Fall 12-2013

# New Fault Tolerant Multicast Routing Techniques to Enhance Distributed-Memory Systems Performance

Masoud Esmail Masoud Shaheen
*University of Southern Mississippi*

The University of Southern Mississippi

# NEW FAULT TOLERANT MULTICAST ROUTING TECHNIQUES TO ENHANCE DISTRIBUTED-MEMORY SYSTEMS PERFORMANCE

by

Masoud Esmail Masoud Shaheen

Abstract of a Dissertation
Submitted to the Graduate School
of The University of Southern Mississippi
in Partial Fulfillment of the Requirements
for the Degree of Doctor of Philosophy

December 2013

ABSTRACT

NEW FAULT TOLERANT MULTICAST ROUTING TECHNIQUES TO ENHANCE DISTRIBUTED-MEMORY SYSTEMS PERFORMANCE

by Masoud Esmail Masoud Shaheen

December 2013

Distributed-memory systems are a key to achieve high performance computing and the most favorable architectures used in advanced research problems. Mesh connected multicomputer are one of the most popular architectures that have been implemented in many distributed-memory systems. These systems must support communication operations efficiently to achieve good performance. The wormhole switching technique has been widely used in design of distributed-memory systems in which the packet is divided into small flits. Also, the multicast communication has been widely used in distributed-memory systems which is one source node sends the same message to several destination nodes. Fault tolerance refers to the ability of the system to operate correctly in the presence of faults. Development of fault tolerant multicast routing algorithms in 2D mesh networks is an important issue. This dissertation presents, new fault tolerant multicast routing algorithms for distributed-memory systems performance using wormhole routed 2D mesh. These algorithms are described for fault tolerant routing in 2D mesh networks, but it can also be extended to other topologies. These algorithms are a combination of a unicast-based multicast algorithm and tree-based multicast algorithms. These algorithms works effectively for the most commonly encountered faults in mesh networks, f-rings, f-chains and concave fault regions. It is shown that the proposed routing algorithms are effective even in the presence of a large number of fault

regions and large size of fault region. These algorithms are proved to be deadlock-free. Also, the problem of fault regions overlap is solved. Four essential performance metrics in mesh networks will be considered and calculated; also these algorithms are a limited-global-information-based multicasting which is a compromise of local-information-based approach and global-information-based approach. Data mining is used to validate the results and to enlarge the sample. The proposed new multicast routing techniques are used to enhance the performance of distributed-memory systems. Simulation results are presented to demonstrate the efficiency of the proposed algorithms.

The University of Southern Mississippi

# NEW FAULT TOLERANT MULTICAST ROUTING TECHNIQUES TO ENHANCE DISTRIBUTED-MEMORY SYSTEMS PERFORMANCE

by

Masoud Esmail Masoud Shaheen

A Dissertation
Submitted to the Graduate School
of The University of Southern Mississippi
in Partial Fulfillment of the Requirements
for the Degree of Doctor of Philosophy

Approved:

Dia Ali_____
Director

Ahmed Abukmail_____

Joe Zhang_____

Jonathan Sun_____

Ras Pandey_____

Adel Ali_____

Susan A. Siltanen_____
Dean of the Graduate School

December 2013

ACKNOWLEDGMENTS

This is to thank all of those who have assisted me in this effort. I wish to express my fruitful thanks and sincere appreciation to Dr. Dia Ali, committee chairman, for his academic supervision, scientific discussion, helping, guiding, and his continuous valuable support throughout this work. I also thank Dr. Ahmed Abukmail, Dr. Chaoyang Zhang, Dr. Adel Ali, Dr. Ras Pandey and Dr. Zheng Sun, the members of my graduate committee, for their guidance and suggestions. Especially Dr. Abukmail, gratitude is due for all his advice, suggestion of the problem, friendship, helpful guidance, and active support. I also thank the school of computing staff, especially Ms. Crystal McCaffrey; without their knowledge and assistance, this study would not have been successful.

I would like to thank my family members, especially my wife, Asmaa Hassan, my daughters, Jomana and Hana, and my son, Yassin, for supporting and encouraging me to pursue this degree. Without my wife's encouragement, I would not have finished the degree. I also would like to thank my mom, dad, and sister, Rania, for their infinite support throughout everything.

TABLE OF CONTENTS

LIST OF TABLES

Table

LIST OF ILLUSTRATIONS

Figure

# LIST OF PSEUDOCODE

Pseudocode

# CHAPTER I

## INTRODUCTION

Many of today's advanced research problems need great computing power at high speeds. Parallel computer systems, which emphasize parallel processing, are the most advantageous architectures to obtain a greater computing power needed by many of today's advanced research problems. These problems include artificial intelligence, expert system, robotics, signal processing, petroleum exploration, fluid mechanics, fusion energy research, medical diagnosis, military defense, weather forecasting, high-energy physics, space sciences, and servicing web servers. In many of these applications, input data arrives at very high rates, and the processed outputs must be generated very rapidly in order to be useful. In conventional sequential digital computers, a single memory buffer serves as the only gate between the high-speed memory and the central processing unit. This makes it necessary to organize all computational tasks in a strictly sequential fashion, hence the more complex the computing task, the more time consuming by the computation. Although advances in hardware technology have led to continuing increases in the speed of individual arithmetic operations, these have been greatly overshadowed by the increasing complexity of many simulation problems. To attain high speeds of the digital computers, the parallelisms have been used in their hardware design. The implementation of these techniques has given arise to the parallel computer systems.

Distributed-memory systems, which have unshared distributed memories among processors, are the most advantageous architectures in building a massively parallel computer system. These systems need switching techniques to transmit messages among processors. The wormhole switching technique has been widely used in the design of these systems. The multicast pattern, in which one processor sends the same message to

multiple processors, is the most fundamental communication pattern. Fault tolerance is a central issue facing the design of interconnection networks for distributed-memory systems. The suited topologies for current distributed-memory systems are 2D mesh, Hypercube and tours. This research will concentrate on studying the fault-tolerant multicast wormhole routings in mesh networks.

## Problem Statement

The multicast pattern, in which one processor sends the same message to multiple processors, is the most fundamental communication pattern used in distributed-memory systems. A familiar problem in the current design of distributed memory system is that fault tolerance is not considered early enough in the design process. It is critical that with the increased complexity and functionality of distributed-memory systems today that the fault tolerance abilities become a part of the system design. Early concern of fault managing abilities of a system can result in more reliable systems. Several researchers conducted as a result of software design and routing techniques errors were presented [1]. Hence, efficient fault-tolerant multicast routing algorithms are critical to the performance of distributed-memory systems. An effective multicast routing must be deadlock-free and should minimize network traffic steps and network traffic time. Tree-based techniques offer a very promising means of achieving extremely efficient multicast routing. These techniques forward a message copy to multiple output channels. Tree-based algorithms have advantageous over other multicast techniques. Hence, the multicasting using tree-based techniques in routed mesh multicomputer will be studied in this research. Important issues such as deadlock and fault tolerance are critical to the performance of distributed-memory systems.

In this research, the above mentioned factors and issues will be studied to improve distributed-memory systems performance. Also, the most accepted criteria such as network traffic steps, network latency steps, network traffic time and network latency time for the communication patterns will be considered to evaluate the performance.

## Significance of the Study

Many research works have been devoted throughout the last era to enhance the performance of distributed-memory systems (multicomputer). As the number of nodes in distributed-memory systems network is increasing, the time necessary to deliver data between the nodes is significant in whole system performance. In addition, it will affect the possible granularity level of parallel processing in running an application program. Distributed-memory systems (multicomputer) are the focal of this research for its several significant benefits. First, distributed-memory systems are scalable, which means that its efficiency increases as the number of node increases. Second, there is no switch and bus contention. Third, there are no cache coherency difficulties. Every processor is responsible for its own data and does not need to worry about placing copy of it in its own local cache and having another processor reference the original.

In order to design a good routing algorithm, we should consider a switching technique (switching techniques determine how messages are forwarded through the network) that will satisfy the basic needs, the message length and buffer size that can be used. This research concentrates on wormhole switching for its several advantages. Wormhole switching technique makes more efficient use of buffers and helps to create deadlock-free algorithms. An entire packet (message) need not be buffered to deliver to the next node, reduce traffic time and latency time. This reduces latency (delay) and traffic noticeably compared to other switching techniques.

Another consideration in the design of parallel processing systems is the set of pathways over which the processors, memories, and switches connect to each other. Those are the connections that define the network topology of the machine. Mesh network topology has been studied in this research because it has the following advantages:

- It is easy detection and isolation of faults in the network.
- Messages can be delivered from different devices concurrently and use alternative paths in case of failure take place or performance degradation.
- Mesh can endure high traffic.
- Expansion and modification in topology can be done without difficulties on other nodes. Distributed-memory systems using mesh topology as their essential architecture have been around for years. A number of large research and commercial multicomputer systems have been built based on mesh topologies, such as Blue Gene Supercomputer.

It is important to find new fault-tolerant multicast routing techniques in the area of distributed-memory systems which exploit parallel computing facilities to:

- Reduce a lot of concerned factors such as network latency steps, network traffic steps, network latency time and network traffic time.
- Apply these routing techniques on 2D mesh network topology.
- Apply it on fault regions with different ships – regular (convex, concave) and irregular - and solve a problem with overlap fault regions.

As shown in the next few chapters, our proposed fault tolerant routing algorithms have achieved the above mentioned factors. In addition, they are compromise of two

basic techniques (tree-based and unicast-based) and they exploit the advantages of each one of these two techniques.

The rest of the dissertation is organized as follow:

In Chapter II, a background of parallel computer memory architectures are considered. Also, Basic network topologies and network switching techniques are investigated. In addition, they are surveyed.

In Chapter III, an overview of fault tolerant multicast routing algorithms is presented. Also, a brief introduction to deadlock, Fault model and fault tolerance were given. Moreover, fault tolerant multicast routing algorithms for regular and irregular fault regions were studied.

In Chapter IV, FTDM and its improved version *i*FTDM fault tolerant multicast routing algorithms are proposed. Also, simulation study for both of algorithms is conducted.

In Chapter V, data mining for proposed routing algorithm is presented to validate the results. In addition, three tools (WEKA, EXCEL and MATLAB) are used to do regression analysis.

In Chapter VI, an efficient fault tolerant multicast routing algorithm, Yassin, for wormhole routed 2D mesh multicomputer is presented. Four essential performance metrics in mesh networks, network traffic steps, network latency steps, network traffic time and network latency time are evaluated.

In Chapter VII, list of some possible future work and summary of this dissertation are presented.

# CHAPTER II

## BACKGROUND

Parallel computer systems, which emphasize parallel processing, are the most favorable architectures to increase the computing power. Parallel processing continues to hold the promise of the solution of the more complex problems by connecting a number of powerful computer processors together into a single system. These connected processors assist in solving a single problem that exceeds the capability of any one of the processors. Parallel processing systems provide cost-effective means to high system performance through concurrent activities.

Multiprocessor systems, distributed-memory, shared-memory, and distributed-shared memory are currently the most promising parallel systems to further increase computer performance. Distributed-memory systems have unshared distributed memories among processors of the systems. Shared-memory systems use a single physical memory shared by all processors. In a distributed-shared memory system, the shared-memory is physically distributed to all processors, and a collection of all local memories forms a global address space accessible by all processors. The interconnection networks are used for internal connections among processors, memory modules, and I/O devices in a shared-memory system or among nodes in a distributed memory system. The interconnection networks depend on several factors including topology, routing algorithms, and switching techniques. The network topology defines how the nodes are interconnected by channels. The routing algorithm is defined as the path chosen by a packet to reach its destination. The switching technique determines how and when the router switch is set when a packet header reaches an intermediate node.

## Parallel Computer Memory Architectures

A parallel computer is a system that emphasizes parallel processing. The parallel processing is a suitable manner of information processing that exploits the computing process. Parallel computer systems can be characterized as *pipeline* processors, *vector* processors, *array* processors, *systolic* processors, and *shared memory (multiprocessor)* systems [2]. Pipeline processors refer to those digital machines that provide overlapped data processing in the central processor, in the I/O processor, and in the memory hierarchy. The pipeline processing concept in a computer system is similar to assembly lines in an industrial factory. To achieve pipelining, the input task must be divided into a sequence of subtasks. Vector processors are designed to manipulate vector instructions over vector operands, all the elements of a vector are subjects to a particular instruction simultaneously. They work efficiently only if the arithmetic operations to be performed are vectorized, that is, arranged as continuous streams of data. Although the vector processors are remarkably fast in certain situations, it proved to be very difficult in practical simulation problems to arrange the computations to be performed in sufficiently long vectors. Array processors are well suited for the applications for which they are designed, general purpose computations. They consisted of a one or two dimensional array of processors, with nearest neighbor interconnections. Such an interconnection pattern is very natural for spatially decomposed problems like partial differential equations and image processing. Furthermore, there is host computer supervision which controls the growth of the computation by passing the next instruction to processors. A systolic processor is an extension of the pipelining concept. While a pipeline is a one dimensional, unidirectional flow, the systolic system permits multidirectional flow including feedback. The systolic processors are designed for a special purpose, such as

solving systems of linear equations with special structure, or performing fast Fourier transforms.

Multiprocessor systems, which contain several processors of approximately comparable capabilities, are currently the most promising architectures to further increase computer performance. They have been shown to be very competent for solving problems that can be partitioned into tasks with homogeneous computation and communication patterns. Depending on how the memory is shared, there exist three models of multiprocessor systems, *shared memory systems*, *distributed-memory systems*, and *distributed shared memory systems* [3]. Shared memory systems use a single physical memory shared by all processors. Such systems are also called *uniform memory access* (UMA) systems. Distributed-memory systems have unshared distributed memories among processors of the systems. Such systems are also called *message passing multicomputers*. In distributed shared memory systems, the shared memory is physically distributed to all processors and a collection of all local memories forms a global address space accessible by all processors. Such systems are also, called *non-uniform memory access* (NUMA) systems. In the next three subsections, the three models of multiprocessor systems are discussed. An introduction to High Performance Computing and parallel computer was presented by Hager and Wellein [4].

*Shared-Memory Systems*

The main property of shared-memory systems is that all processors in the system have access to the same memory; there is only one global address space. Typically, the main memory consists of several memory modules whose number is not necessarily equal to the number of processors in the system, as shown in Figure 1. The Intel Paragon

[5], the Thinking Machines Corp, CM-5 [6], and the Meiko CS-2 [7] are examples of shared-memory architectures. In these systems, communication and synchronization between the processors are done implicitly via shared variables.

The processors are connected to the memory modules via some kind of interconnection network. These systems are called *uniform memory access* (UMA), since all processors access every memory module in the same way concerned latency and bandwidth. Each main memory location in the memory is located by a number called its address. Addresses start at 0 and extend to 2n-1 when there are n bits (binary digits) in the address. To extend the single processor model, there are multiple processors connected to multiple memory modules, such that each processor can access any memory module. They are divided into two types, symmetric and asymmetric. In a symmetric shared-memory system, all processors have equal access to all peripheral devices, and they are equally capable of running the operating system kernel and the I/O service routines. In an asymmetric shared memory system, only one processor can execute the operating system and handle I/O, while the other processors execute user codes under supervision of the master processor.

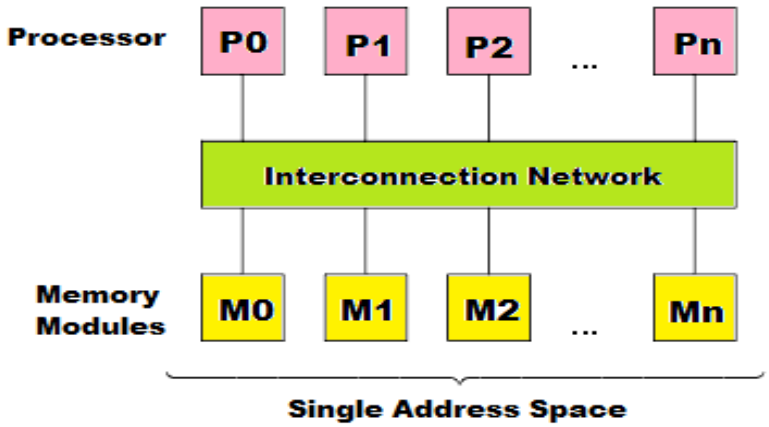



*Figure 1*. [3] A shared memory architecture.

Programming of shared memory systems involves having executable code stored in the shared-memory for each processor to execute. The data for each program will also be stored in the shared memory, and hence each program could access all the data if needed. Programmers create the executable code and shared data for the processors that can be done in different ways, but the final result is to have each processor execute its own program or code sequences from the shared memory.

Shared-memory systems have some advantages, such as a parallel program can be written as a collection of processes that act on common set variables. Hence, writing efficient parallel programs in the shared memory systems is easier than other models. The work needed in shared memory systems to distribute the computation and data over the processors is less than the distributed-memory systems.

*Distributed-memory Systems*

A distributed memory system consists of multiple autonomous processing nodes with local memory modules connected by a common interconnection network. There is no common address space, i.e. the processors can access only their own memories. Communication and synchronization between the processors are done by exchanging messages over the interconnection network. Each computer consists of a processor and local memory. In principle, there are no limits to the number of processors or the total memory, other than the cost of constructing the system. The SGI Origin2000, Cray T3E, and IBM RS/6000 SP are examples of distributed memory architectures [8]. Figure 2 illustrates the connection between the computer modules and a message transfer system.

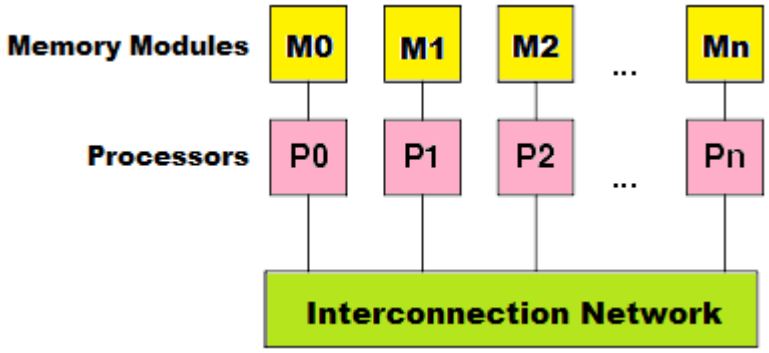



*Figure 2.* [3] Distributed memory architecture.

By using dedicated routers, distributed-memory systems decouple computation and communication functionality in order to improve the performance of both. In such systems, a computer node is attached to a router; which handles a message passing among nodes. A boundary router may be connected to I/O and peripheral devices. The message, which passes between any two nodes, involves a sequence of routers and channels. Several pairs of external channels are connected to define the network topology. A crossbar switch within the router allows the simultaneous transmission of a message between different input and output channels. Moreover, two messages may be transmitted concurrently in reverse directions between neighboring routers. A pair of internal channels connects a router to its local processor/memory. One channel of each pair, injection channel, injects messages into network. The other channel, ejection channel, consumes messages from the network. The architecture of a generic node in distributed-memory systems is illustrated in Figure 3.

Distributed-memory systems can be classified into two approaches, *medium-grain* and *fine-grain* systems. The medium-grain system consists of a few tens of large processors. It uses large word sizes and memory capacities. Examples of medium-grain machines are iPSC/2, and nCUBE2 [8]. The fine-grain system consists of several

thousands of processors. It uses small word sizes and very small memory capacities. Examples of fine-grain machines are Caltech Mosaic [9] and j-machine [10].

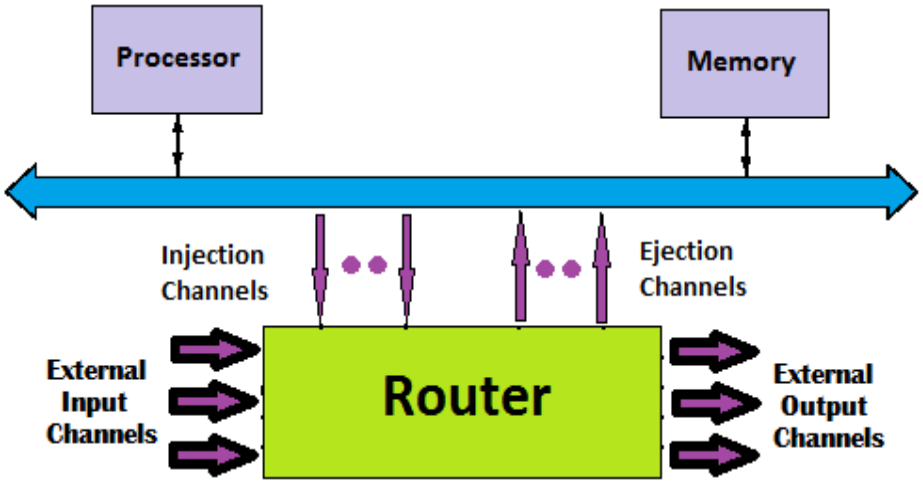



*Figure 3.* [3] Generic node architecture.

Programming of distributed-memory systems still involves separating the problem into parts that are intended to be executed concurrently to solve the problem. Programming could use a parallel or extended sequential language, but a common approach uses message-passing library routines that are inserted into a conventional sequential program for message passing. A problem is divided into a number of concurrent processes. Processes may be executed on individual computers.

Distributed-memory systems have several advantages, such as they require relatively design effort less than shared-memory systems. As the number of processors in the system increases, some points are noticed:

1) The memory size increases in distributed-memory systems, while in shared memory systems, it does not increase.

2) The total memory bandwidth increases in distributed-memory systems, while in shared-memory systems, it remains constant, independent of the number of processors.

3) The processing capability of the system in distributed-memory systems, increases while in shared-memory systems; it may be decreased because of the synchronization [11].

Hence, distributed-memory systems are more scalable than shared-memory systems in building massively parallel computers. A Good overview of distributed-memory systems and parallel computing are presented by Dally [12], Gebali [13].

*Distributed-Shared-Memory Systems*

In a distributed-shared-memory system, the shared memory is physically distributed to all processors, called local memories. The collection of all local memories forms a global address space accessible by all processors. Such systems are also called *non-uniform-memory-access* (NUMA) systems in which the access time varies with the location of the memory word. The memory access time for a local address is less than the access time for remote address, attached to other processors, through the interconnection network. Besides distributed memories, globally shared memory can be added to a multiprocessor system. In this case, there are three memory-access patterns, the fastest is local memory access, the next is global memory access, and the slowest is remote memory access. Examples of such systems are Cedar system [8], SGI Origin [14], Stanford Dash [15], and Stanford Flash [16].

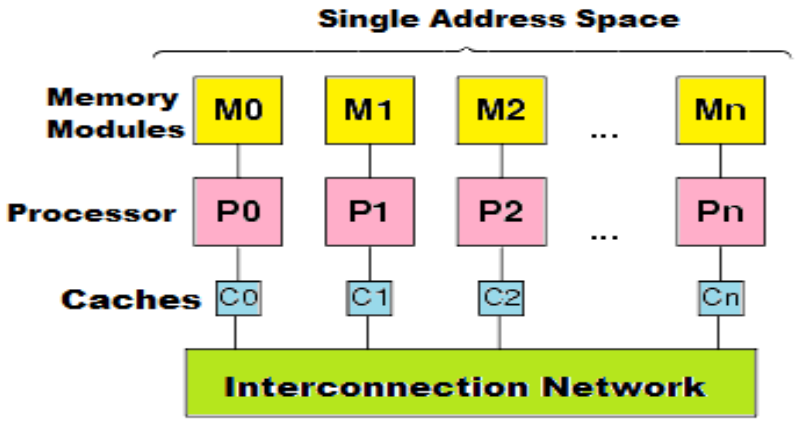



*Figure 4.* [3] A distributed shared memory architecture.

The *cache-only memory* architecture is a special case of a non-uniform-memory access, in which the distributed memories are converted to caches. All caches form a global address space. The processors are divided into numerous clusters. Every cluster is itself an UMA or a NUMA multiprocessor. The clusters are linked to universal shared-memory modules. A distributed-shared-memory systems configuration is shown in Figure 4 [8].

## Basic Network Topologies

In this subsection, different topologies of direct networks are discussed. Each computer found on the network is known as a network node. All topology has its advantages and disadvantages: generally correlated to the price, complexity, dependability and traffic.

### *Linear Topology*

In a linear topology, Figure 5, all nodes connected to LAN as branches on a common line. In a linear network with N nodes, the internal nodes have degree equal to 2 and the terminal nodes have degree equal to 1 while the diameter is N-1. Many devices connect to a single cable *backbone*. If the backbone is broken, the whole part fails. Linear

topologies are fairly easy to set up and do not need much cabling compared to the alternatives.

*Figure 5*. Linear topology.

*Ring Topology*

In a ring topology, Figure 6, all nodes on the same circuit, which forms a continuous loop, is obtained by connecting the two terminal nodes of a linear array with one extra link. It is symmetric with a constant node degree of equal to 2. The diameter is N/2 for a bi-directional ring and N for unidirectional ring. All messages pass through a ring in the equivalent direction. A breakdown in any cable or device disconnects the loop and hence it takes down the whole segment.

