allow messages to reach their destinations even in the presence of faulty components (links and nodes).

A fault-tolerant routing algorithm is *optimal* if it finds an optimal feasible path for every message, whenever a path exists. A path is feasible if it contains no faulty nodes. A path is optimal if it is the shortest feasible path. In direct interconnection networks, the diameter is usually a measure of the performance degradation caused by faults. Indeed, for a number of networks including hypercubes and k-ary n-cubes, it has been proved [24, 45] that an upper bound on the length of fault-free paths between non-faulty nodes in faulty networks (*fault diameter*) is closely related to the diameter of the network. A network is *k-node connected* if there exists at least k *node-disjoint paths* from any node S to any node D in the network [24]. Two paths are node-disjoint if they do not have any common intermediate node.

Although many fault-tolerant routing algorithms have been proposed for common multicomputer networks such as hypercubes [15, 16, 32, 43], very little research has been devoted to developing fault-tolerant routing for other versions of k-ary n-cubes, such as tori. Moreover, previous studies on fault-tolerant routing have focused on designing algorithms with strict conditions imposed on the number of faulty nodes or their locations in the network.

Most existing fault-tolerant routing algorithms have assumed that a node knows either only the status of its neighbours (such a model is called *local-information-based*) [16, 17, 32] or the status of all nodes (*global-information-based*) [15, 71]. Local-information-based routing yields sub-optimal routes (if not routing failure) due to the insufficient information upon which the routing decisions are made. Global-information-based routing can achieve optimal or near optimal routing, but often at the expense of high communication overhead to maintain up-to-date network-wide fault information. The main challenge is therefore to devise a simple and efficient way of representing *limited global fault information* that allows optimal or near-optimal fault-tolerant routing. There have recently been a number of attempts to design limited-global-information-based algorithms [19, 20, 46, 81, 84].

Among the family of k-ary n-cube direct networks, the binary hypercube has received the most attention in the past, for which a number of fault-tolerant routing algorithms have been proposed [18, 46, 81, 83]. For instance, Gordon and Stout [32] have described a fault-tolerant routing based on "*Sidetracking*", where a message is de-routed to a randomly chosen fault-free neighbouring node when no fault-free neighbour exists along any of the existing optimal paths leading to the destination. With this approach a routing failure may occur (although with low probability for large n), and excessive delay may arise even in the presence of few faulty components [20].

Chen and Shin [17] have proposed a routing strategy based on depth-first search in which *backtracking* is required if all the required forward links cannot be used due to faulty components. The traversed path is recorded and attached to the message. A simplified version of this approach that tolerates fewer faults was presented in [16], where routing is progressive without backtracking, and where a message is routed to its destination on an optimal path with

high probability. Lan [44] has presented a fault-tolerant routing algorithm based on local information, and which guarantees an optimal or near-optimal routing. However, the algorithm is based on a restricted model of fault distribution as it can tolerate only (n-1) faulty nodes (and/or links) in an n-dimensional cube.

Lee and Hayes [46] have used the concept of *unsafe nodes* to design a fault-tolerant routing strategy for the hypercube. Message routing is achieved by avoiding unsafe nodes, which could possibly lead to communication difficulties and excessive delays. Chiu and Wu [20] have used the concept of *unsafe nodes* [46] and its extensions to show that a feasible path of length not more than the Hamming distance plus four can be guaranteed, provided that the number of faulty nodes does not exceed (n-1), where n is the dimension of the hypercube. The concept of unsafe nodes has also been discussed in [83].

Wu in [84] has introduced the concept of *safety levels*, based on limited-global-information, as an enhancement of the unsafe node concept. The safety level is an approximate measure of the number as well as the distribution of faulty nodes. Optimal routing is guaranteed if the safety level of the source node is less than the Hamming distance between the source and destination. Chiu and Chen [19] have proposed a concept called *routing capability*, which further enhances the safety levels concept.

The *safety vectors* algorithm, proposed by Wu [81], uses a similar concept to the routing capability with some extensions related to dynamic routing adaptivity and application to the generalised hypercube [10]. The safety vectors approach requires each processor to maintain a bit vector (safety vector) computed through a number of fault information exchanges between adjacent processors. The algorithm guarantees optimal routing to all destinations that are at a Hamming distance k from node A, if and only if, the k^{th} bit of the safety vector at node A is set

(the safety vectors approach will be described in detail later in chapter 4). The safety vectors approach has been extended in [82] for better handling of faulty links.

The only modest contribution in designing fault-tolerant routing algorithms for k-ary n-cubes was described in [64]. In this work, Ravinkumar and Panda [64] have proposed an adaptive routing algorithm for k-ary n-cubes where a large table of information is stored at each node, containing a sorted list of entries for every candidate destination in the k-ary n-cube. Each entry corresponds to the address of a given destination and a list of optimal neighbours that lead to that destination. This algorithm is very costly in terms of computational complexity and storage due to the size $O(nk^n)$ of the routing table at each node. Also, the algorithm selects an intermediate node randomly among the candidate nodes through a complex procedure during routing.

The following three measures have often been used to give indications of the performance of a given fault-tolerant routing algorithm [6, 81, 83]:

- *Unreachability*: is the percentage of messages that cannot be routed towards their destinations due to failures over the total number of generated messages.

- *Deviation from optimality*: is the average difference between the minimal routing distance and the actual routing distance.

- *Looping*: is the percentage of messages continuously looping in the network without reaching their destinations over the total number of generated messages.

1.3. Motivations

Many studies have proposed fault-tolerant routing algorithms in the past [16, 17, 20, 44].

However, most of these studies have imposed strict conditions on the number of faulty nodes in the network and their locations. Some studies have defined and used the concepts of safe and unsafe nodes according to their location in a faulty network (see [44, 46, 83, 84] for more detailed definitions). Such studies often impose strict conditions on the number of safe, unsafe, and faulty nodes in order to ensure message delivery.

Although many fault-tolerant routing algorithms have been proposed for common multicomputer networks, such as the hypercube, very little research has been devoted to developing fault-tolerant routing for other versions of k-ary n-cubes, e.g. 2 and 3-dimensional tori. Furthermore, most existing fault-tolerant routing algorithms are either global-information-based, and as a result suffer from costly communication overhead, or local-information-based, and as a result are unable to make optimal routing decisions. Therefore the challenge is to design an efficient fault-tolerant routing algorithm based on limited-global-information. The main objective of this thesis is to contribute towards filling this gap by introducing and investigating new and efficient fault-tolerant routing algorithms for k-ary n-cubes.

Motivated by the above observations, this thesis introduces two new limited-global-information-based fault-tolerant routing algorithms for k-ary n-cubes, namely the unsafety vectors and probability vectors. In the unsafety vectors algorithm, each node A starts by determining the set of faulty or unreachable neighbours. Then, node A performs $(n\lfloor k/2 \rfloor - 1)$ exchanges with its neighbours to determine its faulty set containing all faulty or unreachable nodes at different distances from node A. For node A, the l-level unsafety set S_l^A, $1 \leq l \leq m$, where m is an adjustable parameter between 1 and $n\lfloor k/2 \rfloor$, represents the set of all nodes at distance l from A which are faulty or unreachable from A. Equipped with these unsafety sets, each node calculates numeric unsafety vectors and uses them to achieve efficient fault-tolerant routing.

The new fault-tolerant routing algorithm routes messages to their destinations over an optimal path in the network. If all optimal paths are faulty, then the new algorithm seeks the shortest available path to route messages toward their destinations. The algorithm routes messages to their reachable destinations without any strict conditions on the number of faults or their locations, and it can deal with both faulty nodes and faulty links during routing. Furthermore, the algorithm exhibits good performance characteristics in terms of the achieved routing distance and reachability.

Most fault-tolerant routing algorithms reported in the relevant literature, including the proposed unsafety vectors [6, 7], use a *deterministic* approach (non probability based) to reflect information about faults in the network. Motivated by this observation, we develop another new routing algorithm, referred to as "*probability vectors*", that achieves fault-tolerance in k-ary n-cubes using a new probabilistic approach. To compute the probability vectors, a node first determines its faulty set, which represents the set of all its neighbouring nodes that are faulty or unreachable due to faulty links. Each node then calculates a probability vector, where the k^{th} element represents an estimated of the probability that a destination node at distance k cannot be reached through an optimal path due to a faulty node or link. The probability vectors are used by all the nodes to achieve an efficient fault-tolerant routing in the network. This new algorithm has the advantage of being the first fault-tolerant routing algorithm that uses the probability approach. Moreover, it is simpler to implement than those algorithms that use the deterministic approach. Each node maintains probability information about nodes at distance k for every k. The routing algorithm is source-destination independent, and always chooses the neighbour with the best probability of reachability towards the desired destination.

In the second part of this study, we compare the relative performance merits of the two proposed

algorithms, the unsafety and probability vectors in terms of communication overhead, computation complexity, average routing distance, deviation from optimality, and unreachability. We also show how to adapt the new algorithms, unsafety and probability vectors, to the hypercube in order to conduct a comparative study against the safety vectors algorithm, that has been recently proposed for the hypercube. This study highlights an important advantage of our proposed algorithms and their ability to be easily adapted to other topologies. Moreover, results from the comparative study reveal that the new algorithms exhibit superior performance characteristics over the safety vectors.

1.4. Outline of the Thesis

Chapter 2 presents the unsafety vectors approach as a new deterministic fault-tolerant routing algorithm for the k-ary n-cube. The calculations of the unsafety vectors and the routing algorithm along with its properties are presented. A simulation study that evaluates the unreachability, deviation from optimality, and looping of the new algorithm is also described.

Chapter 3 presents the second new approach, namely the probability vectors algorithm, for the k-ary n-cube. The calculations of the probability vectors, the routing algorithm along with its properties, and related simulation results are presented. Furthermore, this chapter compares the relative performance merits of the two proposed algorithms, the unsafety and probability vectors, in the k-ary n-cube.

Since there exist hardly any fault-tolerant routing algorithms for the k-ary n-cube against which to compare the two routing algorithms proposed in this work, Chapter 4 adapts the unsafety vectors algorithm to the hypercube, and conducts a comparative analysis against the recently proposed safety vectors routing algorithm for the hypercube.

As in Chapter 4, Chapter 5 adapts the probability vectors algorithm for the hypercube, and conducts a comparative study against the safety vectors algorithm.

Finally, chapter 6 summarises the results presented in this thesis, and discusses some possible directions for future research.

Chapter 2

The Unsafety Vectors: A New Fault-Tolerant Routing Algorithm for *k*-Ary *n*-Cubes

2.1 Introduction

Most practical multicomputers [8, 38, 58, 80] have employed *k*-ary *n*-cubes for low-latency and high-bandwidth inter-processor communication. The *k*-ary *n*-cube has an *n*-dimensional grid structure with *k* nodes in each dimension such that every node is connected to its neighbouring nodes in each dimension by direct channels. The two most popular instances of *k*-ary *n*-cubes are the hypercube (where k=2) and torus (where n=2 or 3). The former was used in early multicomputers such as the iPSC/2 [58] and iPSC/860 [80] while the latter has been adopted in more recent systems, like the CRAY T3D [38] and CRAY T3E [8].

Routing in fault-free cubes has been extensively studied in the past [15, 27, 46, 81, 83]. As the network size scales up the probability of processor and link failure also increases. It is therefore essential to design fault-tolerant routing algorithms that allow messages to reach their destinations even in the presence of faulty components (links and nodes). There have been a number of recent studies reported [18, 46, 81, 83] that have described fault-tolerant routing

algorithms based on limited-global-information. Most of these algorithms, however, have been developed for the hypercube [18, 46, 81, 83]. As a result, few studies have considered the other versions of the k-ary n-cubes, such as tori. In fact, most of the existing research on k-ary n-cubes has dealt with the practical and implementation issues associated with fault-tolerant routing [27, 31, 35]. Except for the research of [64], there has been hardly any study that investigates the *topological* properties of k-ary n-cubes for the provision of efficient fault-tolerant routing algorithms.

In [64], the authors have described a fault-tolerant routing algorithm for k-ary n-cubes, which requires a large information table in each node. The table contains an entry for every other node in the network, containing a list of optimal neighbours leading to a particular destination node, along with the associated probabilities. The algorithm suffers from high computational complexity and storage cost since nodes maintain global information about the network.

This chapter introduces a new limited-global-information-based routing algorithm for the high-radix k-ary n-cube, like the torus. The proposed new algorithm uses a *greedy* approach by giving higher attention to the immediate next routing step in avoiding faulty neighbourhoods. As we shall see later in this chapter, the algorithm uses the concept of "*unsafety vectors*" to considerably reduce the storage requirement for maintaining fault information, compared to the algorithm proposed in [64]. In the proposed algorithm, each node A starts by determining the set of faulty or unreachable neighbours. Then, node A performs $(n\lfloor k/2 \rfloor - 1)$ exchanges with its neighbours to determine its faulty set containing all faulty or unreachable nodes at different distances from node A. For a node A, the l-level unsafety set S_l^A, $1 \le l \le m$, where m is an adjustable parameter between 1 and $n\lfloor k/2 \rfloor$, represents the set of all nodes at Lee distance l from A which are faulty or unreachable from A (a definition of a Lee distance is provided in the

sequent).

Equipped with these unsafety sets, we show how each node calculates numeric unsafety vectors, and uses them to achieve efficient greedy fault-tolerant routing. The larger the value of m is, the better the routing decisions should be, but at the expense of more computation and communication overhead.

The amount of the limited-global-information-based using the unsafety vectors (f addresses, where f is the number of faulty nodes which is a small fraction of k^n) is substantially smaller than the amount of information usually needed by global-information-based algorithms which is in the order of k^n. The simplicity and reduced size of the routing information results in faster routing decisions and decreases the amount of exchanged information. Global-information-based algorithms have the advantage of achieving optimal routing. However, our proposed limited-global-information-based algorithm achieves near optimal routing with a big reduction both in the amount of exchanged routing information and in the complexity of the routing algorithm.

This chapter also includes a performance evaluation of the proposed algorithm through extensive simulation experiments. The results reveal that the algorithm performs near optimal routing for practical values of the numbers of faulty nodes. The obtained measures of routing distances and percentages of reachability are very efficient even when the parameter m is at its lowest value of 1 corresponding to minimum communication overhead. Before presenting the new fault-tolerant routing algorithm, this chapter reviews first some background information (preliminaries and notation) that will be useful for the subsequent sections.

2.2 Preliminaries and Notation

The k-ary n-cube, Q_n^k, is an undirected graph with k^n vertices (nodes). Each node A is labelled in the form $A=a_{n-1},a_{n-2},\ldots,a_0$, where each a_i digit satisfies: $0 \le a_i < k$. Two nodes $A=a_{n-1},a_{n-2},\ldots,a_0$ and $B=b_{n-1},b_{n-2},\ldots,b_0$ are joined by a link if, and only if, there exists i, $0 \le i < n$, such that $a_i = b_i \pm 1 \pmod{k}$ and $a_j = b_j$ for $i \neq j$. For the sake of clarity, we will omit writing *mod k* in similar expressions in the remainder of our discussion. Q_n^k has degree $2n$ and diameter $n\lfloor k/2 \rfloor$. The shortest path between nodes A and B is equal to their *Lee distance* [14] given by

$$d_L(A,B) = \sum_{i=0}^{n-1} w_i \text{, where } w_i = \min_{0 \le i < n}(|a_i - b_i|, k - |a_i - b_i|).$$

The *Hamming distance* between two nodes A and B, denoted $H(A, B)$, is the number of digits at which their labels differ. A path between A and B is an *optimal path* if its length is equal to $d_L(A,B)$.

A routing algorithm R for a network G can be viewed as a function that returns the address of the next node to visit in order to achieve routing between a given source and a given destination. A fault-tolerant routing algorithm is a routing algorithm that is able to function in a network with faulty components (nodes and links).

Consider two nodes A and D where A is the source and D is the destination of a message exchange. Let $A^{(i+)}$ and $A^{(i-)}$ represent the two neighbours of node A along the i^{th} dimension and let $A^{(i\pm)}$ denote $A^{(i+)}$ or $A^{(i-)}$. The symbol $i\pm$ denotes the positive or negative direction along dimension i. If $a_i \neq d_i$, a neighbour $A^{(i\pm)}$ of A is called a *preferred neighbour* for routing from node A to D if $d_L(A^{(i\pm)}, D) = d_L(A, D) - 1$. We say in this case that $i\pm$ is a *preferred direction.* If $a_i \neq d_i$, a neighbour $A^{(i\pm)}$ such that $d_L(A^{(i\pm)}, D) \ge d_L(A, D)$ is called a *spare neighbour.*

Neighbours other than preferred or spare are called *disturb neighbours*. For routing from A to D, a disturb neighbour $A^{(i,1)}$ of A corresponds to the case $a_i = d_i$ and therefore the i^{th} digit is disturbed. Routing through a disturb neighbour increases the total routing distance by at least two over the minimum distance. Routing through a spare neighbour increases the total routing distance by at least one over the minimum distance. A minimal path can be obtained by performing a preferred direction move at every routing step. With respect to routing from node A to D, node T is called a preferred *transit node* if $d_L(T, D) < d_L(A, D)$.

We make the following assumptions for the proposed algorithm and performance study. Similar assumptions have been made in earlier related works, e.g. [27, 64, 81].

i) A faulty k-ary n-cube contains faulty nodes and/or links. The fault pattern remains fixed for the duration of calculations of unsafety sets. In other words, the faulty sets calculation has to be restarted if additional faults occur before completing the calculation.

ii) Each node can determine the status of its own links and the status of its neighbouring nodes.

iii) Node failures are fault-stop failures.

2.3 The Proposed Fault-Tolerant Routing Algorithm

Our proposed fault-tolerant routing algorithm uses the concept of unsafety sets. The l-level unsafety set of a node A contains faulty or unreachable nodes at distance l from A. The unsafety sets of a node A are obtained from the faulty set F_A, which comprises those nodes which are either faulty or unreachable from A. F_A is first initialised to the set of faulty immediate neighbours of A. F_A is then updated by performing $(n\lfloor k/2 \rfloor - 1)$ exchanges of this set between non-faulty immediate neighbours. After each such exchange, new faulty nodes are detected and

added to F_A. Added to those detected faulty nodes is the set of unreachable neighbours of A due to faulty links. After determining F_A, node A calculates m unsafety sets, where m is an adjustable parameter between 1 and $n\lfloor k/2 \rfloor$, denoted S_1^A, S_2^A,..., S_m^A as defined below:

Definition 2.1: The l-level unsafety set S_l^A, $1 \le l \le m$, for node A is given by

$$S_l^A = \{B \in F_A \mid d_L(A,B) = l\}$$

```
Algorithm Find_Unsafety_Sets (A: node)
/* called by node A to determine its faulty and unsafety sets */
  F_A = set of faulty immediate neighbours;
  for l := 1 to n⌊k/2⌋-1 do {
      for i:=1 to n do {
          if A^(i+) ∉ F_A then {
              send F_A to A^(i+);
              receive F_A^(i+) from A^(i+);
              F_A = F_A ∪ F_A^(i+); }
          if A^(i-) ∉ F_A then {
              send F_A to A^(i-);
              receive F_A^(i-) from A^(i-);
              F_A = F_A ∪ F_A^(i-);
      } } }
  for l:=1 to n do {
  if link (A, A^(i+)) faulty then F_A := F_A ∪ { A^(i+) };
  if link (A, A^(i-)) faulty then F_A := F_A ∪ { A^(i-) };
  }
  for l := 1 to m do S_l^A = {B ∈ F_A | d_L(A,B) = l}
End.
```

Fig. 2.1: The algorithm for calculating faulty and unsafety sets.

Definition 2.2: If for some node A, $\left|S_1^A\right| = 2n-1$, then node A is called a *dead-end node.*

The l-level unsafety set S_l^A represents node A's view of the set of nodes at Lee distance l from A which are faulty or unreachable. As it will be subsequently seen in this study, when the routing distance exceeds a certain threshold, network partitioning is assumed. In this case the desired destination D is considered in a different network partition. The detecting node propagates to all the reachable nodes the fact that D is unreachable. All these nodes add node D to the set F_A. Fig. 2.1 gives an outline of the "Find_Unsafety_Sets" algorithm that node A uses to determine its faulty and unsafety sets.

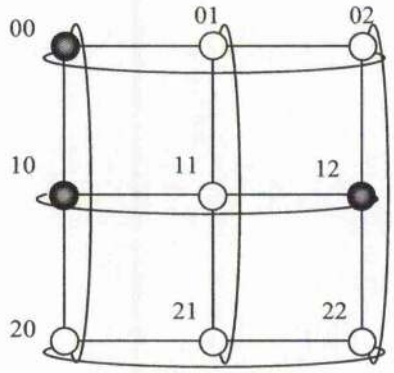


Fig. 2.2: An example of a 3-ary 2-cube with three faulty nodes.

Example 2.1: Consider a 3-ary 2-cube with three faulty nodes, as shown in Fig. 2.2 (faulty nodes are represented in dark colour). Table 2.1 shows the corresponding unsafety sets associated with each node A.

Table 2.1: The unsafety sets of nodes in a 3-ary 2-cube with 3 faulty nodes

Node A	00	01	02	10	11	12	20	21	22
S_1^A	Faulty	{00}	{00,12}	Faulty	{10,12}	Faulty	{00,10}	{ }	{12}
S_2^A	Faulty	{10,12}	{10}	Faulty	{00}	Faulty	{12}	{00,10,12}	{00,10}

2.4. Outline of the Algorithm

Definition 2.3: For a given source-destination pair of nodes (A, D), the (A, D)-*unsafety vector* $U^{A,D} = (u_1^{A,D},\ldots,u_l^{A,D},\ldots,u_m^{A,D})$ is given by:

$$u_l^{A,D} = |\{T \in S_l^A \text{, such that } T \text{ is a } (A, D)\text{-preferred transit node}\}|.$$

In other words, the element $u_l^{A,D}$ is the number of faulty or unreachable (A, D)-preferred transit nodes at distance l from A. $u_l^{A,D}$ can be viewed as a measure of routing unsafety at distance l from A, hence the name *unsafety vectors* for $U^{A,D}$. We also define an ordering relation '<' for numeric vectors as follows. For any two numeric vectors $U = (u_1, u_2,\ldots, u_m)$ and $V = (v_1, v_2, \ldots, v_m)$, $U < V$ iff $\exists\ i$, $1 \le i \le m$, such that $u_i < v_i$, and $u_j = v_j$ for all $j < i$. This ordering relation will correspond to a greedy approach in making routing decisions.

Fig. 2.3 provides a description of the proposed UV_Routing algorithm. When a node A has to forward a message M towards its destination D it uses UV_Routing in order to achieve fault-tolerance routing in the network. The algorithm checks first if the destination is a reachable immediate neighbour in which case the message is delivered to destination. If not, UV_Routing tries to forward the message to a non-faulty intermediate preferred neighbour that is associated with the least unsafety vector. If all preferred neighbours are faulty then the algorithm tries to route through a 'disturb' neighbour with the least unsafety vector as a second choice or through a non-faulty spare neighbour with the least unsafety vector as a third choice. Notice that routing through a disturb neighbour increases the routing distance by at least 2 but routing through a spare neighbour (long cycle) may increase the routing distance by k-2 in the worst case. The worst case may occur when the source node and the destination are at distance 1 on this spare dimension and the fault situation imposes moving on the long part of the cycle of this dimension

```
Algorithm UV_Routing (M: message; A, S, D: node)
/* called by node A to route the message M initiated at source node S
towards its destination node D */
 if A =S then M.Route_distance = 0
 if M.Route_distance ≤ d_L(A, D) + (k-2) × |F_A| then
 {
  if A = D then exit;  /* destination reached */
  M.Route_distance: =M.Route_distance + 1
  Let A^(i±) be the reachable preferred neighbour with least
     (A^(i+), D)-unsafety vector U^(A(i±),D) and A^(i±) is not dead-end
     or (A^(i±) =D)
  if A^(i±) exists then send M to A^(i+)
     else { Let A^(i±) be the reachable Disturb neighbour with least
           (A^(i±), D)-unsafety vector U^(A^(i±),D) and A^(i±) is not dead-end
           if A^(i±) exists then send M to A^(i±)
           else { Let A^(i∓) be the reachable spare neighbour with least
                  (A^(i∓), D)-unsafety vector U^(A(i∓,D)) and A^(i∓) is not
                  dead-end;
                  if A^(i∓) exists then send M to A^(i∓)
                  else failure /* destination unreachable */
                }
           }
 }
 else Handle looping

End.
```

Fig. 2.3: The proposed fault-tolerant UV_Routing algorithm.

(of length k-1). Therefore k-2 extra moves are imposed. If an immediate non-faulty neighbour is not available then the destination is unreachable. Routing failure occurs in such cases.

Example 2.2: Consider a 3-ary 2-cube with three faulty nodes, depicted in Fig. 2.2. To route a message from the source node 22 to the destination node 01, first node 22 has two preferred neighbours 02 and 21, but since node 21 has the least number of faulty preferred neighbours, the UV_Routing algorithm will route to it as an intermediate node, and then finally to its destination node 01.

Notice from the description of the UV_Routing algorithm, given in Fig. 2.3, that looping is detected if the routing distance exceeds the specified limit (Lee distance plus f (k-2) where f is the number of faulty nodes). Since each faulty node may cause a derouting and an increase in the routing distance by a value ranging between 1 and k-2, the maximum increase in the routing distance should not exceed $f(k$-2). The proposed algorithm can be improved to minimise the effect of looping. Since looping occurs when a destination is not reachable from the source, we can add the destination node to the faulty set of the node that detected the looping. When looping occurs ($n\lfloor k/2 \rfloor$-1) exchanges of information between all neighbours are then initiated to propagate the new information among reachable nodes in the whole k-ary n-cube.

2.5 Performance Analysis

In this section, we first analyse the complexity of the calculations in the UV_Routing algorithm, and then analyse the performance of the algorithm using software simulation.

The calculation of the unsafety sets and then the calculation of the unsafety vectors involve information exchanges between the network nodes. This calculation is performed in $n\lfloor k/2 \rfloor$phases. In each phase, each node sends at most $2n$ messages and receives at most $2n$

messages concurrently. Therefore the computation time complexity is $O(n^2k)$ and the total number of generated messages is $O(n^2k^{n+1})$. Notice that the computation time complexity of the routing algorithm in [64] is $O((nk^n)^2)$ and the total number of generated messages is $O((nk^n)^2k^n)$. Furthermore, the storage complexity of the UV_Routing algorithm is at most a set of f node addresses at each node, where f is the total number of faulty nodes (practically a small fraction of k^n). On the other hand, the storage complexity in [64] is in the order of k^n tuples at each node, each tuple contains a node address and a probability of successful routing of that node for a given source-destination pair. This excessive communication and computation cost effectively reduces routing performance. The UV_Routing algorithm compares favourably with the routing algorithm in [64] with respect to both communication cost and storage cost.

This section also analyses the performance of the proposed fault-tolerant routing algorithm. We have developed a simulation program. The nodes and links were coded according to the topological properties of the simulated network. Simulation results were tested for different sizes of the networks and compared against manual calculation of the routing vectors. The simulation results have matched our manual calculation results. Furthermore, we have also tested our simulator on an existing safety vectors algorithm [81]. The obtained results have been found to be in good agreement with those presented in [81].

Simulation experiments have been carried out for a 10-ary 3-cube with 1000 nodes with different random distributions of faulty nodes. We started our experiments with a non-faulty k-ary n-cube and then the number of faulty nodes was gradually increased up to 75% of the network size with random fault distribution. A relatively large number of source-destination pairs (30,000) have been randomly generated at each run. We have made sure that these generated pairs cover all possible source-destination distances. In the first two sets of results reported below (in Figs. 2.4,

and 2.5 respectively), the parameter m is fixed at its lowest value, i.e. m=1, where m is an adjustable parameter between 1 and $n\lfloor k/2 \rfloor$, corresponding to the lowest communication overhead introduced by the algorithm since the exchange of information is only between neighbours. We will report at the end of this section (in Fig. 2.6) on an experiment that assesses the effects of the parameter m on the performance of the algorithm. However, before presenting the results, we define the following variables, which will be used to quantify some performance measures.