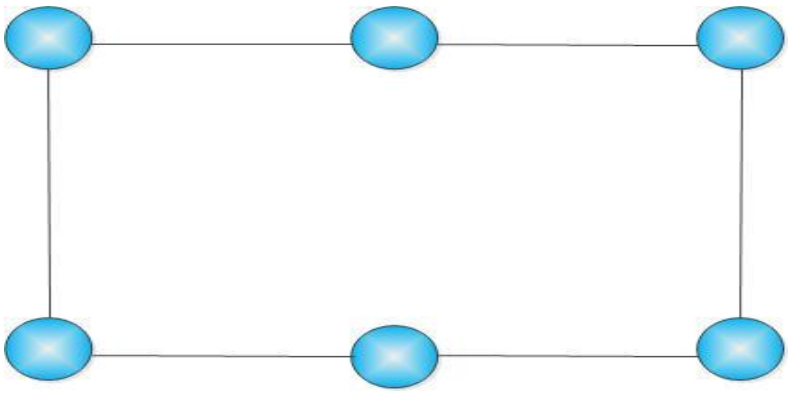


*Figure 6*. Ring topology.

*Star Topology*

In a star topology, Figure 7, every node attached to disconnect lines that direct to center. In a star network with N nodes, the degree of the central node is N-1 while that of other nodes is 1 and the diameter is 2. The star architecture has been used in systems with a centralized supervisor node. A star network has a central connection point – like a hub

or a switch. While it takes more channels, the benefit is that if a channel fails, only one node will be brought down.

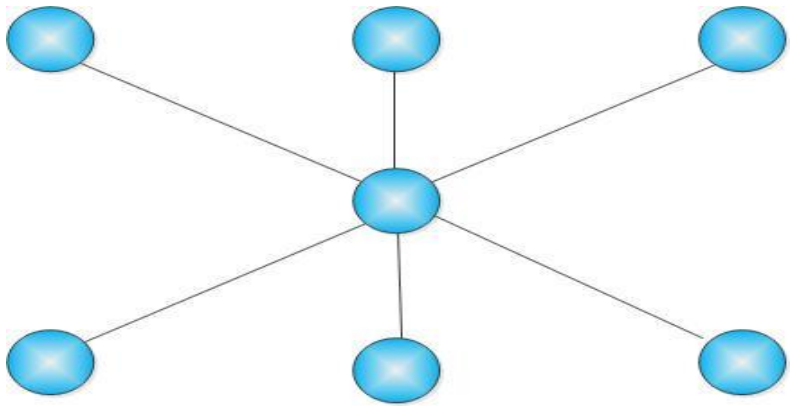


*Figure 7*. Star topology.

*Tree Topology*

In a tree topology, Figure 8, all nodes attached to separate lines that lead to a hub, and then the hubs are connected together (like the branches on a tree) to the main network backbone. The binary tree of k-level contains 2–1 nodes. The maximum node degree is 3 and the diameter is 2(k-1). The tree topology is a combination of linear and star topologies. They are very common in larger networks.

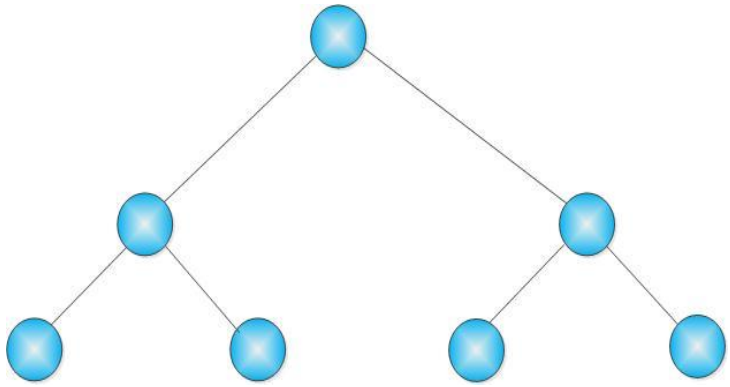
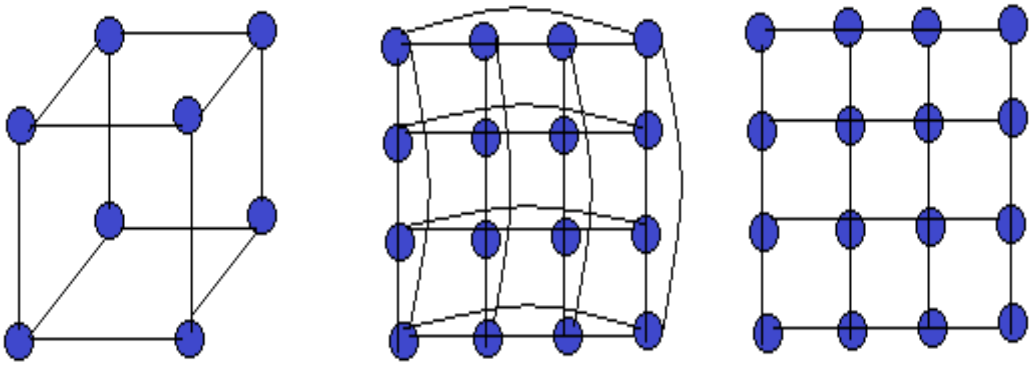


*Figure 8*. Tree topology.

*Hypercube Topology*

A high-dimensional binary n-cube is called *hypercube* topology. An *n- cube* consists of $N=2^n$ nodes spanning along *n* dimensions, with two processing nodes in each dimension. Two nodes *x* and *y* are neighboring nodes if and only if $y_j = ( x_j \pm 1 )$ *mod k* for one *j* and $x_j = y_j$ for all $i \neq j$, $1 < j, j < n$. The first generation multicomputer such as Intel iPSC/1 and n CUBE/2, implemented the hypercube topology. For example, iPSC/1 consists of 128 nodes form a 7-dimensional hypercube with 512 k bytes of local memory per node and 8 I/O ports. While nCUBE/2 consists of 8192 nodes form a 13-dimensional hypercube with 512 Gbytes of local memory per node and 64 I/O boards [17]. Figure 9 (a) illustrates an example of 3-cube networks.



**(a)** **(b)** **(c)**

*Figure 9.* Example of (a) Hypercube, (b) Torus and (c) Mesh.

*Torus Topology*

A low dimensional *k*-ary n-cube is called torus topology. Unlike a hypercube, a torus may contain more than two nodes per dimension. The torus topology has ring connections along each row and along each column of the array, i.e. *wraparound* channels have been added to connect each edge node to the corresponding node on the opposite edge. Each dimension in a *k*-ary n-cube contains the same number of nodes

while dimensions of a torus may contain different numbers of nodes. Two nodes $x$ and $y$ are neighboring nodes if and only if $y_j = (x_j \pm 1) \ mod \ k$ for one $j$ and $x_i = y_i$ for all $i \neq j$, $1 < i, j < n$. Examples of torus architectures include the Torus Routing Chip and the Cray T3D [8]. Figure 9 (b) illustrates an example of ( $3 \times 3$ ) 2D torus networks.

*Mesh Topology*

Mesh network topology is one of the most important interconnection networks. Distributed-memory systems which use mesh topology as their essential architecture have been around for years. They utilize mesh topology because of its simplicity, reliability and good scalability. Also, their significance in achieving high performance, fault tolerant computing for mesh topology has been the focus of research. A 2D mesh with n x n nodes has an internal node degree of 4 (four neighbors), one in each of four directions: east, south, west, and north. A number of large research and commercial multicomputer systems have been built based on 2D and 3D mesh topologies, including Illiac IV, MPP, DAP, CM-2, Intel paragon, Goodyear MPP and Blue Gene Supercomputer [8]. All mesh communication channels and MRCs are built on a backplane. The 3D-Smesh network is implemented in the third generation of multicomputers. The Mosaic C project is designed to use VLSI-implemented nodes, each containing a 14-MIPS processor, 20-Mbytes/s routing channels, and 16 Kbytes of RAM integrated on a single chip [8]. Mosaic consists of 16,348 nodes [18]. Figure 9 (c) illustrates an example of ($3 \times 3$) 2D mesh networks.

## Network Switching Techniques

Switching concerns the form in which link resources are allocated to messages. In most distributed-memory systems, a message enters the network from a source node and is switched towards its destination through a series of routers at intermediate nodes.

The switching technique is the mechanism that removes data from an input channel and puts it on an output channel. Also, the switching technique defines the hardware and software protocols for transmitting and buffering data when sending a message between neighboring routers. The transmission time which is extremely dependent on the switching technology is used to direct messages through the network. Different switching techniques have been proposed for supporting communication across the network. The most common techniques, including *store-and-forward*, *virtual cut through*, *circuit* switching, and *wormhole* switching, are presented in the next subsections.

For each switching technique the computation of the transmission time of an $M$ bit message in the absence of any traffic will be considered. The phit size and flit size are supposed to be equal to the physical data channel width of $W$ bits. The routing header is assumed to be one flit, thus the message size is $M + W$ bits. A router can make a routing decision equal to $t_r$ seconds. The physical channel between two routers works at $B$ Hz, i.e., the physical channel bandwidth is $BW$ bits per second. The propagation delay across this channel is denoted by $t_m = 1/B$. When a path has been structure through the router, the switching delay is denoted by $t_w$. The router internal data paths are supposed to be coordinated to the channel width of $W$ bits. Thus, in $t_s$ seconds, a $W$ bit flit can be transferred from the input of the router to the output [17].

*Store-and-Forward (SF) Switching*

Store-and-forward mechanism had been used to route messages in many multicomputer systems. This switching technique is sometimes called *packet switching*. In store-and-forward switching, a message destined for a node that is not directly connected to the source node must be received in its entirety at each intermediate node before forwarded to the next node. Therefore, the transmission time, the delay from the

beginning of sending a message at the source node until the destination node receives it, is proportional to the distance between the source and destination nodes. The hop step consists of copying the whole packet from one output buffer to the next input buffer. Routing decisions are completed by each intermediate node only after the entire packet was totally buffered in its input buffer.

The transmission time of store-and-forward switching message can be computed as follows [17].

$$T_{SF} = D \times ( t_r + (t_m + t_w) \times \lceil (M+W) / W \rceil )$$

A time space diagram of the progress of a packet across three links is shown in Figure 10.

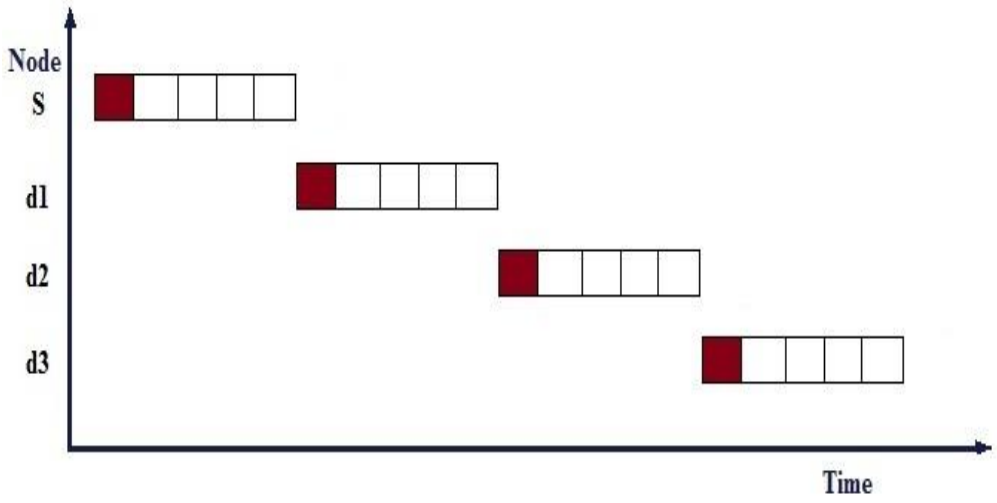



*Figure 10.* [17] Time space diagram of store-and-forward switching message.

The store-and-forward technique is valuable when messages are short and frequent, since one transmission makes busy at most one channel from the whole path. The necessity to buffer the entire packet makes the router design expensive and slower, or the packet size is restricted. The communication latency is proportional to the product of the packet size and distance between the source node and destination nodes. As a result

designers try to pursue shortest path routing and use small diameter networks. Routing algorithms based on store-and-forward switching techniques can found in [19].

*Virtual Cut-Through (VCT) Switching*

Virtual cut-through (VCT) switching is the most complicated and expensive technique among switching techniques. In virtual cut-through a message is buffered at an intermediate node only when the desired outgoing channel (or channels if there is routing choice) is/are already in use. In this technique transmission time, in the absence of contention, becomes largely independent of the distance travelled by the message.

The transmission time of a message that effectively cuts through each intermediate node can be computed as follows [17].

$$T_{VCT} = D \times (t_r + t_m + t_w) + max\,(t_m\,,\, t_w) \times \lceil M/W \rceil)$$

Figure 11 illustrates a time space diagram of a message transmitted using virtual cut-through switching wherever the message is congested after the first link waiting for an output channel to be free [20].

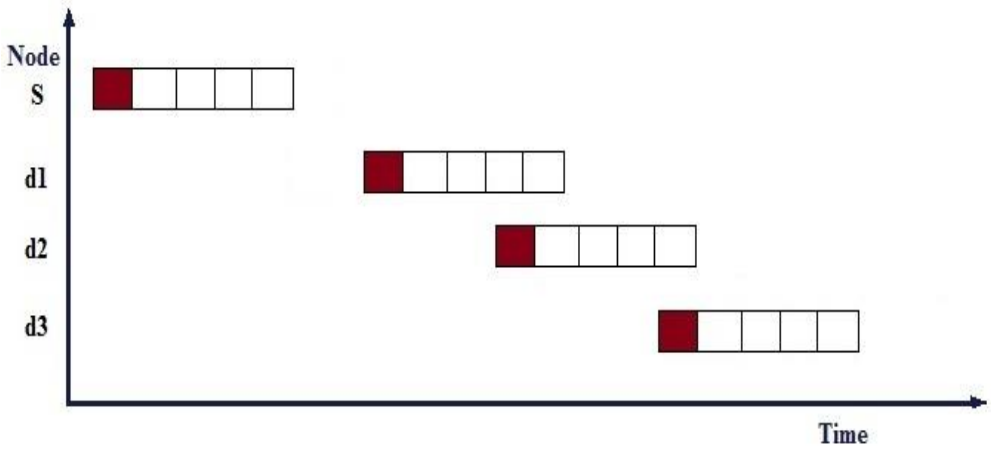



*Figure 11.* [20] Time space diagram of a virtual cut-through switching message.

The header flit is the one that holds routing information and consequently each incoming data flit is merely sent along the same output channel as its predecessor.

Consequently, broadcast of different packets cannot be inserted or multiplexed over one physical channel [20], [21]. Routing algorithms based on virtual cut-through switching techniques can be found in [19].

*Circuit Switching (CS)*

In circuit switching technique, the communication between a source node and a destination node has two phases: *circuit establishment* and *message transmission*. A physical path from the source node to the destination node is held in reserve proceeding to the broadcast of data by inserting a *routing probe*, which holds destination address and several control information. This routing probe developments towards the destination node keeping physical links as it is transferred through intermediate nodes. When the probe reaches the destination node, a whole path has been established and a response is conveyed back to the source node [17]. The circuit is unrestricted either by the destination node or by the last bits of the message. The acknowledgments in the Intel iPSC/2 routers [22] are multiplexed in the opposite direction on the similar physical line as the message.

The transmission time of a circuit switched message can be computed as follows [17].

$$T_{circuit} = t_{setup} + t_{data}$$

$$= D \times (t_r + 2 \times (t_m + t_w)) + ((1/B) \times \lceil M/W \rceil)$$

A time space diagram of the transmission of a message is shown in Figure 12. The shaded boxes signify the times during which a link is busy. The space between these boxes is the time to process the routing header, and the intra-router propagation delays. The clear box defines the time the links are busy conveying data through the circuit [17].

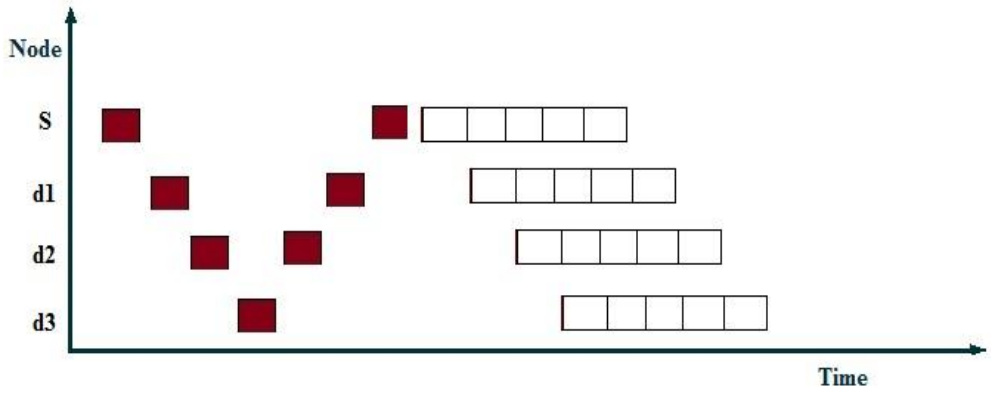



*Figure 12*. [17] Time space diagram of a circuit switching message.

The circuit switching technique is valuable if messages are random and long, i.e. the transmission time is longer than the setup time. In case of short messages, the entire physical circuit is earmarked during the entire setup and transmission part. At each router on the path, the probe is buffered; however, the data bits are not. The circuit operates as a single wire form the source to the destination. Also, messages are not necessary to be divided into fixed-length packets, but they can be conveyed as incessant flow of bits along the setup circuit. Therefore, there are no restrictions on the length of conveyed data [17].

*Wormhole (WH) Switching*

In most parallel computer systems, a message comes into the network from a source node and is switched or routed to its destination using a sequence of intermediate nodes. Throughout this dissertation, all our algorithms are based on wormhole switching technique. In wormhole switching (sometimes referred to as wormhole routing), a packet is transmitted between the nodes in units of flits, the smallest units of a message on which flow control can be performed. The header flits of a message hold all the essential routing information and remaining flits hold the data elements. The flits of the message are routed through the network in a pipelined fashion. From the time when only the header

flit hold the routing information, all the trailing flits follow the header flits alongside. Flits of two altered messages cannot be inserted at any intermediate node. When the header flit is congested, then all the trailing flits reside in the buffers at the intermediate nodes [17].

The transmission time of a wormhole switched message can be computed as follows [17].

$$T_{WH} = D \times (t_r + t_m + t_w) + max\ (t_m\ ,\ t_w) \times \lceil M/W \rceil)$$

Wormhole routing worked on a good way on simple, small, inexpensive, and fast routers. Consequently, it is the most mutual switching technique used currently in commercial machines. In addition, wormhole routers use frequently only input buffering. The main disadvantage of this switching technique is obstructive resources in case of stalled pipelines. Subsequently blocking chains of buffers can simply cause snowball effect, WH switching is very deadlock-prone. This subject is correlated to the idea of virtual channels.

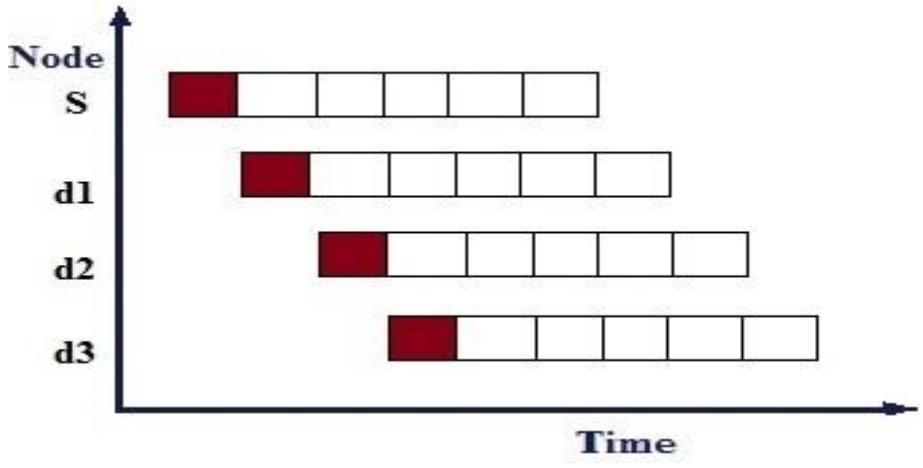



*Figure 13*. [17] Time space diagram of wormhole switching message.

The time space diagram of a wormhole switched message is shown in Figure 13. The clear rectangles illustrate the propagation of flits through the physical channel. The shaded rectangles illustrate the propagation of header flits through the physical channels.

Wormhole routing is the supreme extensively used switching technique in massively parallel computer systems. Examples of the multicomputer systems that apply wormhole routing are nCUBE-2 [23] (hypercube), Symult 2010 [24], Intel Paragon [5], and Intel/DARPA's Touchstone Delta [5] (2D mesh), MIT-J-machine [25] and Caltech Mosaic (3D mesh), and Cray T3D [25] (3D torus). Also, wormhole has been approved in systems that use indirect switch based networks, such as TMC CM-5 and IBM SP series [26]. Performance of wormhole routing switching technique has been studied for several topologies. A simulation study of wormhole routing in 2D mesh was introduced in Chittor and Enbody [27] and a performance analysis of k-ary n-cube networks was introduced in Dally [12]. Wormhole routing techniques in direct networks were surveyed in Chen et al. [28], Ni and McKinley [29].

## Multicast Routing Algorithm

The communication mechanism is one of the most important research areas in parallel computing systems. Its support provided by the system software and hardware for delivering a message from one node to another node. Hence, efficient communication mechanism among nodes is critical to the performance of message passing systems. In this section, the essential communication operation, multicast, is discussed. Also, to demonstrate the classifications of the multicast, a survey of deadlock-free multicast routing algorithms for direct networks is introduced.

Multicast communication has numerous uses in distributed-memory systems and large-scale multiprocessors. Firstly, numerous parallel applications, including parallel

search algorithms and parallel graph algorithms, have been shown to benefit from the use of multicast services. Secondly, multicast is useful in the SPMD (single-program, multiple-data) mode of computation, in which the same program is executed on a different processor with different data. In particular, multicast is essential to numerous operations, such as replication and barrier synchronization, that are supported in data parallel languages. Thirdly, if a distributed shared-memory paradigm is supported, then multicast services may be used to efficiently support shared-data termination and updating. Finally, it is useful in many parallel numerical algorithms, including matrix multiplication, matrix transpose, and Gaussian elimination [30], [31].

Providing support for multicast communication involves several requirements. First, it is desirable that the message delays from the source to each of the destinations are as small as possible. One solution is by sending a detach copy of the message to each destination along the shortest paths, but the enlarged traffic load resulting from these copies might delay the progress of the message. The second requirement is that the amount of network traffic must be minimized. The third necessity is that the routing algorithm is not being computationally complex. Therefore, heuristic algorithms are used and must be deadlock-free.

Basic multicast routings can be classified as unicast-based, tree-based and path-based [30]. In unicast-based multicast routing algorithms, a source node sends messages to its set of destinations through sending a series of separate unicast messages to each destination. It requires a great number of startups to send a message to a large set of destinations. Tree-based algorithms endeavor to distribute the message to all destinations in a single multi-head worm that splits at some routers and replicates the data on many

output ports. Path-based routing algorithms permit a worm to hold sorted list of multiple destination addresses in its header flits.

In the next subsections, the three multicast techniques are discussed and some algorithms of each technique are surveyed.

*Unicast-Based Techniques*

In unicast-based multicast algorithms, a source node broadcast a message to its set of destination by sending multiple unicast messages, which are routed independently through the network. In these techniques, no local processors other than the source and destination processors are required to handle the message, but only routers at the intermediate nodes are involved in forwarding the message [32]. Hence, the message is passed from a source to a destination node in one step. They require no additional hardware support, but additional software is added to support multicast. They have budding to achieve well when the average number of destinations for each multicast message is small [33]. In addition, routing of each individual message can take place using unicast routing; hence, there will be no additional deadlocks in this solution. Many recent distributed shared memory multiprocessors use this technique to perform cache invalidation in directory schemes [34], [35].

The *separate addressing* is one of the unicast-based multicasting routing techniques, in which the source node sends directly a separate copy of the message to all destination nodes [8]. The separate addressing routing, sometimes called *individual*, requires *d* startup latency to complete a multicast with *d* destinations. A communication step is the time required for a message to be sent from one node to another. In wormhole routed networks, the message startup latency is generally several orders of magnitude larger than network latency. Hence, it is desirable to minimize the number of startup

latencies used to deliver the message to all of its destinations. Figure 14 illustrates an example of a separate addressing algorithm on a 2D mesh.