- *Total*: total number of generated messages
- *Routing_Distance*: number of links crossed by a message.
- *Lee_Distance*: Lee distance between the message source and destination.
- *Fail_Count*: number of routing failure cases.
- *Looping_Count*: number of messages that cross a number of links beyond a maximum threshold before being discarded.

Using the above variables we propose the following three performance measures as the basis for studying the new UV_Routing algorithm.

- Average percentage of deviation from optimality $= \frac{1}{Total} \sum \frac{Routing\_Distance - Lee\_Distance}{Lee\_Distance} \times 100$
- Percentage of unreachability $= \frac{Fail\_Count}{Total} \times 100$
- Percentage of looping $= \frac{Looping\_Count}{Total} \times 100$

The average percentage of deviation from optimality indicates how close the achieved routing is to the minimal distance routing. The percentage of unreachability measures the percentage of

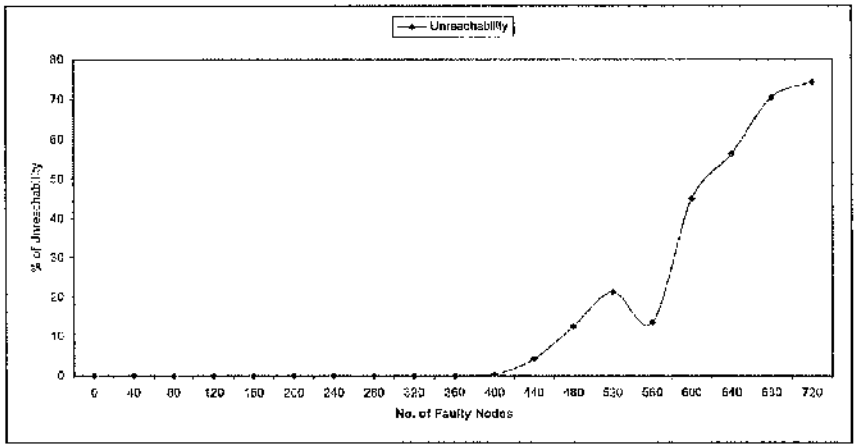
messages that the algorithm failed to deliver to destination due to faulty components. The percentage of looping indicates the ratio of messages that failed to reach destinations due to network partitioning; network partitioning occurs when the faulty nodes and links divide the network to two or more disconnected parts. We believe that these three measures give realistic indication on the performance of a fault-tolerant routing algorithm and are adequate for the purpose of our present study.

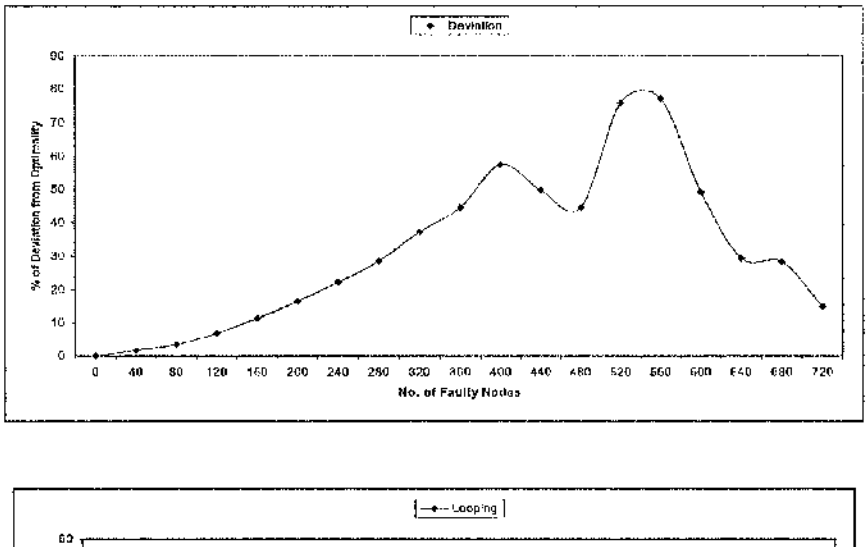
Fig. 2.4 reveals that the UV_Routing algorithm achieves high reachability with low to moderate deviation from optimality. The average percentage of deviation from optimality grows almost linearly with number of faulty nodes as long as this later does not exceed 50% of the total number of faulty nodes. The proposed algorithm is capable of routing messages using optimal or near optimal distance paths even when there is a large number of faulty components. This is due to the fact that the algorithm repeatedly chooses to route through areas of the network with the least number of faults in the neighbourhood, applying a greedy approach giving more weight to the nearest neighbourhoods. As a result, the algorithm tends to select paths that diverge from areas with high counts of faulty components. The figure also reveals that the percentage of looping remains practically negligible when the percentage of faulty nodes remains less than 20%. When there are a high number of faulty nodes in the network, the number of messages reaching destination becomes low, justifying the drop in the deviation and looping measures in Fig 2.4.

An experiment was conducted to assess the performance behaviour of the proposed algorithm when the network size increases. For the sake of illustration, we have fixed the value of n to 3, and increased the value for k from 2 up to 9 (for a network size varying from 8 to 729). For each network size, our algorithm has been tested by setting the percentage of faulty nodes to 10% of

the network size, then to 20%, 30%, 40%, and 50% of the network size. At each run, a total of 10,000 source-destination pairs were selected randomly. The results presented in Fig. 2.5 show that the performance properties of the new routing algorithm are maintained as the network size is scaled up. Furthermore, good performance levels are achieved without imposing any restriction on the system size or impractical restrictions on the number of faults in the network.

As we have mentioned earlier, each node calculates its m unsafety sets after calculating its F_A through ($n\lfloor k/2 \rfloor$-1) exchanges with its neighbours. The adjustable parameter m can take values between 1 and $n\lfloor k/2 \rfloor$, representing the level of complexity of the proposed algorithm. The larger the value of m is, the better the routing decisions are expected to be. However, this is done at the expense of more computation and communication overhead. The results in Fig. 2.6 reveal that when m is increased to 3, the algorithm has a slightly better performance than in the case m=1, shown in Figs. 2.4 and 2.5. It is worth noting that we have found that the same conclusions are obtained even when m is increased beyond 3. The small improvement in the routing performance for higher values of m does not justify the higher communication overhead. We can therefore conclude that the proposed algorithm yields good performance in terms of routing distances and percentages of reachability even when the parameter m is at its lowest value of 1, corresponding to minimum overhead.





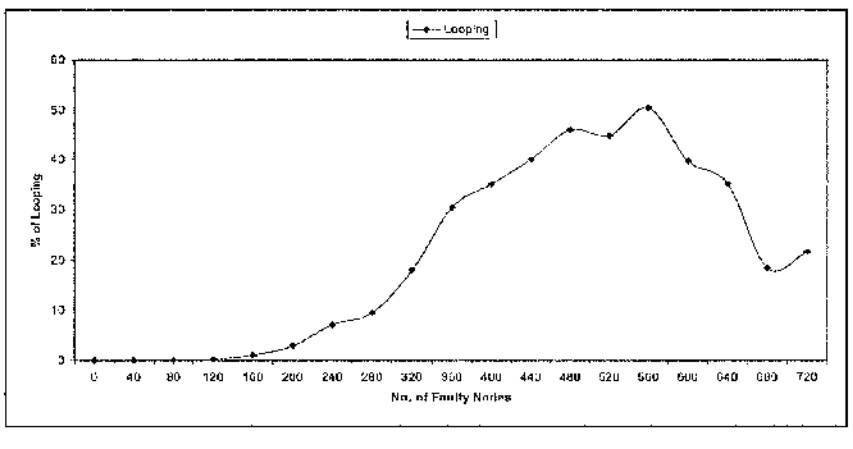
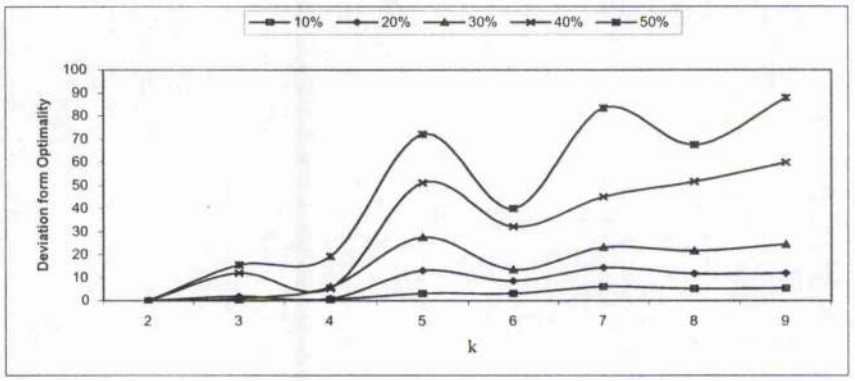
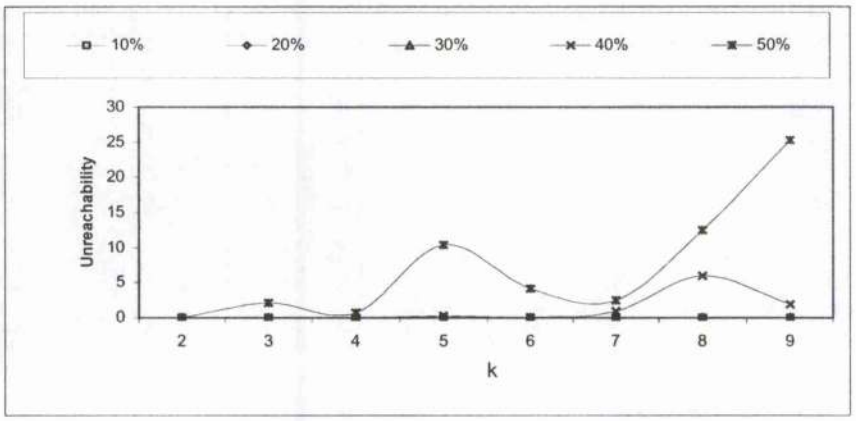


Fig. 2.4: Average percentage of deviation, percentage of unreachability, and percentage of looping in the UV_Routing algorithm.





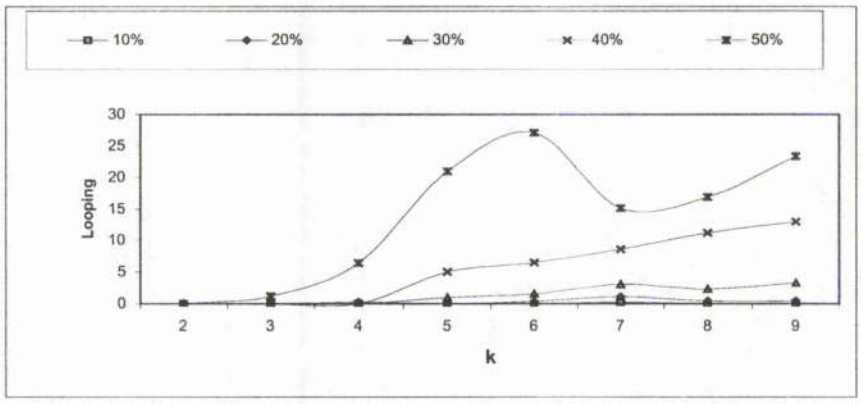
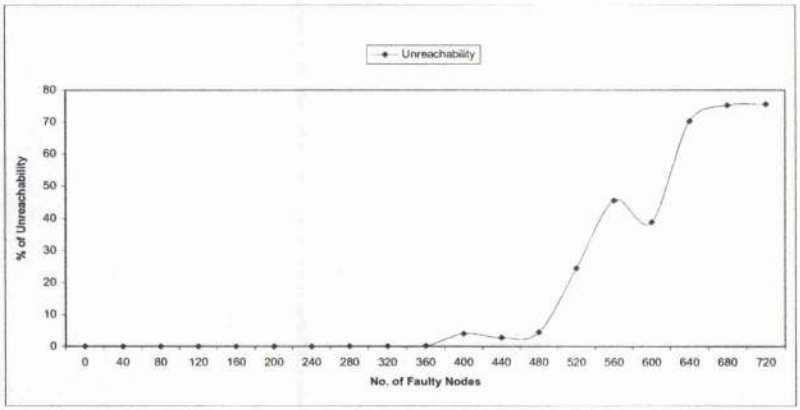
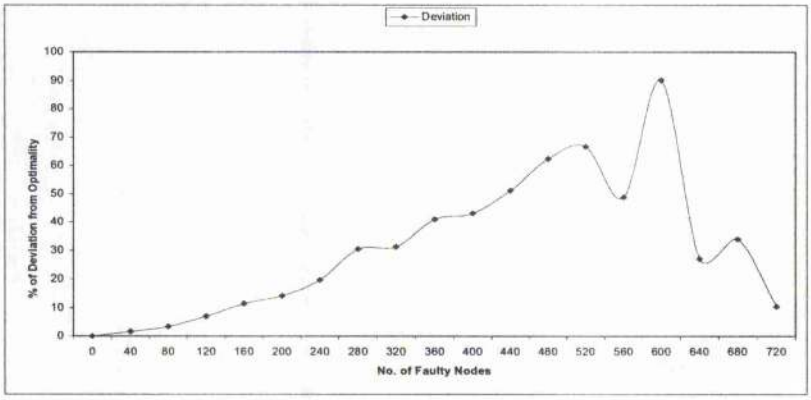


Fig 2.5: Average percentage of deviation, percentage of unreachability, and percentage of looping in the UV_Routing algorithm where n=3 and k value varying from 2 to 9.





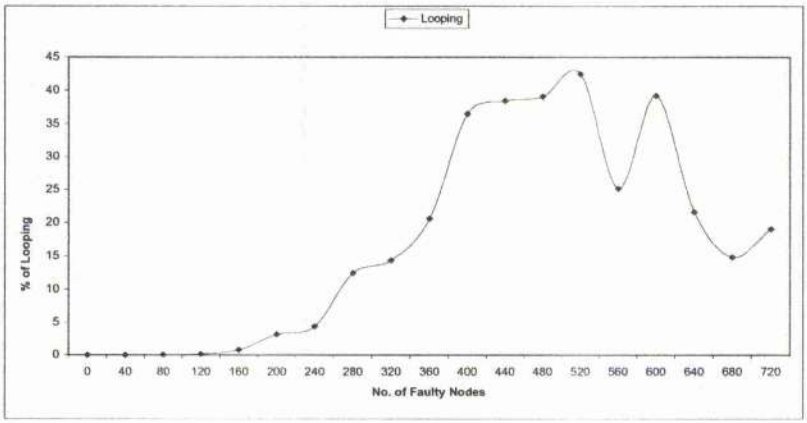


Fig 2.6: Average percentage of deviation, percentage of unreachability, and percentage of looping in the UV_Routing algorithm when m=3.

2.6 Conclusions

There have been a number of studies recently reported in the literature that have described fault-tolerant routing algorithms based on limited-global-information. Most of these algorithms, however, have been discussed in the context of the hypercube. Relatively little research has considered the other versions of the k-ary n-cube, such as the torus. This chapter has proposed a new fault-tolerant routing algorithm for high-radix k-ary n-cubes based on the concept of unsafety vectors. As a first step in this algorithm, each node A determines its view of the faulty set F_A of nodes which are either faulty or unreachable from A, by performing $(n\lfloor k/2 \rfloor - 1)$ exchanges of fault information with its reachable neighbours. Node A then calculates m unsafety sets denoted S_1^A, S_2^A,..., S_m^A where m is an adjustable parameter between 1 and $n\lfloor k/2 \rfloor$. The l-level unsafety set represents the set of all nodes at Lee distance l from A which are faulty or unreachable from A due to faulty links or nodes. Equipped with these unsafety sets each node calculates its unsafety vectors and uses them to achieve fault-tolerant routing in the k-ary n-cube. Larger values of m result in higher communication overhead with little improvement of routing performance. A performance analysis of the proposed algorithm has been conducted using software simulation. The results obtained have revealed that the new algorithm provides good performance in terms of the routing distance and percentage of unreachability even when the parameter m is at its lowest value of 1.

Chapter 3

A New Probability-Based Fault-Tolerant Routing for *k*-Ary *n*-Cubes

3.1 Introduction

We have introduced in the previous chapter the *Unsafety Vectors* [6, 7] as a limited-global-information-based fault-tolerant routing algorithm for k-ary n-cubes. This algorithm imposes no strict conditions on the number of faulty nodes or their locations. Each node A starts by determining the set of faulty or unreachable neighbours. Node A then performs $(n\lfloor k/2 \rfloor\text{-}1)$ exchanges with its neighbours to determine its faulty set containing all faulty or unreachable nodes at different distances from node A. For node A, the l-level unsafety set S_l^A, $1 \le l \le m$, where m is an adjustable parameter between 1 and $n\lfloor k/2 \rfloor$, represents the set of all nodes at a Lee distance l from A which are faulty or unreachable from A (see [14] for more details on Lee distances). Equipped with these unsafety sets, node A calculates numeric unsafety vectors and uses them to achieve efficient greedy fault-tolerant routing.

All fault-tolerant routing algorithms reported in the literature [6, 7, 18, 20, 81], including that proposed in Chapter 2, use a *deterministic* approach in that they use exact information about

faults in the network. We introduce in this chapter a new routing algorithm that achieves fault-tolerance in k-ary n-cubes using a new probabilistic approach [3].

The new algorithm uses the concept of "probability vectors" to considerably reduce the storage requirement for maintaining fault information, compared to the existing algorithms [64]. In the proposed algorithm, each node A starts by determining the set of faulty or unreachable neighbours. Then each node A calculates its probability vector $P^A = (P_1^A, ..., P_{n\lfloor k/2 \rfloor}^A)$. The l^{th} element, P_l^A, of the probability vector is an estimation of the probability that a destination node at distance l from A cannot be reached from A using a minimal path due to faulty nodes and links. An analysis through extensive simulation experiments is performed to assess the performance of the proposed algorithm. The results presented here reveal that the new algorithm performs near optimal routing for practical values of the numbers of faulty nodes. Moreover, the results reveal that the algorithm exhibits good performance levels in terms of the achieved routing distances and percentages of reachability even when a large number of faulty nodes exist.

The reader is referred to Section 2.2 in Chapter 2 for the notation and basic definitions used in the present chapter. The same assumptions listed at the end of section 2.2 in Chapter 2 are also used in this chapter. The remainder of the chapter is organised as follows. Section 3.2 presents the new fault-tolerant routing algorithm for the k-ary n-cube. Section 3.3 presents an analytical study of the proposed algorithm. Section 3.4 conducts a performance evaluation of the new algorithm through simulation experiments. Section 3.5 compares the performance of the proposed algorithm against that introduced in the previous chapter. Finally, Section 3.6 concludes this chapter.

3.2 The Proposed Probability-Based Fault-Tolerant Routing Algorithm

The proposed limited-global-information-based fault-tolerant routing algorithm uses the concept of probability vectors. The probability vector of a node A is denoted $P^A = (P_1^A, ..., P_{n\lfloor k/2 \rfloor}^A)$, where

P_l^A is an estimation of the probability that a destination node at Lee distance l cannot be reached from node A using a minimal path due to faulty nodes and links. To calculate its probability vector, node A starts by determining the faulty set F_A, which comprises those neighbouring nodes that are either faulty or unreachable from A due to faulty links. After determining F_A, node A then calculates its probability vector $P^A = (P_1^A, ..., P_{n\lfloor k/2 \rfloor}^A)$ through $n\lfloor k/2 \rfloor$-1 exchanges of information with neighbours (as described below). The probability vectors are used by all the nodes to perform efficient fault-tolerant routing in the network.

Definition 3.1: The faulty set F_A of a node A is defined as $F_A = \bigcup_{1 \le i \le n} f_A^i$, where f_A^i is given by

$$f_A^i = \left\{ B \in \left\{ A^{(i+)}, A^{(i-)} \right\} \middle| \, B \text{ is faulty or link}(A,B) \text{ is faulty} \right\} \tag{3.1}$$

To simplify our calculation of the probability vectors, we also assume that all the nodes at Lee distance l-1 from $A^{(i-)}$ are at Lee distance l from A. The implications of this assumption will be addressed later in this section. Notice that node A considers the other end of a faulty link as a faulty node. The l^{th} element P_l^A of P^A is an estimation of the probability that a destination at Lee distance l from A is not minimally reachable, i.e. reachable using a minimal path, from A. Since node A has $|F_A|$ faulty or unreachable immediate neighbours, and only one of the $2n$ edges incident from A constitutes a minimal path to a specific destination at Lee distance one, the first element of the probability vector, P_1^A, is given by

$$P_1^A = \frac{|F_A|}{2n} \tag{3.2}$$

For a destination at Lee distance l and Hamming distance h, let q be the number of preferred neighbours. When k is odd, $q = h$. However, when k is even q can be either $q = h$ or $q = 2h$. The latter case arise when the source and destination are diametrically opposite on a given dimension i, and routing to any of the two neighbours, $A^{(i+)}$ or $A^{(i-)}$, is considered as a preferred move on that dimension. When this is the case for all dimensions, the number of preferred

neighbours will be $q=2h$. Let $R_l^{A^{(i\pm)}}$ be the probability that a destination at Lee distance l from A is minimally reachable via its neighbour $A^{(i\pm)}$. Minimal reachability via $A^{(i\pm)}$ is only possible if $A^{(i\pm)}$ is a preferred neighbour. The probability for $A^{(i\pm)}$ to be a preferred neighbour is $q/2n$. Assuming that all the nodes which are at Lee distance l-1 from $A^{(i\pm)}$ are at Lee distance l from A then

$$R_l^{A^{(i\pm)}} = \begin{cases} 0 & \text{if node } A^{(i+)} \text{ is faulty} \\ \\ \frac{q}{2n}(1 - P_{l-1}^{A^{(i+)}}) & \text{otherwise} \end{cases} \tag{3.3}$$

Notice that $A^{(i\pm)}$ means $A^{(i+)}$ or $A^{(i-)}$. If every node at Lee distance l from A were reachable minimally via exactly one of its $2n$ neighbours, then the probability of reaching minimally a node at Lee distance l from A would have been $\sum_{i=1}^{n}(R_l^{A^{(i+)}} + R_l^{A^{(i-)}})$ since probabilities can be added when the events are disjoint. However, a destination node at Lee distance l from A can be reached minimally via q preferred neighbours (not only one). Adding these probabilities includes therefore a redundancy factor whose effect could be reduced by dividing this summation by q. Therefore, the probability of reaching minimally a destination at Lee distance l from A can be approximated as $\frac{1}{q}\sum_{i=1}^{n}(R_l^{A^{(i+)}} + R_l^{A^{(i-)}})$. Hence,

$$P_l^A = 1 - \frac{1}{q}\sum_{i=1}^{n}(R_l^{A^{(i+)}} + R_l^{A^{(i-)}})$$

$$= 1 - \frac{1}{q}\sum_{i=1}^{n}(\frac{q}{2n}(1 - P_{l-1}^{A^{(i+)}}) + \frac{q}{2n}(1 - P_{l-1}^{A^{(i-)}}))$$

$$P_l^A = 1 - \frac{1}{2n}\sum_{i=1}^{n}((1 - P_{l-1}^{A^{(i+)}}) + (1 - P_{l-1}^{A^{(i-)}})) \tag{3.4}$$

The resulting expression can be also intuitively interpreted as follows. The ability of a node A to reach minimally destinations at Lee distance l depends only on the ability of its neighbours

reaching minimally destinations at distance l-1. For instance, if each neighbour $A^{(i\pm)}$ of A can reach minimally all nodes at Lee distance l-1 then A can reach minimally all nodes at Lee distance l. On the other extreme, if for each neighbour $A^{(i\pm)}$ of A, $A^{(i\pm)}$ cannot reach minimally any node at Lee distance l-1 then A cannot reach minimally any node at Lee distance l. We therefore propose to approximate the probability of reaching minimally destinations at distance l from A by the average of node A's neighbours probabilities of reaching minimally destinations at distance l-1, i.e., $1-P_l^A = \frac{1}{2n}\sum_{i=1}^{n}((1-P_{l-1}^{A^{(i+)}})+(1-P_{l-1}^{A^{(i-)}}))$.

Algorithm Compute_Probability_Vector (A: node)

/ called by node A to determine its probability vector* $(P_1^A, P_2^A, ..., P_{n\lfloor k/2 \rfloor}^A)$ **/*

$P_1^A = \frac{|F_A|}{2n}$;

for $l := 2$ *to* $n\lfloor k/2 \rfloor$ *do*

{ *send* P_{l-1}^A *to all neighbours*

$R_l^{A^{(i\pm)}} = 0$;

for $i := 1$ *to* n *do* {

receive $P_{l-1}^{A^{(i+)}}$ *from* $A^{(i+)}$;

receive $P_{l-1}^{A^{(i-)}}$ *from* $A^{(i-)}$;

$R_l^{A^{(i\pm)}} = R_l^{A^{(i\pm)}} + (1-P_{l-1}^{A^{(i+)}}) + (1-P_{l-1}^{A^{(i-)}})$; }

$P_l^A = 1 - \frac{1}{2n} R_l^{A^{(i\pm)}}$;

}

End.

Fig. 3.1: The algorithm for calculating the probability vector in the k-ary n-cube.

The probability vector $(P_1^A, P_2^A, ..., P_{n\lfloor k/2 \rfloor}^A)$ is computed for each node A using the equations (3.2)-(3.4).). If a node A has a faulty neighbour $A^{(i\pm)}$, then A assumes the probability vector of $A^{(i\pm)}$ to be (1, 1,…, 1). The following algorithm implements this probability vector calculation.

Next, we analyse the effect of the assumption that a node at Lee distance l-1 from $A^{(i\pm)}$ is at Lee distance l from A by deriving an error margin caused by the approximations used in the calculation of the elements of the probability vector as a result of this assumption. A node D at Lee distance l-1 from $A^{(i\pm)}$ it is either at Lee distance l or l-2 from A. In fact if k is odd and D is diametrically opposite to $A^{(i\pm)}$ on dimension i, then D would be at Lee distance l-1 from A. This special case only occurs with probability $1/(k\text{-}1)$. Let S_1 be the number of nodes at Lee distance l-1 from $A^{(i\pm)}$ and at Lee distance l from A, and let S_2 be the number of nodes at Lee distance l-1 from $A^{(i\pm)}$ and at Lee distance l-2 or l-1 from A. So the error margin of assuming that the nodes at Lee distance l-1 from $A^{(i\pm)}$ are at Lee distance l from A is estimated as the ratio

$$e_r = \frac{S_2}{S_1 + S_2} \tag{3.5}$$

The number of nodes in S_1 corresponds to the number of ways we can distribute l-1 moves over the n dimensions in either direction except for dimension i where the moves must be in the same direction as the move from A to $A^{(i\pm)}$. The number of nodes in S_2 corresponds to the number of ways we can distribute l-1 moves over the n dimensions in either direction except for dimension i where the moves must be in the opposite direction of the move from A to $A^{(i\pm)}$ with at least on such move on the i^{th} dimension.

From this description it is clear that $S_2 < S_1$. Furthermore, the error ratio, $e_r = \frac{S_2}{S_1 + S_2}$, is equal to $\frac{1}{2n}$ for l=2. Therefore, $\frac{1}{2n} \le e_r \le \frac{1}{2}$ for $2 \le l \le n\left\lfloor \frac{k}{2} \right\rfloor$. This error is reduced by giving

preference to preferred neighbours in the selection of the next node, guaranteeing a decrease in the Lee distance to the destination, and thus reducing the effect of the error caused by the approximations used in the probability calculations.

Example 3.1: In a fault-free 3-ary 3-cube, all the nodes calculate the first element of their probability vectors. Since there are no faulty nodes then $P_1^A = 0$ for all the nodes. In the next stage, all nodes collect the first elements of the probability vector of their neighbours to calculate the 2nd element of their probability vectors using equation 3.4. Obviously, calculations at each stage depend on the calculations of the previous stage. In a given stage, all the nodes perform their own calculations simultaneously in the same stage. After completing the 3rd stage, $n\lfloor k/2 \rfloor = 3$, for the fault-free 3-ary 3-cube, the probability vector for any node A is (0, 0, 0) which means the probability of not minimally reaching a destination at any Lee distance form A is 0.

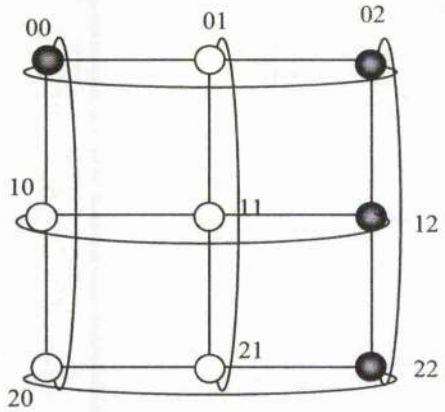


Fig. 3.2: An example of a 3-ary 2-cube with four faulty nodes.

Table 3.1: The faulty sets and probability vectors in a 3-ary 2-cube with 4 faulty nodes

Node A	00	01	02	10	11	12	20	21	22
F_A	Faulty	{00,02}	Faulty	{00,12}	{12}	Faulty	{00,22}	{22}	Faulty
P_1^A	Faulty	0.500	Faulty	0.500	0.250	Faulty	0.500	0.250	Faulty
P_2^A	Faulty	0.625	Faulty	0.688	0.563	Faulty	0.688	0.563	Faulty

Example 3.2: Consider now a 3-ary 2-cube with four faulty nodes, as shown in Fig. 3.2 (faulty nodes are represented as black nodes). Table 3.1 shows the corresponding faulty set and probability vectors associated with each node A according to the algorithm presented in Fig. 3.1.

Assume that the source node is 01 and we need to calculate the probability of reaching minimally a destination at Lee distance 2 from the source node. Let us try to compute the exact probability using a probabilistic argument. Node 01 has 2 fault-free neighbours: 11 and 21, and the probability of routing via any of them is ¼ as shown in Fig. 3.3. Node 11 has only one faulty neighbour and the probability of node 11 to reach minimally its own neighbours is ¾. However, notice that not all neighbours of 11 are at Lee distance 2 from the source node causing the error ratio as discussed earlier. Similarly, node 21 has a probability ¾ of reaching minimally its own neighbours. This means that the probability of node 01 reaching a destination at distance 2 via its neighbours is ¼(¾ + ¾)=0.375. Therefore, the probability of not reaching destination at distance two from the source node is 1-0.375=0.625. This result is the same as the value P_2^{01} given by the calculation of Fig 3.1 as shown in Table 3.1.

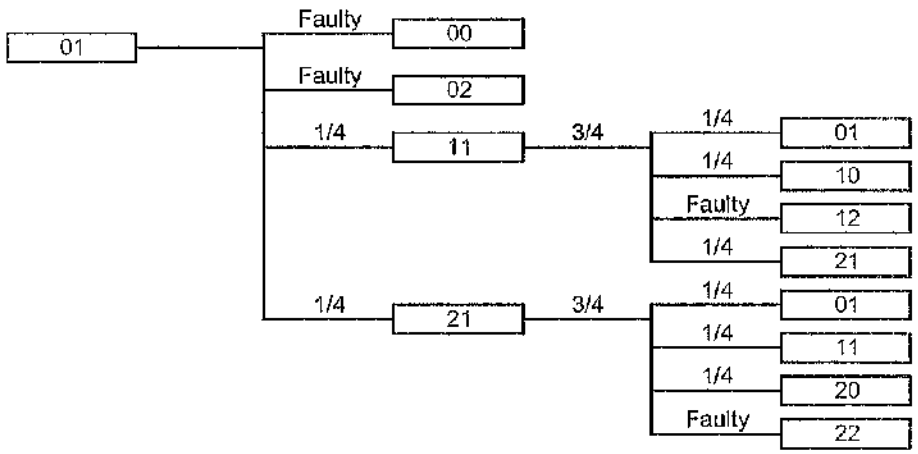


Fig. 3.3: Probabilities of reaching the nodes within distance two from the node 01.

Let us now study the accuracy of these approximate probability calculations. Notice that there are exactly 4 nodes at Lee distance 2 from the node 01, and these are: 12, 22, 20, and 10. Only 2 nodes, 20 and 10, of these 4 nodes are minimally reachable from 01. Therefore, the exact value of P_2^{01} should be 0.5. Our algorithm has estimated it to 0.625. The relative error in this case is $\frac{0.625-0.5}{0.5} = 0.25$ which is within the earlier claimed margin, $\frac{1}{2n} \le e_r \le \frac{1}{2}$.

Algorithm PV_Routing (M: message; A, S, D: node)

/ Called by node A to route the message M initiated at source node S towards its destination node D */*

if A=S then M.Route_distance = 0

if M.Route_distance $\le d_L(A, D) + (k\text{-}2) \times$ *no_faulty_nodes then*

{ M.Route_distance: =M.Route_distance + 1

if D is a reachable neighbour then deliver M to D; exit; / destination reached */*

l = Lee distance between A and D

Let $A^{(i\pm)}$ *be a reachable preferred neighbour with least* $P_{l-1}^{A^{(i\pm)}}$ *value;*

$P_r = l(1-P_{l-1}^{A(i\pm)}) + (l+1)P_{l-1}^{A(i\pm)}$*; /* least expected distance through* $A^{(i\pm)}$ **/*

Let $A^{(j\pm)}$ *be a reachable spare neighbour with least* $P_{l+1}^{A^{(j\pm)}}$ *value;*

If $w_j < \lfloor k/2 \rfloor$ *then* $Sp = (l+2)(1-P_{l+1}^{A(j\pm)}) + (l+3)P_{l+1}^{A(j\pm)}$

else $S_p = (l+1)(1-P_l^{A(j\pm)}) + (l+2)P_l^{A(j\pm)}$ */* least expected distance through* $A^{(j\pm)}$ **/*

if $\exists\ A^{(i\pm)}$ *and* (($\exists A^{(j\pm)}$ *and* $P_r \le S_p$) *or* ($\neg \exists A^{(j\pm)}$)) *then send M to* $A^{(i\pm)}$*;*

else if $\exists\ A^{(j\pm)}$ *and* (($\exists A^{(i\pm)}$ *and* $P_r > S_p$) *or* ($\neg \exists\ A^{(i\pm)}$)) *then send M to* $A^{(j\pm)}$*;*

else failure / destination unreachable */*

} else Detecting_Looping

End.

Fig. 3.4: Outline of the proposed "PV_Routing" fault-tolerant routing algorithm

When a node A has to route a message M towards its destination D, it applies the probability vectors-based routing algorithm, referred to here as "PV_Routing", outlined in Fig. 3.4. If node A can route through a preferred neighbour, $A^{(i\pm)}$, then the associated least expected routing distance is given by

$$P_r = l(1 - P_{l-1}^{A^{(i\pm)}}) + (l+1)P_{l-1}^{A^{(i\pm)}} \tag{3.6}$$

where $P_{l-1}^{A^{(i\pm)}}$ is the estimated probability of not minimally reaching a destination at distance l-1 from the preferred neighbour $A^{(i\pm)}$. This expression is justified by the fact that $(1 - P_{l-1}^{A^{(i\pm)}})$ is the estimated probability of existence of a fault-free minimal path via $A^{(i\pm)}$. If however such a path does not exist (with an estimated probability $P_{l-1}^{A^{(i\pm)}}$), then reaching the destination via $A^{(i\pm)}$ will require at least one more hop longer than the Lee distance l. Notice that in such a case if $A^{(i\pm)}$ and the destination are diametrically opposite on dimension i and k is odd then a path is at least one extra hop longer, otherwise a path is at least two extra hops longer. On the other hand, if node A can route through a spare or a disturb neighbour, $A^{(j\pm)}$, then the least expected routing distance is computed as (using similar arguments as for the calculation of P_r)

$$S_p = \begin{cases} (l+1)(1 - P_l^{A^{(j\pm)}}) + (l+2)P_l^{A^{(j\pm)}} & \text{if } w_j = \lfloor k/2 \rfloor \text{ and } k \text{ is odd} \\ (l+2)(1 - P_{l+1}^{A^{(j\pm)}}) + (l+3)P_{l+1}^{A^{(j\pm)}} & \text{otherwise} \end{cases}, \text{ where} \tag{3.7}$$

$$w_j = \min_{0 \le j < n}(|a_j - d_j|, k - |a_j - d_j|).$$

Notice that in this case each spare move may cost the path to be one hop longer than the Lee distance (via a neighbour $A^{(j\pm)}$ diametrically opposite to the destination on dimension j). Fig. 3.4 gives an outline of the proposed fault-tolerant routing algorithm PV_Routing used by node A. When node A has to forward a message M towards its destination D, the algorithm PV_Routing checks first if the destination is a reachable immediate neighbour in which case the message is delivered to destination. If not, PV_Routing tries to forward the message to a non-faulty

intermediate (preferred, disturb, or spare) neighbour that is associated with the least expected routing distance to the desired destination. PV_Routing selects the forwarding neighbouring node using the probability-based estimations of the least expected routing distance (equation 3.6, and 3.7). If an immediate non-faulty neighbour is not available then the destination is unreachable. Routing failure occurs in such cases.

Example 3.3: Consider a 3-ary 2-cube with four faulty nodes, as described in Fig. 3.2. To route a message from the source node 20 to the destination node 01, the PV_Routing algorithm checks first if node D is a reachable immediate neighbour to deliver the message directly to it. In our case where D is not an immediate neighbour, the algorithm tries to forward the message to a non-faulty intermediate (preferred, disturb, or spare) neighbour that is associated with the least expected routing distance to D. Node 20 has only 2 non-fault neighbours, 10 and 21, the first is a spare and the second is a preferred neighbour. The Lee distance, l, between S and D is 2. Using equations 3.6 and 3.7, we compute the least expected routing distance when routing through the non-faulty neighbours 21 and 10 as

$$P_r(21) = 2(1-0.25) + 3(0.25) = 2.25$$

$$S_p(10) = 3(1-0.688) + 4(0.688) = 3.688$$

Since $P_r(21) < S_p(10)$, the proposed routing algorithm will route to node 21 as an intermediate node, then routes directly to the destination node 01.

The proposed algorithm can detect and minimise the effect of looping. Notice from the above description of PV_Routing in Fig. 3.4 that looping is detected if the routing distance exceeds the specified limit (Lee distance plus $f(k\text{-}2)$ where f is the number of faulty nodes). Since each faulty node may cause a derouting and an increase in the routing distance by a value ranging between 1

and k-2, the maximum increase in the routing distance should not exceed $f(k\text{-}2)$. Since looping occurs when a destination is not reachable from the source node then the algorithm will discard such a message.

3.3 Performance Analysis

This section analyses some performance properties of the proposed PV_Routing algorithm in terms of the achieved minimum and average routing distances for various types of the k-ary n-cube. In the remainder of the chapter, we assume that there are f faulty nodes in the network, and that all the N nodes are equally likely to be faulty with a failure probability $p=f/N$. Furthermore, we assume that the source and destination nodes are non-faulty. In this section we only consider faulty nodes. Faulty link cases can be thought of as faulty node cases by considering the other end node of a faulty link as a faulty node. Let us now start by deriving a lower bound on the probability of minimum distance routing using the new algorithm.

3.3.1 A Lower Bound for the Probability of Minimum Distance Routing

For any two nodes at Hamming distance h, $h \geq 2$, and Lee distance l, $h \leq l \leq n\lfloor k/2 \rfloor$, the k-ary n-cube with $k \geq 5$ is known [24] to embed a family π of $2n$ node-disjoint paths of the following lengths:

h paths of length l, $h \leq l \leq n\lfloor k/2 \rfloor$,

$2n-2h$ paths of length l+2, and

h paths of length l+4

Assume there exists a "hypothetical" routing algorithm R that attempts to route along a non-faulty path from the family π of shortest possible length before considering other paths. The

following theorem provides a lower bound on the probability of minimum distance routing achieved by the algorithm R.

***Theorem 3.1*:** For any source A and destination D at Lee distance l, $h \le l \le n\lfloor k/2 \rfloor$, and Hamming distance h, $2 \le h \le n$, and $k \ge 2$ the routing algorithm R routes from A to D on a path of length at most $l+4$ with a probability at least $1 - P_l \cdot P_{l+2} \cdot P_{l+4}$, where

$$P_l = \left(1-(1-p)^l\right)^h \tag{3.8}$$

$$P_{l+2} = \left(1-(1-p)^{l+2}\right)^{2n-2h} \tag{3.9}$$

$$P_{l+4} = \left(1-(1-p)^{l+4}\right)^h \tag{3.10}$$

$$p = \frac{f}{k^n} = \frac{f}{N} \tag{3.11}$$

Proof: Let P_l, be the probability that all node-disjoint paths in π of length l are faulty. Such a path is faulty if at least one of its l nodes (other than the source node) is faulty. Each node is faulty with probability $p = f/N$ since there are f faulty nodes and all the N nodes in the network are equally likely to be faulty. Therefore a path of length l from π is faulty with probability $1-(1-p)^l$, and hence $P_l = [1-(1-p)^l]^h$. Similar analysis yields the expressions for P_{l+2} and P_{l+4}. Therefore at least one of the $2n$ paths of π is non faulty with probability $1 - P_l \cdot P_{l+2} \cdot P_{l+4}$. ∎

The PV_Routing algorithm attempts to route through a neighbour that has the highest probability of minimum distance routing. The algorithm keeps all options open and may select from any of the possible paths. As a result, it does not have any preference for a particular family of paths as does algorithm R in Theorem 3.1. It is therefore expected that PV_Routing will perform at least as good as R. In other words, the probability that PV_Routing routes from a source A to a

destination D at Lee distance l on a minimum distance path with at least the probability $1-P_l \cdot P_{l+2} \cdot P_{l+4}$.

Claim 3.1: PV_Routing routes in k-ary n-cubes with $k \geq 5$ between any given pair of nodes at Lee distance l, $h \leq l \leq n\lfloor k/2 \rfloor$, and Hamming distance h, $2 \leq h \leq n$, on a minimum length path with probability at least $1-P_l \cdot P_{l+2} \cdot P_{l+4}$.

This claim is verified experimentally by analysing the performance of the PV_Routing algorithm in order to measure the path lengths against the number of faulty nodes in the network. To this end, simulation experiments have been carried out over an 8-ary 3-cube with 512 nodes with different random distributions of faulty nodes. We started our experiments with a non-faulty k-ary n-cube and then the number of faulty nodes was increased gradually up to 70% of the network size with random fault distribution. Paths from every node to all destinations at Lee distance 6 and Hamming distance 3 (as an average Lee and Hamming distances) were selected. Fig. 3.5 shows the calculation probability of the minimal paths routing analytically and experimentally against the number of faulty nodes in the 8-ary 3-cube when the Lee distance is l=6 and the Hamming distance is h=3.

Other simulation experiments have been carried out over an 8-ary 3-cube with a fixed number of faulty nodes 153 (30% of the nodes) with different random distributions. A total of 30,000 source–destination pairs were randomly selected. Table 3.2 contains the probability of minimum distance routing calculated analytically from claim 3.1 and measured experimentally by the simulation for different Lee distances and Hamming distances.

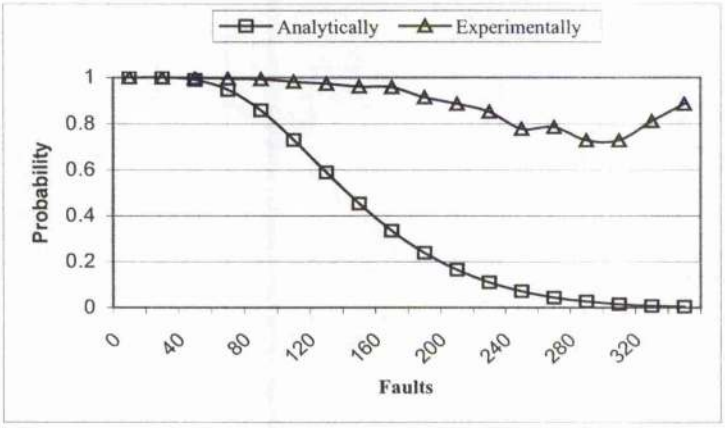


Fig. 3.5: Probability of minimum distance routing against the number of faulty nodes in the 8-ary 3-cube.

Table 3.2: Probability of minimum distance routing for a fixed number of faulty nodes (30% of the nodes) in the 8-ary 3-cube.

Lee Dist.	Hamming Dist.	Analytical Prob.	Experimental Prob.
2	2	0.885	0.985
3	2	0.751	0.957
3	3	0.783	0.985
4	2	0.604	0.937
4	3	0.636	0.964
5	2	0.467	0.934
5	3	0.495	0.950
6	2	0.351	0.931
6	3	0.373	0.945
7	2	0.258	0.936
7	3	0.275	0.944
8	2	0.187	0.936
8	3	0.200	0.943
9	3	0.144	0.943
10	3	0.103	0.937
11	3	0.073	0.928
12	3	0.052	0.918

Both Fig. 3.5 and Table 3.2 confirm that the probability of minimum distance routing for PV_Routing is always better than the corresponding probability for the hypothetical routing algorithm R. This shows that the probability that PV_Routing routes from a source A to a destination D at Lee distance l on a minimum length path is at least $1 - P_l \cdot P_{l+2} \cdot P_{l+4}$.

3.3.2 Average Routing Distance in the k-Ary 2-Cube (or 2-D Torus)

We restrict the discussion in this section to the class of the k-ary 2-cube (or 2-D torus). We will later extend our results to the 3-dimensional tori and then generalise them for the k-ary n-cube. In order to evaluate the average routing distance of PV_Routing, we define a general class of probabilistic routing algorithms. We then evaluate the average routing distance for these algorithms and use it as an approximation for the PV_Routing average routing distance.

We define a hypothetical class of probabilistic routing algorithms PRA as follows:

Definition 3.2: A routing algorithm is called a Probabilistic Routing Algorithm (or PRA for short) if the routing decisions are based on maximising the probabilities of minimum distance routing when selecting a node from the fault-free neighbours.

The following assumptions are also made to simplify the analysis for the PRA algorithms and to derive bounds on the performance of the PV_Routing algorithm:

i) In selecting the next move in PRA, the neighbours are considered in the following order: preferred on the first dimension, preferred on the second dimension, spare on the first dimension, and spare on the second dimension.

ii) After f spare routing moves, the message is discarded to avoid looping.

We now derive an expression for the average routing distance in the PRA algorithm. Since the *k*-ary *n*-cube has a symmetric network topology, we will focus, without loss of generality, our discussion on a particular source node, *A*. We will use the following notation during the derivation:

- $P_{l_1,l_2,s}$: probability of making exactly *s* spare moves when routing between the source node *A* and a destination with Lee distance components (l_1, l_2), where $l = l_1 + l_2$, and l_1 and l_2 are the distances across the first and second dimension, respectively.
- $\overline{D}_l$: average routing distance from the source node *A* to destinations at Lee distance *l*.
- $\overline{D}_{l_1,l_2}$: average routing distance from the source node *A* to destinations with Lee distance components (l_1, l_2).
- w_{l_1,l_2} : ratio of the number of nodes with Lee distance components (l_1, l_2) to the number of nodes at lee distance $l = l_1 + l_2$ from the source node *A*.
- N_l: the number of nodes at Lee distance *l* from the source node *A*.

Lemma 3.1: w_{l_1,l_2} is given by:

$$w_{l_1,l_2} = \begin{cases} \dfrac{1}{4\lfloor k/2 \rfloor - 2l + 2} & l > \lfloor k/2 \rfloor \text{ and } (l_1 = 0 \text{ or } l_2 = 0) \\ \dfrac{1}{2\lfloor k/2 \rfloor - l + 1} & l > \lfloor k/2 \rfloor \text{ and } (l_1 > 0 \text{ and } l_2 > 0) \\ \dfrac{1}{2l} & l \le \lfloor k/2 \rfloor \text{ and } (l_1 = 0 \text{ or } l_2 = 0) \\ \dfrac{1}{l} & l \le \lfloor k/2 \rfloor \text{ and } (l_1 > 0 \text{ and } l_2 > 0) \end{cases} \tag{3.12}$$

Proof: For a given l_1 and l_2, $0 \le l_1, l_2 \le \lfloor k/2 \rfloor$, and a fixed source node $A = (x_1, x_2)$ in the k-ary 2-cube, if $l_1 \neq 0$ and $l_2 \neq 0$ then there are four possible destination nodes with Lee distance components (l_1, l_2) from A. These are: (x_1+l_1, x_2+l_2), (x_1+l_1, x_2-l_2), (x_1-l_1, x_2+l_2), and (x_1-l_1, x_2-l_2). If however $l_1=0$ or $l_2=0$ then there are only two such possible destinations. Furthermore, the number of destinations N_l at a given Lee distance l from the source node A is given by

$$N_l = 4(2\lfloor k/2 \rfloor - l + 1) \qquad l > \lfloor k/2 \rfloor \tag{3.13}$$

This is derived from the fact that the range for each l_1 and l_2 should be $\lfloor k/2 \rfloor$, $\lfloor k/2 \rfloor$-1, $\lfloor k/2 \rfloor$-2, ..., l-$\lfloor k/2 \rfloor$. The number of values in this range is $2\lfloor k/2 \rfloor$-l+1. Furthermore there are four destinations with Lee distance components $(l_1, l-l_1)$ for each case. Using a similar reasoning yields the result $N_l = 4\,l$ for the case $l \le \lfloor k/2 \rfloor$. Hence the claimed result for w_{l_1,l_2} ∎

Lemma 3.2: The average routing distance, $\overline{D}_{l_1,l_2}$, from the source node A to destinations with Lee distance component (l_1, l_2) is given by

$$\overline{D}_{l_1,l_2} = \sum_{s=0}^{f} P_{l_1,l_2,s} \cdot (l_1 + l_2 + 2s), \text{ where} \tag{3.14}$$

$$P_{l_1,l_2,s} = (1-p)P_{l_1-1,l_2,s} + p(1-p)P_{l_1,l_2-1,s} + p^2(1-p)P_{l_1+1,l_2,s-1} + p^3(1-p)P_{l_1,l_2+1,s-1} \tag{3.15}$$

and the probability $P_{l_1,l_2,s}$ satisfies the following boundary conditions:

$$P_{l_1,l_2,s}=\begin{cases}1 & l_1=1,l_2=0,\ s=0\\ 1 & l_1=0,l_2=1,\ s=0\\ (1-p)^{l_1-1} & l_1>0,\ l_2=0,\ s=0\\ (1-p)^{l_2-1} & l_1=0,l_2>0,\ s=0\\ (1-p)P_{l_1-1,l_2,0}+p(1-p)P_{l_1,l_2-1,0} & l_1>0,l_2>0,s=0\\ 0 & l_1=1,\ l_2=0,\ s>0\\ 0 & l_1=0,l_2=1,s>0\\ (1-p)P_{l_1-1,0,s}+p(1-p)P_{l_1+1,0,s-1}+p^2(1-p)P_{l_1,1,s-1} & 1<l_1<\lfloor kl2\rfloor,l_2=0,\ s>0\\ (1-p)P_{l_1-1,0,s}+p(1-p)P_{l_1,1,s-1} & l_1=\lfloor kl2\rfloor,l_2=0,\ s>0\\ (1-p)P_{0,l_2-1,s}+p(1-p)P_{1,l_2,s-1}+p^2(1-p)P_{0,l_2+1,s-1} & l_1=0,1<l_2<\lfloor kl2\rfloor,\ s>0\\ (1-p)P_{0,l_2-1,s}+p(1-p)P_{1,l_2,s-1} & l_1=0,l_2=\lfloor kl2\rfloor,s>0\\ \left.\begin{array}{l}(1-p)P_{l_1-1,l_2,s}+p(1-p)P_{l_1,l_2-1,s}+p^2(1-p)P_{l_1+1,l_2,s-1}\\ +p^3(1-p)P_{l_1,l_2+1,s-1}\end{array}\right\} & 0<l_1<\lfloor kl2\rfloor,0<l2<\lfloor kl2\rfloor,s>0\\ (1-p)P_{l_1-1,l_2,s}+p(1-p)P_{l_1,l_2-1,s}+p^2(1-p)P_{l_1,l_2+1,s-1} & l_1=\lfloor kl2\rfloor,0<l_2<\lfloor kl2\rfloor,s>0\\ (1-p)P_{l_1-1,l_2,s}+p(1-p)P_{l_1,l_2-1,s}+p^2(1-p)P_{l_1+1,l_2,s-1} & l_2=\lfloor kl2\rfloor,0<l_1<\lfloor kl2\rfloor,s>0\\ (1-p)P_{l_1-1,l_2,s}+p(1-p)P_{l_1,l_2-1,s} & l_1=\lfloor kl2\rfloor,l_2=\lfloor kl2\rfloor,s>0\end{cases} \tag{3.16}$$

***Proof*:** Let $\overline{D}_{l_1,l_2}$ be the average routing distance between a given pair of nodes with Lee distance components (l_1,l_2). Since each spare move increases the routing distance by 2 hops, and since messages are discarded after making f spare moves, we can write $\overline{D}_{l_1,l_2}$ as

$$\overline{D}_{l_1,l_2}=\sum_{s=0}^{f}(l_1+l_2+2s)P_{l_1,l_2,s} \tag{3.17}$$

To make s spare moves when routing a message at Lee distance l from its destination, we distinguish the following cases based on the first move:

i) A preferred move on the first dimension leading to a node with a Lee distance component (l_1-1,l_2) from destination, and the remaining route must include s spare moves.

ii) A preferred move on the second dimension leading to a node with a Lee distance component $(l_1, l_2 - 1)$ from destination, and the remaining route must include s spare moves.

iii) A spare move on the first dimension leading to a node with a Lee distance component $(l_1 + 1, l_2)$ from destination, and the remaining route must include s-1 spare moves

iv) A spare move on the second dimension leading to a node with a Lee distance component $(l_1, l_2 + 1)$ from destination, and the remaining route must include s-1 spare moves.

It can be easily verified that $P_{1,0,0} = 1$, $P_{0,1,0} = 1$, $P_{1,0,s} = 0$ and $P_{0,1,s} = 0$ for all $s > 0$ since the source and destination nodes are both assumed to be non faulty. For $l = l_1 + l_2 \geq 2$, $P_{l_1,0,0}$ is the probability that a destination with Lee distance components $(l_1, 0)$ is minimally reachable. This probability is equal to $(1-p)^{l_1 - 1}$ as this requires all $l_1 - 1$ preferred intermediate nodes to be non faulty. Following similar arguments the probabilities $P_{0,l_2,0}$ and $P_{l_1,l_2,0}$ are obtained. For $0 < l_1 < \lfloor k/2 \rfloor$, $0 < l_2 < \lfloor k/2 \rfloor$, and $s > 0$, we therefore can write $P_{l_1,l_2,s}$ as

$$P_{l_1,l_2,s} = (1-p)P_{l_1-1,l_2,s} + p(1-p)P_{l_1,l_2-1,s} + p^2(1-p)P_{l_1+1,l_2,s-1} + p^3(1-p)P_{l_1,l_2+1,s-1} \qquad (3.18)$$

When the destination is at distance $\lfloor k/2 \rfloor$ on one dimension or both dimensions, then the first move can only be a preferred move on that dimension, and in this case

$$P_{l_1,l_2,s} = \begin{cases} (1-p)P_{l_1-1,l_2,s} + p(1-p)P_{l_1,l_2-1,s} + p^2(1-p)P_{l_1,l_2+1,s-1} & l_1 = \lfloor k/2 \rfloor \text{an } 0 < l_2 < \lfloor k/2 \rfloor \\ (1-p)P_{l_1-1,l_2,s} + p(1-p)P_{l_1,l_2-1,s} + p^2(1-p)P_{l_1+1,l_2,s-1} & 0 < l_1 < \lfloor kl2 \rfloor \text{ and } l_2 = \lfloor k/2 \rfloor \\ (1-p)P_{l_1-1,l_2,s} + p(1-p)P_{l_1,l_2-1,s} & l_1 = \lfloor kl2 \rfloor \text{and } l_2 = \lfloor k/2 \rfloor \end{cases} \qquad (3.19)$$

The results of Lemma 3.1 and Lemma 3.2 are used to obtain the following theorem.