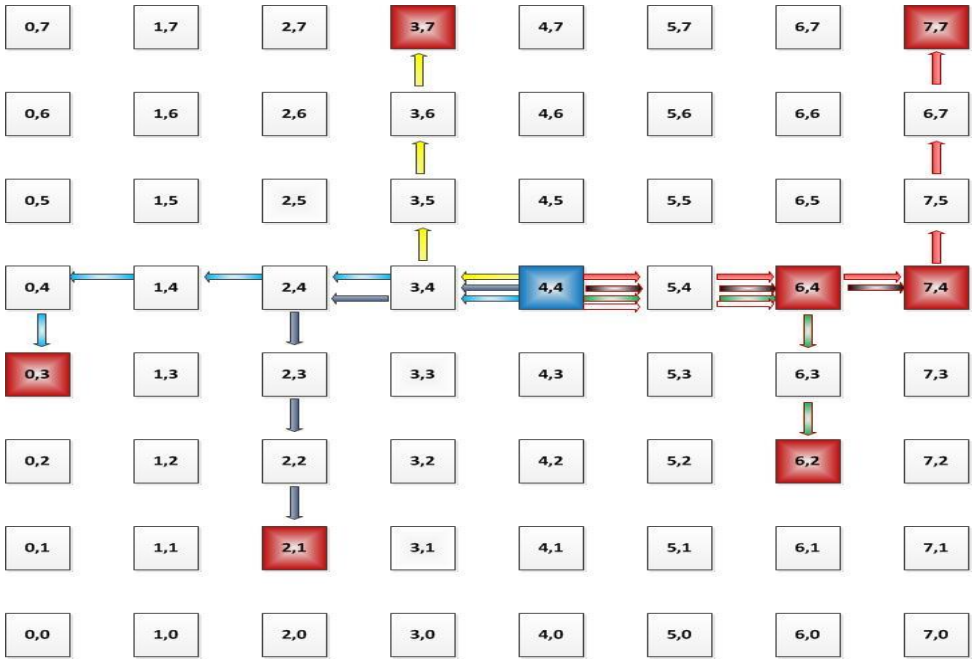



*Figure 14.* An example of a separate addressing algorithm on a 2D mesh.

An alternative unicast-based multicasting technique is to use a *binomial tree* of unicast message [32], in which the number of sources is doubled in all steps. The binomial tree can be considered as a sequence of communication steps, in which each step incurs a startup time. In the first step, the source node sends the message to some subset of the destination nodes. In each subsequent step, each node holds a copy of the message and forwards it to at most one new destination node that has not yet received the message. Such multicast binomial tree algorithms are used to minimize the number of communication startups required for unicast-based multicasting.

The competence of an algorithm is determined by the essential number of communication startups for the multicast to finish. In a binomial tree algorithm, the number of nodes holding the message can at most double with each step. Thus, it can be

easily observed that the lower bound on the number of communication setups required to complete a multicast to *d* destinations using multicast binomial tree routing is equal to $\lceil \log_2 (d+1) \rceil$. A multicast binomial tree algorithm is called optimal if achieves the lower bound of communication steps. Also, the degree of channel contention knowledgeable among the messages of the multicast is significant for the efficiency of the algorithms. The channel contention, which is sometimes called step contention, occurs when two unicast messages in the same communication step contend for a common physical channel. A generalization of step contention is called depth contention. The depth contention happens when a node is scheduled to broadcast the message in communication step, probably competing with a unicast for the same message transmitted at communication step $j$, $j \leq i$. This will guarantee that the depth contention freedom is stronger than the step contention freedom.

Many unicast-based binomial tree multicasting algorithms [30]-[32], [36]-[39] have been proposed. Some of them [31], [32] are briefly discussed in this subsection.

Some binomial tree multicasting routing algorithms for *k-ary n-cube* networks were presented in [31], [32]. Robinson et al. [31] extended the U-mesh algorithm to introduce an optimal binomial tree multicast routing algorithm, U-torus, for n-dimensional torus networks. The U-torus algorithm is applied to unidirectional and bidirectional tori. Also, it uses the deterministic dimension-ordered routing. It avoids contention between the ingredient unicast messages. McKinley et al. [32] proposed binomial tree multicast routing algorithms for one-port *n*-dimensional mesh and hypercube networks that use the deterministic dimension-ordered routing. These algorithms are optimal and prevent contention among the ingredient unicast messages. For example, in the U-mesh algorithm, the source and destination addresses are sorted

into a dimension-ordered chain, denoted Φ, at the time when multicast is initiated by calling the U-mesh algorithm [32]. The source node successively divides Φ in half. If the source node in the lower (upper) half, then it sends a copy of the message to the smallest (largest) node, according to a dimension order relation, in the upper (lower) half. That node will be accountable to deliver the message to other nodes in the upper (lower) half, using the alike U-mesh algorithm. In addition to the data, every message carries the address of destinations for which the receiving node is accountable. At each step, the source continues this procedure until Φ contains only one address. The U-mesh only guarantees contention freedom among the worms of a given step.

Unicast-based multicasting algorithms have some disadvantages. They allow a message to be delivered to only one destination, which leads to multicast operations being implemented as multiple phases of multicast message exchange. Thus, contention freedom must be guaranteed not only among the worms of a given phase, but also among worms in different phases. They also require additional software to support multicast. The essential disadvantage of unicast-based algorithms is the large number of communication startup delays, which they require. The ratio of communication startup time to propagation time is quite high on current generation parallel systems. It is typically in order of 1 to 20 microseconds [39], unicast-based multicasting techniques lead to very high latency. For example, to send a message to 512 destinations with 10 microsecond communication startup time, the separate addressing technique takes 5120 microseconds and binomial tree technique takes 100 microseconds. In addition, multiple unicast messages need more network channels, therefore affecting other network traffic and reducing the overall throughput of the network.

*Path-Based Techniques*

In order to reduce the large number of startups obtained by unicast-based multicasting algorithms, path-based algorithms were proposed [40]. Path-based multicast algorithms allow a worm to contain a sorted list of multiple destinations addresses in its header flits. They use a simple hardware mechanism to allow routers to absorb the message on internal channels while concurrently forwarding a copy of the message on an output channel transmitted to the residual destinations. Current wormhole routers contain logic to take up and forward flits. In this scheme, a message can be delivered to several destinations with the same startup latency as a message sent to a single destination. The destinations of a multicast message are partitioned into a tiny number of subsets, and a copy of the multicast is broadcast to each subset of destinations. Each copy of the message visits its destinations in a predefined order. Diverse copies of a multicast message use displace sets of physical channels and are routed separately of one another, to prevent cyclic dependence and deadlock. Messages pursue shortcuts to decrease path length to assurance that a unicast message constantly follows the shortest path. Duato [41] urbanized the theory leading of design of path-based multicast routing algorithms.

Some path-based multicast routing algorithms for direct networks, [26], [33], [42]-[44] are briefly discussed in this subsection.

The *Hamiltonian* path in the network is used to develop some path-based multicast algorithms for mesh and hypercube networks. It is an undirected path which visits every node in a graph exactly once. The assignment of the label to a node is based on the position of that node in a Hamiltonian path, where the first node in path is labeled 0 and the last node in the path is labeled *N-1,* where *N* is the network size. Two Hamiltonian path-based algorithms for 2D networks, the *dual-path* multicast and the

*multipath* multicast were proposed by Lin et al. [42]. The dual-path routing divides the destination nods into two disjoint subsets, $D_H$ and $D_L$, where every node in $D_H$ has a higher label than that of the source node and where every node in $D_L$ has a lower label than that of the source node. A copy of the multicast message is sent to each subset of the destinations. Each copy of the message visits its destination nodes sequentially according to a defined routing function. Diverse copies of a multicast message use displace sets of physical channels and are routed independently of one another. The multipath algorithm has the same rules of the dual-path algorithm but divides the destination nodes into four disjoint subsets. When the source node is taken as the origin, all the destination nodes in a subset are in one of the four quadrants. The dual-path routing requires only two startups to send a message to any set of destinations, at the same time as the multipath routing requires four startups but often uses short paths to all destinations. The dual-path and the multipath algorithms offer deadlock-free routing of multicast messages. Also, they provide minimal routing of unicast messages, and either algorithm can be used to route unicast and multicast messages simultaneously in a common framework. Figure 15 illustrates an example of the dual path algorithm on a 2D mesh.

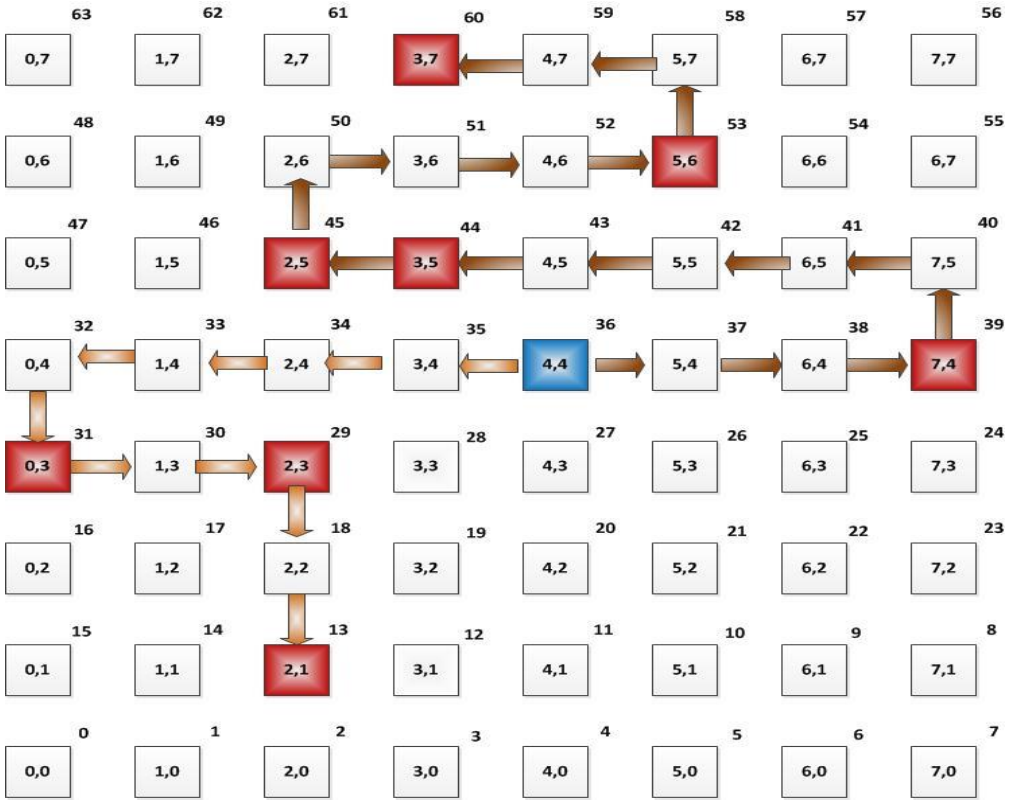



*Figure 15.* An example of the dual path algorithm on a 2D mesh.

Robinson et al. [26] described and evaluated some path-based multicast algorithms for unidirectional wormhole routed torus networks. The first algorithm, *S-torus*, uses a single multi-destination message with single startup to reach the message to all destinations. The second algorithm, *M-torus*, is a generalized multi-phase multicast algorithm, in which a combination of multi-destination messages with multiple startups to reach a message to all destinations. Every copy of the message visits its destination nodes consecutively according to a distinct routing function. These algorithms are deadlock-free and produce contention-free multicast communication by requiring each multi-destination message to visit its destination nodes in an order corresponding to a *Hamiltonian Circuit*. A Hamiltonian Circuit (HC) begins at the node (0, 0) and, at each node $u$ on HC, the next node is the neighbor, $u^d$, that minimizes $d$ under the constraint that $u^d$ does not already

precede *u* on HC, where $u^d$ is the node adjacent to *u* in dimension *d*. The advantage of S-torus is the multicast operation requires only one communication step, which is useful for long messages.

Boppana et al. [33], [43] proposed a multicast routing algorithm called the *column-path* algorithm for mesh and torus networks. In a *k x k 2D* mesh, the column-algorithm partitions the set of destinations of a multicast message into at most *2k*, such that there are at most two message copies directed to each column in the mesh. If a column holds one or more destinations of a multicast communication in the same row or in rows above that of the source, subsequently one message copy will send to service all those destinations. Likewise, if a column holds one or more destinations of a multicast communication in the same row or in rows below that of the source, then one message copy will send to service all those destinations. If all destinations of a column are either below or above the source node, then one message copy will send to service all those destinations. Messages are routed using row-column or e-cube routing method; therefore, the column-path algorithm is well-matched with the e-cube algorithm. Figure 16 illustrates an example of the column-path algorithm on a 2D mesh.

Abd El-Baky [44] proposed two efficient path-based multicast wormhole routing algorithms for 2D torus parallel machines. They used the concept of partitioning the torus into meshes. They entail shifting the origin of the torus network so that the source node always appears to be in (or closer to) the center of the network. This is possible because the torus network is symmetric.

Abd El-Baky [44], the first algorithm, *Torus Dual-Path* (TDP) algorithm uses the vertical wraparound channels to divide the torus into two equal meshes. The first, $M_H$, contains the nodes whose y-coordinates are between that of the source node and that of

the source node plus or minus *n/2* while the other, $M_L$, contains the remaining nodes in the torus. Also, it divides the destination set D into two subsets, $D_H$ and $D_L$, where $D_H$ contains the destination nodes in $M_H$ and $D_L$ contains the destination nodes in $M_L$. The messages will be sent to the nodes in $D_H$ and $D_L$ using the high-channel and the low-channel networks, respectively. Thus, it requires at most two startup times.

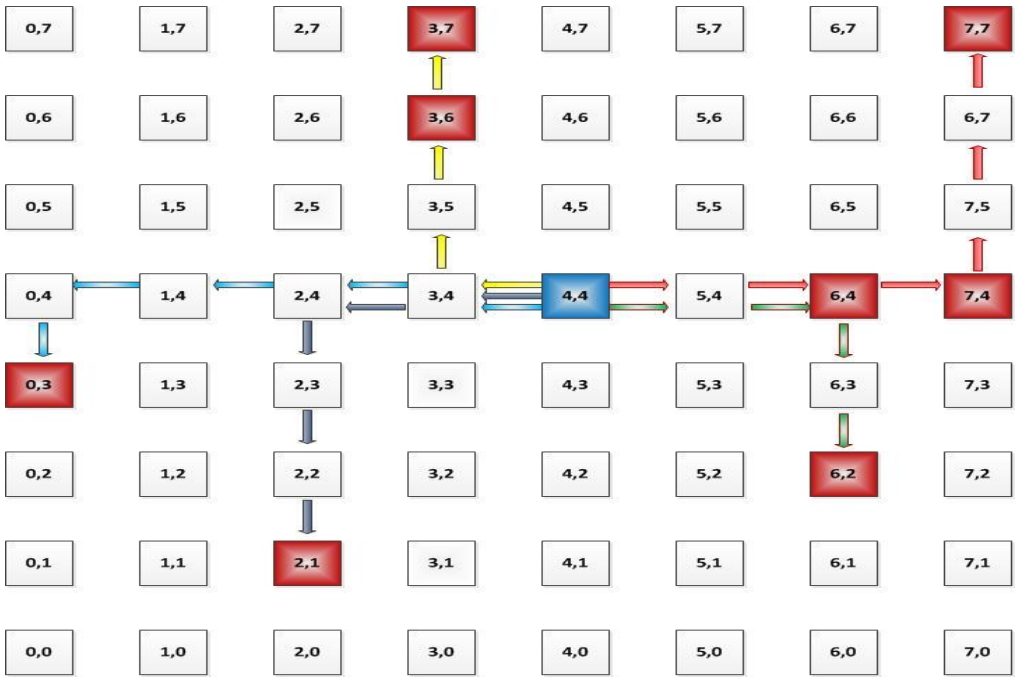



*Figure 16.* An example of the column path algorithm on a 2D mesh.

The second algorithm [44], *Torus Multi-Path (TMP)* algorithm uses the horizontal wraparound channels to divide the torus into four equal width meshes. First, the torus is divided into two meshes, the upper mesh, MU, and the lower mesh, ML. The upper mesh subnetwork MU contains the nodes whose y-coordinates are greater than that of the source node, while the lower mesh subnetwork ML contains the remaining nodes in the torus. Secondly, the two meshes MU and ML are further divided. The mesh MU is divided into two submeshes, the upper center mesh, MUC, which contains the nodes whose x-coordinates are between that of the source node and that of the source node plus

or minus n/2. The other is the upper boundary mesh, MUB which contains the remaining nodes in MU. In a similar manner, the mesh ML is divided into two submeshes, the lower center mesh, MLC, and the lower boundary mesh, MLB. The boundary meshes, MUB and MLB, use the horizontal wraparound channels to connect their partitions. Also, the TMP algorithm divides the destination set D into four subsets, DUC, DUB, DLC and DLB, where DUC contains the destination nodes in MUC and DUB contains the destination nodes in MUB and so on. The messages will be sent from the source node to the nodes in DUC and DUB uses the high-channel network, and to the nodes in DLC and DLB uses the low-channel network. The TMP Algorithm requires at most four startup times. Figure 17 illustrates message transition by using TMP algorithm.

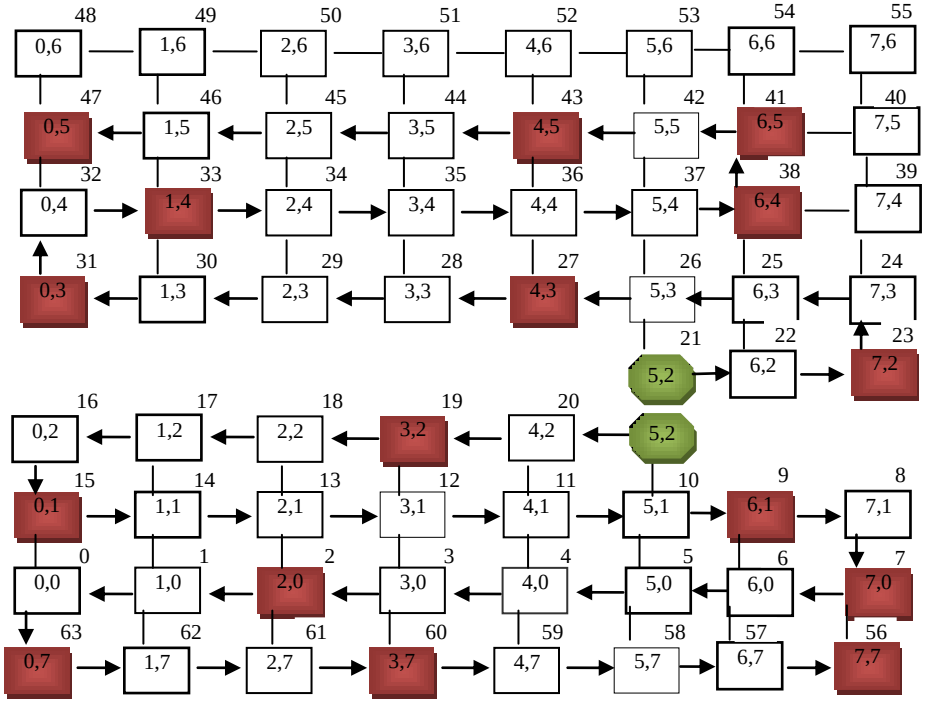



*Figure 17.* [44] Message transition by using TMP algorithm.

*Tree-Based Techniques*

Tree-based routing algorithms challenge to deliver the message to all destinations in single *multi-head* worm that splits at some routers and replicates the data on multiple

output ports. As path-based algorithms, tree-based algorithms use a simple hardware mechanism to allow routers to absorb the message to local processors while concurrently forwarding copies of the message on output channels enroute to the residual destinations. The path followed by every copy may supplementary branch in this method until the message is delivered to all destination nodes. In the literature, two approaches have been planned for replication of data in tree-based schemes, *synchronous* and *asynchronous*. Synchronous replication schemes need that all branches of the multi-head worm proceed in lock–step [30]. As a result any branch of the multi-destination that is blocked can block all other branches. Also, it needs some kind of feedback architecture to guarantee that the flits proceed in lock–step. To prevent deadlock under synchronous replication, deadlock avoidance algorithms have been proposed that arbitrate between multicast packets at a router to prevent cyclic wait. In asynchronous replication schemes, different heads of a multi-head worm can progress independently through the network. Also, an arriving multicast worm at an input buffer of a switch is read by multiple output ports. *Bubble flits* are inserted where required obviating the need for a hardware synchronization mechanism [30]. Asynchronous replication schemes may be ideal for a practical implementation due to the next reasons. Firstly, they do not require the costly feedback architecture required under synchronous replication. Secondly, they are more efficient because blocked branches do not block other branches. Finally, it works effectively because current routers already offer relatively large buffers. To avoid deadlock under synchronous replication, routers must be set with buffers large enough to store the largest packet in the system. Figure 18 illustrates an example of a tree-based algorithm on a 2D mesh.

Many tree-based multicasting algorisms [30], [40], [45]-[52] have been proposed. Some of them, [30], [40], [45], [46], [50]-[52], are briefly discussed in this subsection.

Shaheen [30] proposed a tree-based multicasting wormhole algorithm for arbitrary interconnection topologies, MURA. It is a deadlock-free and provides a general solution for multicasting in any direct network. It needs only fixed-sized input buffers that are independent of highest message length, and it uses a simple asynchronous flit replication mechanism. In this algorithm, wormhole switching is selected as the switching technique to route the message to the destination nodes. A multicast message is first forward to the least common ancestor (LCA), of the set of destinations. When the message has reached at the LCA, all succeeding routing is limited to down tree channels. The channel of the network is defined to be *up* or *down* channels depending on the source node coordinates. The head of the worm will require splitting at the LCA into a multi-head worm and the heads of these multi-head worms may split regularly in order to reach all of the destinations.

Lin and Ni [40], the *double channel* XY routing scheme in 2D mesh networks, was proposed. It is a deadlock-free multicast algorithm based on XY unicast routing. It avoids cyclic channel dependencies by using two virtual channels for each physical channel in the 2D mesh. Thus, it partitions the network into four subnetworks, $N_{E\text{-}N}$ , $N_{W\text{-}N}$, $N_{W\text{-}S}$, and $N_{E\text{-}S}$. The subnetwork $N_{E\text{-}N}$ contains the unidirectional channels with addresses $[(i\ ,j),(i+1,j)]$ and $[(i,j),(i,j+1)]$ and the subnetwork $N_{E\text{-}S}$ contains the channels with addresses $[(i\ ,j),(i+1,j)]$ and $[(i,j),(i,j-1)]$ and so on. It can be easily verified that the four subnetworks use disjoint sets of virtual channels. The destination set D is divided into at most four subsets, $D_{E\text{-}N}$, $D_{W\text{-}N}$, $D_{W\text{-}S}$, and $D_{E\text{-}S}$. The set $D_{E\text{-}N}$ contains the destination nodes to the upper right of the source node and so on. In each subnetwork, the

multicast message is sent to the destinations using a tree that routes the message according to the XY routing. The message will be sent to $D_{E\text{-}N}$ through subnetwork $N_{E\text{-}N}$, to $D_{E\text{-}S}$ through subnetwork $N_{E\text{-}S}$, and so on.

Malumbres et al. [45] proposed an asynchronous tree-based algorithm based on pruning blocked branches, which is effective only for short messages. Also, Wang and Blough [46] proposed an asynchronous tree-based algorithm based on pipelined circuit switching rather than wormhole routing. It avoids deadlock by allowing backtracking but it does not guarantee delivery of all messages.