Theorem 3.2: For any PRA algorithm, the average routing distance $\overline{D}_l$ between a given pair of nodes at Lee distance l in the k-ary 2-cube is given by

$$\overline{D}_l = \left\{ \begin{array}{ll} \sum_{l_1=0}^{l} \overline{D}_{l_1,l-l_1} \cdot w_{l_1,l-l_1} & \text{if } l \le \lfloor k/2 \rfloor \\ \sum_{l_1=l-\lfloor k/2 \rfloor}^{\lfloor k/2 \rfloor} \overline{D}_{l_1,l-l_1} \cdot w_{l_1,l-l_1} & \text{if } l > \lfloor k/2 \rfloor \end{array} \right\} \qquad (3.20)$$

Claim 3.2: The average routing distance between two nodes at Lee distance l in PV_Routing in k-ary 2-cubes is approximated by $\overline{D}_l$.

This claim is intuitively justified by the fact that PV_Routing and PRA algorithms are based on similar probabilistic nature, and therefore we expect them to perform similarly in terms of the achieved average routing distances.

To support this intuitive claim, we have compared the results obtained using the above-derived expressions against those obtained through simulation. We have first solved the equations related to w_{l_1,l_2}, $P_{l_1,l_2,s}$, $\overline{D}_{l_1,l_2}$, and $\overline{D}_l$ given by Lemma 3.1, Lemma 3.2, and Theorem 3.2. These calculations yield the average routing distance vector $\overline{D}_A = (\overline{D}_1, \overline{D}_2, ..., \overline{D}_{n\lfloor k/2 \rfloor})$. We have then performed experiments of the proposed PV_Routing algorithm to measure experimentally the corresponding average routing distances vector.

Table 3.3: The average routing distance between two nodes at Lee distance *l* for different numbers of faulty nodes in the 15-ary 2-cube using PV_Routing.

l	Number of faulty nodes in 15-ary 2-cube (for PV-Routing)												
	0	5	10	15	20	25	30	35	40	45	50	55	60
1	1	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
2	2	2.09	2.12	2.17	2.17	2.24	2.26	2.39	2.39	2.44	2.58	2.70	2.55
3	3	3.14	3.21	3.29	3.30	3.35	3.39	3.59	3.62	3.66	3.70	3.98	3.79
4	4	4.14	4.24	4.33	4.40	4.44	4.54	4.73	4.77	4.86	4.96	5.19	5.03
5	5	5.14	5.26	5.35	5.46	5.52	5.63	5.81	5.91	6.00	6.10	6.26	6.21
6	6	6.14	6.28	6.39	6.51	6.60	6.76	6.96	7.02	7.16	7.30	7.38	7.41
7	7	7.14	7.30	7.42	7.58	7.69	7.86	8.08	8.12	8.28	8.46	8.47	8.54
8	8	8.13	8.29	8.40	8.56	8.68	8.83	9.04	9.07	9.25	9.50	9.44	9.53
9	9	9.12	9.26	9.38	9.53	9.62	9.76	9.94	9.97	10.19	10.42	10.39	10.46
10	10	10.12	10.27	10.38	10.52	10.61	10.75	10.91	10.93	11.18	11.37	11.40	11.42
11	11	11.13	11.28	11.40	11.54	11.61	11.77	11.89	11.97	12.26	12.44	12.52	12.51
12	12	12.14	12.29	12.41	12.55	12.60	12.74	12.85	12.95	13.35	13.46	13.55	13.58
13	13	13.16	13.30	13.42	13.55	13.59	13.75	13.89	13.98	14.38	14.48	14.57	14.60
14	14	14.16	14.31	14.45	14.55	14.60	14.76	14.81	14.91	15.28	15.40	15.57	15.48

Table 3.4: The average routing distance between two nodes at Lee distance *l* for different number of faulty nodes in the 15-ary 2-cube using PRA (Eq 3.20).

l	Number of faulty nodes in 15-ary 2-cube (PRA Probability)												
	0	5	10	15	20	25	30	35	40	45	50	55	60
1	1	1	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
2	2	2.02	2.05	2.08	2.11	2.14	2.18	2.21	2.25	2.29	2.33	2.38	2.41
3	3	3.05	3.10	3.15	3.21	3.28	3.35	3.42	3.49	3.56	3.63	3.70	3.76
4	4	4.07	4.15	4.23	4.32	4.41	4.51	4.61	4.71	4.81	4.90	4.98	5.05
5	5	5.09	5.20	5.30	5.42	5.54	5.67	5.79	5.91	6.03	6.14	6.23	6.29
6	6	6.12	6.24	6.38	6.52	6.67	6.82	6.96	7.10	7.23	7.34	7.42	7.47
7	7	7.14	7.29	7.44	7.60	7.77	7.93	8.08	8.23	8.35	8.44	8.49	8.49
8	8	8.14	8.29	8.45	8.61	8.78	8.94	9.10	9.24	9.35	9.42	9.45	9.42
9	9	9.16	9.32	9.49	9.65	9.82	9.98	10.12	10.25	10.35	10.40	10.41	10.34
10	10	10.2	10.36	10.55	10.73	10.90	11.06	11.20	11.32	11.39	11.42	11.38	11.27
11	11	11.2	11.41	11.61	11.81	11.99	12.15	12.29	12.39	12.45	12.44	12.36	12.18
12	12	12.2	12.45	12.67	12.88	13.07	13.24	13.37	13.45	13.47	13.42	13.27	13.02
13	13	13.2	13.49	13.72	13.94	14.13	14.29	14.39	14.43	14.39	14.26	14.01	14.64
14	14	14.3	14.49	14.68	14.84	14.94	14.98	14.95	14.83	14.62	14.29	14.84	14.27

Tables 3.3 and 3.4 show results for the calculated average routing distances using PV_Routing and PRA (equation 3.20), respectively, in a 225-nodes 15-ary 2-cube where the faulty nodes are increased gradually up to 60 faulty randomly distributed nodes. All possible source-destination pairs have been generated and tested. The experimental results and the analytical results are in close agreement with those obtained using simulation, demonstrating the accuracy of our above analytical derivation. The results also show that PV_Routing can achieve as good a performance as any fault-tolerant routing algorithm, e.g. the PRA algorithm that achieves high ratios of minimal routing in the presence of faults in the network.

A second set of simulation experiments has been carried out for a k-ary 2-cube with a fixed number of faulty nodes (20% of the nodes) with different random distributions. All possible source–destination pairs in the network have been generated and tested. Table 3.5 contains both the calculated probability of the minimal path routing using PV_Routing and theorem 3.2 for different Lee distances. This table also supports the claim that the proposed PV_Routing and hypothetical PRA algorithms exhibit similar performance characteristics in terms of the achieved average routing distances.

3.3.3 The Average Routing Distance in the k-Ary 3-Cube (or 3-D Torus).

Using similar analysis to that introduced for Lemma 3.1 and Lemma 3.2, the average routing distance $\overline{D}_l$ for the PRA algorithm in the k-ary 3-cube can be expressed as

$$\overline{D}_l = \sum_{l_1=0}^{x}\sum_{l_2=0}^{x}\sum_{l_3=0}^{x} D_{l_1,l_2,l_3} \cdot w_{l_1,l_2,l_3}, \text{ where } l_1 + l_2 + l_3 = l \text{ and } x = \min(l, \lfloor k/2 \rfloor) \tag{3.21}$$

$$\overline{D}_{l_1,l_2,l_3} = \sum_{s=0}^{f} (l_1 + l_2 + l_3 + 2s) P_{l_1,l_2,l_3,s} \tag{3.22}$$

Table 3.5: The average routing distance using PV_Routing and PRA (Eq.3.20) for a fixed number of faulty nodes (20% of the nodes) in the *k*-ary 2-cube.

k	Lee Dist	PV_Routing	PRA
3	1	1	1
	2	2	2
5	1	1	1
	2	2.265	2.171
	3	3.315	3.179
	4	4.343	3.999
7	1	1	1
	2	2.184	2.278
	3	3.357	3.442
	4	4.312	4.463
	5	5.240	5.487
	6	6.287	6.184
9	1	1	1
	2	2.300	2.284
	3	3.465	3.530
	4	4.577	4.682
	5	5.528	5.701
	6	6.559	6.744
	7	7.600	7.768
	8	8.474	8.375
11	1	1	1
	2	2.329	2.289
	3	3.487	3.549
	4	4.643	4.780
	5	5.794	5.920
	6	6.815	6.934
	7	7.764	7.965
	8	8.727	9.022
	9	9.655	10.02
	10	10.603	10.502

k	Lee Dist	PV_Routing	PRA
13	1	1	1
	2	2.577	2.295
	3	3.891	3.562
	4	4.923	4.807
	5	6.152	6.023
	6	7.249	7.152
	7	8.176	8.158
	8	9.203	9.171
	9	10.366	10.225
	10	11.425	11.271
	11	12.442	12.226
	12	13.121	12.566
15	1	1	1
	2	2.379	2.293
	3	3.656	3.559
	4	4.817	4.805
	5	5.938	6.031
	6	7.103	7.231
	7	8.269	8.349
	8	9.139	9.348
	9	10.146	10.347
	10	11.120	11.392
	11	12.152	12.447
	12	13.142	13.473
	13	14.065	14.392
	14	14.920	14.618

$$P_{l_1,l_2,l_3,s} = (1-p)P_{l_1-1,l_2,l_3,s} + p(1-p)P_{l_1,l_2-1,l_3,s} + p^2(1-p)P_{l_1,l_2,l_3-1,s} + p^3(1-p)P_{l_1+1,l_2,l_3,s-1} + p^4(1-p)P_{l_1,l_2+1,l_3,s-1} + p^5(1-p)P_{l_1,l_2,l_3+1,s-1} \quad (3.23)$$

$$w_{l_1,l_2,l_3} = \begin{cases} \frac{2}{N_l} & l_1 > 0 \text{ or } l_2 > 0 \text{ or } l_3 > 0 \\ \frac{4}{N_l} & l_1 = 0 \text{ or } l_2 = 0 \text{ or } l_3 = 0 \\ \frac{8}{N_l} & l_1 > 0 \text{ and } l_2 > 0 \text{ and } l_3 > 0 \end{cases} \tag{3.24}$$

$$N_l = \begin{cases} \sum_{l_1=0}^{\lfloor k/2 \rfloor} \sum_{l_2=0}^{\lfloor k/2 \rfloor} \sum_{l_3=0}^{\lfloor k/2 \rfloor} 1 & l_1 + l_2 + l_3 = l \text{ and } l > \lfloor k/2 \rfloor \\ 12l - 6 + 8\sum_{i=0}^{l-2} i & l \le \lfloor k/2 \rfloor \end{cases} \tag{3.25}$$

Table 3.6 shows both PV_Routing results against PRA (equation 3.21) results for the average routing distances for different sizes of the k-ary 3-cube, where the number of faulty nodes is 20% of the total network size. The results demonstrate that the above derived expressions predict the average routing distance with a reasonable degree of accuracy.

Table 3.6: The average routing distance using PV_Routing and PRA (Eq.3.21) for a fixed number of faulty nodes (20% of the nodes) in the k-ary 3-cubes.

k	Lee Dist	PV_Routing	PRA
3	1	1	1
	2	2.024	2.043
	3	3.014	3.014
5	1	1	1
	2	2.277	2.203
	3	3.257	3.277
	4	4.263	4.233
	5	5.309	5.354
	6	6.279	6.347
7	1	1	1
	2	2.297	2.232
	3	3.477	3.408
	4	4.546	4.501
	5	5.539	5.573
	6	6.546	6.643
	7	7.522	7.693
	8	8.540	8.754
	9	9.513	9.726
9	1	1	1
	2	2.294	2.235
	3	3.394	3.426
	4	4.501	4.602
	5	5.565	5.709
	6	6.581	6.785
	7	7.594	7.875
	8	8.597	8.954
	9	9.601	10.012
	10	10.872	11.083
	11	11.912	12.159
	12	12.965	13.096

3.3.4 The Average Routing Distance for the *k*-Ary *n*-Cube (the general case).

We now compute the expected average routing distance in the *k*-ary *n*-cube. To reduce the complexity of the combinatorics involved, and thus simplify the analysis, we assume that the Lee distance l_i $(1 \le i \le n)$ across any of the n dimensions is equal $l_i = k/4$, which is the average routing distance along a dimension assuming that a message can be destined to any node in the network with equal probability. Assuming equal chances of encountering faulty nodes on each of the n dimensions, the number of spare moves on each dimension is at most f/n. Since each spare move causes two extra routing steps, the overall average routing distance can thus be written as

$$\overline{D} = \sum_{i=1}^{n} \sum_{s=0}^{f/n} (k/4 + 2s) P_{i,s} \tag{3.26}$$

where $P_{i,s}$ is the probability that s spare moves are made on dimension i. In order to evaluate $P_{i,s}$ we assume the routing algorithm attempts to make a preferred move on one of dimensions 1, 2, ..., n in this order. If that is not possible, the routing attempts to make a spare move on one of dimensions 1, 2, ..., n in this order. Consequently, a spare move is performed on dimension i if all n preferred moves are not possible (due to faulty nodes or links) and the i-1 spare moves on dimensions 1, 2,..., i -1 are not possible. Hence the probability of making one spare move on dimension i is $p^{n+(i-1)}(1-p)$. The probability of making s spare moves on dimension i is given by

$$P_{i,s} = \left(p^{n-i-1}(1-p)\right)^{s} \quad (1 \le i \le n) \tag{3.27}$$

The following Table 3.7 shows results using the above equations and simulation for estimating the average routing distance in the *k*-ary 3-cube for different sizes, where the number of faulty nodes is 10% of the network size. The results show that in each case PV_Routing and PRA (equation 3.26) average routing distances are in close agreement with each other.

Table 3.7: The PV_Routing and PRA (Eq.3.26) average routing distance for a fixed number of faulty nodes (10% of the nodes) in the k-ary 3-cubes.

k	PV_Routing $\overline{D}$	PRA
2	1.705	1.502
3	2.004	2.253
4	3.029	3.003
5	3.665	3.753
6	4.562	4.503
7	5.266	5.254
8	6.295	6.004
9	6.717	6.754
10	7.775	7.504

3.4 Experimental Performance Analysis

In this section, we first analyse the complexity of the calculations of the PV_Routing algorithm, and then analyse the performance of the algorithm using simulation.

The calculation of the probability vectors involves information exchanges between network nodes. This calculation is performed in $n\lfloor k/2 \rfloor$ phases. In each phase, each node concurrently sends at most $2n$ messages and receives at most $2n$ messages. Therefore the computation time complexity is $O(n^2k)$ and the total number of generated messages is $O(n^2k^{n+1})$. Notice that the computation time complexity of the routing algorithm in [64] is $O((nk^n)^2)$ and the total number of generated messages is $O((nk^n)^2k^n)$. Furthermore, the storage complexity in PV_Routing is $n\lfloor k/2 \rfloor$ real numbers at each node. On the other hand, the storage complexity of the algorithm proposed in [64] is in the order of k^n tuples at each node, each tuple contains a node address and a probability of successful routing of that node for a given source-destination pair. This excessive communication and computation cost effectively reduces routing performance. The PV_Routing

algorithm compares favourably with the algorithm of [64] with respect to both communication cost and storage cost.

This section also obtains experimentally three additional performance measures on the proposed PV_Routing algorithm, namely deviation from optimality, unreachability, and looping. To this end, simulation experiments have been carried out over a 3-ary 3-cube with 27 nodes with different random distributions of faulty nodes. We started our experiments with a non-faulty *k*-ary *n*-cube and then the number of faulty nodes was increased gradually up to 75% of the network size with random fault distribution. A total of 30,000 source-destination pairs were selected randomly at each run. In the first two sets of results reported below (in Figs. 3.6, and 3.7 respectively), However, before presenting the results, we recall the definitions of the following variables and performance measures (see Section 2.5):

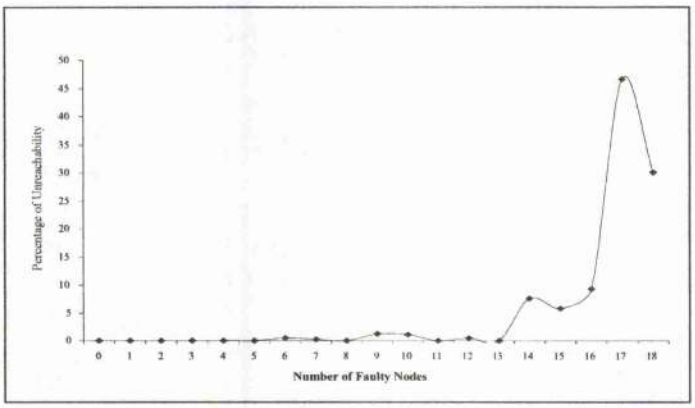
- *Total*: total number of generated messages
- *Routing_Distance*: number of links crossed by a message.
- *Lee_Distance*: Lee distance between the source and destination nodes.
- *Fail_Count*: number of routing failure cases.
- *Looping_Count*: number of messages that cross a number of links beyond a maximum threshold before being discarded.
- Average percentage of deviation from optimality

$$= \frac{1}{Total} \sum \frac{Routing\_Distance - Lee\_Distance}{Lee\_Distance} \times 100$$

- Percentage of unreachability $= \frac{Fail\_Count}{Total} \times 100$
- Percentage of looping $= \frac{Looping\_Count}{Total} \times 100$

Fig. 3.6 reveals that PV_Routing achieves a high reachability with low average percentage of deviation from optimality. The deviation from optimality remains low as long as this number of faulty nodes does not exceed 50% of the total number of nodes, then it grows almost linearly with the number of faulty nodes. The proposed algorithm is capable of routing messages using optimal distance paths even when there are a large number of faulty components. This is due to the fact that the algorithm repeatedly chooses to route through areas of the network with the least number of faults in the neighbourhood by choosing to route to a preferred neighbour with the least probability that a destination at distance l from A is not minimally reachable from A. As a result, the algorithm tends to select paths that diverge from areas with high counts of faulty components. The result also reveals that the percentage of looping remains practically negligible when the percentage of faulty nodes is less than 40%.

Another experiment was carried out to evaluate the performance behaviour of the new algorithm when the network size increases. For the sake of illustration, we have fixed the value of n to 3, and increased the value for k from 2 up to 9; we have found that the same conclusions are reached when other values of n are considered. For each network size, the algorithm has been tested by setting the percentage of faulty nodes to 10% of the network size, then to 20%, 30%, 40%, and 50% of the network size. At each run, a total of 30,000 source-destination pairs were selected randomly. The result presented in Fig. 3.7 shows that the performance properties of the PV_Routing are not affected as the network size is scaled up. This reveals that the proposed algorithm possesses the desirable property of maintaining good performance levels without imposing any restriction on the system size.



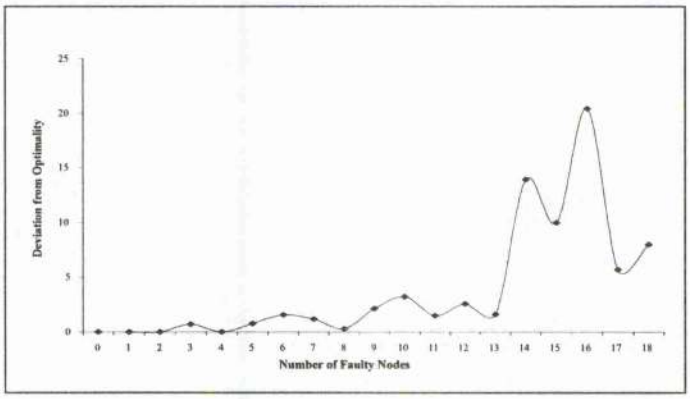
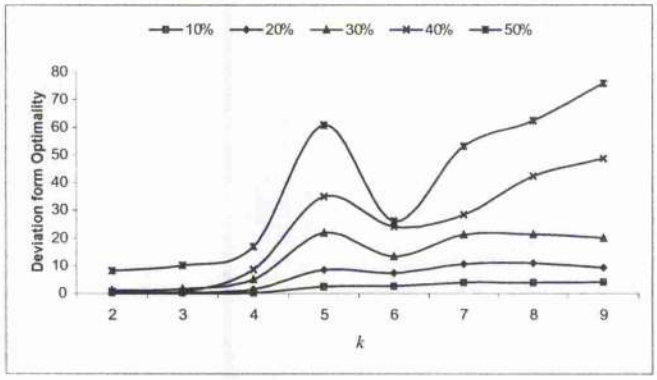
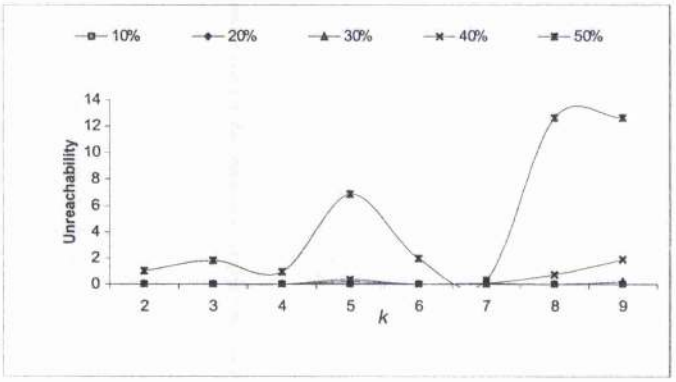


Fig. 3.6: Percentage of unreachability and average percentage of deviation in the proposed PV_Routing algorithm for 3-ary 3- cube.





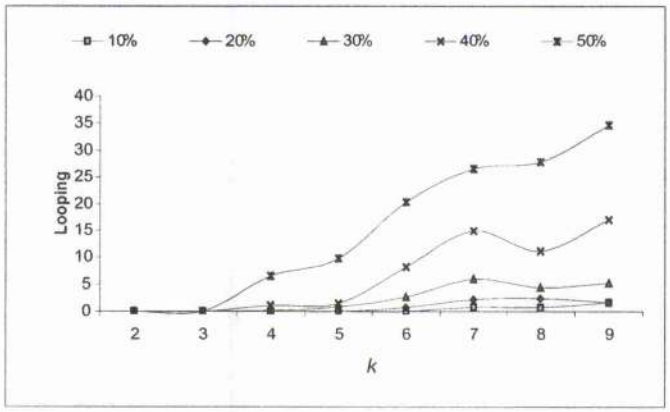


Fig. 3.7: Average percentage of deviation, percentage of unreachability, and percentage of looping in the proposed PV_Routing algorithm for different sizes of the *k*-ary *n*-cube.

3.5 Performance Comparison

This section compares the performance of our two algorithms proposed for the k-ary n-cube, namely PV_Routing and UV_Routing (proposed in Chapter 2), in terms of reachability, deviation from optimality, percentage of looping, average routing distance, communication complexity, and computation overhead.

3.5.1 Comparison of the Performance Merits

We have used the three performance measures defined in the previous chapter (average percentage of deviation from optimality, percentage of unreachability, and percentage of looping) as the basis for comparing the performance of PV_Routing and UV_Routing algorithms. Simulation experiments have been carried out over a 9-ary 3-cube with 729 nodes with different random distributions of faulty nodes. We started our experiments with a non-faulty k-ary n-cube and then the number of faulty nodes was increased gradually up to 50% of the network size with random fault distribution. A total of 50,000 source-destination pairs were selected randomly at each run.

Figs. 3.8, 3.9, and 3.10 reveal that both algorithms achieve a high reachability with low percentage of deviation from optimality. The deviation from optimality remains low as long as the number of faulty nodes does not exceed 25% of the total number of nodes. It then grows almost linearly with the number of faulty nodes. Both algorithms are capable of routing messages using optimal distance paths even when there are a large number of faulty components. This is due to the fact that both algorithms repeatedly choose to route through areas of the network with the least number of faults in the neighbourhood via attempting to maximize the chances of minimal distance routing. As a result, both algorithms tend to select paths that diverge from areas with high counts of faulty components. The results also reveal that the percentage of looping

remains practically negligible when the percentage of faulty nodes is less than 30% for both algorithms. In general, the PV_Routing and UV_Routing algorithms exhibit similar performance in terms of reachability and deviation from optimality.

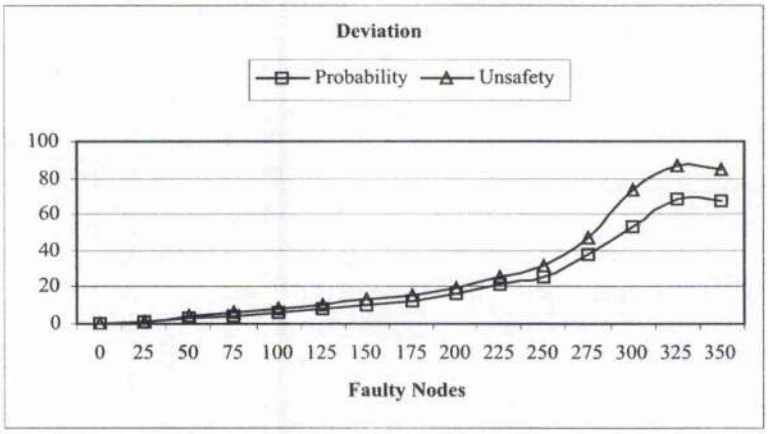


Fig. 3.8: Average percentage of deviation in the PV_Routing and UV_Routing algorithms in the 9-ary 3-cube.

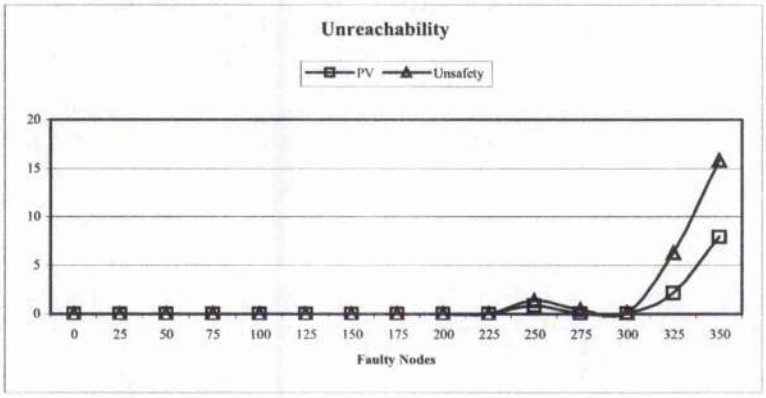


Fig. 3.9: Percentage of unreachability in the PV_Routing and UV_Routing algorithms in the 9-ary 3-cube.

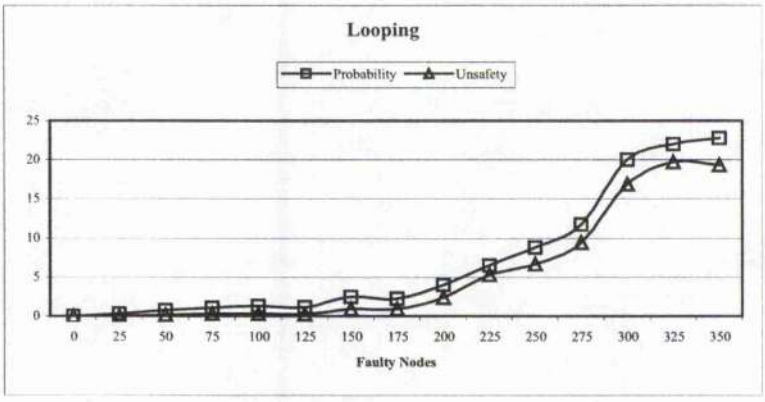


Fig. 3.10: Percentage of looping in the PV_Routing and UV_Routing algorithms in the 9-ary 3-cube.

Another set of performance experiments has been conducted to evaluate the behaviour of the PV_Routing and UV_Routing algorithms when the network size increases. For the sake of illustration, we have fixed the value of n to 3, and increased the value of k from 2 to 9 (for a network size varying from 8 to 729 nodes). For each network size, our algorithms have been tested by setting the percentage of faulty nodes to 10% of the network size, then to 20%, 30%, 40%, and 50%. At each run, a total of 30,000 source-destination pairs were selected randomly. The results presented in Figs. 3.11 through 3.16 show that the performance properties of the PV_Routing and UV_Routing are not affected as the network size is scaled up. This reveals that both algorithms possess the advantageous property of maintaining good performance levels without imposing impractical restrictions on the network size.

As stated above, the experimental performance comparison shows that both algorithms have similar performance in terms or reachability, deviation from optimality, and looping. However, the nature and amount of the calculations of the unsafety vectors are different from those of the probability vectors. This difference is reflected in the computational complexity and communicational overhead for the two algorithms, as will be discussed below.

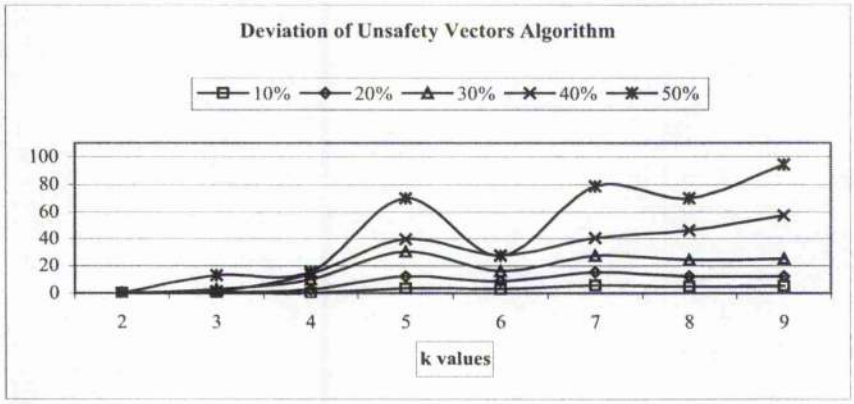


Fig. 3.11: Average percentage of deviation in the UV_Routing algorithm.

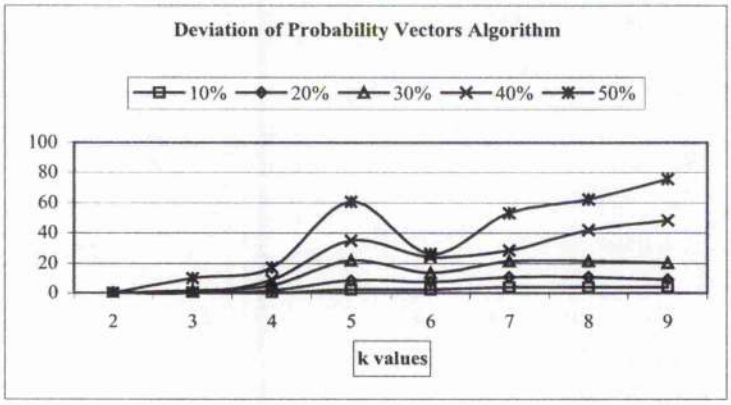


Fig. 3.12: Average percentage of deviation in the PV_Routing algorithm.

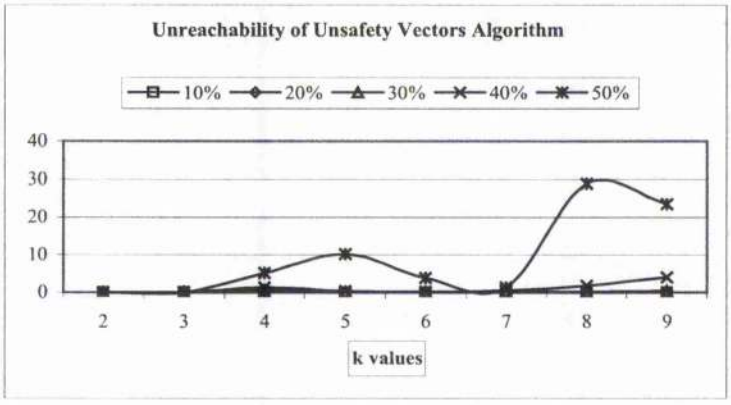


Fig. 3.13: Percentage of unreachability in the UV_Routing algorithm.

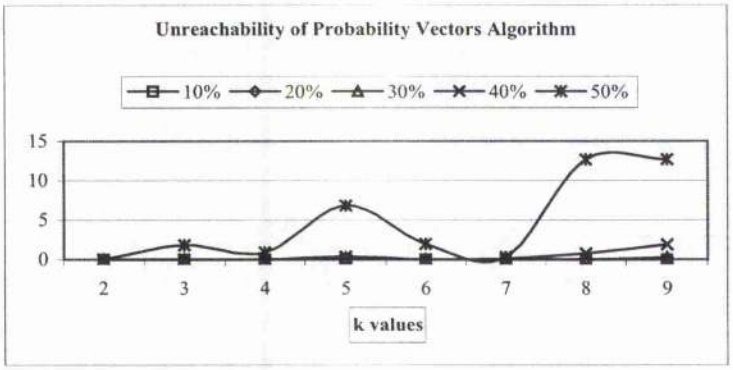


Fig. 3.14: Percentage of unreachability in the PV_Routing algorithm.

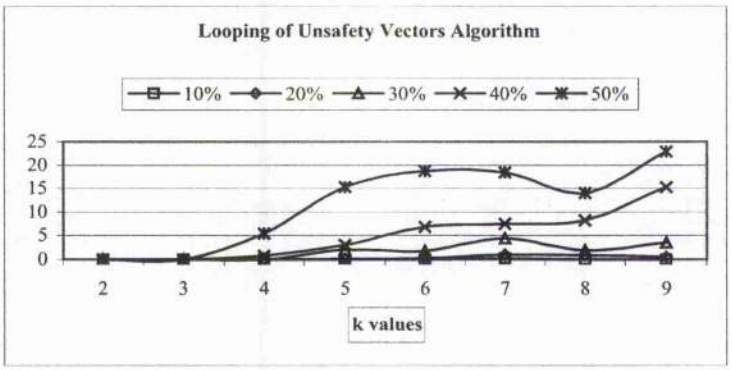


Fig. 3.15: Percentage of looping in the UV_Routing algorithm.

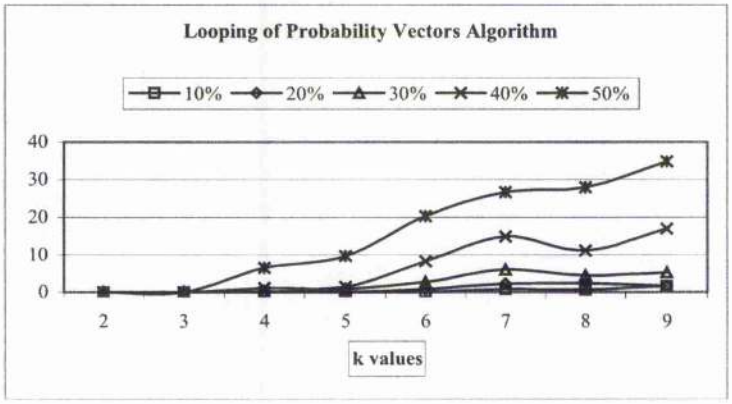


Fig. 3.16: Percentage of looping in the PV_Routing algorithm.

3.5.2 Comparison of the Average Routing Distance

In section 3.3 we have introduced a Probabilistic Routing Algorithm (PRA) model in order to evaluate the average routing distance for probability-based algorithms that satisfy the PRA conditions. Analytical results have been presented based on this model.

We now compare the analytical results against the experimental results using both UV_Routing and PV_Routing algorithms. We first solve the equations (3.21 – 3.24) related to w_{l_1,l_2,l_3}, $P_{l_1,l_2,l_3,s}$, $\overline{D}_{l_1,l_2,l_3}$, and $\overline{D}_l$ given by Lemma 3.1, Lemma 3.2, and Theorem 3.2. These calculations yield the average routing distance vector $\overline{D}_A = [\overline{D}_1, \overline{D}_2, ..., \overline{D}_n]$. We then simulate both PV_Routing and UV_Routing algorithms to measure the experimental average routing distance vector for both algorithms.

Table 3.8 shows both analytical results using PRA and experimental results using both PV_Routing and UV_Routing for different *k*-ary 3-cubes where the number of faulty nodes equals 10% of the nodes with random distribution for these faulty nodes. All possible source-destination pairs have been generated and tested. The experimental and analytical results for both algorithms are in close agreement. This reflects the good performance of both PV_Routing and UV_Routing in achieving high ratios of minimal routing in the presence of faults in the network.

It is worth noting that other experiments have been conducted for a varying the number of faulty nodes. The same conclusion has been reached from those experiments regarding the relative performance behaviour of both algorithms.

Table 3.8: Average routing distances using PV_Routing, UV_Routing, and PRA in *k*-ary 3-cubes.

k	PV_Routing $\overline{D}$	UV_Routing $\overline{D}$	PRA $\overline{D}$
2	1.705	1.718	1.502
3	2.004	2.092	2.253
4	3.029	3.062	3.003
5	3.665	3.756	3.753
6	4.562	4.663	4.503
7	5.266	5.564	5.254
8	6.295	6.307	6.004
9	6.717	7.016	6.754
10	7.775	7.889	7.504

3.5.3 Communication Complexity and Calculation Overheads

The performance of the UV_Routing and PV_Routing algorithms is primarily dependent on the overhead associated with the calculations of the unsafety and probability vectors, respectively, and which involve message-passing communication between network nodes. The UV_Routing algorithm performs in the order of $n\lfloor k/2 \rfloor$ phases. In each phase, each node sends at most $2n$ messages and receives at most $2n$ messages. Therefore the computation time complexity at each phase is $O(n^2k)$ and the total number of generated messages in the network is $O(n^2k^{n+1})$. Notice that the PV_Routing algorithm performs the same order of message exchanges with the difference being that probability vectors messages are substantially of smaller size. This cost is quite substantial (especially in terms of the total number of generated messages).

The PV_Routing algorithm requires from each node A to build a set F_A of all faulty immediate neighbouring nodes. The size of this set is, in the worst case, proportional to the number of faulty nodes in the network. Based on the global information set F_A, node A will then calculate a local

probability vector P_A of $n\lfloor k/2 \rfloor$ numeric components and use it for routing. This compares favourably with global-information-based routing algorithms which require in the order of k^n global information in the k-ary n-cube to be collected and used during decision making.

The excessive communication and computation cost for global-information-based algorithms does not justify the little gain in achieving optimal routing as compared to near optimal routing achieved by the UV_Routing algorithms and PV_Routing. On the other hand, the UV_Routing algorithm requires substantially higher initial computation and communication overhead in calculating the unsafety vectors as compared to the initial calculation of the probability vectors for the following reasons:

i) Unsafety sets are destination dependent while probability vectors are destination independent.

ii) Calculating the unsafety sets involves exchanging the sets F_A between neighbours while calculating the probability vectors involves exchanging single numeric values between neighbours.

3.6 Conclusions

K-ary n-cubes have been one of the most popular networks for multicomputers. This chapter has first introduced the concept of "probability vectors", and then used it to propose an efficient fault-tolerant routing algorithm for k-ary n-cubes. As a first step in the algorithm, each node A determines its view of the faulty set F_A of neighbouring nodes which are either faulty or unreachable from A. Equipped with these faulty sets, node A calculates its probability vector P^A by exchanging fault information with its reachable neighbours. An element P_l^A, $1 \le l \le n\lfloor k/2 \rfloor$,

of the vector is an estimation of the probability that a destination node at distance l cannot be reached from node A using a minimal path due to a faulty node or link along the path. Each node then uses the probability vectors to perform efficient fault-tolerant routing in the k-ary n-cube network.

An analytical study has been presented to derive upper bounds on the average message distance achieved by the proposed algorithm. A performance analysis of the proposed algorithm using simulation experiments has also been reported. The results have revealed the validity of the analytical model and have confirmed that the algorithm provides good performance in terms of the routing distance and percentage of reachability even when the number of faulty nodes in the network is large. The results have also revealed that the proposed algorithm maintains good performance levels as the network size scales up.

This chapter has also compared the performance of the probability vectors algorithm with the one proposed in the previous chapter. The results have shown that both algorithms exhibit similar performance with regards to the achieved average routing distance, reachability, and deviation from optimality. However, the probability vectors algorithm has the advantage of lower storage requirement and communication overhead.

Chapter 4

Adapting The Unsafety Vectors Algorithm to Hypercubes

4.1 Introduction

The hypercube (or the binary n-cube) has been one of the most popular networks for multicomputers due to its attractive topological properties, e.g. regularity, recursive structure, low diameter, and ability to exploit communication locality. Several commercial and experimental systems have employed this network, including the NCUBE-2 [54], iPSC/2 [58], Cosmic Cube [68], and SGI Origin 2000 multiprocessor [75]. There have been a number of attempts to design limited-global-information-based fault-tolerant algorithms for the hypercube, and the paragraph below briefly reviews some of the important algorithms that have been proposed in the literature [18, 46, 81, 83].

Chapter 2 has introduced the unsafety vectors as a new concept for designing a new fault-tolerant routing algorithm for k-ary n-cubes. The new routing algorithm has two important advantages over those already existing in the relevant literature, e.g. [18, 32, 64, 83]. Firstly, it can ensure fault-tolerance under more relaxed assumptions, regarding the number of faulty nodes and their

locations in the network. Secondly, the algorithm is more general in that it can easily be adapted to different topologies, including those that belong to the family of k-ary n-cubes (e.g. tori and hypercubes) and those that do not (e.g., generalised hypercubes and meshes).

As most previous studies on fault-tolerant routing have mainly focused on the hypercube [18, 46, 81, 83], the objective of the present chapter is to demonstrate how the concept of unsafety vectors can be adapted and applied to the hypercube for the design of efficient new fault-tolerant routing for this network. The resulting new algorithm is then compared against the safety vectors algorithm [81]. The reason we have selected the safety vectors algorithm in the present study is because besides being the most recent algorithm proposed in the literature, it has been shown to possess superior characteristics to existing similar algorithms [44, 46, 84].

In the unsafety vectors approach, each node A starts by determining the set of faulty or unreachable neighbours. Then, each node A performs (n-1) exchanges with its neighbours to determine its faulty set containing all faulty or unreachable nodes at different distances from node A. The unsafety sets are derived from the faulty sets according to the Hamming distance between the node and the elements of its faulty set. The k-level unsafety set S_k^A for all $1 \leq k \leq m$, where $1 \leq m \leq n$, represents the set of all nodes at Hamming distance k from A which are faulty or unreachable from A. Each node uses the unsafety sets to calculate numeric unsafety vectors to achieve fault-tolerant routing in the network. The chapter includes an analytical study proving some properties of the proposed algorithm. The performance of the proposed routing algorithm is compared against that of the safety vectors algorithm using simulation. The results demonstrate that the new unsafety vectors algorithm exhibits superior performance characteristics to the existing safety vectors algorithm.

The amount of the limited-global information used in the unsafety vectors is substantially smaller

(at most f addresses, where f is the number of faulty nodes which is typically a small fraction of 2^n) than the amount of information usually needed by global-information-based algorithms which is proportional to the number of nodes 2^n.

The simplicity and reduced size of the routing information result in faster routing decisions and reduce the amount of exchanged information. Global-information-based algorithms have the advantage of achieving optimal routing. However, our proposed limited-global-information-based algorithm achieves near optimal routing with a big reduction in the amount of exchanged routing information and in the complexity of the routing algorithm.

Before presenting the adaptation of the unsafety vectors algorithm for the hypercube, this chapter reviews some background information (preliminaries and notation) that will be useful for the subsequent sections. Then a description of the recently proposed safety vectors approach is presented [81].

4.2 Preliminaries and Notation

The n-dimensional hypercube, Q_n, is an undirected graph with 2^n vertices, representing nodes, which are labelled by the 2^n binary strings of length n. Two nodes are joined by an edge if, and only if, their labels differ in exactly one bit position. The label of node A is written $a_n a_{n-1} \ldots a_1$, where $a_i \in \{0, 1\}$ is the i^{th} bit (or bit at i^{th} dimension). The neighbour of a node A along the i^{th} dimension is denoted $A^{(i)}$. A faulty n-dimensional hypercube contains faulty nodes and/or links. The *Hamming distance* between a node A and a node B, denoted $H(A, B)$, is the number of bits at which their labels differ. In other words, $H(A, B) = |A \oplus B|$ where $\oplus$ denotes the *"exclusive or"* binary operation. A path between two nodes A and B is an *optimal path* if its length is equal to $H(A, B)$.

With respect to a given destination node, D, a neighbour $A^{(i)}$ of node A is called a *preferred neighbour* for the routing from A to D if the i^{th} bit of $A \oplus D$ is 1. We say in this case that i is a *preferred dimension*. Neighbours other than preferred neighbours are called *spare neighbours*. Routing through a spare neighbour increases the routing distance by two over the minimum distance. An optimal path can be obtained by routing through all preferred dimensions in some order. A node T is called an (A, D)- *preferred transit node* if any preferred dimension for the routing from A to T is also a preferred dimension for the routing from A to D.

Example 4.1: Suppose that $A = 1101$ and $D = 1010$. We have $A \ominus D = 0111$. Therefore, among the neighbours of A, nodes 1100, 1111, and 1001 are preferred neighbours and node 0101 is a spare neighbour. Nodes 1000, 1001,1011,1100,1110 and 1111 are preferred transit nodes for the routing from A to D.

We make the following assumptions for the purpose of our present study. These assumptions have also been used in similar previous studies [46, 81, 83, 84].

i) A faulty n-dimensional hypercube contains faulty nodes and/or links. The faults are distributed with equal probabilities across the network nodes and links.

ii) Each node is provided with the status of its own communication links and the status of its neighbouring nodes when routing vectors need to be calculated.

iii) If there is a faulty link between two nodes, then each of the two nodes considers the node at the other end as faulty.

iv) Lower software/hardware layers are responsible of detecting changes in the fault/recovery configuration and activation of the routing vector recalculation.

4.3 The Safety Vectors Approach

Wu [81] has presented a reliable communication scheme for hypercube-based multicomputers using the safety vector concept. In the safety vector approach, each node in the n-dimensional hypercube is associated with a safety vector, which can be considered as an approximated measure of the number and distribution of faults in the neighbourhood. As will be described below, an optimal routing between two nodes at a Hamming distance k is guaranteed if the k^{th} element of the associated safety vector is set to one.

Basically, each node in an n-dimensional hypercube is associated with a bit vector, called a safety vector, calculated through n - 1 rounds of information exchange among neighbouring nodes. In this approach, fault information is captured in a safety vector of n bit numbers, $(u_1, u_2, ..., u_n)$, associated with each node, u. Specifically, u_k represents the routing capability of node u to k-Hamming distance destinations. Based on the topological property of the hypercube, the k^{th} bit of a safety vector can be determined from the $(k-1)^{th}$ bit, $u_{k-1}^{(i)}$, of safety vectors of its neighbours if $k>1$, or directly if $k=1$. The safety vectors $(u_1, u_2, ..., u_n)$ is defined as follows:

- The first bit:

$$u_1 = \begin{cases} 0 & \text{if node } u \text{ is a faulty node or an end node of a faulty link} \\ 1 & \text{otherwise} \end{cases} \tag{4.1}$$

- The k^{th} bit, where $2 \le k \le n$:

$$u_k = \begin{cases} 0 & \text{if } \sum_{i=1}^{n} u_{k-1}^{(i)} \le n-k \\ 1 & \text{otherwise} \end{cases} \tag{4.2}$$

If the k^{th} bit of the safety vector of a node is one, then there exists at least one preferred neighbour that has 1 in the $(k - 1)^{th}$ bit of its safety vector. This neighbour is one step closer to the destination. Using this property inductively, a minimal path can be constructed to any destination which is k-Hamming distance away from a given node. Faulty nodes are assumed to be associated with safety vector (0, 0, ..., 0) which corresponds to the lowest order of safety degree. A node associated with (1, 1, ..., 1) as its safety vector has the highest order of safety degree and the corresponding node is called a *safe node*; otherwise, it is called an *unsafe node*.

4.3.1 Calculation of Safety Vectors

The GLOBAL_STATUS (GS) algorithm, described in Fig. 4.1, calculates the safety vector of each node in the network. Suppose that a source node intends to forward a message to a node k-Hamming distance away. The optimality is guaranteed if the k^{th} bit of its safety vector is 1 or one of its preferred neighbours' safety vector $(k - 1)^{th}$ bit is 1.

Routing starts by forwarding a message to a preferred neighbour with a 1 in the $(k - 1)^{th}$ bit of its safety vector. This node, in turn, forwards the message to one of its preferred neighbours which has 1 in the $(k - 2)^{th}$ bit of its safety vector, and so on. If there is no preferred neighbour that has 1 in the $(k - 1)^{th}$ bit of its safety vectors but there exists a spare neighbour which has 1 in the $(k+1)^{th}$ bit of its safety vector, the message is first forwarded to this neighbour and, then, the optimal routing algorithm is applied. In this case, the length of the path is the Hamming distance plus two, resulting in routing through a spare neighbour.

Algorithm GLOBAL_STATUS (GS)

/* determine safety vector $(u_1, u_2, ..., u_n)$ of node u in n-cube Q_n */

Begin

For all $u \in Q_n$ /* determine the first bit u_1 */

if u is an end node of a faulty link *then* $u_1 = 0$ *else* $u_1 = 1$;

for $k = 2$ *step* 1 *to* n /* determine the k^{th} bit u_k, where $2 \leq k \leq n$ */

for all $u \in Q_n$ /* $(k-1)^{th}$ bits, $u_{k-1}^{(i)}$,of neighbors' safety vectors */

$$\text{if } \sum_{i=1}^{n} u_{k-1}^{(i)} \leq n-k \text{ then } u_k = 0 \text{ else } u_k = 1$$

End.

Fig. 4.1: The algorithm for calculating the elements of the safety vectors for a given node in an *n*-dimensional hypercube.

4.3.2 The Routing Algorithm Using Safety Vectors

The routing process consists of two parts: UNICASTING_AT_SOURCE_NODE, outlined in Fig. 4.2, is applied at the source node to decide the type of routing process and to perform the first routing step. UNICASTING_AT_INTERMEDIATE_NODE, outlined in Fig. 4.3, is used at the intermediate nodes along the message path. After the first routing step, both OPTIMAL_UNICASTING and SUBOPTIMAL_UNICASTING, as described in Fig. 4.2, select a Hamming distance path to route the message to the destination node. Therefore there is no need to distinguish the type of routing process in UNICASTING_AT_INTERMEDIATE_NODE.

The main drawback of the safety vectors approach is that it is too pessimistic in deciding whether there is an optimal path between two nodes when there are failures in the neighbourhood. As will

be subsequently discussed, there are some situations where it is possible to find an optimal path between two nodes, but the safety vectors approach is not capable of locating them.

Algorithm UNICASTING_AT_SOURCE_NODE

Begin

$N = s \oplus d$; $H = |s \oplus d|$;

/ calculate navigation factor N and Hamming distance H */*

if $S_H = 1 \vee \exists i(S^{(i)}_{H-1} = 1 \wedge N^{(i)} = 1)$

*/ * the IIth bit of the safety vector is one or the (H - 1)th bit of the safety vector of a preferred neighbour is one */*

then OPTIMAL_UNICASTING:

send $(m, N^{(i)}$ *to* $s^{(i)})$, *where* $S^{(i)}_{H-1} = 1$ *and* $N^{(i)} = 1$

/ send message m to preferred neighbour* $s^{(i)}$, *where the (H - 1)th bit of its safety vector is one, together with N after resetting bit i */*

else if $\exists i(S^{(i)}_{H+1} = 1 \wedge N^{(i)} = 0)$

/ the (H + 1)th bit of a spare neighbour's safety vector is one */*

then SUBOPTIMAL_UNICASTING:

send $(m, N^{(i)}$ *to* $s^{(i)})$, *where* $S^{(i)}_{H+1} = 1$

/ send message m to spare neighbour* $s^{(i)}$, *where the (H + 1)th bit of its safety vector is one, together with N after resetting bit i */*

else failure

End.

Fig. 4.2: Routing at a source node using safety vectors.

Example 4.2: Consider a 4-dimensional hypercube with four faulty nodes as described in Fig. 4.4. A total of (n-1) rounds of information exchanges are performed to calculate the safety

vectors for all nodes in the network. Table 4.1 shows the formation of the safety vectors after each round of the safety vector calculation algorithm (described in Fig. 4.1).

Algorithm UNICASTING_AT_INTERMEDIATE_NODE
Begin
/* at any intermediate node u with message m and navigation vector N */
if $N = 0$ /* the navigation factor is empty */ then stop /* the current node is the destination node */
else send $(m, N^{(i)}$ to $u^{(i)})$, where $u^{(i)}_{H-1} = 1$ and $N^{(i)} = 1$
/ * send message m to preferred neighbour $u^{(i)}$, where the (H - 1)th bit is one, together with N after resetting bit i */
End.

Fig. 4.3: Routing at an intermediate node using safety vectors

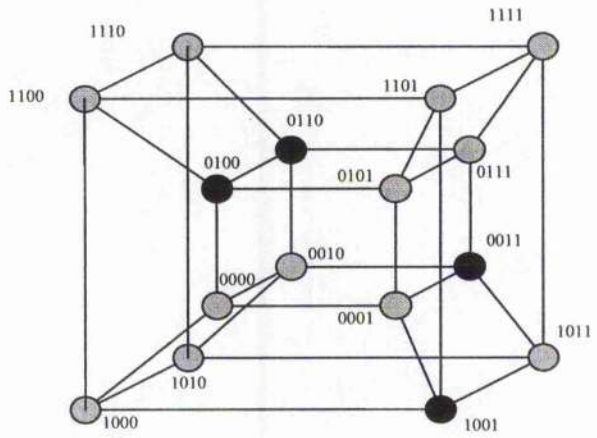


Fig. 4.4: A 4-dimensional hypercube with four faulty nodes (represented in dark colour).

Table 4.1: The safety vectors in a 4-dimensional hypercube with 5 faulty nodes.

(A) The initial safety vector assignments

Node	0000	0001	0010	0011	0100	0101	0110	0111
Safety Vector	{1,1,1,1}	{1,1,1,1}	{1,1,1,1}	{0,0,0,0}	{0,0,0,0}	{1,1,1,1}	{0,0,0,0}	{1,1,1,1}
Node	1000	1001	1010	1011	1100	1101	1110	1111
Safety Vector	{1,1,1,1}	{0,0,0,0}	{1,1,1,1}	{1,1,1,1}	{1,1,1,1}	{1,1,1,1}	{1,1,1,1}	{1,1,1,1}

(B) The safety vectors after the first round

Node	0000	0001	0010	0011	0100	0101	0110	0111
Safety Vector	{1,1,1,1}	{1,0,1,1}	{1,0,1,1}	{0,0,0,0}	{0,0,0,0}	{1,1,1,1}	{0,0,0,0}	{1,0,1,1}
Node	1000	1001	1010	1011	1100	1101	1110	1111
Safety Vector	{1,1,1,1}	{0,0,0,0}	{1,0,1,1}	{1,1,1,1}	{1,1,1,1}	{1,1,1,1}	{1,1,1,1}	{1,1,1,1}

(C) The safety vectors after the second round

Node	0000	0001	0010	0011	0100	0101	0110	0111
Safety Vector	{1,1,0,1}	{1,0,1,1}	{1,0,1,1}	{0,0,0,0}	{0,0,0,0}	{1,1,0,1}	{0,0,0,0}	{1,0,1,1}
Node	1000	1001	1010	1011	1100	1101	1110	1111
Safety Vector	{1,1,1,1}	{0,0,0,0}	{1,0,1,1}	{1,1,1,1}	{1,1,1,1}	{1,1,1,1}	{1,1,1,1}	{1,1,1,1}

(D) The safety vectors after the third (final) round

Node	0000	0001	0010	0011	0100	0101	0110	0111
Safety Vector	{1,1,0,1}	{1,0,1,0}	{1,0,1,1}	{0,0,0,0}	{0,0,0,0}	{1,1,0,1}	{0,0,0,0}	{1,0,1,1}
Node	1000	1001	1010	1011	1100	1101	1110	1111
Safety Vector	{1,1,1,1}	{0,0,0,0}	{1,0,1,1}	{1,1,1,1}	{1,1,1,1}	{1,1,1,1}	{1,1,1,1}	{1,1,1,1}

4.4 The Unsafety Vectors Fault-Tolerant Routing Algorithm

The adapted fault-tolerant routing algorithm, based on the concept of unsafety sets (defined below), presents a remedy for the major limitations of the safety vectors algorithm proposed for the hypercube [81]. These limitations are related to its conservative (pessimistic) routing approach and inapplicability to the network partitioning fault configurations. Before presenting the new algorithm let us first discuss how a node in the hypercube calculates its unsafety sets.