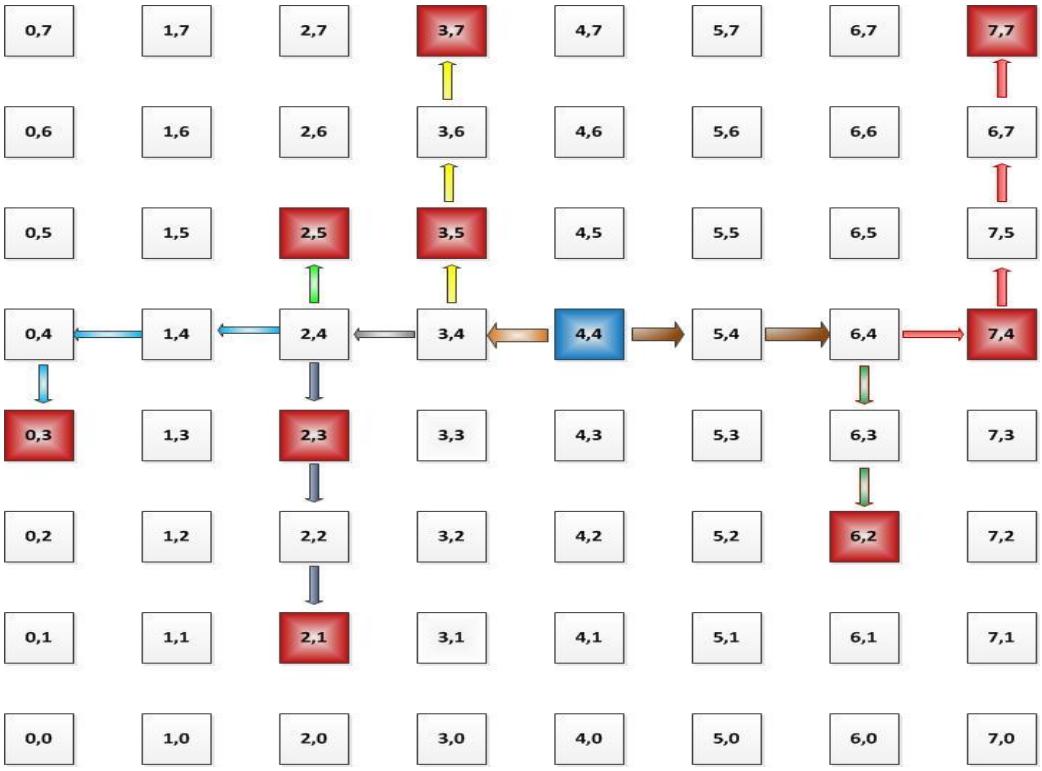



*Figure 18.* An example of a tree-based algorithm on a 2D mesh.

Wu and Chen [51] proposed a tree-based multicast algorithm based on a relatively new switching technique called pipelined circuit switching PCS [52] and multicast-PCS [46]. In a PCS (multicast-PCS), the header (multiheader) is delivered first through the path (tree) setup phase. Once a path (tree) is kept by the header, an acknowledgement is

sent back to the source. As soon as the source receives the acknowledgement, data is sent through the path (tree) in a pipelined fashion. The algorithm always establishes a minimal path to each destination. Also, fault information of a fault block is spread to a restricted number of nodes in the neighborhood so that multiheader can keep away from the fault before reaching it. If the source satisfies certain conditions, the algorithm can set up a multicast tree such that each destination (a leave node in the tree) is reachable through a minimal path in the tree. Wu and Chen applied the algorithm in 2D mesh, 3-D mesh (the MIT J-machine [25]) and tori (Cray T3D [53] and Cray T3E), which are meshes with warp around links.

Moharam et al. [50] proposed an efficient algorithm (YOMNA) to find a deadlock-free multicast wormhole routing in 2D mesh parallel machines. YOMNA algorithm assigns a label for each node based on the position of that node in a Hamiltonian path. YOMNA algorithm creates the routing decision at each sending node. The message may be sent through two paths. It divides the network into two subnetworks. The high-channel subnetwork contains all of the channels whose direction is from lower-labeled nodes to higher-labeled nodes, and the low-channel subnetwork contains all of the channels whose direction is from higher-labeled nodes to lower-labeled nodes. At the source node, YOMNA algorithm divides the network into two subnetworks, $N_U$ and $N_L$, where every node in $N_U$ has a higher label than that of the source node and every node in $N_L$ has a lower label than that of the source node. YOMNA algorithm also divides the destination set D into two subsets, $D_U$ and $D_L$, where $D_U$ containing the destination nodes in $N_U$ and $D_L$ are containing the destination nodes in $N_L$. The messages will be sent from the source node to the nodes in $D_U$ using the high-

channel network and to the destination nodes in $D_L$ using the low-channel network. Figure 19 illustrates message transition by using YOMNA algorithm.

The message transmission in YOMNA technique is made according to the following method. Each sending node examines the above (below) neighboring node in the high-channel (low-channel) subnetwork. In the case where the above (below) neighboring is not a destination, the sending node sends the message together with the destination set $D_U$ ($D_L$) to the neighboring which has maximum (minimum) label and having lower (higher) label than that of the first destination in $D_U$ ($D_L$). In the case where the above (below) neighboring is a destination node, the sending node replicates the message and sends it together with its header to the above (below) neighboring. The message header contains the destination nodes which have higher (lower) label than or equal to that of the above (below) neighboring. The sending node sends the other copy of message together with its header to the next horizontal neighboring node. In this case, the message header contains the remaining destination nodes.

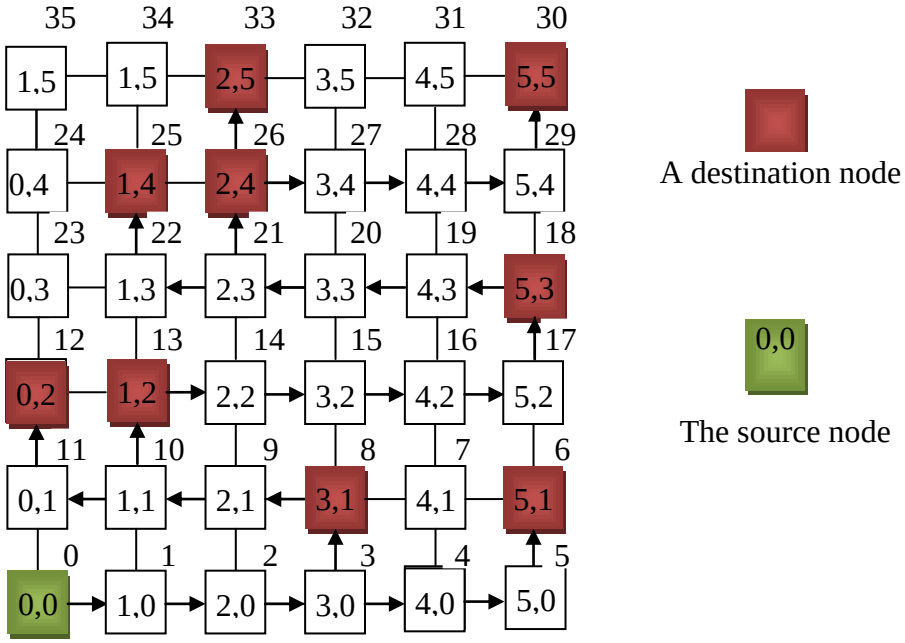



*Figure 19.* [50] Message transition by using YOMNA algorithm.

Upon receiving the message, each receiving node determines whether it is the first destination node. If so, it is removed from the destination nodes and receives the message. At this point, if the sets of the destination nodes are not empty, the algorithm continues according to the previous method. YOMNA algorithm is efficiently used in all cases especially when the size of the network is large (massively parallel systems), and average destination number in the networks is large.

# CHAPTER III

## FAULT TOLERANT MULTICAST ROUTING ALGORITHMS

In distributed-memory systems, packets (messages) usually travel across several intermediate nodes before reaching the destination node. Deadlock occurs when some packets (messages) cannot advance toward their destination because the buffers requested by them are full. In direct networks, packets (messages) often go across several intermediate nodes before reaching the destination node. In switch based networks, packets (messages) frequently traverse numerous switches before getting the destination node. On the other hand, it may happen that some packets are not capable to arrive at their destination nodes, even if exist fault-free paths connecting the source and destination nodes for every packet (message) [17]. There are different situations take place when some messages are not capable to reach their destination node, even if they never block permanently. Once some packets (messages) cannot go forward toward their destination node because the buffers requested by them are full, the state is known as deadlock. A packet (message) may be traveling around its destination node, in no way getting it because the channels required to do so are occupied by other packets (messages). This circumstance is known as livelock. It can barely take place when packets (messages) are permitted to follow non-minimal paths. Deadlocks take place because the number of resources is finite. Moreover, some of these situations may create the others [54].

In distributed-memory systems, a few components such as processors, routers, and communication channels may fail. According to number of parameters, faults are classified into different types, regular and irregular faults. Regular faults consist of convex and concave fault shapes. Other shapes considered as irregular faults. Convex

faults are the most commonly encountered faults in mesh networks. A convex fault is a fault region such that there is a rectangle whose interior contains all and only the faulty components of the fault region and all processors and links on its four boundaries are fault-free. A fault ring consists of the fault-free nodes and channels that are adjacent to one or more components of the associated fault region. There are two complementary approaches to create reliable (failure-free) systems, fault prevention and fault tolerance. Fault prevention approaches deal with ending faults being present in the final system. Fault tolerance refers to the capability of the system to operate correctly in the presence of faults. Fault model and fault tolerance are discussed in next section. A good fault tolerant routing should be simple and use few virtual channels. Fault tolerant routing algorithms for regular and irregular faults are discussed in this chapter.

## Fault Model and Fault Tolerance

Some components such as processors, routers, and communication channels may fail in distributed-memory systems. Fault tolerance refers to the capability of the system to operate properly in the presence of faults. According to number of parameters, faults are categorized into different types.

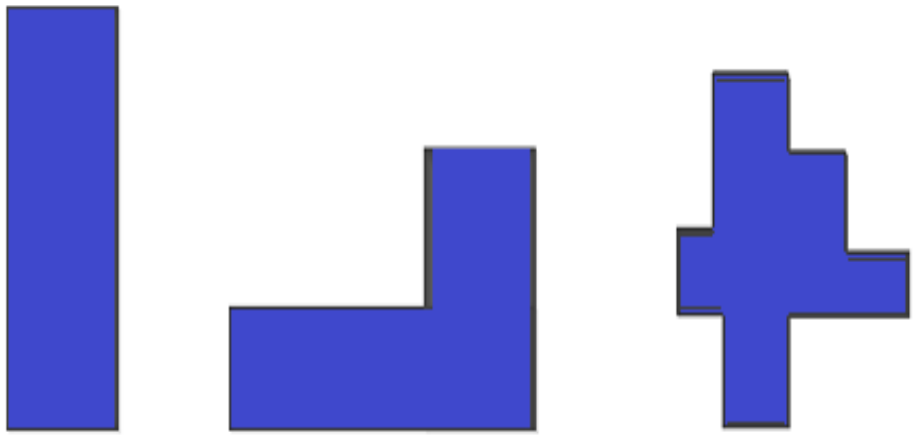
One of the considerations is the level at which components are identified as having failed. Detection mechanisms are assumed to have identified one of two classes of faults. The failure is called node failure when both the processors and their associated routers may fail. The failure is called link failure when any communication channel may fail. In node failures, all physical links incident on the failed node are also marked faulty at adjacent routers. When a physical link fails, all virtual channels on that particular physical link are marked faulty. It is noted that many types of failures will simply be noticeable themselves as link or node failures. For example, the failure of the link

controller, or the virtual channel buffers, appears as a link failure. On the other hand, the failure of the router control unit, or the associated PE, appears as a node failure.

The model of individual link and node failures are lead to patterns of failed components. Adjacent faulty links and faulty nodes are coalesced into *fault regions*. The two most important fault regions are regular (*convex*, *concave)* and irregular faults. Convex faults are the most commonly encountered faults in mesh networks [55], [56]. A convex fault is a fault region such that there is a rectangle whose interior contains all and only the faulty components of the fault region and all processors and links on its four boundaries are fault-free, Figure 20(a). When a fault region touches one or more boundaries of a 2D mesh, the above definition still applies by assuming that there exist non-faulty virtual rows and columns beyond the four boundaries. Hence, all the connected fault regions under consideration are of rectangular shapes. A *fault ring (f-ring)* can be formed around each fault region [57]. Essentially, an f-ring consists of the fault-free nodes and channels that are adjacent (row-wise, column-wise or diagonally) to one or more components of the associated fault region. If a fault region includes boundary nodes, the fault ring reduces to a *fault chain*. Generally, it is assumed that fault regions do not disconnect the network, since each connected network component can be treated as a distinct network. The second type of regular fault regions is called concave fault region, which faults in shape of ┗, ┛ , ┏ and ┓ , Figure 20(b). The third type of fault regions is called irregular fault region, which faults in shape other than shape in the previous two types, Figure 20 (c), [58], [59].

According to how components fail, faults may be identified into three categories. These categories are transient, permanent, and intermittent faults. Transient faults appear

for a time and then disappear. Permanent faults appear at some time and remain forever. Intermittent faults occur and reappear from time to time.



**(a)** **(b)** **(c)**

*Figure 20.* Example of regular (convex, concave) and irregular fault regions.

Depending on when components fail, failures may be either static or dynamic. Static failures are present in the network when the system is running. Dynamic failures appear at random, throughout the operation of the system. Both types of faults are mostly considered to be permanent or transient. When dynamic or transient faults interrupt a message in progress, slices of messages may be left occupying message or flit buffers. Fault recovery structures are necessary to remove such message components from the network to avoid deadlock, mainly if such messages have became corrupted and can no longer be routed [17].

The configuration of fault tolerant routing algorithms is a normal result of the types of faults that can happen, and the capability to identify them. The patterns of component failures and expectations about the behavior of processors and routers in the presence of these failures determine the approaches to achieve deadlock-freedom. This information is captured in a *fault model*. The fault tolerant computing literature is in

general the definition of fault models for the treatment of faulty regions in distributed-memory systems. Designing a fault tolerate system requires the selection of a fault model, a set of possible failure scenarios along with an understanding of the frequency, duration, and impact of each scenario. A simple fault model merely lists the set of faults to be considered; inclusion in the set is decided based on a combination of expected frequency, impact on the system, and providing protection. Most reliable network designs address the failure of any single component, and some designs tolerate multiple failures. In contrast, few attempts to handle the confrontational conditions that might occur in a terrorist attack and cataclysmic events are almost never addressed at any scale larger than a scale of a city [17].

Several additional problems must be considered in the design of a fault-tolerant system beyond the selection of a fault model. A system must be capable of detecting each fault in the model. In addition, it must be able to isolate each fault from the functioning portion of the system in a manner that prevents faulty behavior from spreading. As a fault detection mechanism may detect occurrence of fault, a system must also address the process of fault diagnosis. This process tightens the set of possible faults and allows more efficient fault isolation techniques to be employed.

Significance of the behavior of failed components is also great and the system implementation must defend certain behaviors to guarantee deadlock freedom. The failed node can no longer send or receive any messages and is effectively removed from the network. Otherwise messages designed for these nodes may block indefinitely holding buffers and leading to deadlock. This behavior can be preserved in practice in the absence of global information about the location of faults, by having routers adjacent to a failed node by removing it from the network messages designed for the failed router. The fault

model specifies the extent of the fault information that is available at a node. At one side, only the fault status of adjacent nodes is known. Moreover, the fault status of every node in the network is known. Finally, optimal routing decisions can be made at an intermediate node, i.e., messages can be forwarded along the shortest feasible path in the presence of faults. Conversely, in practice it is difficult to provide global updates of fault information in a timely manner without some form of hardware support. The occurrence of faults during this update period necessitates complex synchronization protocols [60].

Moreover, the increased storage and computation time for globally optimal routing decisions have a significant impact on performance. On one hand, fault information is limited to the status of adjacent nodes. With only local fault information, routing decisions are relatively simple, they can be computed quickly, and updating the fault information of neighboring nodes can perform with an easy way. On the other hand, messages may be forwarded to a portion of the network with faulty components ultimately leading to longer paths. In practice, fault tolerant routing algorithm design is typically a compromise between purely local and purely global fault status information [17].

### Fault Tolerant Routing Algorithms for Regular Faults

Optimal fault tolerant routing algorithm supposed to be simple (low accomplishment cost), uses few virtual channels, supports maximum flexibility in routing, uses the minimal paths when possible, guarantees the delivery of messages, tolerates a large class of fault region patterns and guarantees the deadlock-free routing. Additionally, all these goals should be achieved with a modest hardware constraint. Some approaches called the *global-information-based* assume that each node knows the global distribution of faults. Such an approach is very expensive because the difficult steps to

collect and maintain fault information, and also because it is not scalable. This dissertation focuses on designing a fault tolerant multicast wormhole routing algorithm using a *limited knowledge* based fault information which is a compromise between *local knowledge* based (which needs only the information of neighbor nodes on the routing path and that can take as close to optimal a routing path as possible for any routing instance) and *global-information-based*.

The most used and simplest fault routing algorithm using in regular fault regions (block faults) in 2D mesh networks is e-cube routing algorithm. In dimension order fault-free, messages (packets) are normally routed. When a block fault is encountered, the message can be routed around it. The e-cube algorithm remains deadlock-free by preventing messages from traversing a row after traversing a column. In recent years several fault tolerant routing algorithms for mesh and tori networks have been proposed. Fault tolerant routing algorithms for regular fault regions can be classified into two categories, convex and concave fault regions, depends on the fault shape.

*Convex Fault Region*

Glass and Ni [61] proposed a fault tolerant routing in meshes without virtual channels, the *negative-first*. The negative-first algorithm operates in two phases. In the first phase, the message is delivered adaptively in the negative direction and around fault region, even farther west or south than the destination. It re-labels the channels of the mesh in order to avoid fault region. It performs re-labeling in a purely local fashion, resulting in fast and straightforward reconfiguration. For example, the path that has been taken by message X is illustrated in Figure 21. In the second phase, the message is delivered adaptively in the positive directions to the destination, unless it reaches the destination column, in which case there is only one path to the destination. However, by

permitting the message to be delivered further west and south than the destination; more paths of the destination are created for the second phase. Negative-first routing algorithm can tolerate only one faulty component in a 2D mesh.

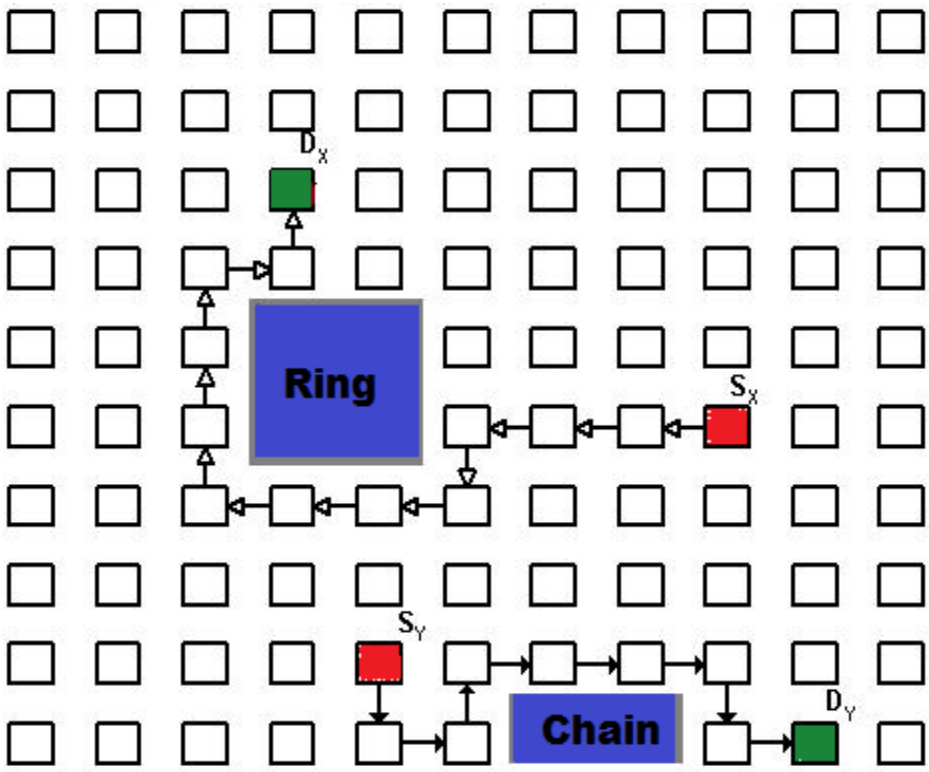



*Figure 21.* [61] An example of the negative-first routing in a 2D mesh.

Glass and Ni [62] proposed modifications to the routing logic of the base negative first routing algorithm to find an alternative path when blocked by a fault region, particularly along an edge of the mesh. The behavior that is permitted in this case is shown by message Y around f-chain fault region in Figure 21. Such a single misroute to avoid a fault region does avoid deadlock. They illustrated that the number of fault regions that can be tolerated by their algorithm is (*n-1*)-fault tolerant in *n*-dimensional meshes with no virtual channels and this is a unicast based multicast routing algorithm. Yet for more than three dimensional meshes, it is not easy to design fault tolerant routing algorithms following their approach [63].

Wu [64] proposed the design of fault tolerant minimal routing methods in 2D meshes that is based on the concept of limited knowledge based fault information. It addresses the issues of the existence of a minimal path at a given source node, limited distribution of fault information, and minimal routing techniques. The disconnected rectangular fault block (convex) is used as the fault model. In addition, Wu [65], proposed a fault tolerant routing algorithm without using virtual channels for mesh networks. The algorithm extended XY-routing techniques, which is based on an odd-even turn model. Wu uses extended convex fault regions (disjoint convex faults), which consists of connected unsafe and faulty nodes. Wu's technique can be applied to 2D meshes having orthogonal faulty blocks (convex polygon). The extended XY-routing technique, however, does not allow routing to some locations (i.e., some nodes cannot fail, and some nodes cannot be destinations, and the convex fault region in the model could include non-faulty nodes).

Rezazadeh et al. [66] proposed a performance-enhancing fault tolerant routing algorithm for Network-on-Chip in Uniform Traffic based on f-cube3 as a solution for increasing the rate of switched and routed packets (messages) in NoCs. They proposed that when a message is not blocked by fault region, all virtual channels could be used. The proposed algorithm requires only one virtual channel per physical channel to ensure deadlock-free in NoCs. Also, Rezazadeh et al. [67] proposed, an improved fault tolerant routing algorithm for mesh network. The proposed algorithm requires only two virtual channels per physical channel to ensure deadlock-free in NoCs. The proposed modification tolerates multiple convex fault regions with overlapped f-rings. A whole column/row fault disconnects mesh networks and is not considered. Mohtashamzadeh et al. [68] proposed an innovative fault tolerant routing algorithm for 2-D mesh network

routing (FTR) as a solution to decrease the delay of the messages deliver over the on-chip interconnection mesh networks. The proposed fault tolerant routing algorithm is a deterministic e-cube routing as long as no faults occur. Also, the proposed routing algorithm is a wormhole-switched routing for 2-D mesh networks and has been used for convex fault regions. The proposed routing algorithm requires virtual channels to ensure deadlock-free in 2D mesh networks. There is no restriction on the number of fault regions tolerated in the proposed routing algorithm.

Xie et al. [69] proposed the two level turn model fault tolerant routing scheme in tori with convex fault regions. The proposed routing algorithm requires less than six virtual channels per physical channel to ensure deadlock-free in tori. The routing algorithm is based on the properties and idea of the turn model for each of his classified of the five message types, which itself could tolerate some faults of delivering for these messages and could work successfully no matter whether the fault regions are connected and no matter where the fault region locates. In addition, Xie et al. [70] presented another fault tolerant wormhole routing scheme in the tori networks with convex faults, called two-level-turn-model scheme, in the tori with revised convex fault regions, which is also based on turn model. The proposed routing algorithm requires only four virtual channels per physical channel to ensure deadlock-free in tori. This algorithm could also tolerate disjointed or overlapped convex fault regions. Safaei et al. [71] proposed an evaluating of the performance of adaptive fault tolerant routing algorithms for wormhole-switched mesh interconnects networks. These networks carry a routing scheme proposed by Boppana and Chalasani [55] as an instance of a fault tolerant. They present a comparative performance study of ten famous adaptive fault tolerant routing algorithms in wormhole switched 2D mesh. The suggested algorithms is extensively used in the researching to

support inter-processor communications in parallel processing computer systems due to its capability to conserve both communication performance and fault tolerant demands in these networks and to achieve high adaptively in these computer systems.

Wu and Chen [72] proposed a fault tolerant tree-based multicast algorithm for 2D meshes based on the idea of the extended safety level which is a vector associated with each node to capture fault information in the neighborhood. They suggested three strategies to develop their ideas. In this algorithm each destination node is reached through a small number of hops. This algorithm can be simply implemented by pipelined circuit switching (PCS) techniques based on limited global information with a simple model. The algorithm has been proved to achieve minimal use of number of hops to deliver the message. Gu et al. [73] proposed improved fault tolerant routing algorithm using a concept of ‘‘balanced ring.’’ The proposed routing algorithm keeps away from early saturation of the network by proposing the concept of balanced ring. The proposed routing algorithm also requires only one virtual channel per physical channel to ensure deadlock-free. With this concept employed, the existing f-ring-based fault tolerant routing algorithm can achieve a more even use of the network resources. The balanced ring is concentric rings of a given fault ring (convex fault region), which can be formed easily.

Zhou and Lau [74] proposed fault tolerant wormhole routing in 2D meshes which is based on the XY routing scheme, which is not adaptive for the more general fault regions. The proposed algorithm requires only two virtual channels per physical channel to guarantee deadlock-free in 2D mesh networks and overlapping of processors along the boundaries of different fault regions is allowed. The proposed fault tolerant routing algorithm can be extended to n-dimensional mesh networks, where an n-dimensional

mesh can be treated as being composed of multiple two-dimensional mesh networks. Also, Zhou and Lau [75] proposed an adaptive fault tolerant routing algorithm with two virtual channels in 2D mesh networks. The proposed routing algorithm can tolerate convex fault regions with overlapping. The proposed routing algorithm requires only two virtual channels per physical channel to ensure deadlock-free in 2D mesh networks. The convex fault model used does not include any non-faulty processors. In addition, Zhou and Lau [76] proposed multi-phase minimal fault tolerant wormhole routing in mesh networks, which is based on the idea of multi-phase minimal routing. The proposed routing algorithm can tolerate convex fault regions with only four virtual channels per physical channel in spite of how processors of different convex fault regions may overlap. Moreover, Zhou [77] proposed fault tolerant wormhole routing with two virtual channels in mesh networks. The proposed routing algorithm can be simply extended to adaptive routing technique. This routing algorithm can tolerate the disjointed convex fault regions with distance at most two hops, which do not include any non-faulty nodes and do not prohibit any routing as long as nodes outside convex fault regions are connected in the mesh networks.

Chalasani and Boppana [78] proposed a fault tolerant routing algorithm to decrease the number of functional nodes that must be marked as faulty nodes. This routing algorithm builds on the idea of fault rings to support more flexible routing around convex fault regions. The proposed routing algorithm uses four virtual channels per physical channel to support more flexible routing around convex fault regions. In this routing algorithm, four virtual networks may be constructed, each comprised of virtual channels of each type. Messages are assigned types based on the relative positions of the source node and destination nodes and dimension ordered routing. In a 2D mesh,

messages are categorized as east-west (EW), west-east (WE), north-south (NS), or south-north (SN) based on the relative values of offsets in the first dimension. Routing is dimension ordered until a message encounters a convex fault region. Depending on the type, the message is routed around the convex fault region as shown in Figure 22. The direction around the convex fault region is chosen based on the relative position of the destination node. The EW and WE messages may turn out to be NS and SN messages. However, the opposite is not true. As a result, dependencies between channel classes are acyclic. Since fault regions are convex faults, dependencies within a fault region are also acyclic – the arguments are similar to those provided for fault tolerant multicast planar adaptive routing.

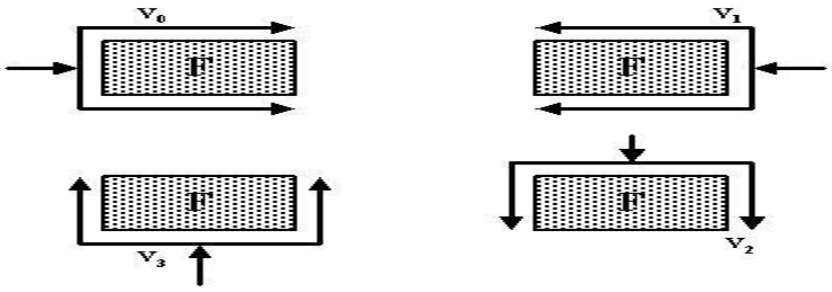



*Figure 22.* [78] Routing restrictions around a fault region.

An example of routing around convex fault regions with overlapping fault rings is shown in Figure 23. Two messages X and Y have destinations and sources as shown. X is an EW message and Y is a WE message. Message Y is routed as a WE message around convex fault region until it reaches the destination column where the type is changed to that of a NS message. The figure also illustrates the path selected by message X. These two messages share a physical link where convex fault regions overlap. Consider the shared link where both messages traverse the link in the same direction towards the destination node. If virtual channels were not used to separate the messages in each fault

ring (convex fault), one of the messages could block the other. An EW message can block a WE message and vice versa, resulting in cyclic dependencies. The separation of the messages into four types (classes), the use of four virtual networks, and acyclic dependencies between these networks avoid occurrence of deadlock.

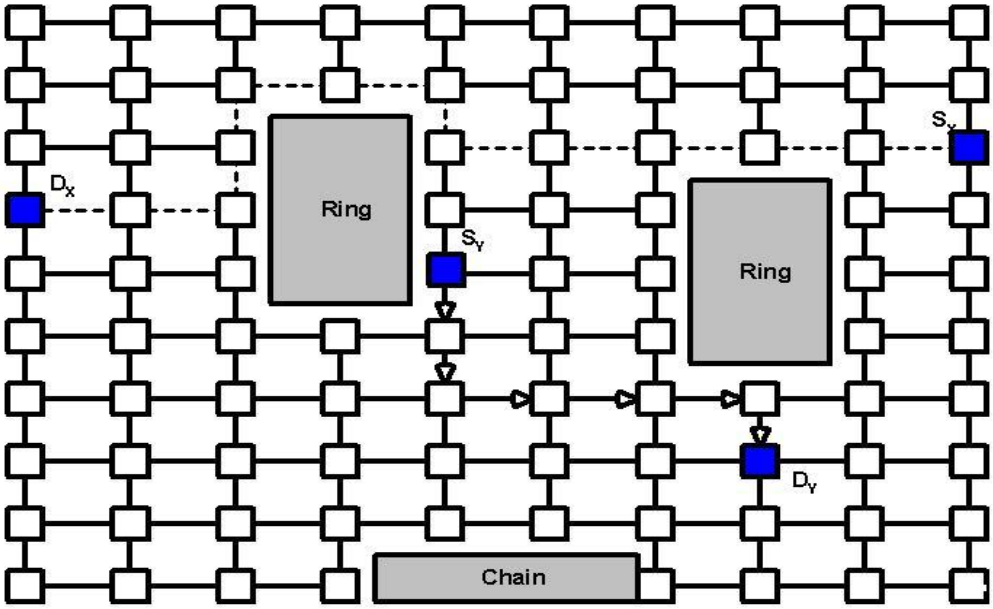



*Figure 23.* [78] An example of routing around overlapping fault rings.

Boppana and Chalasani [55] proposed an adaptive fault tolerant routing algorithm for mesh networks that can tolerate faults of arbitrary rectangular shapes (convex fault regions). The concepts of f-rings and f-chains were used for routing messages around rectangular fault regions. They enhanced routing algorithm for mesh networks based on e-cube routing. This routing algorithm uses two virtual channels to provide non-adaptive deadlock-free routing in networks with non-overlapping f-rings. For more complex fault regions, such as overlapping f-rings and f-chains, the algorithm uses four virtual channels to ensure deadlock-free.

Chang and Chiu [60] proposed a fault tolerant multicast unicast-based routing algorithm, FT-cube2, in 2D mesh networks. In FT-cube2 routing algorithm, the well-

known e-cube routing algorithm improved in order to handle multiple convex fault regions in 2D mesh networks. The proposed routing algorithm requires only two virtual channels per physical channel to guarantee deadlock-free in 2D mesh networks. Also it is local knowledge-based fault information and works correctly for any combination of convex fault regions. In FT-cube2 routing algorithm, normal messages from source node to destination node are routed via e-cube hops. A message is misrouted on an f-ring or f-chain along clockwise or counterclockwise direction specified by table 1 [60]. FT-cube2 will be compared with algorithms proposed in this dissertation.

Table 1

*Misrouted on an f-ring or f-chain* [60]

| Message type | f-ring/f-chain | Direction |
|---|---|---|
| Normal-WE | | Counterclockwise |
| s-WE | | Clockwise |
| EW | n-Chain or nw-Chain<br>Others | Clockwise<br>Counterclockwise |
| SN | w-Chain<br>Others | Counterclockwise<br>Clockwise |
| NS | w-Chain<br>Others | Clockwise<br>Counterclockwise |

*Concave Fault Region*

Fault tolerant multicast routing algorithms in the presence of concave fault regions, which faults in shape of ⌞, ⌟, ⌜ and ⌝ study is not the main research for a lot of researchers in the field of distributed-memory systems performance because most of

them consider concave fault region as a multi connected convex fault regions. In addition, some of them take into consideration other shapes like + as a concave fault region. Several of these studies consider concave fault region, which faults in shape of ┗, ┛ , ┏ and ┓ will presented in this section.

Xie and Xu [79] proposed the two level turn model fault tolerant routing algorithm in tori networks with convex and concave fault regions. This routing algorithm could tolerate the concave fault regions and the convex fault regions both with a few limitations to their shape. The proposed routing algorithm requires at most five virtual channels per physical channel to ensure deadlock-free in tori networks.

Park et al. [80] proposed fault tolerant wormhole routing algorithms in mesh networks in the presence of concave fault regions. They proposed two fault tolerant wormhole routing algorithms that deal with more relaxed shapes of fault rings in the mesh networks. The first fault tolerant routing algorithm uses four virtual channels per physical channel and allows all four sides of fault rings to contain concave shapes. The second fault tolerant routing algorithm permits up to three sides to contain concave shapes using only three virtual channels per physical channel. In their fault models, there might be several f-rings in a 2D mesh networks. They divide also each f-ring (convex or concave) fault region into four portions: the north, south, west and east sides. Both fault tolerant routing algorithms are free of deadlock and guarantee the delivery of messages between any pair of non-faulty and connected nodes in mesh networks. The first fault tolerant routing algorithm, F4, will be compared with algorithms proposed in this dissertation.

## Fault Tolerant Routing Algorithms for Irregular Faults

Convex (rectangular and square) fault regions provide some form of non-decreasing property in coordinates of misrouted messages. This property is exploited to prevent the occurrence of deadlocked message configurations. However, the construction of regular fault regions by the marking of fault-free routers and links as fault region can lead to under significant utilization of resources. These methods cannot be accepted by many of the early methodologies to design fault tolerant routing algorithms. Since a whole knowledge of the patterns of occurrences of fault regions is not assumed, these methodologies are not generally proper to cases where larger fault regions must be supported. Various fault tolerant routing algorithms use virtual channels of irregular faults for mesh networks have been proposed in recent years.

Table 2

*Routing Rules for Irregular Fault Regions* [57]

| Message type | Position of destination | F-ring orientation |
|---|---|---|
| WE | row above current row | Clockwise |
| WE | row below current row | Counterclockwise |
| EW | row above current row | Counterclockwise |
| EW | row below current row | Clockwise |
| NS or SN | either | either orientation |

Chalasani and Boppana [57], the concept of fault rings can be extended in a minimal manner to account for certain classes of irregular fault regions. Consider the class of fault regions in *n*-dimensional mesh networks where any two dimensional cross-

section of the fault region produces a single rectangular fault region. Such a fault model is referred to as an irregular fault model [57]. Figure 24 provides an example of an irregular fault region, and a message is being routed along a fault ring around the fault region. Table 2 describes routing rules considered around fault region and these rules do not apply in case of overlap fault regions. As in previous techniques, for non-overlapping fault rings, and non-faulty boundary nodes, messages types are distinguished by the relative positions of the destination when the message is generated. When the message eventually arrives at the destination column, the message type is changed to NS or SN depending on the relative location of the destination. When a message encounters a fault, the rules for routing the message along the fault ring are shown. There are four virtual channels over each physical channel: $v_0$, $v_1$, $v_2$, and $v_3$. In these techniques, each set of channels implements a distinct virtual network. If a message must travel along a fault ring before encountering a fault region, (as shown in Figure 24 at node X) then the message must continue to be routed in the same direction, along the fault ring. Otherwise the message follows the direction specified in Table 2. Each message type is transmitted in a distinct virtual network. From the routing rules, the channel dependency graph within a virtual network is acyclic. In addition, messages can only transition from WE or EW channels to NS or SN channels but not vice versa. Consequently, the relation between these virtual networks residues acyclic and therefore routing residues deadlock-free.

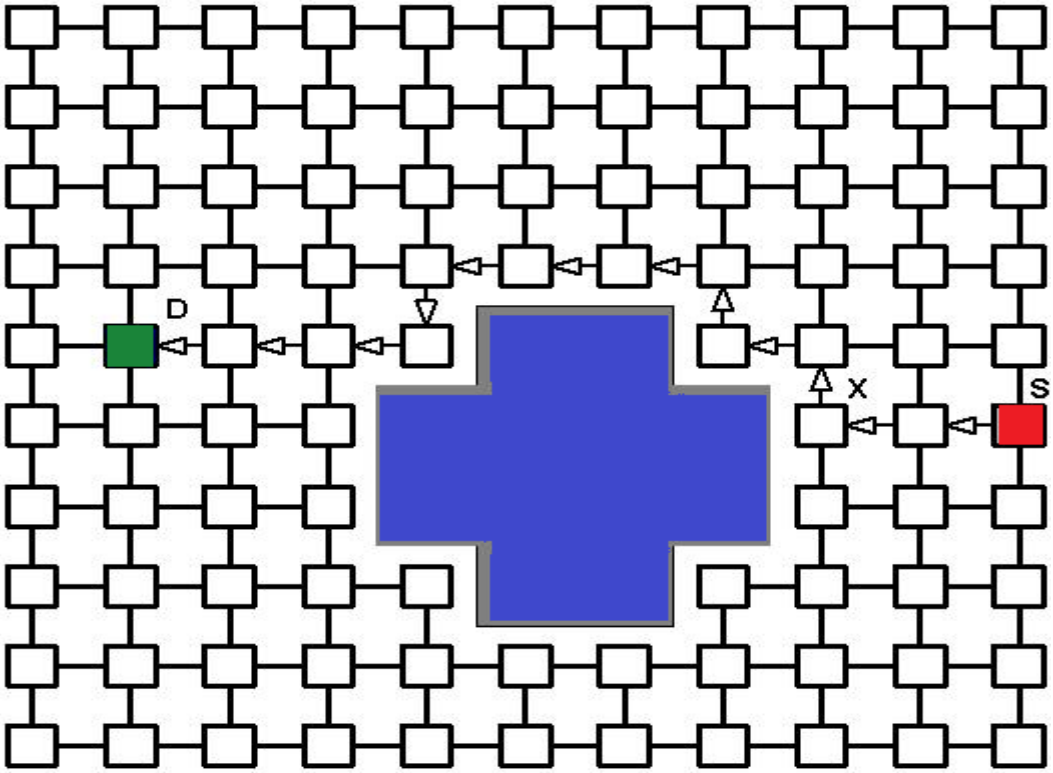



*Figure 24.* [57] An example of routing around an irregular fault.

Fukushima et al. [81] proposed a hardware-oriented fault tolerant multicast routing algorithm for 2D mesh Network-on-Chip without virtual channels. The proposed routing algorithm requires no virtual channels per physical channel to ensure deadlock-free in irregular 2D mesh. The proposed position-route algorithm for non-VC routers needs much less routing complexity. The main idea is to add routing behaviors of the traditional message-based algorithm and to simplify the fault region (ring) selection.

Mejia et al. [82] proposed an efficient fault tolerant routing algorithm for mesh and tori networks. The proposed routing algorithm is a deterministic routing methodology for tori and mesh networks, which accomplishes high performance without the necessity of use virtual channels. This routing algorithm can handle any topology derived from any combination of fault regions when combined with static reconfiguration. The algorithm, called segment-based routing (SR), works by dividing a topology into sub-networks, and

sub-networks into segments. Wu and Wang [83] proposed a fault tolerant and deadlock-free routing in 2D mesh networks using rectilinear-monotone polygonal fault regions. The main idea for it is both source and destination nodes are outside any fault region. In addition, the destination node is not a boundary node of any fault region. Moreover, fault model is static, that is, no new fault regions happen during a routing process, and fault regions (ring) are at least 2 hops away from the four boundaries of a mesh network. Wu's extended X–Y routing to 2D mesh networks that use a new fault region model called minimal-connected-component (MCC). The extended X–Y routing is a deterministic fault tolerant and deadlock-free routing protocol in 2D mesh networks. The proposed routing algorithm requires no virtual channels per physical channel to ensure deadlock-free in 2D mesh networks, as shown on Figure 25.

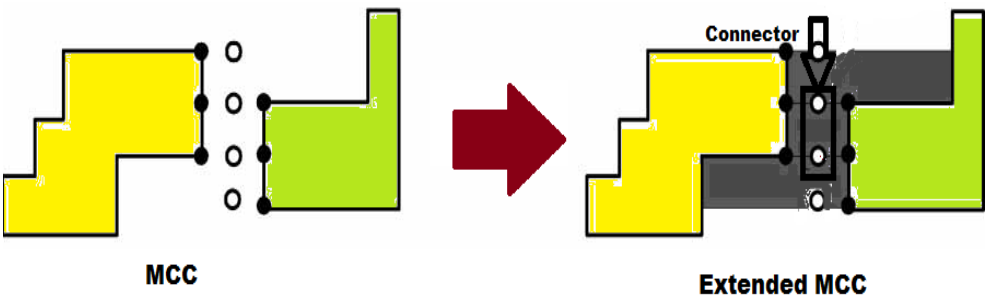



*Figure 25.* [83] MCCs that are 2-hop apart are merged into one (extended) MCC.

Stojmenovic and Nayak [84] proposed fault tolerant multicast routing in mesh networks. The proposed routing algorithm uses local information knowledge of fault regions. In this routing algorithm there is no need for additional resources. It works for an arbitrary number and structure of fault regions, and guarantees delivery to all destination nodes connected to the source node, it also remains optimum in a fault free mesh network. The routing algorithm is extended to faulty k-D mesh networks and k-ary n-cubes, where the delivery will be guaranteed if healthy nodes in every 2D sub-mesh (sub-

tori) stay connected. The proposed routing algorithm requires no virtual channels per physical channel to ensure deadlock-free in 2D mesh networks. Shih [85] proposed a fault tolerant wormhole routing algorithm for hypercube without using any virtual channels. The proposed routing algorithm can tolerate a different pattern of fault regions as long as the number of faulty nodes is no more than n/2, where n is the dimension of hypercube network.

Xiang et al. [86] proposed a fault tolerant routing in mesh/tori networks using planerly constructed fault regions. A new limited-global safety-based measure called the extended local safety information is presented to guide fault tolerant routing, based on which a new path set-up scheme is proposed. The number of virtual channels requisite by the proposed routing technique is linearly proportional to the number of dimensions of the network. In this routing algorithm fault region are created inside separate planes, where many unsafe nodes can be activated. This can significantly advance the computational power of the system and improve the performance of the fault tolerant routing algorithm greatly. In addition, Xiang et al. [87] proposed a practical deadlock-free fault tolerant multicast routing in mesh networks based on the planar network fault model. The proposed routing algorithm requires only two virtual channels per physical channel to ensure deadlock-free in 3D mesh networks. The deadlock-free routing pattern is used to do fault tolerant routing in mesh networks, where PN fault model is presented to guide deadlock-free adaptive fault tolerant routing in wormhole-routed mesh networks. Also, Xiang [88] proposed a new deadlock-free adaptive routing in mesh networks with fault tolerance ability. It is proposed based on a new virtual network partitioning scheme, called channel overlapping. The proposed routing algorithm requires only two virtual channels per physical channel to ensure deadlock-free in 2D mesh networks. The

proposed routing algorithm is also extended to the one in an n-dimensional mesh network with two virtual channels. Xiang proposed planar safety information in mesh networks to guide fault tolerant routing and to categorize fault-free nodes inside 2D planes.

Safaei et al. [89] proposed a performance analysis of fault tolerant routing algorithm in wormhole switched interconnections for 2D tori network using the fault tolerant software-based method. They describe a general model to derive mathematical expressions to study the performance behavior of challenging routing algorithms, regular and irregular fault regions. They consider regular (|-shaped, _-shaped) or irregular (U-shaped, +-shaped, T-shaped, H-shaped) fault regions. The number of virtual channels they use to grantee deadlock-free depend on if it is adaptive or deterministic routing. Safaei and Mortazavi [90] presented a novel routing algorithm for achieving static fault tolerance in 2D mesh networks. The proposed routing algorithm does not require the use of routing tables and is well-suited for use in high performance computing systems. The proposed routing algorithm requires five virtual channels per physical channel to ensure deadlock-free in 2D mesh networks. The main idea is to splits sub-networks of nodes into two parts. Then they count fault region in each part and select the less one.

Youn at el. [91] proposed a fault tolerant routing method that can tolerate irregular fault regions. The proposed routing algorithm requires only two virtual channels per physical channel to ensure deadlock-free in mesh networks. The proposed routing scheme misroutes messages both clockwise and counter clockwise directions to lessen channel contention on f-rings. It is shown that the proposed routing algorithm is deadlock-free in mesh networks when it has non-overlapping multiple f-regions. Wang et al. [92] proposed a fault tolerant multicast routing algorithm on mesh networks. This routing algorithm is highly fault tolerant and has a high success probability to route

messages. The routing algorithm is local information based and a distributed multicast routing algorithm based on the concept of k-sub-mesh in all port mesh networks. The main idea is to divide the mesh m×n into (m/k), (n/k) disjoint k sub-meshes. Then these k-sub-meshes are partitioned into four regions according to the location of the k-sub-mesh that contains the source node.

Duan et al. [93] proposed a fault tolerant routing algorithm for wormhole mesh networks. The proposed routing algorithm is connected and deadlock-free in spite of the various irregular fault regions in mesh networks. In addition, the proposed fault tolerant routing algorithm only works as few virtual channels as possible. Thus the proposed routing algorithm is appropriate to the fault tolerance mesh network. Since it chooses the path around fault regions according to the local fault information, the presented routing algorithm takes routing decisions quickly and is applicable in interconnection networks.

Jiang et al. [94] proposed a fault information model for fault tolerant adaptive and minimal routing in 3D mesh networks. In this fault tolerant routing model, they have rewritten the MCC model in 2D mesh networks without using global information based for this reason the shape information at boundaries can be used to guarantee the presence of a minimal path and to form a minimal routing by making routing decisions at intermediate nodes along the path. In addition, they extended the MCC model in 2D mesh networks to 3D mesh networks. This fault information model is limited global-information model.

Chen and Chiu [95] proposed a fault tolerant routing algorithm for mesh networks with irregular fault regions. In this routing algorithm a flag bit is introduced for guiding misrouted messages. The proposed routing algorithm necessitates only three virtual channels per physical channel to ensure deadlock-free in 2D mesh networks. This routing

algorithm is able to handle irregular fault regions whose associated fault rings overlap. In addition, this routing scheme can be used to deliver messages when fault regions touch the boundaries of the mesh.

# CHAPTER IV

## FTDM AND *i*FTDM ROUTING ALGORITHMS

One of the important issues in parallel computing is how to powerfully accomplish routing in a faulty network, where each element fails with various probabilities. Routing is a task where a source node sends a message to a destination node. Network topology is an important factor that affects routing algorithms.

Mesh connected networks have been widely used in most multicomputer systems. These computers generally use the e-cube routing algorithm with wormhole switching because of its simplicity. The main idea of e-cube algorithm is to route a message first along the row and then along the column in a 2D mesh. It is important to note that e-cube provides deadlock-free shortest path routing without needing virtual channels [55]. Distributed-memory systems are the most advantageous architectures in building a massively parallel computer system. These systems need switching techniques to broadcast messages among processors. The wormhole switching technique has been widely used in the design of parallel computer systems. The basic idea of wormhole routing is that a message is partitioned into flow control flits. Each flit of a message is chosen as the header flit, which is responsible for leading the message on the network. The multicast pattern, in which one processor (node) sends the same message to multiple processors (nodes), is the most fundamental communication pattern used on multicomputer. Fault tolerance is a central issue facing the design and implementation of interconnection networks for distributed-memory systems. This work focuses on studying the fault tolerant multicast wormhole routings in a 2D mesh networks.

In recent years, fault tolerant routing in direct networks has been deservedly gaining a lot of attention. The model of individual link and node failures produces

patterns of failed elements. Fault regions result from the closest faulty links and faulty nodes. The two most important fault regions are regular (convex, concave) and irregular. A good fault tolerant routing should be simple (low implementation cost), use few numbers of virtual channels, assure the delivery of messages, tolerate many types of fault patterns, and assure deadlock-free routing while minimizing disabled processors to ease the routing algorithm. Furthermore, all these goals should be achieved with less consideration for hardware requirement.

In this Chapter, a new fault tolerant routing algorithms in wormhole-switched 2D mesh multicomputer is presented. It can tolerate convex faults without using virtual channels. The proposed routing algorithm, called Fault Tolerant Deadlock-free Multicast (FTDM) works perfectly for the most common faults in 2D mesh networks, f-rings and f-chains. In addition, an improved version of FTDM which is called *i*FTDM is presented. Both algorithms are a unicast/tree based multicast routing algorithm. The *i*FTDM can tolerate convex faults with overlapping. Four essential performance metrics in mesh networks – network traffic steps, network latency steps, network traffic time and network latency time – will be considered and calculated for both algorithms.

## FTDM Fault Model

Many applications of interconnect networks require high reliability and availability. A large parallel computer requires that it's interconnect network operates without packet loss for ten thousands of hours. Thus, these networks must employ an error control mechanism to continue operation without interruption, and possibly without packet loss, despite the failure of a component. The failure of a processing element and its associated routers is referred to as a node failure, and the failure of any communication channel is referred to as a link failure. In our fault model, both node

failures and link failures are considered. The fault model is the base for the fault tolerant routing algorithms. Types of faults, structures of fault regions and processes to component failures determine the approaches to design deadlock-free routing algorithm.