4.4.1. Calculation of Unsafety Sets

Definition 4.1: *The first-level unsafety set* S_1^A of a node A is defined as

$$S_1^A = \bigcup_{1 \le i \le n} f_A^i \text{, where } f_A^i \text{ is given by}$$

$$f_A^i = \begin{cases} \{A^{(i)}\} & \text{if } A^{(i)} \text{ is faulty} \\ \phi & \text{Otherwise} \end{cases}$$

S_1^A is the set of faulty or unreachable neighbours of A.

Definition 4.2: An *isolated node* is associated with first-level unsafety set containing n addresses of faulty nodes, i.e., $\left|S_1^A\right| = n$.

Definition 4.3: If for some node A, $\left|S_1^A\right| = n-1$, then node A is called a *dead-end node.*

Each node then determines the faulty set F_A, which comprises those nodes that are either faulty or unreachable from A due to faulty nodes or links. This is achieved by performing $(n-1)$ exchanges of the sets of detected faulty nodes with the reachable neighbours. After determining F_A, node A calculates m unsafety sets, $1 \le m \le n$, denoted S_1^A, S_2^A,..., S_m^A (defined below).

Definition 4.4: The k-level unsafety set $S_k^A, 1 \le k \le m$, for node A is given by

$$S_k^A = \{B \in F_A \mid H(A,B) = k\}$$

The k-level unsafety set S_k^A represents node A's view of the set of nodes at Hamming distance k from A which are faulty or unreachable from A due to faulty nodes and links. Notice that if the network is disconnected due to faulty nodes and links, A's view about unreachable nodes may not be accurate. In this case message looping will occur. We later present (see Section 4.4.3) a method for detecting and handling such looping. Fig. 4.6 provides an outline of the Find_Unsafety_Sets algorithm that node A uses to determine its faulty and unsafety sets.

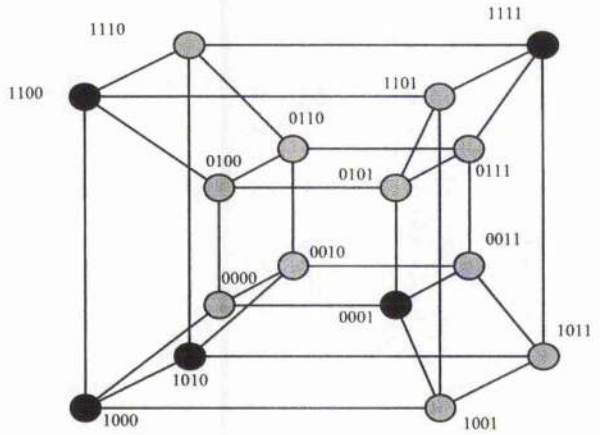


Fig. 4.5: A 4-dimensional hypercube with five faulty nodes (represented in dark colour).

Example 4.3: Consider a four-dimensional hypercube with five faulty nodes, as shown in Fig. 4.5 (faulty nodes are represented as black nodes). Table 4.2 shows the corresponding first-level unsafety set, S_1^A, associated with each node A. The Find_Unsafety_Sets algorithm calculates the sets S_k^A for all $1 \le k \le m$, $1 \le m \le n$, after calculating F_A. To achieve this, (n-1) exchanges of fault information are performed among neighbouring nodes.

Let $m=n$ and for the sake of a specific illustration we show how to compute the unsafety sets associated with node A=0000. First, the node assigns the addresses of its immediate faulty neighbours to its faulty set F_A. Then each node performs n-1 exchanges of the new elements of its faulty set F_A with the immediate non-faulty neighbours. After determining F_A, node A calculates m unsafety sets denoted S_1^A, S_2^A,..., S_m^A according to the Hamming distance between node A and each element of F_A. So, the faulty set for node A in our example, given in decimal representation, F_A ={1, 8, 10, 12, 15}, and the unsafety sets are S_1^A ={1, 8), S_2^A ={10, 12}, S_3^A ={}, and S_4^A ={15}.

```
Algorithm Find_Unsafety_Sets (A: node)
 /* called by node A to determine its faulty set and unsafety sets*/
F_A = set of faulty immediate neighbours;
 for k := 1 to n-1 do
   {
     for i:=1 to n do
         if  A^(i) ∉ F_A  then
         {
              send F_A  to  A^(i) ;
              receive F_A^(i)  from  A^(i) ;
              F_A = F_A  ∪ F_A^(i);
         }
   }
 for i := 1 to n do
       if link(A,  A^(i)) faulty then F_A = F_A  ∪ { A^(i) };
 for k := 1 to m do
   S_k^A = {B ∈ F_A | H(A,B) = k}
End.
```

Fig. 4.6: Faulty and unsafety sets calculation in the hypercube.

Table 4.2. The first level unsafety sets of nodes in a 4-dimensional hypercube with 5 faulty nodes.

Node	0000	0001	0010	0011	0100	0101	0110	0111
S_1^A	{1,8}	Faulty	{10}	{1}	{12}	{1}	{ }	{15}

Node	1000	1001	1010	1011	1100	1101	1110	1111
S_1^A	Faulty	{1,8}	Faulty	{10,15}	Faulty	{12,15}	{10,12,15}	Faulty

4.4.2. The Unsafety Vectors Routing Algorithm

Definition 4.5: For a given source-destination pair of nodes (A, D), we define the *(A, D)-unsafety vector* $U^{A,D} = (u_1^{A,D},...,u_k^{A,D},...,u_m^{A,D})$ where its k^{th} element is given by

$$u_k^{A,D} = |\{\ T \in\ S_k^A, \text{ such that } T \text{ is an } (A, D)\text{-preferred transit node}\}|.$$

In other words, $u_k^{A,D}$ is the number of faulty or unreachable (A, D)-preferred transit nodes at distance k from A. $u_k^{A,D}$ can be viewed as a measure of routing unsafety at distance k from A when routing to destination D, hence the name *unsafety vectors* for $U^{A,D}$. We also define an ordering relation '<' for numeric vectors as follows. For any two numeric vectors $U = (u_1, u_2,\ldots, u_m)$ and $V = (v_1, v_2, \ldots, v_m)$, $U < V$ iff $\exists\ i$, $1 \le i \le m$, such that $u_i < v_i$, and $u_j = v_j$ for all $j < i$. Fig. 4.7 shows the proposed fault-tolerant routing algorithm that each node in the network applies to route a message towards a destination node D. In the rest of the present chapter, we will refer to the new routing algorithm as the "unsafety vectors algorithm" to contrast it with the "safety vectors algorithm proposed by Wu [81].

```
Algorithm Unsafety Vectors (M: message; A, S, D: node)
/* called by node A to route message M initiated at source S towards its
destination node D */
if A =S then M.Route_distance = 0
if M.Route_distance<= H(A,D) + 2 × |F_A| then
{   M.Route_distance:=M.Route_distance + 1
    if A = D then exit;          /* destination reached */
    if ∃ a preferred non-faulty neighbour A^(i) such that
      u_j^(A^(i),D) ≤ j    ∀ j, 1≤ j ≤ H(A,D)−1    then send to A^(i)  /* Theorem 4.1*/
    Let A^(i) be the reachable preferred neighbour with least
          (A^(i),D)-unsafety vector U^(A^(i),D) and A^(i) is not dead-end
    if A^(i) exists then
       send M to A^(i)
    else
        {
        Let A^(J) be the reachable spare neighbour with least (A^(J),D)-unsafety
         vector U^(A^(J),D) and A^(J) is not dead-end;
        if A^(J) exists then
           send M to A^(J)
        else failure      /* destination unreachable */
        }
}
 else Handle looping /* will be discussed in section 4.4.3 */
End.
```

Fig. 4.7: A description of the unsafety vectors routing algorithm in the hypercube.

Example 4.4: Consider the hypercube depicted in Fig. 4.5. Consider the source node A=0010 and the destination node D=1101, and assume m=1. According to the unsafety vectors algorithm, the source node A will route a message to a preferred neighbour associated with the least number of preferred faulty nodes in its unsafety sets, which is node 0110. By performing the same operations the message will be routed to node 0100 then 0101 and finally to its destination 1101.

4.4.3. Handling of Message Looping

The unsafety vectors algorithm can be improved to minimise the effect of looping. Notice from the description of the unsafety vectors algorithm given in Fig. 4.7 that looping is detected if the routing distance exceeds the specified limit (Hamming distance plus $2f$ where f is the number of faulty nodes). Since each faulty node may cause a derouting and an increase in the routing distance by a value 2, the maximum increase in the routing distance should not exceed $2f$. Since looping occurs when a destination is not reachable from the source we can add the destination node to the faulty set of the node that detected the looping. When this occurs (n-1) exchanges of information between all neighbours are then initiated to propagate the new information among reachable nodes in the whole hypercube. Experimental simulations showed that the percentage of looping decreases significantly to less than 1%, regardless of the number of faulty nodes in the network, when we include this simple mechanism to handle message looping.

4.4.4. Properties of the Unsafety Vectors Algorithm

The new routing algorithm is capable of routing messages in a faulty network via fault-free minimal paths if they exist, otherwise the algorithm routes messages via fault-free near-minimal paths if they exist. The unsafety vectors algorithm always attempts to route messages along minimum distance paths between the source and destination nodes if they exist.

Theorem 4.1: *Given a non-faulty pair of source and destination nodes (A, D), if* $u_i^{A,D} \leq i$, for all $1 \leq i \leq H(A,D)-1$, *then there exists at least one minimal fault free path between the source A and the destination D.*

Proof: (By induction on $H(A, D)$). For $H(A, D)=2$, assume that $u_1^{A,D} \leq 1$. Let $A^{(i)}$ and $A^{(j)}$ be the two (A, D)-preferred transit nodes. Since $u_1^{A,D} \leq 1$ either $A^{(i)}$ or $A^{(j)}$ is non faulty (assume it is $A^{(i)}$). Therefore, the path $A \rightarrow A^{(i)} \rightarrow D$ is minimal and fault-free.

Let us assume that the property is satisfied for $H(A, D) \leq k$ for some $2 \leq k \leq n$. Now, consider the case $H(A, D) = k+1$. Since $U_k^{A,D} \leq k$ at least one of the $k+1$ (A, D)-preferred transit nodes at distance k from A is non faulty. Let B be such a node. Notice that the set of (A, B)-preferred transit nodes at any distance i from A, $1 \leq i \leq k$, is a subset of the set of (A, D)-preferred transit nodes at the same distance i from A. Therefore $U_i^{A,B} \leq U_i^{A,D} \leq i$.

By induction hypothesis, there exists a minimal fault free path from A to B. Hence the existence of a minimal fault-free path from A to D going through B. ∎

Corollary 4.1: *The unsafety vectors algorithm selects a preferred neighbour positioned on a minimal fault-free path if there exists at least one such a path between the source A and the destination D.*

Theorem 4.2: *Let* $A^{(i)}$ *and* $A^{(j)}$ *be two non faulty (A, D)-preferred neighbours of A. If all preferred neighbours of* $A^{(j)}$ *are faulty and at least one preferred neighbour of* $A^{(i)}$ *is non faulty then the unsafety vectors algorithm does not route messages of destination D via* $A^{(j)}$.

Proof: Since $u_1^{A^{(i)},D} < u_1^{A^{(j)},D}$ then $U^{A^{(i)},D} < U^{A^{(j)},D}$. Therefore, $U^{A^{(j)},D}$ is not the smallest vector and therefore $A^{(j)}$ is not selected as a forwarding node.

Example 4.5. Consider a four-dimensional hypercube with seven faulty nodes, as depicted in Fig. 4.8. Table 4.3 shows the unsafety set and the safety vector associated with each node. Consider a source A=1110 and a destination D=1001. The safety vectors algorithm fails to route a message between the pair (A, D) since the third bit (h=3) of the safety vector of the source node is not one and since none of the preferred neighbours has one at the second bit (h-1) of their safety vectors. Also none of the spare neighbours has one at the fourth bit (h+1) of their safety vectors. On the other hand, the unsafety vectors algorithm is capable of achieving an optimal route between the source and destination (A, D) in this case. The unsafety vectors algorithm will route the message to the intermediate node 1010 since it has the least number of preferred faulty nodes in its unsafety set, then to node 1000, and finally to the destination node 1001. While the safety vectors approach is not able to route at all (unreachable destinations), the following paths are achieved using the unsafety vectors approach

- (0011 → 0001 → 1001 → 1101)
- (1110 → 1010 → 1000 → 1001 → 0001)
- (1010 → 1110 → 1111 → 1101)
- (1111 → 1101 → 1001 → 0001 → 0011)

Table 4.3. The first level unsafety sets and the safety vectors of a 4-dimensional hypercube with 5 faulty nodes.

Node	0000	0001	0010	0011	0100	0101	0110	0111
S_1^A	Faulty	{0,5}	Faulty	{2,7,11}	{0,5,6,12}	Faulty	Faulty	Faulty
$S.V$	0,0,0,0	1,0,0,0	0,0,0,0	1,0,0,0	1,0,0,0	0,0,0,0	0,0,0,0	0,0,0,0

Node	1000	1001	1010	1011	1100	1101	1110	1111
S_1^A	{0,12}	{11}	{2,11}	Faulty	Faulty	{12,15}	{6,12}	{7,11}
$S.V$	1,0,0,0	1,1,0,0	1,0,0,0	0,0,0,0	0,0,0,0	1,0,0,0	1,0,0,0	1,0,0,0

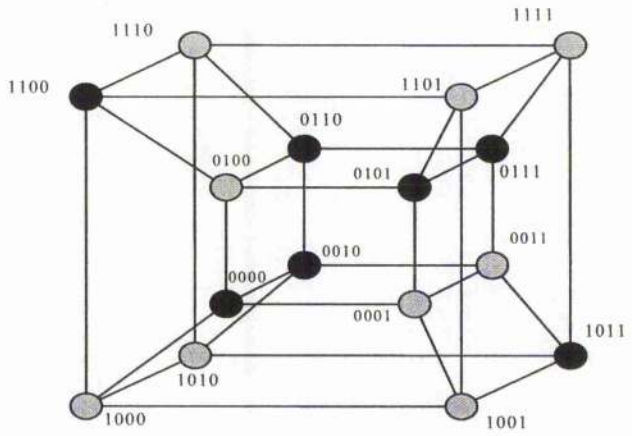


Fig. 4.8: A 4-dimensional hypercube with seven faulty nodes (represented in dark colour).

4.5 Performance Comparison

This section starts by analysing the complexity of the calculations of unsafety vectors phase, and then performs performance comparison against the safety vectors algorithm [81].

The performance of the unsafety vectors calculations is dependent on the performance the unsafety sets calculations algorithm, described in Fig. 4.6, which involves message-passing communications between the hypercube nodes. The unsafety sets calculations are performed in the order of n phases. In each phase, each node sends at most n messages and receives at most n messages. So, the computation time complexity is $O(n^2)$ and the total number of generated messages is $O(n^2 2^n)$.

Notice that the safety vectors algorithm performs the same order of message exchanges, with the difference being that our messages are relatively of larger size. This cost is quite substantial (especially in terms of the total number of generated messages). In real systems the frequency of

fault occurrence is rather low and this cost is incurred only when a fault occurs. Therefore this relatively high communication cost can be tolerated in practical situations.

Our algorithm requires from each node A to build an unsafety set of all reachable faulty nodes. The size of this set is in the worst case proportional to the number of faulty nodes in the network. Based on the unsafety set, node A will then calculate a local unsafety vector S^A of n components (numbers) and use it efficiently for routing. This compares favourably with global-information-based routing algorithms which require in the order of 2^n global information in the n-dimensional hypercube to be collected and used during decision making. This excessive communication and computation cost for global-information-based algorithms does not justify the little gain in achieving optimal routing as compared to near optimal routing achieved by limited-global-information-based algorithms as demonstrated in the simulation results of this section.

A simulation study of both the unsafety and safety algorithms has been carried out over hypercubes of different sizes. However, we report below the results for a 1024 node network only as the general conclusions have been found not to change much for other system sizes. We have considered in our experiments different random distributions of faulty nodes in the network; we started with a non-faulty hypercube in the first experiment, and then increased in each subsequent experiment the number of faulty nodes gradually up to 75% of the network size with random fault distribution. A total of 30,000 source-destination pairs were selected randomly in each simulation run.

As in the previous Chapters 2 and 3, we use the following three performance measures as the basis for the comparative analysis (see Chapter 2 for more details on the calculations of these measures)

- Percentage of unreachability $= \frac{Fail\_Count}{Total} \times 100$

- Average percentage of deviation from optimality

$$= \frac{1}{Total} \sum \frac{Routing\_Distance - Hamming\_Distance}{Hamming\_Distance} \times 100$$

- Percentage of looping $= \frac{Looping\_Count}{Total} \times 100$

In all the reported results, the parameter m has been set to its lowest value (m=1) in the unsafety vectors algorithm. As expected, our simulation experiments have confirmed that larger values of m greatly improve performance, but, of course, at the expense of increased communication overhead. Results in Fig. 4.9 reveal that even with the modest value of m=1 the unsafety vectors algorithm achieves much higher reachability than the safety vectors algorithm with low to moderate deviation from optimality, as depicted by Fig. 4.10. The figure also shows that the deviation from optimality becomes noticeable for the unsafety vectors algorithm only when the percentage of faulty nodes exceeds 50% of the total number of nodes in the network. Fig. 4.11 reveals that message looping in the unsafety vectors algorithm remains very low (practically zero) when the percentage of faulty nodes is less than 30%. From the three figures, we can conclude that the proposed algorithm exhibits superior performance characteristics over the existing safety vectors algorithm under realistic network working conditions.

The unsafety vectors algorithm is more capable of routing messages using optimal distance paths, especially for a large number of faulty components. Under high fault rates our algorithm is capable of routing a large percentage of messages for which the safety vectors algorithm announces a routing failure. This is due to the fact that the unsafety vectors algorithm repeatedly chooses to route through areas of the hypercube with the least number of faults in the neighbourhood, applying a greedy approach that gives more weight to the nearest neighbourhood.

The safety vectors algorithm, on the other hand, routes via a neighbour only if that neighbour guarantees optimal routing to all destinations at the desired distance, not just to the desired destination. Furthermore, owing to its inherent properties, the unsafety vectors algorithm tends to select paths that diverge from areas with high counts of faulty components.

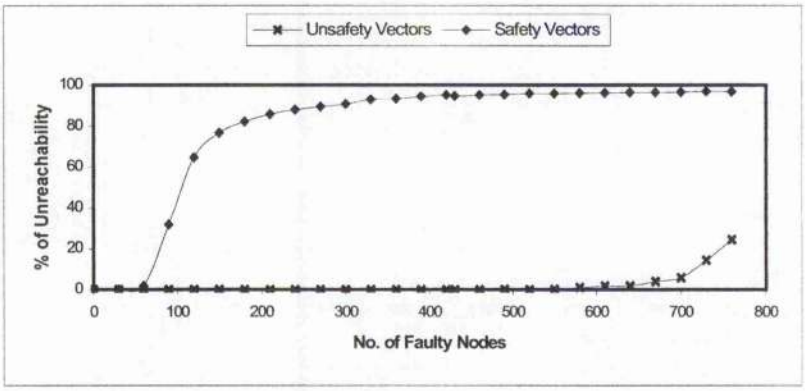


Fig. 4.9: Percentage of unreachability in the unsafety vectors and safety vectors algorithms.

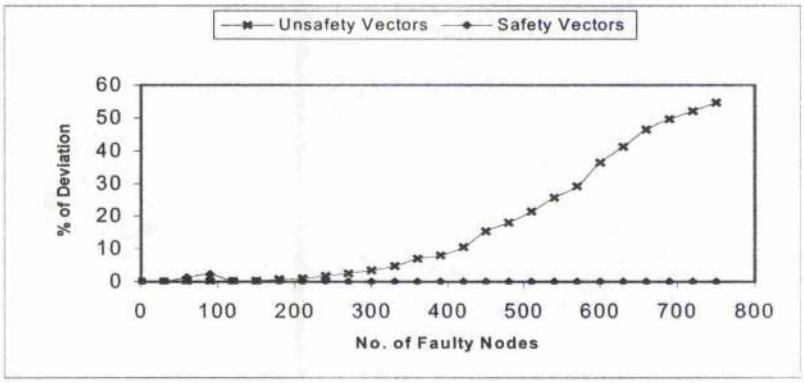


Fig. 4.10: Average percentage of deviation in the unsafety vectors and safety vectors algorithms.

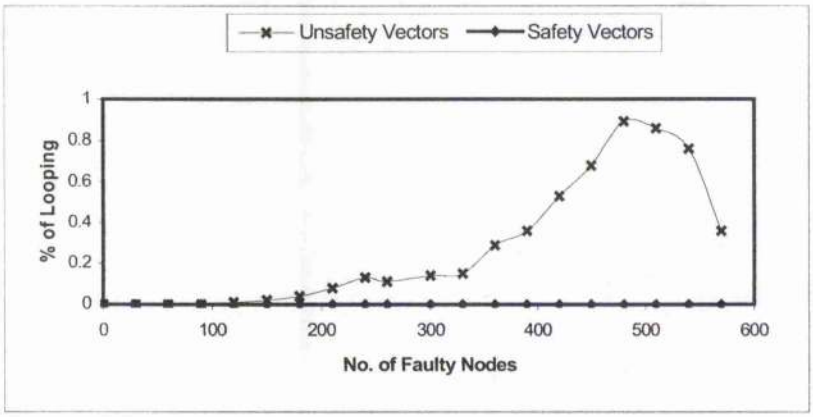


Fig. 4.11: Percentage of looping in the unsafety vectors and safety vectors algorithms.

4.6 Conclusions

This chapter has adapted the new fault-tolerant routing based on the concept of unsafety vectors for the hypercube in order to compare it with existing fault-tolerant routing algorithms. As a first step in this algorithm, each node A determines its view of the faulty set F_A of nodes that are either faulty or unreachable from A. This is achieved by performing (n-1) exchanges with the reachable neighbours. Node A then calculates m unsafety sets denoted S_1^A, S_2^A,..., S_m^A, where $1 \leq m \leq n$. The k-level unsafety set, S_k^A, represents the set of all nodes at Hamming distance k from A which are faulty or unreachable from A due to faulty links or nodes. Nodes use these unsafety sets to compute unsafety vectors and use them to achieve a fault-tolerant routing algorithm in the hypercube. A comparison between the unsafety vectors and safety vectors algorithm has been presented. Results for the achieved routing distance and percentage of reachability have revealed that the new algorithm outperforms the safety vectors algorithm even when the parameter m is set to 1, corresponding to the case where a node exchanges fault information with its neighbours only.

Chapter 5

Adapting the Probability Vectors Algorithm to Hypercubes

5.1. Introduction

In Chapters 2 and 3, we have introduced two new algorithms, based on the unsafety and probability vectors, respectively, for providing fault-tolerant routing in the k-ary n-cube. While the first algorithm uses a deterministic approach, which has been widely employed by existing algorithms in the past, the second algorithm is the first limited global information algorithm that uses a probabilistic approach to achieve fault-tolerance. The two algorithms have two important advantages over those existing in the literature. They both ensure fault-tolerance under more relaxed assumptions regarding the number of faulty nodes and their locations in the network. Moreover, they are more general in that they can easily be adapted to different topologies, including those that belong to the family of k-ary n-cubes (e.g. tori and hypercubes) and those that do not (e.g., generalised hypercubes [10]).

Since very little work has been carried out on providing fault-tolerant routing for the k-ary n-cube, Chapter 3 has compared the relative performance merits of the unsafety and probability vectors approaches on this network. Furthermore, since previous work has focused mostly on the hypercube, Chapter 4 has adapted the previous unsafety vectors approach to the hypercube in order to conduct a comparative study against existing routing algorithms, such as the safety vectors proposed in [81].

Motivated by the observation that most algorithms proposed for the hypercube, including the unsafety vectors [6, 7], use a *deterministic* approach, i.e. they use exact information about faults in the network, this chapter adapts the probability vectors approach to the hypercube, and evaluates its performance against the existing safety vectors algorithm. The rest of this chapter is organised as follows. Section 5.2 presents the adapted fault-tolerant algorithm and derives some of its properties. Section 5.3 presents an analysis of the probability-based fault-tolerant routing. Section 5.4 presents a comparative evaluation between the probability vectors and the safety vectors algorithms. Finally, Section 5.5 concludes this chapter.

5.2 The Adapted Probability Vectors Routing Algorithm

The same assumptions outlined in Section 4.2 are used for the purpose of the present study. To simplify our calculation of the probability vectors, we also assume that all the nodes at distance k-1 from $A^{(i)}$ are at distance k from A. The effect of this assumption will be addressed in the next section. In the adapted probability vectors algorithm, each node A determines its faulty set F_A of faulty or unreachable neighbours uses this faulty set to calculate its estimated probability vectors P_k^A, $(1 \le k \le n)$, and then to perform efficient routing in the hypercube based on these probability vectors.

5.2.1 Calculating the Faulty Sets

Definition 5.1: *The faulty set* F_A of a node A is defined as follows:

$$F_A = \bigcup_{1 \le i \le n} f_A^i \text{, where } f_A^i \text{ is given by}$$

$$f_A^i = \begin{cases} \{A^{(i)}\} & \text{if } A^{(i)} \text{ is faulty} \\ \phi & \text{Otherwise} \end{cases} \tag{5.1}$$

5.2.2 Calculating the Probability Vectors

After determining its faulty set, F_A, node A calculates its probability vector $P^A = (P_1^A, P_2^A, ..., P_n^A)$. The element P_k^A in this vector is an estimation of the probability that a destination at distance k from A is not minimally reachable from A. With respect to a source node A, a path is faulty if it includes at least one faulty or unreachable node. Since node A has $|F_A|$ faulty or unreachable immediate neighbours, and only one of the n edges (or links) incident from A constitutes a minimal path to a specific destination at distance one, we derive the probability P_1^A as:

$$P_1^A = \frac{|F_A|}{n} \tag{5.2}$$

In order to compute the other elements P_k^A, $k \geq 2$, let $R_k^{A^{(i)}}$ be the probability that a destination at distance k from A is minimally reachable via its neighbour $A^{(i)}$. Minimal reachability via $A^{(i)}$ is only possible if $A^{(i)}$ is a preferred neighbour. The probability for $A^{(i)}$ to be a preferred neighbour is k/n. If we assume that all the nodes which are at distance k-1 from $A^{(i)}$ are at distance k from A we can write $R_k^{A^{(i)}}$ as

$$R_k^{A^{(i)}} = \begin{cases} 0 & \text{if node } A^{(i)} \text{ is faulty} \\ \frac{k}{n}(1 - P_{k-1}^{A^{(i)}}) & \text{otherwise} \end{cases} \tag{5.3}$$

If a node at destination k from A were reachable minimally via exactly one of its n neighbours, then the probability of reaching minimally a destination at distance k from A would be given by $\sum_{i=1}^{n} R_k^{A^{(i)}}$ since probabilities can be added when the events are disjoint.

However, a destination at distance k from A can be reached minimally via k-preferred neighbours (not only one). Adding these probabilities includes therefore a redundancy factor whose effect could be reduced by dividing this summation by k. Therefore, the probability of reaching minimally a destination at distance k from A can be approximated by $\frac{1}{k}\sum_{i=1}^{n} R_k^{A^{(i)}}$.

Hence,

$$P_k^A = 1 - \frac{1}{k}\sum_{i=1}^{n} R_k^{A^{(i)}}$$

$$= 1 - \frac{1}{k}\sum_{i=1}^{n} \frac{k}{n}(1 - P_{k-1}^{A^{(i)}})$$

$$P_k^A = 1 - \frac{1}{n}\sum_{i=1}^{n}(1 - P_{k-1}^{A^{(i)}}) \qquad (5.4)$$

The resulting expression can be also intuitively interpreted as follows. The ability of a node A to reach minimally destinations at distance k depends only on the ability of its neighbours to reach minimally destinations at distance k-1. For instance, if each neighbour $A^{(i)}$ of A can reach minimally all nodes at distance k-1 then A can reach minimally all nodes at distance k. On the other extreme, if for each neighbour $A^{(i)}$ of A, $A^{(i)}$ cannot reach minimally any node at distance k-1 then A cannot reach minimally any node at distance k. We therefore propose to approximate the probability of reaching minimally destinations at distance k from A by the average probability of reaching minimally destinations at distance k-1 from the neighbours of node A, i.e. $1 - P_k^A = \frac{1}{n}\sum_{i=1}^{n}(1 - P_{k-1}^{A^{(i)}})$.