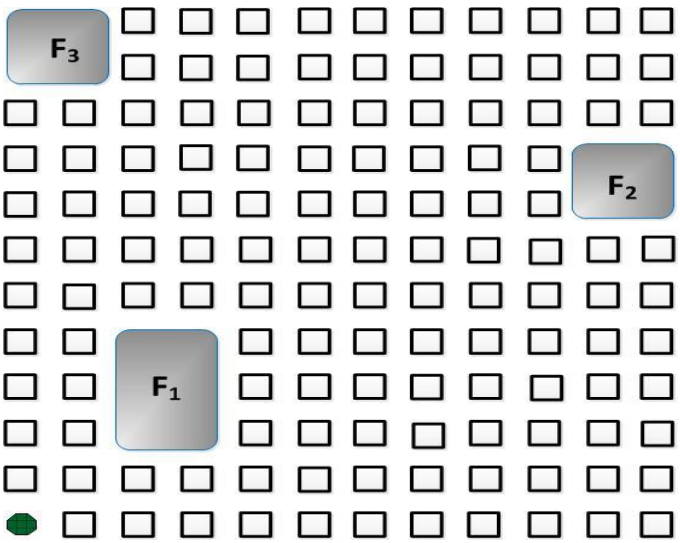



*Figure 26.* FTDM Fault model.

FTDM algorithm considers convex faults (also known as block faults), which are the most commonly encountered faults in mesh networks [96]. A convex fault is a fault region such that there is a rectangle whose interior contains all and only the faulty components of the fault region and all processors and links on its four boundaries are fault-free. A fault ring (f-ring) consists of the fault-free nodes and channels that are adjacent to one or more components of the associated fault region, $F_1$, as shown in Figure 26. If a fault region includes boundary nodes, the fault ring reduces to a fault chain (f-chain), $F_2$ and $F_3$, as shown in Figure 26. In FTDM, fault information of a fault (faulty block) is distributed to a limited number of nodes (0, $y_{bi}$) in case of odd rows or (m, $y_{bi}$) in case of even rows in order to avoid the fault before reaching it. Because fault information is distributed to a limited number of nodes, FTDM is a limited-global-

information-based multicasting, which is a compromise of local-information-based approach and global-information-based approach.

## FTDM Routing Algorithm

Most fault tolerant routing algorithms which were proposed in the literature recently concentrate on unicast-based multicast algorithms [60], [62], [88]. Unicast-based algorithms require a startup time for each destination. Also, unicast-based multicast algorithms are incompetent because they permit a message to be delivered to only one destination, which leads to multicast operations being implemented as multiple phases of multicast message exchange. Hence, contention freedom must be guaranteed not only among the worms of a given phase, but also among worms in different phases.

In this section, the details of new fault tolerant deadlock-free multicast routing algorithm, FTDM, for 2D meshes is covered. FTDM is a unicast/tree-based multicast algorithm, which attempts to deliver the message to all destinations in two phases. In the first phase the message is delivered as a unicast-based to X-coordinate nodes – nodes (0, $y_{bi}$) in case of odd rows or (m-1, $y_{bi}$) in case of even rows – of each *true fault regions* at these nodes – *central nodes*. We consider each node of them as a source node that has a message with a header containing destinations in the three locations around the fault. In the second phase, the message is delivered from the central nodes in a tree-based fashion, which attempts to route the message to all destinations in a single multi- head worm that splits at some routers and replicates the data on multiple output ports.

To define the path routing functions, which determines the next node for which the path of FTDM will be visited, some definitions are introduced:

1) Let $f_{bi}= (x_{bi}, y_{bi})$, and $f_{ei}= (x_{ei}, y_{ei})$ be the coordinates of each fault.

2) The fault region number *i*, $F_i$, is described by two nodes, $f_{bi}$, $f_{ei}$, where $f_{bi}$ is located in the southwest corner of the fault region, while $f_{ei}$ is located in the northeast corner of the fault region,

$F_i = \{(x, y): x_{bi} < x < x_{ei} \wedge y_{bi} < y < y_{ei}\}$.

3) Width of a fault region $F_i$ is defined as follow: $d_{Fi} = | x_{ei} - x_{bi} |$

4) The variable $d_x$ is equal to 1 if the direction of the message path is from west to east or -1 if it is from east to west.

5) *LN* is the label of last node, $(x_{ei}, y_{bi})$, of a fault region which the message path visits. The value of LN is zero if the message path is in a non-fault region, while it is non zero if the message path is in a fault region.

6) $L_1$, $L_2$and $L_3$ are three locations around each true fault regions as in Figure 27, and $L_4$ is a location in case if the 1st fault region is an f-ring. In Figure 27 we define three fault regions $F_1$, $F_2$ and $F_3$. Also, the notation $L_{A, B}$ means location A for fault number B (i.e. $L_{3,1}$ means location three for the 1st fault region, $F_1$.)

7) *True fault regions* are the main fault regions which have three locations around them and may have other fault regions on locations, $L_3$ or $L_1$, with $f_{bi} = (x_{bi}, y_{bi})$, and $f_{ei} = (x_{ei}, y_{ei})$ less than it.

8) *Central nodes* are the nodes which the source node sends a copy of a message in the first phase in a unicast fashion,

9) Consider a source node as one of the central nodes if the first fault region is f-ring.

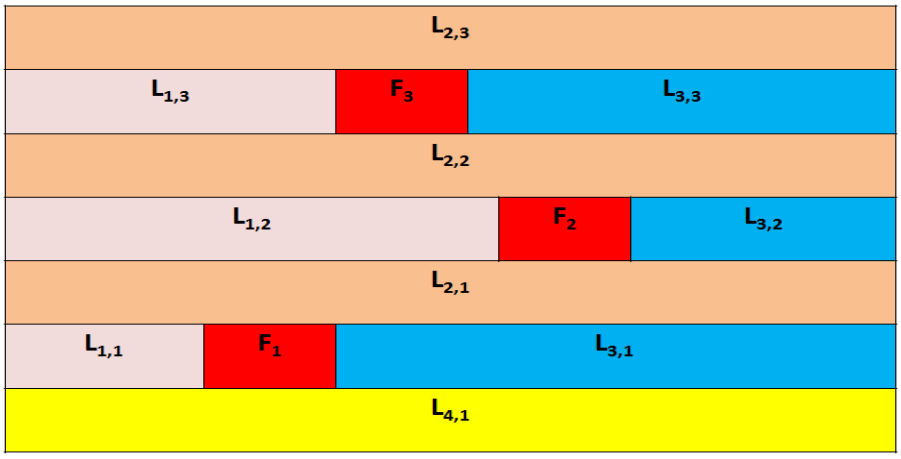



*Figure 27*. Locations around fault regions.

*Routing Functions*

FTDM assigns a label for each node based on the position of that node in a Hamiltonian path. The *Hamiltonian* path in a network is an undirected path that visits each node in a graph just once where the first node in the path is labeled *1* and the last node in the path is labeled *N*, where *N* is the network size [40]. The label assignment function Q for an $m \times n$ 2D mesh using a Hamiltonian path can be expressed in terms of the x- and y-coordinates of nodes as follows:

$$Q(p_i) = Q(x_i, y_i) = \begin{cases} y_i \times n + x_i + 1 & y_i \text{ is even} \\ y_i \times n + n - x_i & y_i \text{ is odd} \end{cases}$$

FTDM creates the routing decision at each sending node. The path followed by a message in the first phase is simply unicast-based in which a source node sends a separate message to each central node beside a copy to $L_4$ in case the first true fault region is f-ring using XY routing algorithm. The path followed by a message in the second phase is defined by one of the two routing functions. Each function is defined as a function of the node currently holding a message and the destination node of this

message. The function returns a neighboring node of the current node to which the message must be forwarded. Let c be a current node, and d is a destination node.

The first routing function used in FTDM is defined as:

R (c, d) = w, where

$$Q(w) = \begin{cases} \max\{Q(z):\ Q(z) \le Q(u)\} \text{ if } Q(c) < Q(d) \\ \max\{Q(z):\ Q(z) \ge Q(u)\} \text{ if } Q(c) > Q(d) \end{cases} \quad \text{and z is a neighboring node of c}$$

Lin and Ni proved [40] that for two arbitrary nodes *c* and *u* in a 2D mesh, the path selected by the routing function R is the shortest path between them. As proved in [40], this routing function is deadlock-free even using the path based on facility. FTDM which uses the routing function R in each region does not contain any fault nodes.

The second routing function used in FTDM at a fault region $F_i$ is defined as:

R′(c, d) = w, where

$$w = \begin{cases} (x_c, y_c\text{-}1) & \text{if } x_d = x_c \\ (x_c, y_c\text{+}1) & \text{if } x_d = x_c + d_x \times d_{Fi} \\ (x_c + d_x, y_c) & \text{otherwise} \end{cases}$$

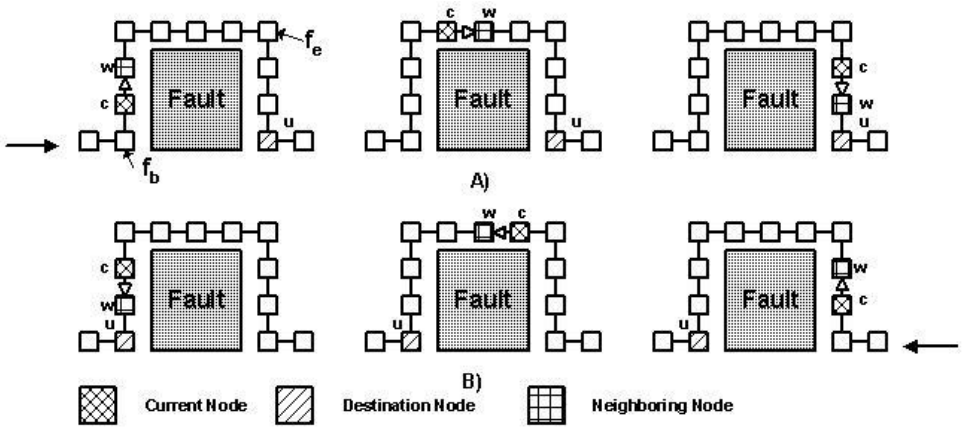



*Figure 28.* The routing path using R'.

**Algorithm FTDM**

**Input**: The message *msg***,** Label node LN, central nodes $CN_k$, destination set D, and fault region $F_i$ .

**Output**: $\forall\ d_j \in D$, Receive($d_j$ , *msg*)

**Procedure**:

[1]/* Phase 1 (unicast-based): Send copies of message to $CN_k$

[a] If $c = d_1$ then

a. 1) D = D-{c}

a. 2) Receive(c, *msg*)

[b] If $D = \phi$ then stop

[c] Send separate messages to $CN_k$ using XY routing.

[d] Modify header of messages, *msg*, and put in each header $D_k$ destinations, which k is the number of central nodes (plus one if first fault is f-ring)

[e] Let each $CN_k$ as a new source node

[f] Go to phase 2

[2]/* Phase 2 (tree-based): Send *msg* using R and R' functions

[a] If $c = d_1$ then

a. 1) D = D-{c}

a. 2) Receive(c, *msg*)

[b] If $D = \phi$ then stop

[c] At each new source node, send two copies of message, *msg*

c. 1) $1^{st}$ copy contains destinations on $L_1$ and $L_2$ using R

c. 2) $2^{nd}$ copy contains destinations on $L_3$. Using R' to route a message around the fault region until the message reach to LN, and then use R.

c. 3) If $L_3$ have another faults then recursively apply FTDM.

[d] Repeat the above steps until each destination in the message header is reached.

*Pseudocode 1*. FTDM Routing Algorithm

FTDM uses the routing function R' in fault regions only. Figure 28 illustrates the different cases of the routing function R' and the way it works around the fault region. The direction of the message path may be from west to east, Figure 28(A) or from east to west, Figure 28(B). It is clear that, the path selected by the routing function R' is the shortest path between the two nodes c and u. Also, it is clear that the routing function R' is deadlock-free, because it works on three boundaries only of each fault region, i.e., the cycle is not complete.

***Lemma****:* FTDM algorithm is deadlock-free

*Proof:* There are two phases as follow:

Phase 1: Unicast-based multicast routing

The separate addressing is one of the unicast-based multicasting techniques, which has been already proven to be deadlock-free [8], because in separate addressing, the source node sends directly a separate copy of the message to every destination node, then no cyclic dependency can be created among the channels.

Phase 2: Tree-based multicast routing has two cases and they are as follows:

Case 1: Nonfault regions

Because a cyclic dependency among resources is a necessary condition for deadlock, since a message is routed at any node according to the routing function R, which is proved deadlock-free [42], and monotonic order of requested channels is guaranteed. Therefore, a cycle cannot exist within this path in the network; hence, no cyclic dependency can be created among the channels.

Case 2: Fault regions

Since a message is routed at any faulty nodes according to routing function R´, and a message never visits an f-ring and f-chain more than twice (at most as a row message and once as a column message), then a cycle cannot exist within this path in the network. Hence, FTDM algorithm is deadlock-free.

*Results and Discussions*

A simulation study has been conducted to test the proposed new fault-tolerant multicast routing algorithm. To evaluate the performance of FTDM and to compare its performance with FT-cube2 routing algorithms, simulations on a 50×50 2D mesh were conducted, double channels were used. The two algorithms were written using C++ language and were implemented on a PC. In this section, we present the simulation results and analysis. In the simulation, the wormhole switching routing technique is

chosen as the switching technique. In addition, FTDM routing algorithm is also applicable with other switching techniques. The notation F is used to represent the number of fault regions, R is the number of rows, and C is the number of columns. This configuration creates different networks with a number of processors ranging from 100 to 1080. The average number of destinations is ranging from 10 to 100 and using three fault regions.

*Network latency steps and network traffic steps analysis.* In this subsection, two essential performance metrics in direct networks, network latency steps and network traffic steps, are calculated. The network latency step is the greatest number of channels which the message takes to reach its destination nodes. The network traffic step is the total number of channels used to deliver the message to all destinations. They affect the overall performance of parallel computing systems and the granularity of parallelism that can be exploited from the system [30].

Now, the network latency steps and network traffic steps are calculated for FTDM and FT-cube2 routing algorithms. The following formulas can be used to calculate the network latency steps for FTDM.

Our partitioning of the 2D mesh around each fault regions into $L_{i1}$, $L_{i2}$, $L_{i3}$ and $L_4$, will result in partitioning the destination nodes D into $D_{i1}$, $D_{i2}$, $D_{i3}$,and $D_{i4}$ respectively where i is ranging from 1 to F, and F is number of fault regions. In addition, ($c_X$, $c_Y$) is the coordinate of central node.

- dist ($d_i$, $d_{i-1}$) = $|x_{di} - x_{di-1}| + |y_{di} - y_{di-1}|$
- Lat (D) = $\sum_{i=1}^{|D|}$ dist ($d_i$, $d_{i-1}$)

Lat (D) is depends on the start coordinates and end coordinates for each location.

- $D_i = \{(x, y) : (x, y) \in D \wedge x > x_{bni} \wedge y < y_{ei} \}$
- $L_{i1} = Lat(D_{i1}) + |(c_X - x_{di})| + |(c_Y - y_{di})|$,
- $OO_i = |X_{end} - X_{start}| - 1$,

  Where $X_{start} = 0$ and $X_{end} = x_{bi} + 1$
- $UU_i = y_{bi} + 1$
- $Lf_i = 2*|(y_{ei} - y_{bi} - 1)| + |(x_{ei} - x_{bi})|$
- $L_4 = Lat(D_{i4}) + |(S_X - x_{di})| + |(S_Y - y_{di})|$
- $L_{i2} = Lat(D_{i2})$,
- $L_{i3} = Lat(D_{i3})$
- $Traffic_{(i)} = L_{i1} + L_{i2} + L_{i3} + OO_i + UU_i + Lf_i$
- $LP_{(i)} = L_{i1} + L_{i2} + UU_i$

  Where LP is the left path
- $RP_{(i)} = L_{i3} + OO_i + UU_i + Lf_i$

  Where RP is the Right path

The latency step of FTDM, FTDM_Latency, is given by:

$$FTDM\_Latency = Max(LP_{(i)}, RP_{(i)}, L_4) \quad (1)$$

The traffic steps of FTDM, FTDM_Traffic, is given by:

$$FTDM\_Traffic = Traffic_{(i)} + L_4 \quad (2)$$

The latency step of FT-cube2, FT_Latency, is given by:

$$FT\_Latency = Max\{ Flat_i, 1 \leq i \leq |D| \} \quad (3)$$

Where $Flat_i = |x_{di} - S_x| + |y_{di} - S_y| + 2*|y_{ei} - y_{bi}|$

The traffic steps of FT-cube2, FT_Traffic, is given by:

$$FT\_Traffic = \sum_{i=1}^{|D|} Flat_i \quad (4)$$

*Case study*. As an example, to demonstrate the difference between FTDM algorithm and FT-cube2 algorithm, a 15×15 2D mesh network is considered, Figure 29. The source node is (0, 0) and 15 destinations node, colored with dark. The number of fault regions equals 3 and they are f-ring type. The dimension for each fault is, $F_1$ = {(3, 1), (6, 4)}, $F_2$ = {(9, 5), (11, 8)}, $F_3$ = {(4, 9), (7, 12)}. The dimension of central nodes is CN's = {(0, 0), (0, 2), (0, 6), (0, 10)}. As shown in Figure 29, in first phase four messages sent to each central node, each message has destinations in locations corresponding to a fault region for each central node.

By applying FTDM algorithm, in Figure 29, the message is transferred from the node (0, 0) to central nodes in a unicast –based multicast routing fashion. Then a message routed on non-faulty region with function R and around fault regions with function R′ using tree-based multicast routing fashion. The latency steps are equal 29, and traffic steps are equal 117.

By applying FT-cube2 algorithm, the message is transferred from the node (0, 0) to all destinations on a separated message using a unicast based multicast routing fashion. Network latency steps are equal 25, and network traffic steps are equal 213. Hence, in case of network traffic steps, FTDM algorithm is more effective than FT-cube2 algorithm in case of large number of destinations. Also, in case of network latency steps, FTDM algorithm is close to FT-cube2 algorithm.

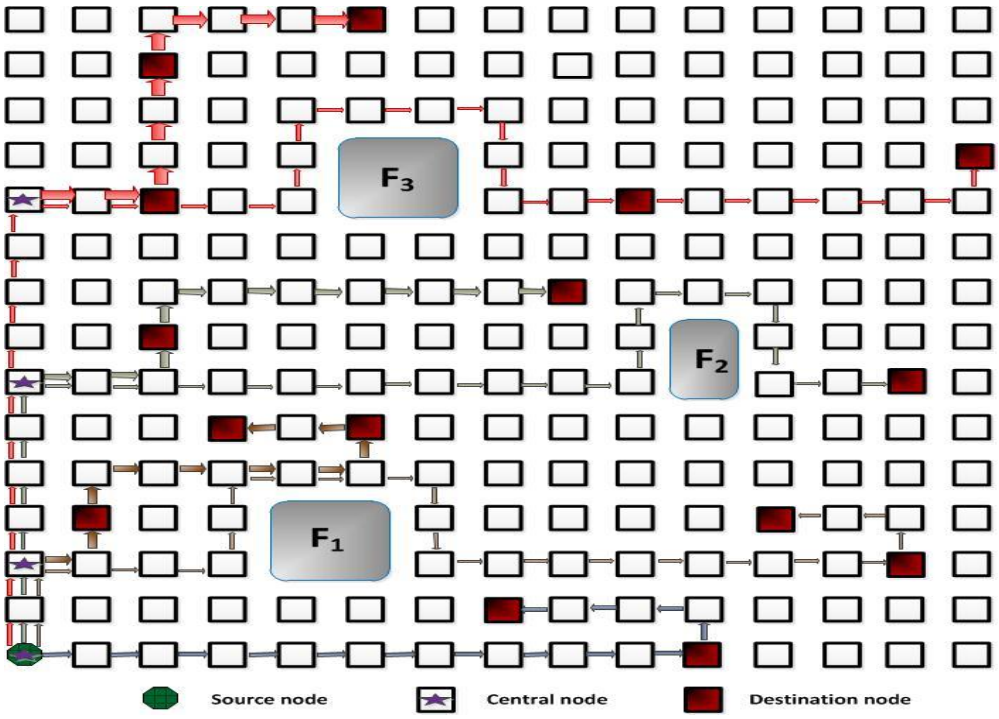



*Figure 29* An example of FTDM routing algorithm.

*Network latency time and network traffic time analysis.* In this subsection, another two important and essential performance metrics in parallel computer systems, network latency time and network traffic time, are calculated. The network latency time is the longest message transmission time involved. The network traffic time is the overall time required to deliver the message to all destination nodes [33]. In general, they are not totally independent. This means that achieving minimum network traffic time may not essentially achieve minimum network latency time at the same time, and vice versa. Network latency time depends on network latency steps while network traffic time depends on network traffic steps. The startup time also affects the value of the network latency and network traffic times. The startup time is the time acquired by the system in preparing the message at the source node to deliver the message to the network and at the destination node to receive the message from the network. It depends on the design of system software within the nodes and the interface between nodes and routers on mesh

networks. From this research, network traffic steps and network traffic time are more significant criteria of measuring the efficiency of fault tolerant multicast routing algorithms.

Also, in this subsection network latency time and network traffic time are calculated for FTDM algorithm and FT-cube2 algorithm. The worst case of network latency time of FTDM algorithm can be calculated by:

$$FTDM\_Latency\_Time = t_{header} * D_{latency\_steps} + t_{copy} * F_{latency\_steps} + t_{channel} * FTDM\_Latency + t_{startup} * (cn+1) \quad (5)$$

Where the time $t_{channel}$ is the channel time between two neighbor nodes and is multiplied by the network latency steps computed by FTDM algorithm, FTDM_Latency. The channel time, $t_{channel}$, equals the sum of the router latency time, $t_r$, and the channel propagation time, $t_p$. The time, $t_{startup}$ is the startup time. The time, $t_{header}$, is the time taken to modify the message header at each destination, so it is multiplied by $D_{latency\_steps}$, which is a set of destinations participating in the longest path. Finally, the time, $t_{copy}$, is the time taken to copy the message at each fault region participates in the longest path, so it is multiplied by $F_{latency\_steps}$, which is a number of fault regions participate in the longest path.

The worst case of network traffic time of FTDM algorithm can be calculated by:

$$FTDM\_Traffic\_Time = t_{header} * |D| + t_{copy} * |F| + t_{startup} * (cn+1) + t_{channel} * FTDM\_Traffic \quad (6)$$

The channel time $t_{channel}$ is multiplied by the network traffic steps computed by FTDM algorithm, FTDM_Traffic. The time $t_{startup}$ is multiplied by two because FTDM algorithm requires at most two startups to deliver a message to any set of destinations, one startup time for each subnetwork of the mesh. The time $t_{header}$ is multiplied by all

destinations, |D|. Finally, the time $t_{copy}$ is multiplied by |F| because FTDM algorithm requires at most |F| message replications, one at each fault region.

The worst case of network latency time of FT-cube2 algorithm can be calculated by:

$$FTcube2\_Latency\_Time = t_{startup} * |D| + t_{channel} * FT\_Latency \quad (7)$$

The channel time $t_{channel}$ is multiplied by the network latency steps computed by FT-cube2 algorithm, FT_Latency. The time $t_{startup}$ is multiplied by |D| because FT-cube2 algorithm requires one startup time to each destination.

Finally, the worst case of network traffic time of FT-cube2 algorithm can be calculated by:

$$FTcube2\_Traffic\_Time = t_{startup} * |D| + t_{channel} * FT\_Traffic \quad (8)$$

The channel time $t_{channel}$ is multiplied by network traffic steps computed by FT-cube2 algorithm, FT_Traffic. The time $t_{startup}$ is multiplied by |D| because FT-cube2 algorithm requires one startup time to each destination node.

*Network latency steps and network traffic steps results.* The equations from 1 to 4 are used to calculate network latency steps and network traffic steps for both algorithms in 2D mesh network. Figure 30 and Figure 31 show the results. The continuous line represents results of FTDM, while the dotted line represents results of FT-cube2.

Figure 30 plots the latency steps for various values of the average number of destination nodes, ranging from 10 to 100. The figure shows that network latency steps computed by FTDM increases as the number of destination nodes increases. The increase in network latency steps will begin to be less significant as the number of destination nodes increase. The increase is not affected by type of the fault region (f-ring and f-chain). Network latency steps computed by FT-cube2 is nearly constant as number of

destination nodes increases. This is because FTDM is a unicast/tree-based multicast routing algorithm, while FT-cube2 is unicast-based multicast routing algorithm.