The probability vector $(P_1^A, P_2^A, ..., P_n^A)$ is computed for each node A using the equations (5.2)-(5.4). If a node A has a faulty neighbour $A^{(i)}$, then A assumes the probability vector of $A^{(i)}$ to be (1, 1,..., 1). The following algorithm implements this probability vector calculation.

Algorithm Compute_Probability_Vector (A: node)

/ called by node A to determine its probability vector* $(P_1^A, P_2^A, ..., P_n^A)$ **/*

$P_1^A = \frac{|F_A|}{n}$;

for $k := 2$ *to* n *do*

{ *send* P_{k-1}^A *to all neighbours*;

$R^A = 0$; */* summation of reachability via all neighbours of A */*

for $i := 1$ *to* n *do* {

receive $P_{k-1}^{A^{(i)}}$ *from* $A^{(i)}$;

$R^A = R^A + (1 - P_{k-1}^{A^{(i)}})$; }

$P_k^A = 1 - \frac{1}{n} R^A$;

}

End.

Fig. 5.1: The algorithm for calculating the probability vector in the hypercube.

An upper bound on the error caused by assuming that the nodes at distance k-1 from $A^{(i)}$ are at distance k from A can be estimated as the ratio of nodes at distance k-1 from $A^{(i)}$ but not at distance k from A, and is given by

$$\frac{\binom{n-1}{k-2}}{\binom{n}{k-1}} = \frac{k-1}{n} \qquad (5.5)$$

Notice that this error ratio increases as k increases. The impact of this error is reduced by giving preference to preferred neighbours in the selection of the next node guaranteeing a decrease of the distance to the destination and therefore reducing the effect of this estimated error.

Example 5.1: Let us consider the calculation of the probability vectors in a fault-free 4-dimensional hypercube. All the nodes calculate the first element of their probability vectors. Since there are no faulty nodes then $P_1^A = 0$ for all the nodes. In the next stage, all nodes collect the first elements of the probability vector of their neighbours to calculate the 2^{nd} element of their probability vectors using equation 5.4. Obviously, calculations at a given stage depend on the calculations of the previous stage. In each stage, all the nodes perform their own calculations simultaneously. After completing the 4^{th} stage in the fault-free 4-dimensional hypercube, the probability vector for any node A is (0, 0, 0, 0), i.e. the probability of not minimally reaching a destination at any distance form A is 0.

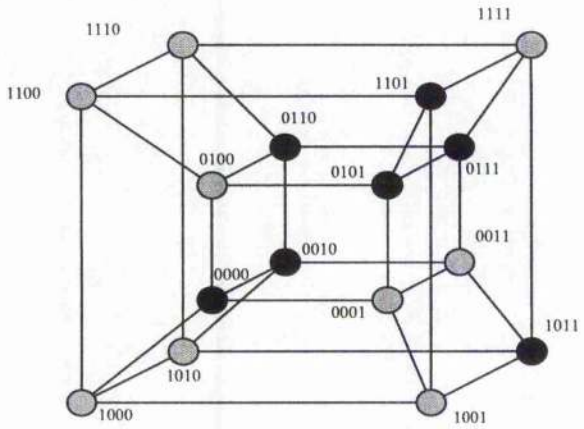


Fig. 5.2: A 4-dimensional hypercube with 7 faulty nodes.

Example 5.2: Consider now a 4-dimensional hypercube with seven faulty nodes (faulty nodes are indicated by dark colour), as shown in Fig.5.2. Table 5.1 shows the probability vectors associated

with each node calculated using the algorithm outlined in Fig 5.1.

Table 5.1: The probability vectors in a 4-dimensional hypercube with 7 faulty nodes.

Node A	0000 Faulty	0001	0010 Faulty	0011	0100	0101 Faulty	0110 Faulty	0111 Faulty
P_1^A	—	[0.50,	—	[0.75,	[0.75,	—	—	—
P_2^A	—	0.81,	—	0.88,	0.81,	—	—	—
P_3^A	—	0.89,	—	0.95,	0.89,	—	—	—
P_4^A	—	0.95]	—	0.97]	0.94]	—	—	—

Node A	1000	1001	1010	1011 Faulty	1100	1101 Faulty	1110	1111
P_1^A	[0.25,	[0.50,	[0.50,	—	[0.25,	—	[0.25,	[0.75,
P_2^A	0.56,	0.69,	0.63,	—	0.56,	—	0.63,	0.81,
P_3^A	0.72,	0.38,	0.80,	—	0.75,	—	0.75,	0.91,
P_4^A	0.85]	0.90]	0.87]	—	0.84]	—	0.86]	0.94]

Suppose that the source node is 0001. Let us compute the exact probability of reaching minimally a destination at distance 2 from node 0001 using a probabilistic argument. Node 0001 has 2 fault-free neighbours, 0011 and 1001. The probability of routing via any of them is ¼ as shown in Fig. 5.3. Node 0011 has only one fault-free neighbour and the probability of node 0011 reaching minimally its own neighbours is ¼. Notice that not all neighbours of 0011 are at distance 2 from the source node. Now, node 1001 has a probability ½ of reaching its own neighbours. As a result, the probability of node 0001 reaching minimally a destination at distance 2 via its neighbours is ¼ (¼ + ½)= 0.1875. Therefore, the probability that a destination at distance 2 from the source node is not minimally reachable = 1 - 0.1875 = 0.8125. This result is the same as the value P_2^{0001} shown in table 5.1 (and calculated using the algorithm of Fig 5.1).

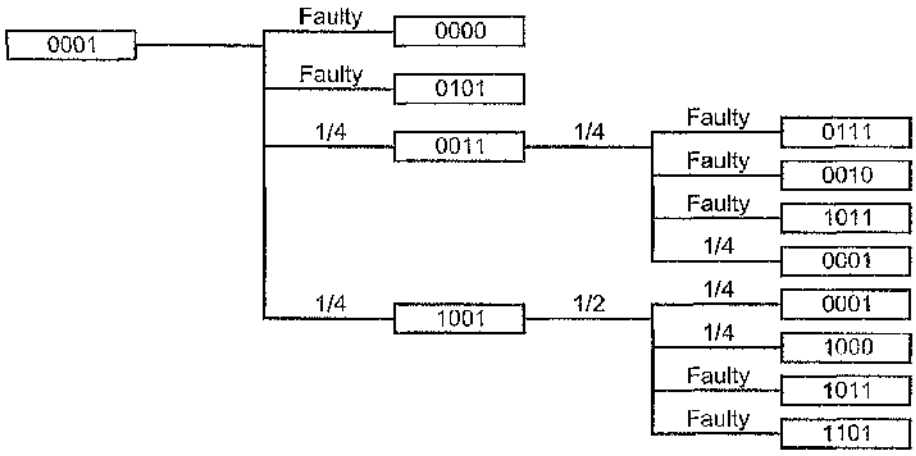


Fig. 5.3: Probability distribution of the nodes within distance two from the node 0001.

Let us now study the accuracy of these approximate probability calculations. Notice that there are exactly 6 nodes at distance 2 from the node 0001 which are {1000, 0100, 0010, 0111, 1011, and 1101}. Only the node 1000 of these 6 nodes is minimally reachable from 0001. Therefore, the exact value of P_2^{0001} should be 0.833. Our algorithm has estimated it to 0.8125. The relative error in this case is $\frac{0.833-0.8125}{0.833}=0.025$ which does not exceed the earlier derived error bound $\frac{k-1}{n}=0.25$.

5.2.3 A Probability-Based Fault-Tolerant Routing Algorithm

When a node A has to forward a message to its destination, it applies the probability-based routing outlined in Fig 5.4 in order to achieve fault-tolerance. We will refer to this as the "probability vectors" algorithm to contrast it with the safety vectors algorithm proposed by Wu [81]. The probability vectors algorithm checks first if the destination is a reachable immediate neighbour in which case the message is delivered directly to the destination. If not, the proposed

algorithm tries to forward the message to a non-faulty intermediate (preferred or spare) neighbour that is associated with the least expected routing distance to the desired destination. If the message is routed through a preferred neighbour, $A^{(i)}$, then the associated least expected routing distance is calculated as follows:

$$P_r = h(1 - P_{h-1}^{A(i)}) + (h+2)P_{h-1}^{A(i)} \qquad (5.6)$$

Algorithm Probability_Vectors (M: message; A, S, D: node)
/ called by node A to route message M initiated at source S towards its destination node D */*

if A =S then M.Route_distance = 0

if M.Route_distance<= H(A,D) + 2 × no_faulty_nodes then

{ M.Route_distance:=M.Route_distance + 1

if D is a reachable neighbour then deliver M to destination D; exit; / destination reached */*

h = Hamming distance between A and D

Let $A^{(i)}$ *be a reachable preferred neighbour with least* $P_{h-1}^{A(i)}$ *value;*

$P_r = h(1 - P_{h-1}^{A(i)}) + (h+2)P_{h-1}^{A(i)}$ *; /* least expected routing distance if we route through* $A^{(i)}$ **/*

Let $A^{(j)}$ *be a reachable spare neighbour with least* $P_{h+1}^{A(j)}$ *value;*

$S_p = (h+2)(1 - P_{h+1}^{A(j)}) + (h+4)P_{h+1}^{A(j)}$ *; /* least expected routing distance if we route through* $A^{(j)}$ **/*

if $\exists\ A^{(i)}$ *and ((*$\exists\ A^{(j)}$ *and* $P_r \le S_p$*) or (* $\neg \exists\ A^{(j)}$*)) then send M to* $A^{(i)}$*;*

else if $\exists\ A^{(j)}$ *and ((*$\exists\ A^{(i)}$ *and* $P_r > S_p$*) or (* $\neg \exists\ A^{(i)}$*)) then send M to* $A^{(j)}$*;*

else failure / unreachable destination */ }*

else Detect looping

End {Algorithm}

Fig. 5.4: Outline of the adapted probability vectors fault-tolerant routing algorithm in the hypercube.

where $P_{h-1}^{A^{(i)}}$ is the estimated probability of not minimally reaching a destination at distance h-1 from the preferred neighbour $A^{(i)}$. This expression is justified by the fact that $(1-P_{h-1}^{A^{(i)}})$ is the estimated probability of existence of a fault-free minimal path via $A^{(i)}$. If however such a path does not exist (with estimated probability $P_{h-1}^{A^{(i)}}$) then reaching the destination via $A^{(i)}$ will require the path to be at least two more hops longer than the Hamming distance (h). On the other hand, if the message is routed through a spare neighbour, $A^{(j)}$, then the least expected routing distance is calculated as follows (using similar arguments as for the calculation of P_r):

$$S_p = (h+2)(1-P_{h+1}^{A(j)}) + (h+4)P_{h+1}^{A(j)} \tag{5.7}$$

The probability vectors algorithm selects the forwarding neighbouring node with least expected routing distance using these probability-based estimations of the least expected routing distance. If an immediate non-faulty neighbour is not available then the destination is unreachable. Routing failure occurs in such cases.

Example 5.3: Consider a 4-dimensional hypercube with seven faulty nodes (faulty nodes are indicated by dark colour), as shown in Fig. 5.2. Table 5.1 shows the probability vectors associated with each node.

For instance, to route a message from node A=0001 to destination node D=1010, the probability vectors algorithm checks first if node D is a reachable immediate neighbour to deliver the message directly to it. In our case D is not an immediate neighbour, the algorithm tries to forward the message to a non-faulty intermediate (preferred or spare) neighbour that is associated with the least expected routing distance to D. Node A has 2 preferred non-faulty neighbours 1001 and 0011. It has no non-faulty spare neighbours. The Hamming distance, h, between S and D is 3. Using equation 5.6, we compute the expected routing distance when routing through the preferred non-faulty neighbours, 1001 and 0011, as:

$$P_r(1001) = 3(1-0.69) + 5(0.69) = 4.38$$

$$P_r(0011) = 3(1-0.88) + 5(0.88) = 4.76$$

The algorithm routes to node 1001 since it has the least expected routing distance. Then for the same reason, the message will be forwarded to node 1000, and finally to the destination node 1010. It is worth noting that the safety vectors approach [81] fails to route for this source-destination pair. The following theorem presents a sufficient condition for optimal routing using the proposed algorithm.

Theorem 5.1: If node A has a reachable preferred neighbour $A^{(i)}$ such that $P_{k-1}^{A^{(i)}} = 0$, $k \geq 2$, then the probability vectors algorithm will perform minimal distance routing to any destination at distance k.

Proof: (by induction on k) Let k =2 and let $A^{(i)}$ be a preferred neighbour satisfying $P_1^{A^{(i)}} = 0$. This implies that $F_{A^{(i)}} = \phi$ (from the definition of P_1^A). Therefore $A^{(i)}$ has no faulty or unreachable immediate neighbours. Furthermore, it is clear from the description of the probability vectors algorithm provided in Fig. 5.4 that in this case the message is forwarded either to $A^{(i)}$ or to a similar preferred neighbour. $A^{(i)}$ will then deliver the message to destination.

Assume that the claim is true for a distance $k \geq 2$. Consider the routing from a node A to a destination D at distance k+1. Assume there exists a preferred neighbour $A^{(i)}$ such that $P_k^{A^{(i)}} = 0$. The proposed routing algorithm will select this preferred neighbour or a similar one since the least expected routing distance, Pr, of such a neighbour is k which is the minimum Pr value. Since $A^{(i)}$ is at distance k, $k \geq 2$, from destination D, $A^{(i)}$ has at least one preferred neighbour $A^{(i,j)}$ (with respect to the same destination D).

Since $P_k^{A^{(i)}}$ is the probability that a destination at distance k from $A^{(i)}$ is not minimally reachable

and $P_k^{A^{(i)}} = 0$, this implies that all minimal paths of $A^{(i)}$ to a destination at distance k are non-faulty. Furthermore, any minimal path from $A^{(i,j)}$ to a destination at distance k-1 is a sub-path of a minimal path from $A^{(i)}$ to a destination at distance k. Hence all minimal paths from $A^{(i,j)}$ to a destination at distance k-1 are non-faulty (otherwise there would exist a faulty minimal path from $A^{(i)}$ to a destination at distance k). Therefore $P_{k-1}^{A^{(i,j)}} = 0$.

The induction hypothesis implies that the proposed routing algorithm will perform minimal routing from $A^{(i)}$ to a destination at distance k yielding in the overall a minimal routing from A to the destination D at distance $k+1$. ∎

5.3 Analysis of the Probability Vectors Fault-tolerant Routing Algorithm

In this section, we analyse the properties of the new routing algorithm. In particular, we derive analytical expressions that predict the average routing distance in this algorithm. In the remainder of the section, we assume that there are f faulty nodes in the network, and that all the n nodes are equally likely to be faulty with failure probability p. Furthermore, we assume that the source and destination nodes are non-faulty. In this section we consider only faulty nodes. Faulty link cases can be thought of as faulty node cases by considering the other end node of a faulty link as a faulty node. Let us first define a "hypothetical" class of probabilistic routing algorithms. We then evaluate the average routing distance for these algorithms and use it as an approximate for the probability vectors average routing distance.

Definition 5.2: A routing is called a Probabilistic Routing algorithm (PRA) if it satisfies the following assumptions:

i) A message is discarded after making over f spare moves in the network.

ii) The routing decisions at a given node are based on maximising the probability of minimal distance reachability when selecting a fault-free neighbour.

A PRA algorithm routes messages depending on a probabilistic value. The maximum number of spare moves along a given path is f. The total path length in this case is the Hamming distance between the source and destination nodes plus $2f$. We use the following notation in the theorem, which will be stated below.

$P_{k,s}$: Probability of making exactly s spare moves when routing between two nodes at Hamming distance k.

$P_{k,f+1}$: Probability of making more than f spare moves (i.e., probability of discarding a message).

$\overline{D}_k$: Average routing distance assuming message is not discarded.

Theorem 5.2: In the PRA algorithm, the average routing distance $\overline{D}_k$ between a given pair of nodes at Hamming distance k is given by

$$\overline{D}_k = \frac{1}{1-P_{k,f+1}} \sum_{s=0}^{f} (k+2s) P_{k,s} \text{ , where} \tag{5.8}$$

$$P_{1,0} = 1 - \frac{1}{NF} \sum_{\text{non faulty } A} P_1^A \text{ , with } NF \text{ being the number of non-faulty nodes} \tag{5.9}$$

$$P_{1,s} = 0 \text{ , } 1 \le s \le f \tag{5.10}$$

$$P_{1,f+1} = \frac{1}{NF} \sum_{\text{non faulty } A} P_1^A \tag{5.11}$$

$$P_{k,0} = 1 - \frac{\sum_{\text{non faulty } A} P_k^A}{2^n - f} \text{ , } k \ge 2 \tag{5.12}$$

$$P_{k,s} = (1 - p^k)P_{k-1,s} + p^k P_{k+1,s-1},\ 1 \le s \le f+1,\ 2 \le k < n \tag{5.13}$$

$$P_{n,s} = (1 - p^n)P_{n-1,s},\ 1 \le s \le f+1 \tag{5.14}$$

Proof: Every spare move increases the routing distance by two hops. Since messages are discarded after making f spare moves, we can write

$$\overline{D}_k = \sum_{s=0}^{f} (k + 2s) \cdot \text{Prob[making } s \text{ spare moves given that the message is not discarded]} \tag{5.15}$$

$$\overline{D}_k = \sum_{s=0}^{f} (k + 2s) \frac{P_{k,s}}{1 - P_{k,f+1}} \tag{5.16}$$

It can be easily shown that $P_{1,0} = 1$ and $P_{1,s} = 0$, $\forall s \ge 1$, since the source and destination nodes are both assumed non faulty. For $k \ge 2$, the probability, $P_{k,0}$, that a destination at Hamming distance k is minimally reachable, is given by

$$P_{k,0} = \sum_{\text{non faulty } A} \text{Prob[source node is } A] \cdot (1 - P_k^A) \tag{5.17}$$

where the P_k^A is computed using the same calculation for the probability vectors (equations 5.2, 5.3 and 5.4). For a message at distance k from its destination to make s spare moves, it either starts by making a first preferred move (with probability $1 - p^k$) leading to a node at distance $k-1$ from destination for which the remaining routing will include s spare moves (with probability $P_{k-1,s}$), or it starts by making a first spare move (with probability p^k) leading to a node at distance $k+1$ from destination with remaining s-1 spare moves to make (with probability $P_{k+1,s-1}$). Therefore, $P_{k,s}$ can be written as:

$$P_{k,s} = (1 - p^k)P_{k-1,s} + p^k P_{k+1,s-1},\ 1 \le s \le f+1,\ 2 \le k < n-1 \tag{5.18}$$

When the destination is at distance n, the first move can only be a preferred move, and therefore

$$P_{n,s} = (1-p^n)P_{n-1,s}\,,\ 1 \le s \le f+1 \qquad (5.19)$$

Notice that $\overline{D}_k$ is calculated using $P_{k,s}$ which is defined recursively using as a basis the expression (5.17) of $P_{k,0}$. This expression in turn calculated using the same probabilities, P_k^A, used in the calculation of the probability vectors. It is therefore expected for these calculations to provide an estimate of the average routing distance in the probability vectors algorithm.

Claim: The average routing distance in the probability vectors algorithm is approximated by $\overline{D}_k$ given in theorem 5.16.

To support this intuitive claim, we have compared the results of the average routing distance obtained using the above-derived expressions against those obtained through simulation. We have programmed the calculation provided in theorem 5.2 in order to obtain analytically the vector of average routing distances in an n-dimensional hypercube, denoted by $\overline{D}_A = (\overline{D}_1, \overline{D}_2, ..., \overline{D}_n)$, where $\overline{D}_k$ is the average routing distance from a node A to destinations at Hamming distance k, $(1 \le k \le n)$. We then used a simulation program that mimics the behaviour of the proposed algorithm in the network in order to measure experimentally the vector of average routing distances, denoted by $\overline{D}_E$.

Fig. 5.5 depicts both the analytical results using the PRA routing algorithm and experimental results using the proposed algorithm for a 128-node 7-dimensional hypercube, where the number of faulty nodes was set to 40, and distributed randomly over the network nodes. A total of 30,000 source-destination pairs were also selected randomly. The experimental results demonstrate that the analytical expressions predict the average routing distance with a reasonable degree of accuracy. Since we assume that both the source and destination nodes are non-faulty, the average routing distance to a destination at distance one is always one. In this case, as revealed by Fig.

5.5, the analytical and experimental results are identical. The figure also reveals that, as predicted in our claim, the experimental average distance, which corresponds to the actual average routing distance in the probability vectors algorithm, is always smaller than the analytically-derived average distance for the PRA algorithm. This fact reflects the good performance of the probability vectors algorithm as a fault-tolerant routing that can achieve high ratios of minimal routing in the presence of faults in the network.

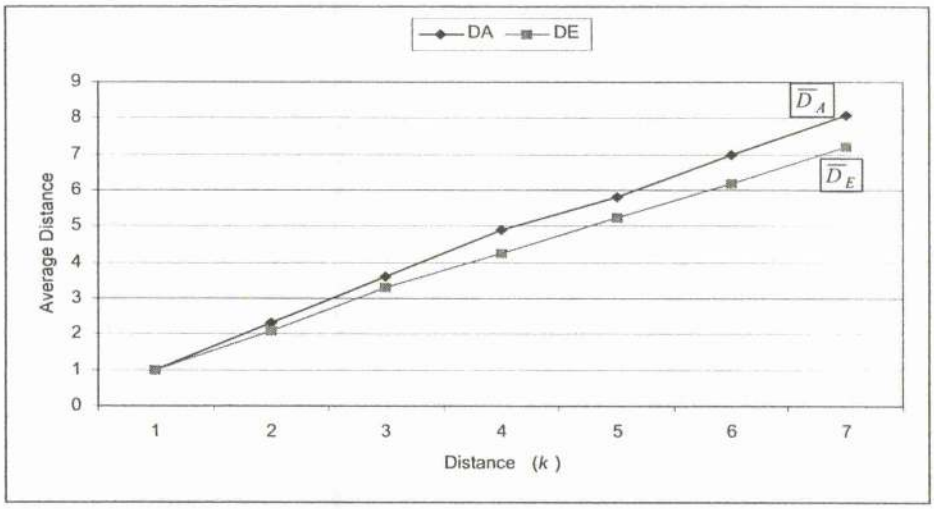


Fig. 5.5: The average distance in the 7-Dimensional hypercube calculated analytically ($\overline{D}_A$) and experimentally ($\overline{D}_E$).

5.4 Performance Considerations and Comparison with Safety Vectors

This section first presents an analysis of the complexity of the calculation phase of the probability vectors. The performance of the probability vectors calculations is dependent on the number of unreachable immediate neighbours. As outlined in Fig. 5.1, such calculations are performed in n-1 phases. In each phase, each node sends a real number to all its neighbours and receives a real number from each neighbour concurrently. Therefore the computation time complexity is $O(n^2)$

and the total number of generated messages is $O(n^2 2^n)$. Notice that the safety vectors algorithm performs the same order of message exchanges with the difference that our messages are of relatively larger size. This cost is quite large (especially in terms of the total number of generated messages). In real systems the frequency of fault occurrence is rather low and this cost is incurred only when a fault occurs. Therefore this relatively high communication cost can be tolerated in practical situations.

The proposed algorithm requires from each node A to know the number of its unreachable immediate neighbours, node A will then calculate a local probability vector P_A of n components (real numbers) and use it efficiently for routing. This compares favourably with global-information-based routing algorithms which require in the order of 2^n global information in the n-dimensional hypercube to be collected and used during decision making. This excessive communication and computation cost for global-information-based algorithms does not justify the little gain in achieving optimal routing as compared to near optimal routing achieved by limited-global-information-based algorithms as demonstrated in the simulation results of this section.

Let us now report results from simulation experiments comparing the performance of the adapted probability vectors algorithm against that of the safety vectors algorithm [81]. A simulation study has been conducted of both algorithms over a 10-dimensional hypercube (1024 nodes) with different random distributions of faulty nodes. We started initially with a non-faulty hypercube. Then, the number of faulty nodes was increased gradually up to 75% of the network size with random fault distributions. A total of 20,000 source-destination pairs were selected randomly during each simulation run. The three performance measures, namely the percentage of unreachability, average percentage of deviation from optimality, and percentage of looping

introduced in section 4.5 are used for the purpose of the present comparison.

The results depicted in Fig. 5.6 reveal that the probability vectors algorithm achieves much higher reachability with low to moderate deviation from optimality. The results in Fig. 5.7 may suggest that the safety vectors algorithm provides better performance in terms of deviation from optimality. However, the difference becomes substantial only when the number of faulty nodes is high (exceeding 50%). But when the number of faults becomes high (exceeding 50%), the safety vectors algorithm delivered less than 25% of the total number of messages. A more realistic comparison should consider the fact that the proposed probability vectors algorithm delivers messages to their destinations in most cases while the safety vectors algorithm has a substantial unreachability ratio, as revealed in Fig 5.6. The deviation from optimality becomes substantial (30%) in the new algorithm only when the percentage of faulty nodes exceeds 60% of the total number of nodes. Furthermore, the looping percentage, as shown in Fig 5.8, remains practically zero as long as the percentage of faulty nodes is less than 40%.

The probability vectors algorithm can route more messages through the network than the safety vectors algorithm using minimal or near minimal distance paths especially in the presence of a large number of faulty nodes. Under high fault rates the proposed algorithm is able to route a large percentage of messages for which the safety vectors algorithm announces a routing failure. This is due to the fact that the new algorithm tends to select paths that diverge from areas with high counts of faulty components. The safety vectors algorithm on the other hand routes via a neighbour only if that neighbour guarantees optimal routing to all destinations at the desired distance not just to the desired destination.

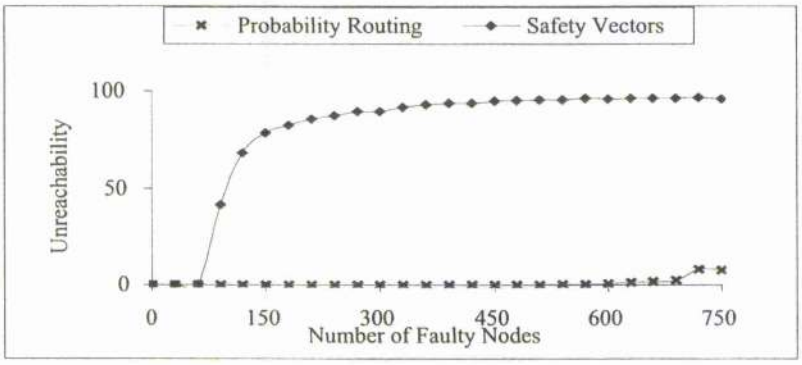


Fig. 5.6: Percentage of unreachability in the probability and safety vectors algorithms.

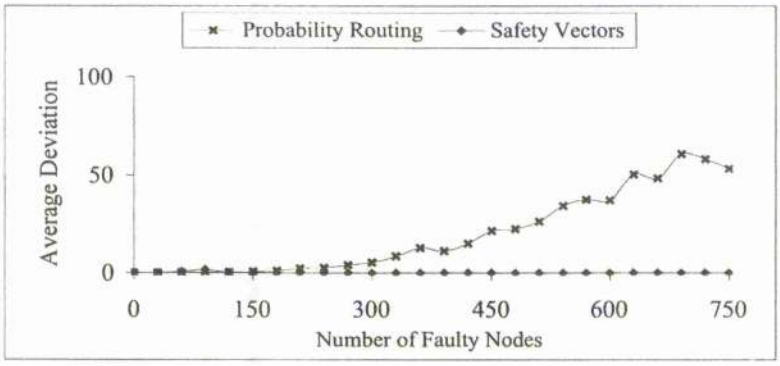


Fig. 5.7: Average percentage of deviation from optimality in the probability and safety vectors algorithms.