Figure 31 plots network traffic steps for various values of average number of destination nodes, ranging from 10 to 100. The figure shows that the traffic steps computed by FTDM is nearly constant (slight increase) as number of destination nodes increases. Network traffic steps computed by FT-cube2 are increase as the number of destination nodes increases. The increasing rate of network traffic steps computed by FT-cube2 is large because each destination needs a separate message path.

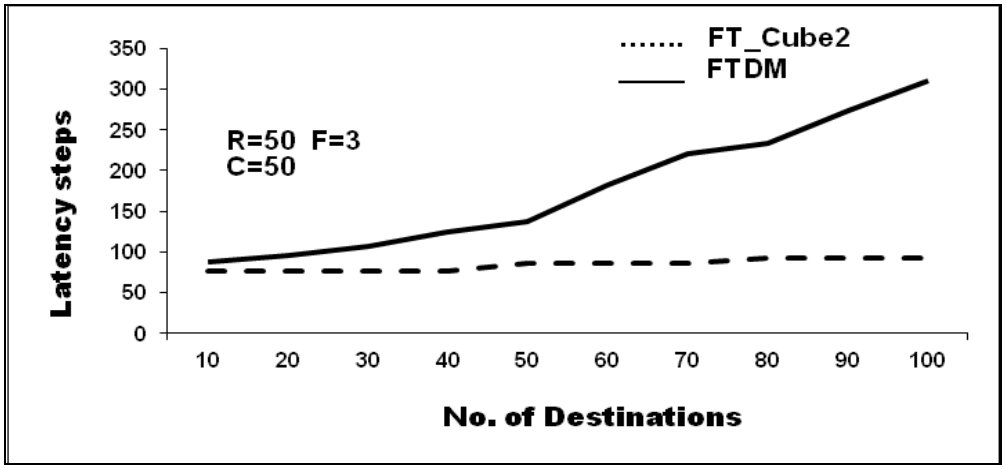



*Figure 30*. Latency Steps Vs. No. of Destinations.

It is obvious that the network traffic steps computed by FTDM is less than that computed by FT-cube2.

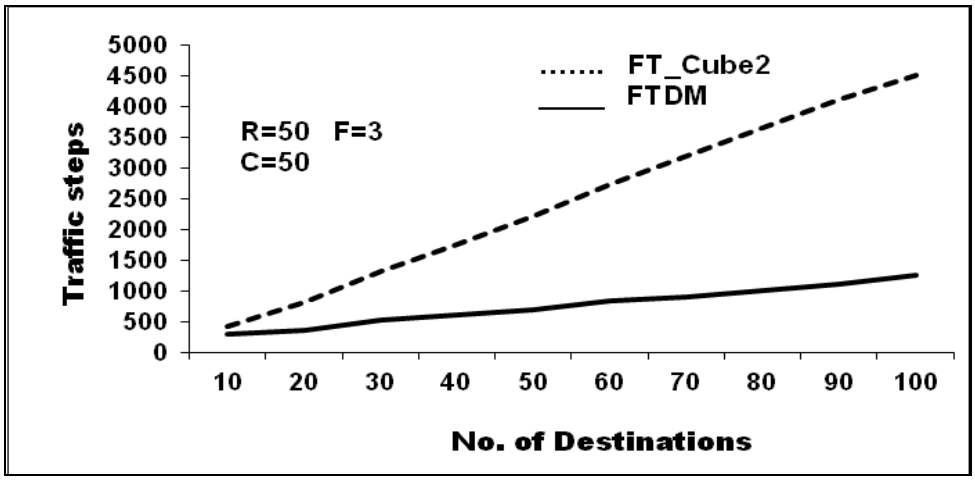



*Figure 31*. Traffic Steps Vs. No. of Destinations.

*Network latency time and network traffic time results.* The equations from 5 to 8 are used to calculate network latency time and network traffic time for both algorithms. Figures 32 and 33 show the results.

Figure 32 plot network traffic time for various values of average number of destination nodes, ranging from 10 to 100. The figure shows that network traffic time computed by FTDM algorithm is nearly constant as number of destination nodes increases, while network traffic time computed by FT-cube2 algorithm increases and FTDM algorithm is less than FT-cube2 algorithm. This is because network traffic time values depend on network traffic steps values.

Figure 33 plots network latency time for various values of average number of destination nodes, ranging from 10 to 100. The figure shows that network latency time computed by the two algorithms increases as the number of destination nodes increases. Clearly, at small average number of destination nodes, FTDM algorithm outperforms FT-cube2 algorithm, while at large average number of destination nodes, FT-cube2 algorithm is less than FTDM algorithm. At medium average number of destination nodes, the two curves nearly are the same.

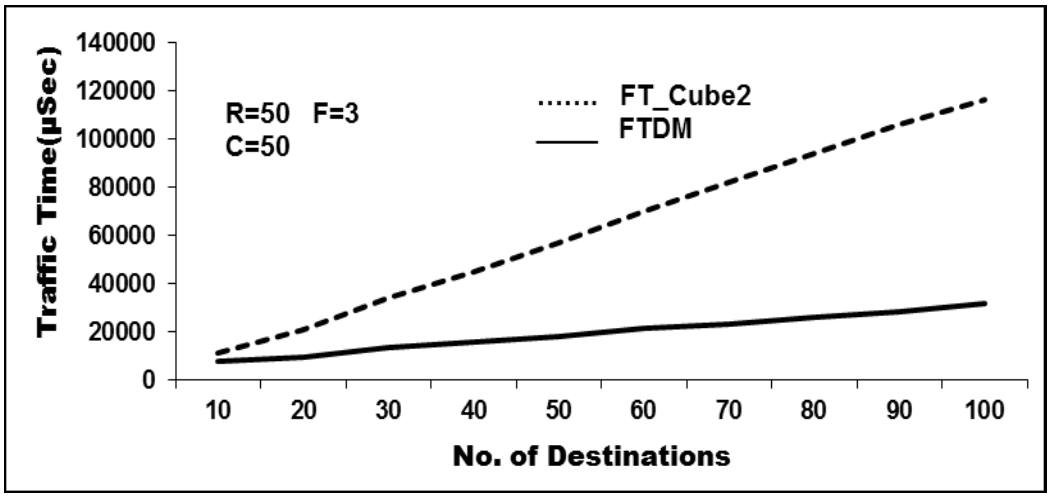



*Figure 32*. Network Traffic Time Vs. No. of Destinations.

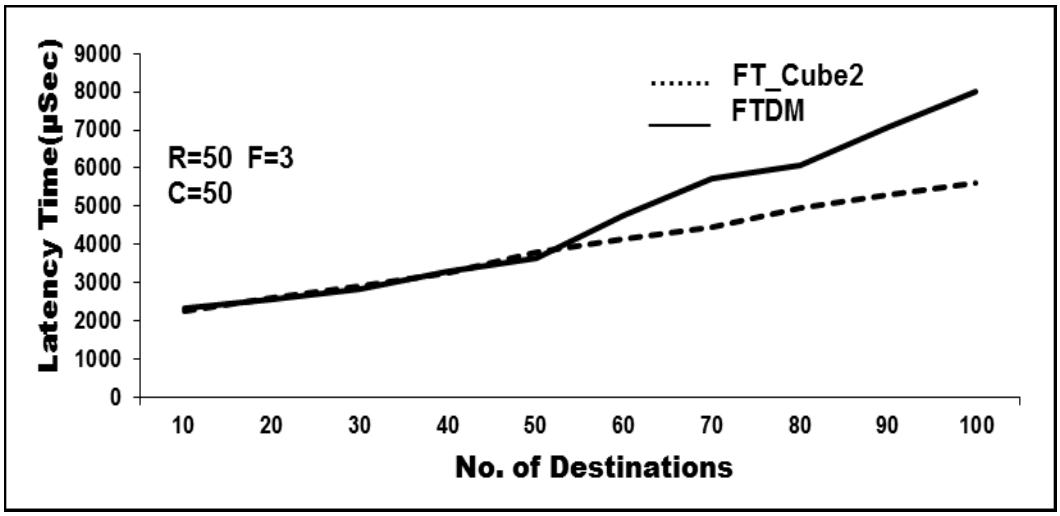



*Figure 33*. Network Latency Time Vs. No. of Destinations.

Generally, from the previous figures, As number of destinations network traffic steps and network traffic time computed by FTDM algorithm is nearly constant, while that computed by FT-cube2 algorithm increases and FTDM algorithm is very effective than FT-cube2 algorithm.

As explained in the previous section network traffic steps and network traffic time are more significant criteria of measuring the efficiency of fault tolerant multicast routing algorithms. Hence, FTDM is more efficient than FT-cube2.

## iFTDM Routing Algorithm

Most fault tolerant multicast routing algorithms which were proposed recently concentrate on unicast-based multicast algorithms. Unicast-based algorithms require a startup time for each destination and this requires more work. Also, they are incompetent because they permit a message to be delivered to only one destination, which leads to multicast operations being implemented as multiple phases of multicast message exchange. Hence, contention freedom must be guaranteed not only among the worms of a given phase, but also among worms in different phases [96].

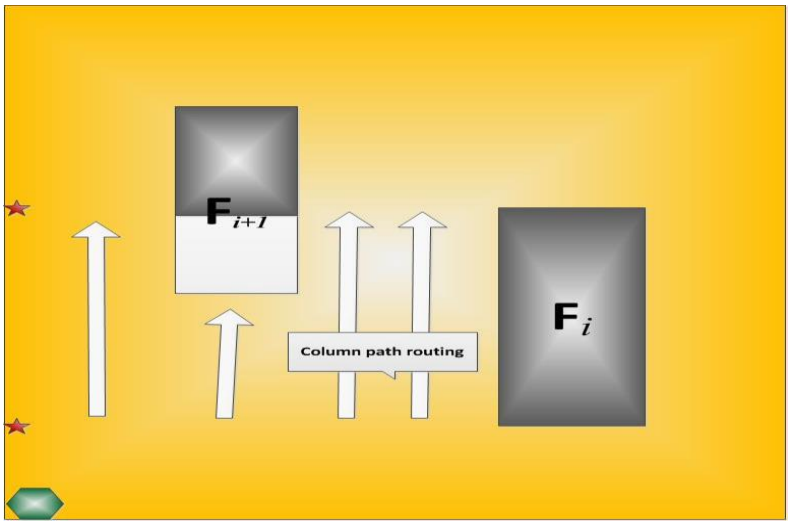
Furthermore, many fault tolerant multicast routing algorithms which were proposed recently concentrate on number of destinations as the main parameter that must be consider in calculating network latency steps and network traffic steps in mesh networks [60], [87]. On the other hand, the effect of number of fault regions and fault region size are not taken into consideration when they calculate network latency steps and network traffic steps for their algorithms.

In this section, a new fault tolerant deadlock-free multicast routing algorithm, *i*FTDM for 2D mesh networks, is introduced. *i*FTDM (*i*mproved FTDM) is a unicast/tree-based multicast algorithm, which attempts to deliver the message to all destinations in two phases, the same method as FTDM work. In the first phase the message is delivered as multicast unicast-based to X-coordinate nodes – nodes (0, $y_{bi}$) in case of odd rows or (m-1, $y_{bi}$) in case of even rows – of each *true fault regions* at these nodes; *central nodes*. We consider each node of them as a source node that has a message with header containing destinations in the three locations around the fault. In the second phase, the message is delivered from the central nodes in multicast tree-based fashion, which attempts to route the message to all destinations in a single multi-head worm that

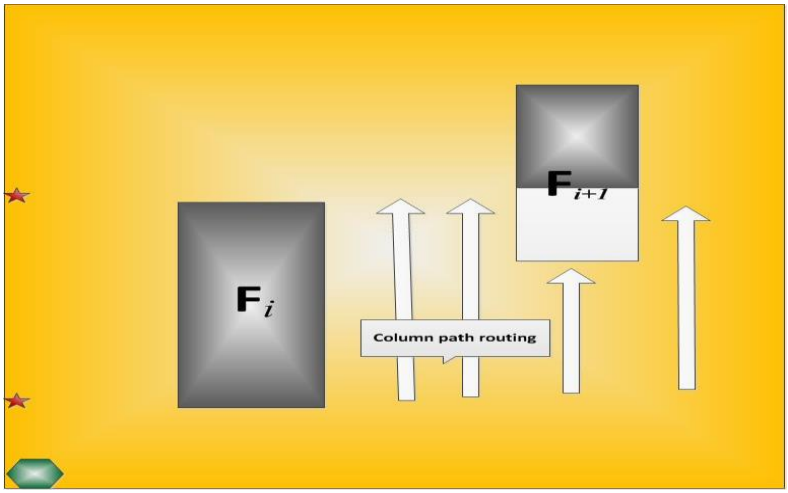
splits at some routers and replicates the data on multiple output ports. In a 2D mesh $L_1$, $L_2$ and $L_3$ are three locations around each true fault regions as in Figure 27, and $L_4$ is a location in case if the 1st fault region is an f-ring.

*Overlapping convex fault regions*

The proposed fault tolerant multicast routing algorithm, *i*FTDM, does tolerate overlapping convex faulty regions. If there is an overlapping fault region, then there is no $L_2$ will formed in between the overlap fault regions at this point there are two cases of overlapping, the first case is overlap on $L_1$ which $y_b$ of $F_{i+1}$ less than $y_e$ of $F_i$ as shown in Figure 34(a), the second case is overlap on $L_3$ which $y_b$ of $F_i$ less than $y_e$ of $F_{i+1}$ as shown in Figure 34(b). In Figure 34 star shapes represents central nodes.




(a)




(b)

*Figure 34.* Overlapping on $L_1$ and $L_3$.

In order to tolerate the overlapping of fault regions, we need to use the column path routing technique as we described it in the background chapter.

The *i*FTDM routing algorithm uses the same routing functions, R and R′, which FTDM used. Both R and R′ were proved deadlock-free. Also, *i*FTDM assigns a label for each node based on the position of that node in a Hamiltonian path.

***Lemma***: *i*FTDM algorithm is deadlock-free

*Proof:* Because *i*FTDM use R, R′and column path routing to route a message on a 2D mesh with convex faults and all of these are deadlock-free (no cyclic dependency can be created among the channels) as mentioned. Then *i*FTDM is deadlock-free.

**Algorithm (FTDM vs *i*FTDM)**

**In previous section we introduced FTDM algorithm,**

**Input**: The message *mess***,** Label node LN, central nodes $CN_k$, destination set D, and fault region $F_i$ .

**Output**: $\forall\ d_j \in D$, Receive($d_j$ , mess)

**Procedure**:

[1]/* Phase 1 (unicast-based): Send copies of message to $CN_k$

**[a]** If $c = d_1$ then

a. 1) D = D-{c}

a. 2) Receive(c, *mess*)
**[b]** If D = ϕ then stop
**[c]** Send separate messages to $CN_k$ using XY routing.
**[d]** Modify header of messages, *mess*, and put in each header $D_k$ destinations, which k is the number of central nodes (plus one if first fault is f-ring)
**[e]** Let each $CN_k$ as a new source node
**[f]** Go to phase 2

[2]/* Phase 2 (tree-based):
**[a]** If c = $d_1$ then
a. 1) D = D-{c}
a. 2) Receive(c, *mess*)
**[b]** If D = ϕ then stop
**[c]** At each new source node, send two copies of message, *mess*, we have three cases:
***Case 1 (No overlap)***
c. 1) 1st copy contains destinations on $L_1$ and $L_2$ using R
c. 2) 2nd copy contains destinations on $L_3$. Using R' to route a message around the fault region until the message reach to LN, and then use R.
c. 3) If $L_3$ have another faults then recursively apply *i*FTDM.

**In this section we proposed the following improvement (*i*FTDM) algorithm:**
***Case 2 (overlap in* $L_1$*)***
c. 1) 1st copy contains destinations on $L_1$ using *column path* routing.
c. 2) 2nd copy contains destinations on $L_3$. Using R' to route a message around the fault region until the message reach to LN, and then use R.
c. 3) If $L_3$ have another faults then recursively apply *i*FTDM.
***Case 3 (overlap in* $L_3$*)***
c. 1) 1st copy contains destinations on $L_1$ using R
c. 2) 2nd copy contains destinations on $L_3$. Using R' to route a message around the fault region until the message reach to LN, and then use *column path* routing.
c. 3) If $L_3$ have another faults then recursively apply *i*FTDM.
**[d]** Repeat the above steps until each destination in the message header is reached.

*Pseudocode 2*. *i*FTDM Routing Algorithm

*Results and Performance Analysis*

Because of *i*FTDM is an improved version of FTDM, the same simulation study used in FTDM where F is used to represent the number of fault regions, R is the number of rows, and C is the number of columns. This configuration creates different networks with a number of processors ranging from 100 to 1080. The size of fault region is ranging from 2×2 to 20×20 using 25 destination nodes. The number of fault regions is ranging

from 1 to 10 using 30 destination nodes. Using equations from 1 to 8 to calculate network traffic steps, network traffic time, network latency steps and network latency time as mentioned in FTDM section.

*Network latency steps and network traffic steps results.* By using equations from 1 to 4 network latency steps and network traffic steps for both algorithms in 2D mesh are calculated. Figures 35, 36, 37 and 38 show the results. The continuous line represents results of *i*FTDM, while the dotted line represents results of FT-cube2.

Figure 35 plots network latency steps for various values of number of fault regions, ranging from 1 to 10 and |D| is equal to 30. The figure shows that, network latency steps computed by *i*FTDM algorithm decreases as number of fault regions increases, while network latency steps computed by FT-cube2 algorithm is nearly constant. Obviously, at small number of fault regions, network latency steps computed by FT-cube2 algorithm is less than that computed by *i*FTDM algorithm, while at large number of fault regions, network latency steps computed by *i*FTDM algorithm is less than that computed by FT-cube2 algorithm.

Figure 37 plots network latency steps for various sizes of one fault region, ranging from 4 to 400 where R x C= 2x2 to 20x20 and |D| is equal to 25. The figure shows that network latency steps computed by *i*FTDM algorithm decreases as size of the fault region increases, while network latency steps computed by FT-cube2 algorithm is nearly constant. So, when the size of the fault region increases, the number of used channels increases. Noticeably, at small fault region sizes, network latency steps computed by FT-cube2 algorithm is less than that computed by *i*FTDM algorithm while at large fault region sizes, network latency steps computed by *i*FTDM algorithm is less than that computed by FT-cube2 algorithm.

Figure 36 plots network channel traffic steps for various values of number of fault regions, ranging from 1 to 10 and |D| equals 30. The figure shows that network traffic steps computed by *i*FTDM algorithm is nearly constant (slight increase) as number of fault regions increases, while network traffic steps computed by FT-cube2 algorithm nearly constant, but greater than with that is calculated by *i*FTDM algorithm.

Figure 38 plots network traffic steps for various sizes of one fault region, ranging from 4 to 400 where R x C=2x2 to 20x20 and |D| is equal to 25. The figure shows that network traffic steps computed by *i*FTDM algorithm is nearly constant (slight decrease) as size of the fault region increases, while network traffic steps computed by FT-cube2 algorithm is nearly constant, but greater than what is calculated by *i*FTDM algorithm. Also, this is because the path computed by FT-cube2 algorithm circles around the fault region.

In all tested cases, network traffic steps computed by *i*FTDM is less than that computed by FT-cube2.

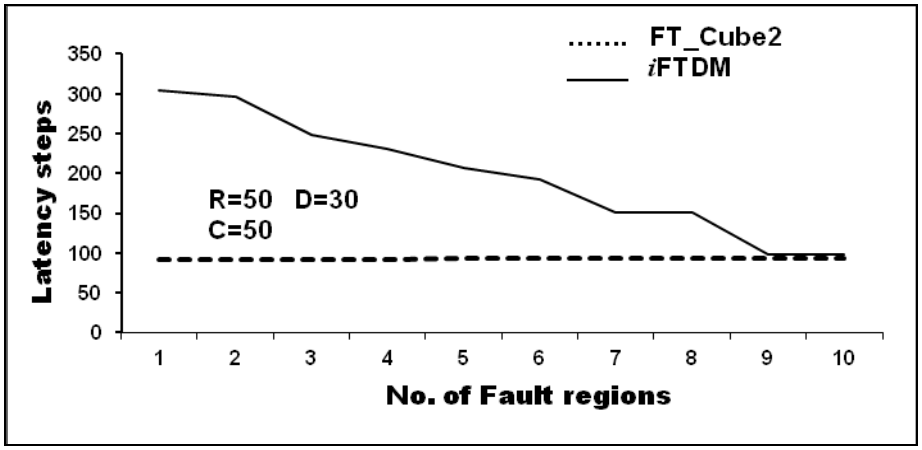



*Figure 35.* Latency Steps Vs. No. of Fault regions.

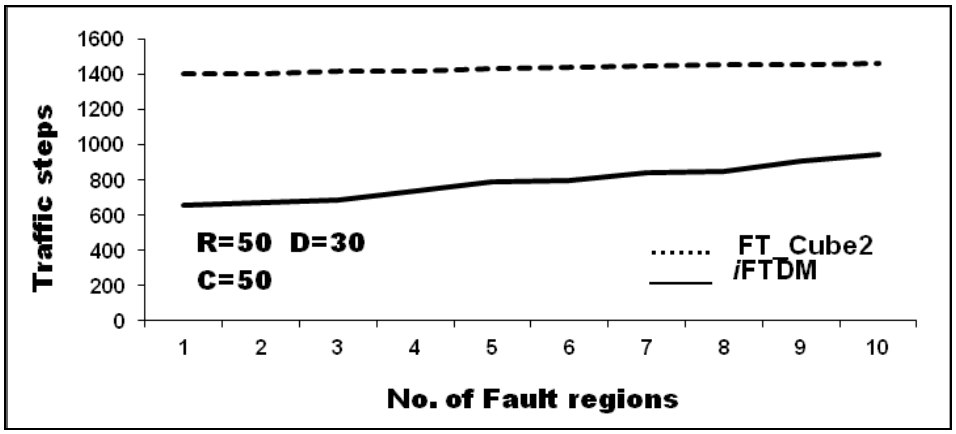



*Figure 36.* Traffic Steps Vs. No. of Fault regions.

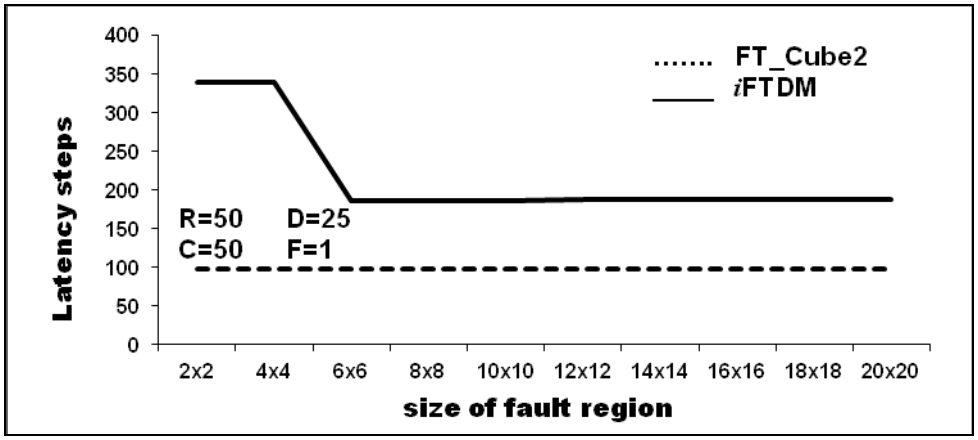



*Figure 37.* Latency Steps Vs. Size of fault region.

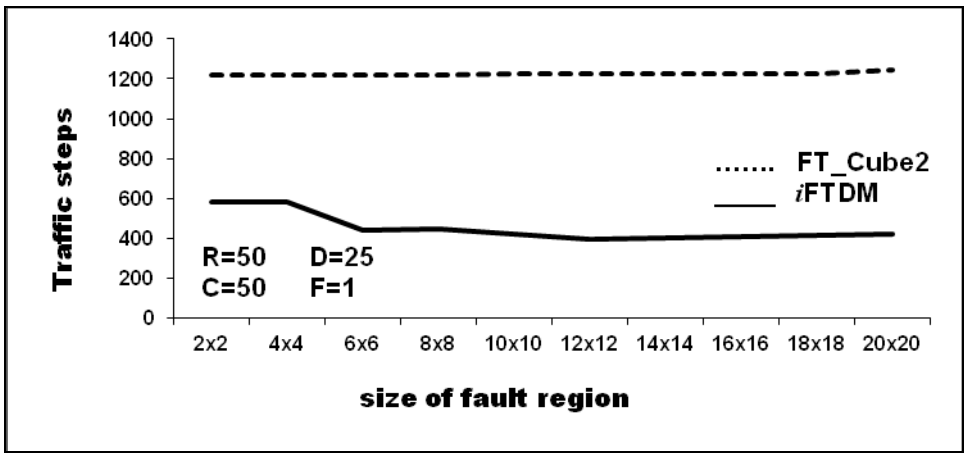



*Figure 38.* Traffic Steps Vs. Size of fault region.