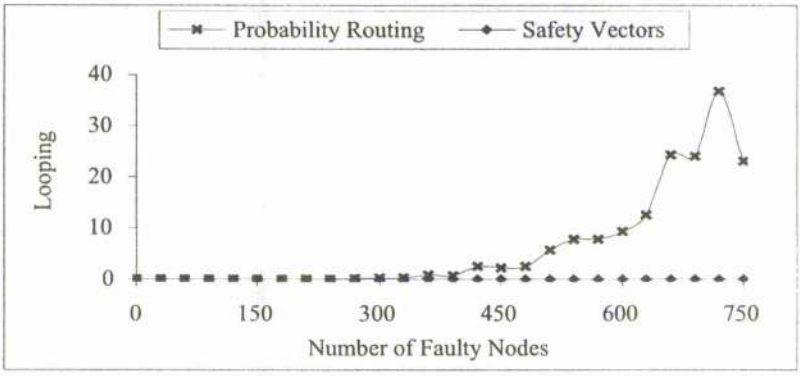


Fig. 5.8: Percentage of looping in the probability and safety vectors algorithms.

5.5 Conclusions

This chapter has adapted the probability vectors approach for the hypercube, and conducted a performance comparison against an existing fault-tolerant routing algorithm. In the adapted probability vectors algorithm, each node A determines first the faulty set, F_A, which represents the set of all neighbouring nodes which are faulty or unreachable from A due to faulty nodes or links. Using this faulty set as a basis and through message exchanges with neighbouring nodes, node A derives recursively an approximation of a numeric probability vector P^A. This probability vector is then used by node A to perform an efficient fault-tolerant routing in the hypercube. A probability-based analysis of some properties of the adapted probability vectors algorithm has been presented. The performance of the new algorithm has also been compared against that of the existing safety vectors algorithm. The results have shown that the new algorithm has a higher percentage of reachability than the safety vectors algorithm. Furthermore, even in situations where both algorithms achieve reachability, the probability vectors algorithm exhibits lower deviation from optimality.

Chapter 6

Conclusions and Future Directions

Routing in fault-free networks has been extensively studied in the past. As the network size scales up, the probability of processor and link failure also increases. It is therefore essential to design fault-tolerant routing algorithms that allow messages to reach their destinations even in the presence of faulty components (links and nodes). Although many fault-tolerant routing algorithms have been proposed for common multicomputer networks, very little research has been devoted to developing fault-tolerant routing for k-ary n-cubes, low-dimensional versions in particular, which are an important class of interconnection networks widely used in experimental and commercial parallel systems. The following points summarise the major contributions of the research work reported in the present thesis:

- A new limited-global-information-based fault-tolerant routing algorithm, using the concept of unsafety vectors, has been proposed for k-ary n-cubes. This algorithm uses

a *deterministic* approach where a numeric vector is used to reflect the status of the network. Although the deterministic approach has been widely employed by other algorithms in the past, the new algorithm has an important advantage over those existing in the literature in that it can ensure fault-tolerance under more relaxed assumptions regarding the number of faulty nodes and their locations in the network.

- An extensive simulation study has been conducted in order to evaluate the performance of the new fault-tolerant routing algorithm based on the unsafety vectors. The study has considered the performance of the algorithm for different network sizes. Three common performance measures have been used, notably the percentage of reachability, average percentage of deviation from optimality, and percentage of looping. Results presented here have revealed that for a practical number of faulty nodes, the new algorithm achieves good performance levels in terms of the routing distance and percentage of reachability.

- A performance study has been conducted to measure the effects of the level of complexity (m) on the unsafety vectors, with m being the number of calculations performed by a given node to determine its unsafety sets. The larger the value of m is, the better the routing decisions are, but at the expense of more computation and communication overhead. The results have demonstrated that the proposed algorithm yields good performance in terms of routing distances and percentages of reachability even when the parameter m is at its lowest value of 1, corresponding to minimum communication overhead.

- Another new limited-global-information-based fault-tolerant routing algorithm using the concept of probability vectors has also been proposed for k-ary n-cubes. The

algorithm is the first in the literature that uses *probabilities* to achieve fault-tolerance. The algorithm ensures fault-tolerance under relaxed assumptions regarding the number of faulty nodes and their locations in the network. It is also simpler to implement and has lower computational and storage overhead, when compared to algorithms that are based on the deterministic approach.

- A simulation study has also been conducted to evaluate the performance of the algorithm based on the probability vectors. The results have revealed a good outcome for the achieved percentage of reachability and routing distance. Moreover, the analysis has also shown that in addition to being simple to implement, the algorithm exhibits low computational overhead during message routing.

- A comparative study of the relative performance merits of the two proposed algorithms, the unsafety and probability vectors, has revealed that for practical number of faulty nodes, both algorithms achieve good performance levels in k-ary n-cubes. However, as stated above, the probability vectors algorithm has the advantage of being simpler to implement.

- While existing algorithms have addressed specific topologies, this study has shown that one of the main attractive features of our proposed algorithms is the ease with which they can be adapted to different k-ary n-cube topologies, e.g. hypercube and torus.

- Previous studies have focused mostly on fault-tolerance in the hypercube. Both the unsafety and probability vectors algorithms have been adapted for the hypercube in order to compare their performance against the recently proposed safety vectors

algorithm. Extensive simulation experiments have confirmed that the new algorithms exhibit superior performance and fault tolerance characteristics over the safety vectors algorithm.

- An analytical study that examines the properties of the proposed probability vectors algorithm has been presented. The study has confirmed that the algorithm is capable of routing messages in a faulty network via fault-free minimal paths if they exist, otherwise it routes messages via fault-free near-minimal paths if they exist. The algorithm has been shown to be able to perform minimal distance routing to any destination at distance k from a given source node that has a reachable preferred neighbour with a zero in the k-1 element of its probability vector.

There are several interesting issues and open problems that require further investigation. These are summarised below.

- Meshes are another class of networks that have been widely used in multicomputers due to their simple structures and ease of implementation. Unlike k-ary n-cubes, however, they are based on an asymmetric topology because nodes at the edges have fewer neighbours than those located in the center. An obvious step of this work would extend both the proposed unsafety vectors and probability vectors approaches to meshes. However, this would require careful re-examination of the major factors, such as preferred neighbours, spare neighbours, minimal path characterisation, diameter, and degree, in order to deal with the inherent asymmetry of these networks.

- Multicast (one-to-many) and broadcast (one-to-all) are important communication operations required by many real-world parallel applications found in Science and

Engineering [52, 71, 83]. A lot of research activities have been devoted in the past to devising efficient algorithms to support these operations [11, 26, 34, 53]. Moreover, some research work has tried to augment those algorithms to deal with faults in the network. A possible continuation of this research would extend the ideas of the probability and unsafety vectors to support efficient fault-tolerant multicast and broadcast operations.

- The switching technique determines the way a message visits intermediate routers when crossing the network. Among the well-known switching techniques are packet switching [79], circuit switching [79] and wormhole routing [27]. The switching technique considerably influences the router architecture, and has great impact on network performance. For instance, wormhole switching provides a magnitude improvement in network performance over packet switching since it makes latency less sensitive to the message distance under light traffic [27]. While this thesis has discussed the idea of the unsafety vectors and probability vectors at the "network topology" level, an interesting line of research would investigate implementation-related issues and study the performance of both algorithms when a particular switching technique, e.g. wormhole switching, is used.

- Analytical models that measure message latency and throughput in k-ary n-cubes have been widely proposed in the literature [49, 50, 59, 67]. All these models, however, have been discussed in the context of fault-free networks, and as a result there has been relatively little activity in the analytical modelling of fault-tolerant routing algorithms. Another interesting research work would develop analytical models for both the probability and unsafety vectors and validate their accuracy

through simulation experiments. Such models will be a very useful performance tool as they can be used to assess the performance of the proposed algorithms in large networks, that are not feasible to simulate due to the amount of time and computing resources required to run large simulations.

References

[1] S.B. Akers, D. Hared, B. Krishnamurthy, The star graph: An Attractive alternative to the *n*-cube, *Proc. Int. Conf. Parallel Processing*, University Park, Pennsylvania, 1987, vol. 923, pp.393-400.

[2] B.F. Almohammad, B. Bose, Fault-tolerant communication algorithms in toroidal networks, *IEEE Trans. Parallel & Distributed Systems*, vol. 10, no.10, pp.976-983, 1999.

[3] J. Al-Sadi, K. Day, M. Ould-Khaoua, Probability vectors: A new fault-tolerant routing algorithm for *k*-ary *n*-cubes, *Proc. 17th ACM Symposium on Applied Computing*, Madrid, March 10-14, 2002, pp. 830-834.

[4] J. Al-Sadi, K. Day, and M. Ould-Khaoua, Fault-tolerant routing in the binary *n*-cube using unsafety sets, *Proc. Int. Conf. Parallel & Distributed Processing: Techniques & Applications (PDPTA'99)*, Las-Vegas, June 29-July 1, 1999, pp. 2190-2194.

[5] J. Al-Sadi, K. Day, M. Ould-Khaoua, Probability-based fault-tolerant routing in hypercubes, *Proc. Europar'2000*, Lecture Notes in Computer Science, Springer-Verlag, Munich, Aug.29-Sept. 1, 2000, pp. 935-938

[6] J. Al-Sadi, K. Day, M. Ould-Khaoua, Unsafety vectors: A new fault-tolerant routing for *k*-ary n-cubes, *Microprocessor and Microsystems*, vol. 25, no. 5, 2001, 2000, pp. 239-246.

[7] J. Al-Sadi, K. Day, M. Ould-Khaoua, A new fault-tolerant routing for *k*-ary *n*-cubes using unsafety vectors, *Proc. The 2000 Arab Conference on Information Technology (ACIT'2000)*, Zarka Private University, Jordan, Oct.31-Nov. 2, 2000, pp. 119-126.

[8] E. Anderson, J. Brooks, C. Grassl, S. Scott, Performance of the Cray T3E multiprocessor, *Proc. Supercomputing Conference*, San Jose, California, 1997, CD-ROM http:/www.supercomp.org/sc97/processing.

[9] W.C. Athas, C.L. Seitz, Multicomputers: message passing concurrent computers, *IEEE Computer*, vol. 21, no. 8, 1988, pp. 9-24.

[10] L.M. Bhuyan, P.D. Agarwal, Generalised hypercube and hyperbus structures for a computer network, *IEEE Trans. Computers*, vol. 33, no. 4, 1984, pp. 323-333.

[11] K. Bolding, W. Yost, The express broadcast network: A network for low cost broadcast of control messages, *Proc. Int. Conf. Algorithms & Architectures for Parallel Processing*, Brisbane, Australia, vol. 1, April 1995, pp. 93-102.

[12] R. Boppana, S. Chalasani, Fault-tolerant communication with partitioned dimension-order routers, *IEEE Trans. Parallel & Distributed Systems*, vol. 10, no.10, 1999, pp.1026-1039.

[13] S. Borkar, R. Cohen, G. Cox, S. Gleason, T. Gross, H.T. Kung, M. Lam, B. Moore, C. Peterson, J. Pieper, L. Rankin, P.S. Tseng, J. Sutton, J. Urbanski, and J. Webb, iWarp: An integrated solution to high-speed parallel computing, *Proc. Supercomputing '88*, Orlando, Florida, Nov. 1988, pp. 330-339.

[14] B. Bose, B. Broeg, Y. Kwon, Y. Ashir, Lee distance and topological properties of *k*-ary *n*-cubes, *IEEE Transactions on Computers*, vol. 44, no. 8, 1995, pp. 1021-1030.

[15] M.S. Chen, K. G. Shin, On hypercube fault-tolerant routing using global information, *Proc. 4th Conf. Hypercube Concurrent Computers & Applications*, Monterey, California, March 1989, pp. 83-86.

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

[17] M.S. Chen, K.G. Shin, Depth-first search approach for fault-tolerant routing in hypercube multicomputers, *IEEE Trans. Parallel & Distributed Systems*, vol. 1, no. 2, 1990, pp. 152-159.

[18] G.M. Chiu, K.-S. Chen, Fault-tolerant routing strategy using routing capability in Hypercube Multicomputers, *Proc. Int. Conf. Parallel and Distributed Systems*, Tokyo, Japan, 1996, pp. 396-403.

[19] G.M. Chiu, K.-S. Chen, Use of routing capability for fault-tolerant routing in hypercube multicomputers, *IEEE Trans. Computers*, vol. 46, no. 8, 1997, pp. 953-958.

[20] G.M. Chiu, S.P. Wu, A Fault-tolerant routing strategy in hypercube multicomputers, *IEEE Trans. Computers*, vol. 45, no. 2, 1996, pp. 143-156.

[21] R. Cypher, F. Meyer, C. Scheideler, B. Voking, Universal algorithms for store-and-forward and wormhole routing, *Proc. 28th ACM Symp. on Theory of Computing*, Philadelphia, Pennsylvania, 1996, pp. 356-365.

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

[23] W.J. Dally *et al.*, The Reliable Router: A reliable and high-performance communication

substrate for parallel computers, *Proc. Parallel Computer routing and Communication, in Lecture Notes in Computer Science,* Seattle, Washington, vol. 853, 1994, pp. 241-255.

[24] K. Day, A. Al-Ayyoub, Fault diameter of *k*-ary *n*-cube networks, *IEEE Trans. Parallel & Distributed Systems*, vol. 8, no. 9, 1997, pp. 903-907.

[25] P.W. Dowd, M. Carrato, IIigh speed routing in a parallel processing environment: a simulation study, *Proc. 24th Annual Simulation Symposium.* New Orleans, Louisiana, *IEEE Computer. Society. Press*, 1991, pp. 60-72.

[26] J. Duato, Theory of deadlock-free adaptive multicast routing in wormhole networks, *IEEE Trans.* Parallel Distributed Systems, vol. 6, no. 9, 1995, pp. 976-987.

[27] J. Duato, S. Yalamanchili, L. Ni, Interconnection networks: An engineering approach, *IEEE Computer Society Press*, 1997.

[28] J. Duato, A necessary and sufficient condition for deadlock-free routing in cut-through and store-and-forward networks, *IEEE Trans. Parallel Distributed Systems*, vol. 7, no. 8, 1996, 841-854.

[29] S. Felperin, P. Raghavan, E. Upfal, A theory of wormhole routing in parallel computers, *IEEE Trans. Communications*, vol. 45, no. 6, 1996, pp. 704-713.

[30] S. Fujita, A fault-tolerant broadcast scheme in the star graph under the single-port, half-duplex Communication model, *IEEE Trans. Parallel & Distributed Systems,* vol. 10, no. 10, 1999, pp. 1123-1126.

[31] P.T. Gaughan, S. Yalamanchili, Adaptive routing protocols for hypercube interconnection networks, *IEEE Computer*, vol. 26, no. 5, 1993, pp. 12-24.

[32] J.M. Gordon, Q.F. Stout, Hypercube message routing in the presence of faults, *Proc. 3rd Conf. Hypercube Concurrent Computers and Applications,* Pasadena, California, Jan. 1988, pp. 251-263.

[33] A. Gottlieb *et al*, The NYU Ultracompuetr- Designing a MIMD Shared Memory Parallel Computer, *IEEE Trans. Computers*, vol. 32, no. 2, 1983, pp.175-189.

[34] S. Graham, S. Seidel, The cost of broadcasting on star graphs and *k*-ary hypercubes, *IEEE Trans. Computers*, vol. 42, no. 6, 1993, pp. 756-759.

[35] L. Gravano, G. Pifarre, P. Berman, J. Sanz, Adaptive deadlock- and livelock-free routing with all minimal paths in torus networks, *IEEE Trans. Parallel & Distributed Systems*, vol. 5, no. 12, 1994, pp. 1233-1251.

[36] Q.P. Gu, S. Peng, Unicast in hypercube with large number of faulty nodes, *IEEE Trans. Parallel & Distributed Systems*, vol. 10, no. 10, Oct. 1999, pp. 964-975.

[37] K.T. Herley, G. Bilardi, Deterministic simulations of PRAMs on bounded-degree networks, *SIAM J. Computing*, vol. 23, no. 2, 1994, pp. 276-292.

[38] R.E. Kessler, J.L. Schwarzmeier, CRAY T3D: A new dimension for Cray Research, *Proc.38th IEEE Computer Soc. Intl. Conference. (CompCon)*, Houston, Texas, Feb. 1993, pp. 176-182.

[39] S.Y. Kim, K.Y. Chwa, Optimal embeddings of multiple graphs into a hypermesh, *Proc. Int. Conf. Parallel & Distributed Systems (ICPADS'97)*, Hsinchu, Taiwan, 1997, pp. 436-443.

[40] J. Konicek *et al*, The organization of the Cedar system, *Proc. Int. Conf. Parallel*

Processing, Austin, Texas, 1991, pp. 49-56.

[41] M. Kunde, Block gossiping on grids and tori: deterministic sorting and routing match the bisection bound, *Proc. European Symposium Algorithms*, Bad Honnef, Germany, Lecture Notes in Computer Science, Springer_Verlag, vol. 726, 1993, pp. 272-283.

[42] M. Kunde, Routing and sorting on mesh connected processor arrays, *Proc. VLSI Algorithms and Architectures*, Corfu, Greece, Lecture Notes in Computer Science, Springer_Velag, vol. 319, July 1988, pp. 423-433.

[43] Y. Lan, A fault-tolerant routing algorithm in hypercubes, *Proc. Int. Conf. Parallel Processing*, Aug. 1994, pp. III163-III166.

[44] Y. Lan, An adaptive fault-tolerant routing algorithm for hypercube multicomputers, *IEEE Trans. Parallel & Distributed Systems*, vol. 6, no. 11, 1998, pp. 1147-1152.

[45] S. Latifi, Combinational analysis of fault-diameter of the *n*-cube, *IEEE Trans. Computers*, vol. 42, no. 1, 1993, pp. 27-33.

[46] T.C. Lee, J.P. Hayes, A fault-tolerant communication scheme for hypercube computers, *IEEE Trans. Computers*, vol. 41, no. 10, 1992, pp. 1242-1256.

[47] D. Linder, J. Harden, An adaptive and fault-tolerant wormhole routing strategy for *k*-ary hypecubes, *IEEE Trans. Computers* , vol. 40, no. 1, 1991, pp. 2-12.

[48] K.J. Liszka *et al*, Problems with comparing interconnection networks: Is an alligator better than an armadillo?, *IEEE Concurrency*, vol. 5, no. 4, 1997, pp. 18-28.

[49] S. Loucif, M. Ould-Khaoua and L.M. Mackenzie, The "Express Channel" concept in

hypermeshes and *k*-ary *n*-cubes, *Proc. 8th IEEE Symp. Parallel & Distributed Processing*, IEEE Computer Society Press, New Orleans, Louisiana (USA), Oct. 1996.

[50] S. Loucif, H. Sarbazi-Azad, M. Ould-Khaoua, Message latency in *k*-ary *n*-cubes with hop-based routing, *IEE Proceedings-Computers and Digital Techniques*, vol. 148, no. 2, 2001, pp. 89-94.

[51] A. Lubiw, Counterexample to a conjecture of Szymanski on hypercube routing, *Information Processing Letters*, vol. 35, no 2, 1990, 57-61.

[52] P.K. Mckinley, C. Trefftz, Efficient broadcast in all-port wormhole-routed hypercubes, *Proc. 7th Int. Conf. Parallel Processing*, New Port Beach, California, 1993, pp. 288-291.

[53] P.K. Mckinley *et al*, Unicast-based multicast communication in wormhole-routed networks, *IEEE Trans. Parallel & Distributed Systems*, vol. 5, no. 12, 1994, pp. 1254-1265.

[54] *n*CUBE Systems, *N-cube Handbook*, *n*CUBE, 1986.

[55] L.M. Ni, P.K. McKinley, A survey of routing techniques in wormhole networks, *IEEE Computer*, vol. 26, no. 2, 1993, pp. 62-76.

[56] M. Noakes *et al*, The J-machine multicomputer: An architectural evaluation, *Proc. 20th Int. Symp. Computer Architecture*, San Diego, California, May 1993. pp. 224-235.

[57] A.G. Nowatzyk *et al.*, S-Connect: From networks of workstations to supercomputer performance, *Proc. 22nd International Symp. Computer Architecture*, Santa Margherita, Italy, June 1995, pp. 71-82.

[58] S.F. Nugent, The iPSC/2 Direct-Connect Communication Technology, *Proc. Conf. Hypercube Concurrent Computers & Applications*, Pasadena, California, vol. 1, Jan. 1988, pp. 51-60.

[59] M. Ould-Khaoua, H. Sarbazi-Azad, An analytical model of adaptive wormhole routing in hypercubes in the presence of hot-spot traffic, *IEEE Transactions Parallel & Distributed Systems*, vol. 12, no. 3, 2001, pp. 283-288.

[60] Paragon XP/S Product Overview. Intel Corporation, Supercomputer Systems Division, Beaverton, Or, 1991.

[61] G.F. Pfister *et al*, The IBM Research Parallel Processor Prototype (RP3): Introduction and Architecture, *Proc. Int. Conf. Parallel Processing*, Los Alamitos, California, Aug. 1985, pp. 764-771.

[62] C.S. Raghavendra, P.J. Yang, S.B. Tien, Free dimension– an effective approach to achieving fault-tolerance in hypercubes, *Proc. 22nd Int. Symp. Fault-Tolerant Computing*, Boston, Massachusetts, 1992, pp. 170- 177.

[63] J. Rattler, Concurrent Processing: A new direction in scientific computing, *Proc. Of National Computer Conf.*, AFIPS press, vol. 54, 1985, pp. 157-166.

[64] C.P. Ravikumar, C.S. Panda, Adaptive routing in *k*-ary *n*-cubes using incomplete diagnostic information, *Microprocessors and Microcomputers*, vol. 20, 1997, pp. 351- 360.

[65] Y. Saad, M.H. Schultz, Data communication in hypercubes, *J. parallel Distributed Computing*, vol. 6, no. 1, 1989, pp. 115-135.

[66] Y. Saad, M.H. Schultz, Topological properties of hypercubes, *IEEE Trans on computers*,

vol. 37, no. 7, 1988, pp. 867-872.

[67] H. Sarbazi-Azad, M. Ould-Khaoua, L.M. Mackenzie, An accurate performance model of adaptive wormhole routing in *k*-ary n-cube interconnection networks, *Performance Evaluation*, vol. 43, no. 2-3, 2001, pp. 165-179.

[68] C.L. Seitz, The cosmic Cube, *CACM*, vol. 28, 1985, pp. 22-23.

[69] C.L. Seitz, The Hypercube Communication Chip, Dep. Comp. Sci., CalTech, Display File 5182:DF:85, 1985.

[70] H. Shen, F. Chin Y. Pan, Efficient fault-tolerant routing in multi-hop optical WDM networks, *IEEE Trans. Parallel & Distributed Systems*, vol. 10, no. 10, 1999, pp. 1012-1025.

[71] J.-P. Sheu, M.-Y. Su, A multicast algorithm for hypercube multiprocessors, *Proc. Int. Conf. Parallel Processing*, University Park, Pennsylvania, 1992, pp.18-22.

[72] J.F. Sibeyn, Deterministic routing and sorting on rings, *Proc.8th IEEE Int. Parallel Processing Symp.*, 1994, pp. 406-410.

[73] H.J. Siegel, C.B. Stunkel, Trends in parallel machine interconnection networks, *IEEE Computer. Science. & Eng.*, 1996, pp. 69-71.

[74] H.J. Siegel *et al*, PASM: A partitionable SIMD/MIMD System for Image Processing and Pattern Recognition, *IEEE Trans. Computers*, vol. 30, no. 12, 1981, pp. 934-947.

[75] Silicon Graphics, Origin 200 and Origin 2000, *Technical Report*, 1996.

[76] H. Sullivan, T.R. Bashkow, D. Klappholz, A large scale homogeneous fully distributed

parallel machine. *Proc. 4th Ann. Symp. Computer Architecture*, IEEE, Now York, NY, vol. 5, Mar. 1977, pp. 105-117.

[77] T. Szymanski, Hyper-meshes: Optical interconnection networks for parallel processing, *Journal Parallel & Distributed Computing*, vol. 26, 1995, pp. 1-23.

[78] T. Szymanski, On the permutation capability of a circuit-switched hypercube, *Proc. Int. Conf. Parallel Processing, IEEE Society Press*, vol. 1, Aug. 1989, pp. 103-110.

[79] A.S. Tanenbaum, Computer Networks, Prentice-Hall Int., Inc. (2nd Ed.) 1989.

[80] B. Vanvoorst, S. Seidel, E. Barscz, Workload of an iPSC/860, *Proc. 8th Scalable High-Performance Computing Conf.*, Knoxville, May 1994, pp. 221-228

[81] J. Wu, Adaptive fault-tolerant routing in cube-based multicomputers using safety vectors, *IEEE Trans. Parallel and Distributed Systems*, vol. 9, no. 4, 1998, pp. 321-334.

[82] J. Wu, F. Gao, Z. Li, and Y. Min, Optimal fault-tolerant routing in hypercubes using extended safety vectors, *Proc. of 7th Int. Conf. Parallel and Distributed Systems (ICPADS)*, Iwate, Japan, July 2000, pp. 264-271.

[83] J. Wu, E.B. Fernandez, Broadcasting in faulty hypercubes, *Microprocessors and microprogramming*, vol. 39,no. 1, 1993, pp. 43-53.

[84] J. Wu, Reliable unicasting in faulty hypercubes using safety levels, *IEEE Trans. Computers*, vol. 46, no. 2, 1997, pp. 241-247.

Publications During Research

1- J. Al-Sadi, K. Day, M. Ould-Khaoua, Unsafety vectors: A new fault-tolerant routing for *k*-ary *n*-cubes, *Microprocessors and Microsystems*, vol. 25, no. 5, 2001, pp. 239-246.

2- J. Al-Sadi, K. Day, M. Ould-Khaoua, Fault-tolerant routing in hypercubes using probability vectors, *Parallel Computing*, vol. 27, no. 10, 2001, pp. 1381-1399.

3- Al-Sadi, K. Day, and M. Ould-Khaoua, Probability-based fault-tolerant routing in hypercubes, *The Computer Journal*, vol. 44, no. 5, pp. 368-373, 2001.

4- J. Al-Sadi, K. Day, M. Ould-Khaoua, Unsafety vectors: A new Fault-tolerant routing for the binary *n*-cube, *Journal of Systems Architecture*, vol. 47, no. 9, 2002, pp. 783-793.

5- J. Al-Sadi, K. Day, M. Ould-Khaoua, Probability-based fault-tolerant routing in hypercubes, *Proc. Europar'2000*, in Lecture Notes in Computer Science, Springer-Verlag, Munich, Aug. 29-Sept. 1, 2000, pp. 935-938.

6- J. Al-Sadi, K. Day, and M. Ould-Khaoua, Fault-tolerant routing in the binary *n*-cube using unsafety sets, *Proc. Int. Conf. Parallel & Distributed Processing: Techniques & Applications (PDPTA'99)*, Las-Vegas, June 29-July 1, 1999, pp. 2190-2194.

7- J. Al-Sadi, K. Day, M. Ould-Khaoua, A new fault-tolerant routing for *k*-ary *n*-cubes using

unsafety vectors, *Proc. 2000 Arab Conference on Information Technology (ACIT'2000)*, Zarka Private University, Jordan, Oct.31-Nov. 2, 2000, pp. 119-126.

8- J. Al-Sadi, K. Day, M. Ould-Khaoua, Probability vectors: A new fault-tolerant routing algorithm for *k*-ary *n*-cubes, *Proc. 17th ACM Symposium on Applied Computing*, Madrid, March 10-14, 2002, pp. 830-834.

9- J. Al-Sadi, K. Day, M. Ould-Khaoua, Efficient fault-tolerant routing in multicomputer networks, Technical Report TR-2001-88, Department of Computing Science, University of Glasgow, April 2001.

10- J. Al-Sadi, K. Day, M. Ould-Khaoua, Performance evaluation of a probability-based fault-tolerant routing algorithm, Technical Report TR-2001-89, Department of Computing Science, University of Glasgow, April 2001.

11- J. Al-Sadi, K. Day, M. Ould-Khaoua, A new fault-tolerant routing algorithm for *k*-ary *n*-cube networks, *under review, International Journal of High-Speed Computing*, 2001.

12- J. Al-Sadi, K. Day, M. Ould-Khaoua, Performance analysis of a fault-tolerant routing algorithm in hypercubes, *under review, Journal of Supercomputing*, 2001.

13- J. Al-Sadi, K. Day, M. Ould-Khaoua, Analysis of Fault-Tolerant Routing Algorithms in *k*-Ary *n*-Cubes Networks, *under review*, *Journal of Information Science and Engineering (JISE)*, 2001.