*Network latency time and network traffic time results.* The equations from 5 to 8 are used to calculate network latency time and network traffic time for both algorithms. Figures from 39 to 42 show the results.

Figure 39 plots network traffic time for various values of number of fault regions, ranging from 1 to 10 and |D| is equal to 30. The figure shows that network traffic time computed by *i*FTDM algorithm is nearly constant as number of fault regions increases, while network traffic time computed by FT-cube2 algorithm increases and *i*FTDM algorithm is better than FT-cube2 algorithm.

Figure 40 plots network latency time for various values of number of fault regions, ranging from 1 to 10 and |D| equals 30. The figure shows that network latency time computed by *i*FTDM algorithm decreases as number of fault regions increases, while network latency time computed by FT-cube2 algorithm nearly constant. Also, at small number of fault regions, network latency time computed by FT-cube2 algorithm is less than that computed by *i*FTDM algorithm while at large number of fault regions, network latency time computed by *i*FTDM algorithm less than that computed by FT-cube2 algorithm.

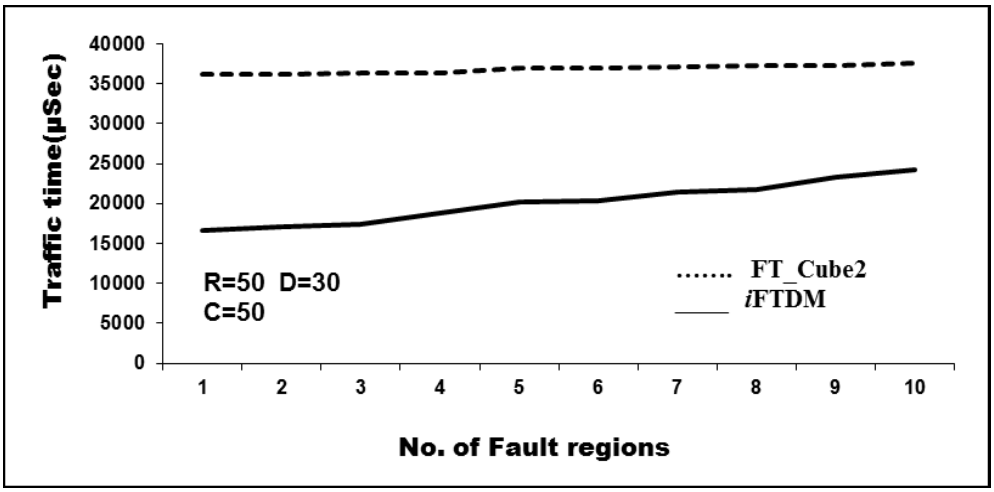



*Figure 39.* Network Traffic Time Vs. No. of Fault regions.

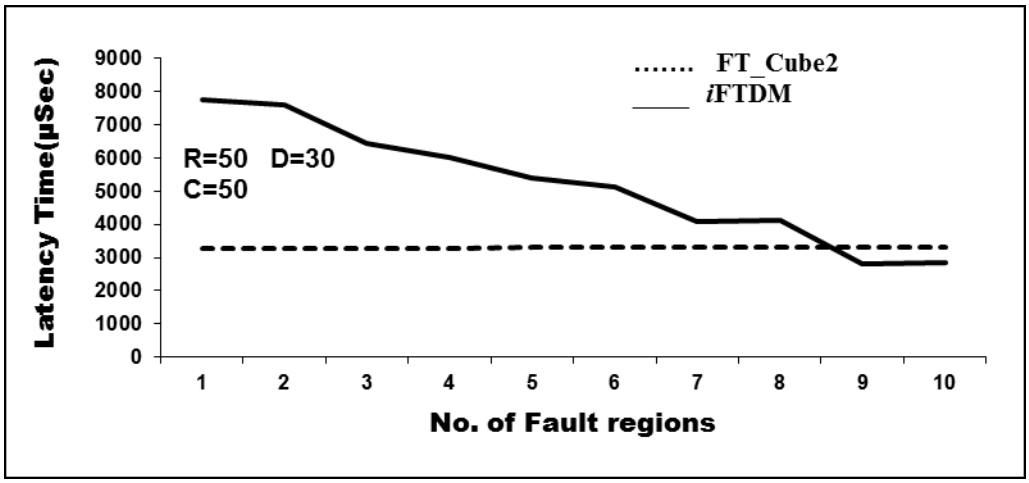



*Figure 40.* Network Latency Time Vs. No. of Fault regions.

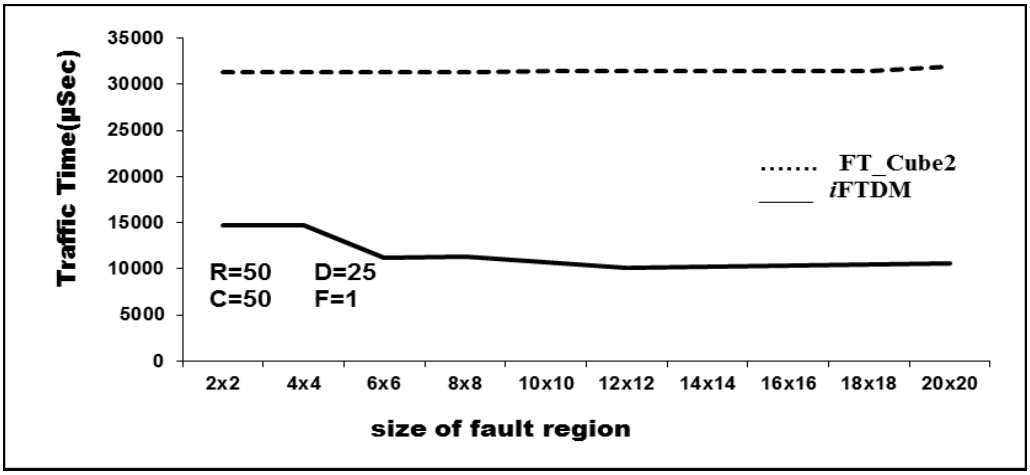



*Figure 41.* Network Traffic Time Vs. Size of fault regions.

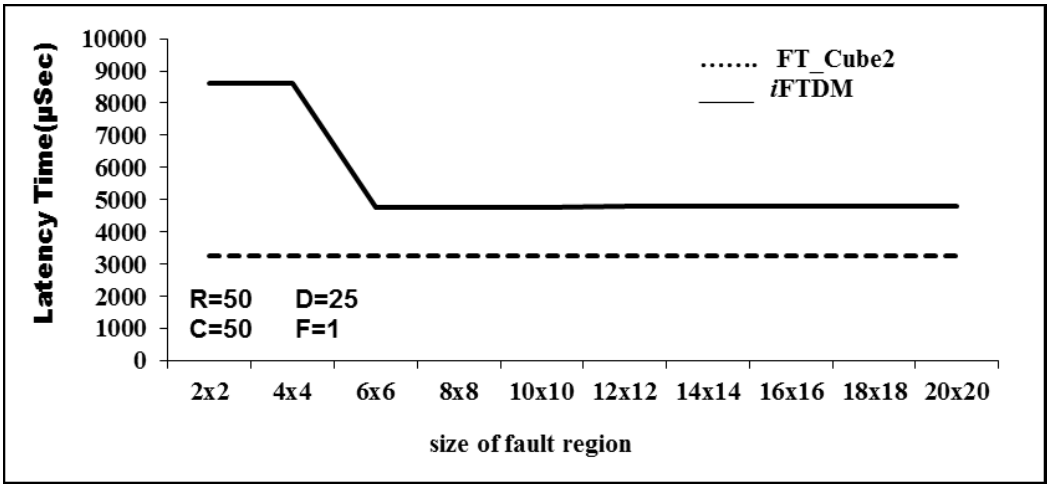



*Figure 42.* Network Latency Time Vs. Size of fault regions.

Figure 41 plots network traffic time for various sizes of one fault region, ranging 2×2 to 20×20 and |D| equals 25. The figure shows that network traffic time computed by *i*FTDM algorithm is nearly constant with a slight decrease as size of the fault region increases, while network traffic time computed by FT-cube2 algorithm nearly constant and less than *i*FTDM algorithm.

Figure 42 plots network latency time for various sizes of one fault region, ranging from 2×2 to 20×20 and |D| is equal to 25. Clearly, network latency time computed by *i*FTDM algorithm decreases as size of the fault region increases, while network latency time computed by FT-cube2 algorithm nearly constant and less than *i*FTDM.

In general, from the previous figures, the following notes can be observed:

- As size of the fault region and number of fault regions increase, network traffic steps and network traffic time computed by *i*FTDM algorithm is nearly constant, while that computed by FT-cube2 algorithm increases and *i*FTDM algorithm is very effective than FT-cube2 algorithm.
- As size of the fault region and number of fault regions increase, network latency steps and network latency time computed by *i*FTDM algorithm increases, while that computed by FT-cube2 algorithm is nearly constant. In most tested cases, network latency steps and network latency time computed by *i*FTDM algorithm is larger than that computed by FT-cube2 algorithm. Also, at a small number of fault regions, network latency time computed by FT-cube2 algorithm is less than that computed by *i*FTDM algorithm while at a large number of fault regions, network latency time computed by *i*FTDM algorithm less than that computed by FT-cube2 algorithm.

As explained in the previous section network traffic steps and network traffic time are more significant criteria of measuring the efficiency of fault tolerant multicast routing algorithms. Hence, *i*FTDM is more efficient than FT-cube2.

# CHAPTER V

## DATA MINING FOR PROPOSED ROUTING ALGORITHMS

Data mining is the process of mining patterns and new information from data. Data mining is seen as a progressively significant tool by recent business to transform data into business intelligence giving an informational benefit. Marketing, surveillance, fraud detection, and scientific discovery are a wide range of profiling practices presently used by data mining. On the whole, spatial data mining, or knowledge discovery in spatial bases data is the mining of implicit knowledge, spatial relations and the discovery of interesting characteristics and patterns that are not explicitly represented in the data bases. These methods are essential in understanding spatial data and in capturing intrinsic connections between spatial and non-spatial data. Furthermore, such exposed connections and relationships can be used to present data in a brief manner and to rearrange spatial databases to accommodate data semantics and accomplish high performance [97].

Traditional statistical techniques such as linear regression were basically hands-on technologies that operated on small, static datasets to validate hypotheses or models. Conversely, new data mining tools and methods are capable of not only understanding tremendously large and complex datasets but also inferring those relationships as trends and predictions [98], [99].

Distributed-memory systems are the most favorable architectures used in advanced research problems. 2D mesh networks are popular architectures that have been implemented in many distributed-memory systems. These systems must support communication operations efficiently to achieve good performance. Development of fault tolerant multicast routing algorithms in 2D mesh networks is an important issue. FTDM, fault tolerant deadlock-free unicast/tree-based multicast routing algorithm for 2D mesh

multicomputer [96] is compared with another fault tolerant deadlock-free unicast-based multicast routing algorithm for 2D mesh algorithm called FT-cube2 [60]. FT-cube2 algorithm matches and differs in some points with our algorithm as presented in the previous chapter. Chapter IV contains a detailed analysis of results calculated by several significant criteria such as network latency steps, network traffic steps, network latency time and network traffic time for both algorithms with studying the effect of a changing number of destination nodes.

In this chapter, data mining techniques have been used to validate results obtained from algorithms presented in Chapter IV. This has been done by enlarging the number of destinations network traffic steps, which have been used to analyze data results. Data mining regression analysis is one of the best features to understand mapping and relationship between dependent and independent variables. Regression analysis using different data mining software tools is presented in the next section.

## Regression Analysis

Regression is an easy and simple technique to use. This model can be as easy as one input variable (dependent) and one output variable (independent). Obviously, it can acquire more complexity than that, including many input variables. There are several independent variables, when taken together produce a result with a dependent variable. The regression model is then used to predict the result of an unidentified dependent variable, given the values of the independent variables [97].

Regression analysis involves any techniques or methods for modeling and analyzing numerous variables when the focus is on mapping the relationship between a dependent variable and one or more independent variables. Additionally, regression analysis helps us to understand how the typical value of the dependent variable changes

when any one of the independent variables is varied, while the other independent variables are held fixed. Regression analysis is broadly used in many applications for prediction and forecasting, where its use has considerable intersection with the field of machine learning. Regression analysis is also used to understand and recognize which among the independent variables are correlated to the dependent variable, and to discover types of these relationships [97].

## Methods and Analysis

Multiple regressions have been done for FTDM result data using WEKA [100], EXCEL [101] and MATLAB [102]. By changing the number of destinations, network traffic steps have been observed for both FTDM and FT-cube2 algorithms. In the following sections regression analysis produced by this data mining software is introduced.

### *Data Mining Using WEKA*

Data mining software are not exclusively the domain of big businesses and expensive software. Actually, there is good and free software which has almost all the same features and is as good as other expensive. WEKA is the product of the University of Waikato (New Zealand) and was first implemented and developed in its modern form in 1997. It uses the GNU General Public License (GPL). WEKA software is written in the Java™ language and contains a GUI for interacting with data files and producing visual results [100]. In addition, WEKA has a general API. Hence, WEKA can be embedding in application; like any other library. WEKA startup screen shown in Figure 43.

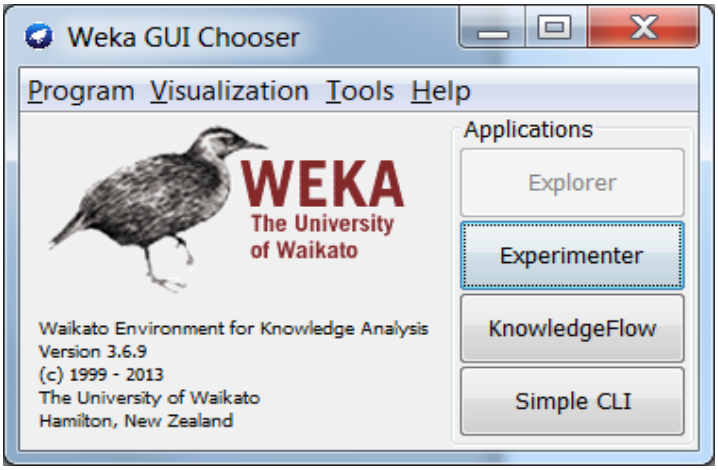



*Figure 43*. WEKA startup screen.

Our data are numerical and the relation measure here is between number of destinations and network traffic steps of FTDM and FT-Cube2 fault tolerant routing algorithms. The following is a brief description of steps of using WEKA software:

- *First step* is building the data set for WEKA. In order to load data into WEKA, put it into a format that will be understood and accepted by WEKA. WEKA's most popular method for loading data is in the Attribute-Relation File Format (ARFF), where the type of data being loaded is defined. After that, supplies the data itself. In the file each column and what each column contains are defined. In the case of the regression model, numeric or a date column are limited. Finally, each row of data in a comma-delimited format is supplied.
- *Second step* is starting WEKA, and then choose the *Explorer*. The next step is go to the *Explorer* screen, with the *Preprocess* tab selected. Select the *Open File* button and select the ARFF file. With WEKA review the data can do. In the left section of the *Explorer* window, it outlines all of the columns in FTDM result data (Attributes) and the number of rows of data supplied (Instances). By

selecting each column, the right section of the *Explorer* window will also give information about the data in that column of data set.

- *Third step* is creating the regression model with WEKA. To create the model, select the *Classify* tab. Then chose the desired model and where the data is that it should use to build the model. However it may be noticeable that to use the data supplied in the ARFF file, there are actually different options, some more advanced than what we'll be using. The other three selections are supplied *test set*, where you can supply a different set of data to build the regression model: *Cross-validation*, which lets WEKA build a model depending on subsets of the supplied data and then average them out to create a final model; and *Percentage split*, where WEKA takes a subset of the supplied data to build a final model. These other choices are useful with different models, depending on the nature of data sets. With regression, FTDM result data can simply choose *Use training set* to build the desired model data set supplied in ARFF file [100].

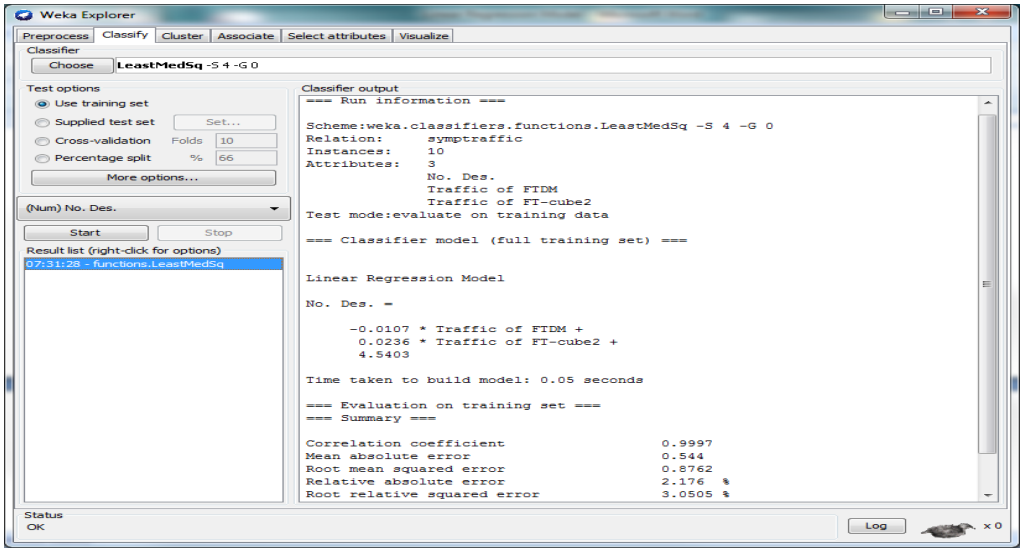



*Figure 44*. Regression output model in WEKA.

- Finally, the last step to creating the model is to choose the dependent variable (the column we are looking to predict). We know this should be No. of destinations, since that's what we're trying to determine for FTDM result data. Right below the test options, select a combo box that lets you choose the dependent variable. Then select the column No. of destinations, the output, as shown in Figure 44.

*Data Mining Using EXCEL*

The following is a brief description of steps of using EXCEL software:

- *First step* is to prepare data by writing each attribute in only one column, and then we decide the dependent and independent attributes [101].
- *Second step* is adding-in the data analysis tool pack EXCEL software. Statistical data analysis such as descriptive statistics and regression needs the Excel Data Analysis add-in. The default configuration of Excel software does not spontaneously support descriptive statistics and regression analysis.
- *Third step* is to do multiple regressions using the data analysis Add-in. In this case the data obtained from both routing algorithms are numerical and the relation between number of destinations and network traffic steps of FTDM and FT-cube2 routing algorithms. In the next lines a brief analysis for this case is introduced. We have regression with an interrupt and regressors, network traffic steps of FTDM and FT-cube2 algorithms. The population regression model is given by: $y = \beta1 + \beta2\ x2 + \beta3\ x3 + u$

It is supposed that the error term u is independent with a constant variance. The estimated regression line is given by: $y = b1 + b2\ x2 + b3\ x3$

Using the data analysis add-in and regression the only change over one-variable regression is to contain more than one column in the Input X Range. However, we need

adjoining columns to do this regression. If this is not the case in the original data, then columns need to be copied to get the regressors in adjoining columns.

SUMMARY OUTPUT

| *Regression Statistics* | |
|---|---|
| Multiple R | 0.999783033 |
| R Square | 0.999566112 |
| Adjusted R S | 0.999442144 |
| Standard Err | 0.715099808 |
| Observation | 10 |

ANOVA

| | *df* | *SS* | *MS* | *F* | *gnificance F* |
|---|---|---|---|---|---|
| Regression | 2 | 8246.420426 | 4123.21 | 8063.102 | 1.7E-12 |
| Residual | 7 | 3.579574151 | 0.511368 | | |
| Total | 9 | 8250 | | | |

| | Coefficients | Standard Error | t Stat | P-value | Lower 95% | Upper 95% | Lower 95.( | Upper 95.0% |
|---|---|---|---|---|---|---|---|---|
| Intercept | 0.402546705 | 2.358224537 | 0.170699 | 0.86929 | -5.17377 | 5.978862 | -5.17377 | 5.978862 |
| Traffic of FT( | 0.005371196 | 0.010982003 | 0.489091 | 0.639729 | -0.0206 | 0.03134 | -0.0206 | 0.03134 |
| Traffic of FT- | 0.020409198 | 0.002472204 | 8.255468 | 7.45E-05 | 0.014563 | 0.026255 | 0.014563 | 0.026255 |

*Figure 45*. Regression output model in EXCEL.

The regression output has three components, as shown in Figure 45: regression statistics table, ANOVA table and regression coefficients table (which are important to build the relation between dependent and independent variables).

*Data Mining Using MATLAB*

MATLAB is popular data mining software tool which is a numerical computing environment developed by MathWorks. Implement multiple linear regressions to find the best fit of dependent variable as a linear function of independent variables [102]. For example, in the X-and-Y data:

X = [1766 1422 1077 1120 1040 1023 840];

Y = [1.22 0.97 1.32 1.56 0.72 0.35 1.55];

We believe that there is a straight line relationship between the number of X and the number of Y, and we need to find the parameters a and b in the linear formula

Y = b + a * X

MATLAB facilities can do this with [a, b] = regress (Y', X')

There are two important observations about this: (1) the first returned argument is the slope a in the linear formula, the second is the y-intercept term b. (2) the ' operator has been used to transpose the data, which is in row vector form, into column data. When there is more than one independent variable, regression will carry out multiple regressions. Actually, this is done by packaging the independent variables into a matrix, with one dependent variable per column [102].

NO_Dest = b + a1 * Traffic_FTDM + a2 * Traffic_FT_cube2

The parameters a1, a2, and b can be found using regression:

[a,b]=regress(NO_Dest', [Traffic_FTDM', Traffic_FT_cube2'])

Our data are numerical and consists of two values, one for each of the two independent variables used in the regression. The following is a brief description of steps of using MATLAB software:

- *First step* is putting data in the following format:

  NO_Dest=[10 20 30 40 50 60 70 80 90 100];

  Traffic_FTDM=[305 367 538 619 698 841 911 1017 1109 1253];

  Traffic_FT_cube2=[431 821 1316 1755 2212 2734 3187 3651 4114 4515];

- *Second step* is writing the regression function for three (multiple) variables:

  [a,b]=regress(NO_Dest', [Traffic_FTDM', Traffic_FT_cube2'])

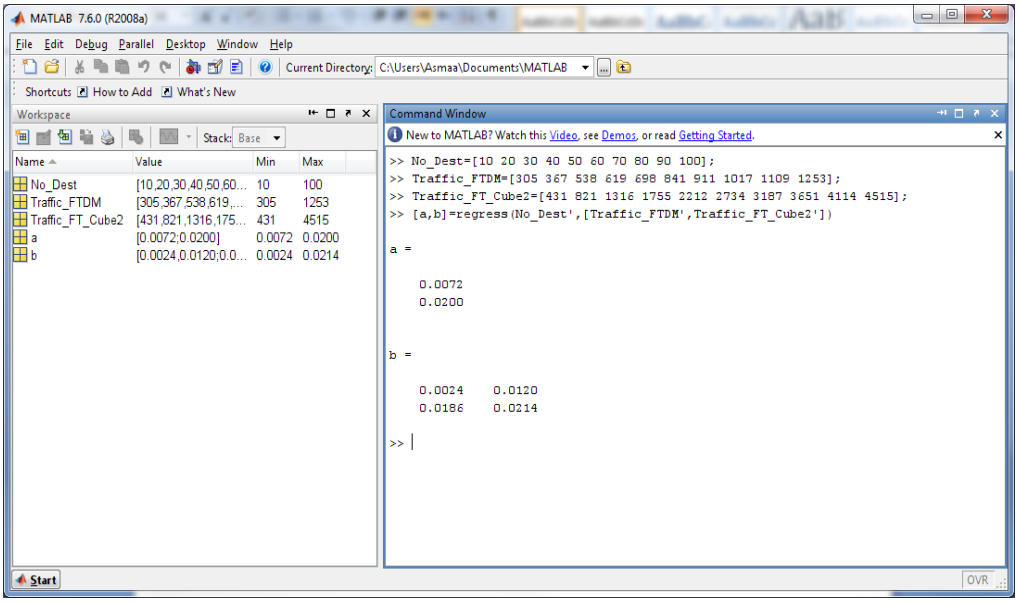



*Figure 46*. Regression output model in MATLAB.

## Results

We obtained from using these three data mining programs and from plotting our data that as number of destinations increases, network traffic steps computed by FTDM and *i*FTDM routing algorithm are less than that computed by FT-cube2 routing algorithm. MATLAB was used to enlarge data set to 1000 destinations instead of 100 on the original simulations as shown in Figure 47 in case of FTDM. Also, MATLAB was used to enlarge data set to 1000 fault regions instead of 10 on the original simulations as shown in Figure 48 in case of *i*FTDM

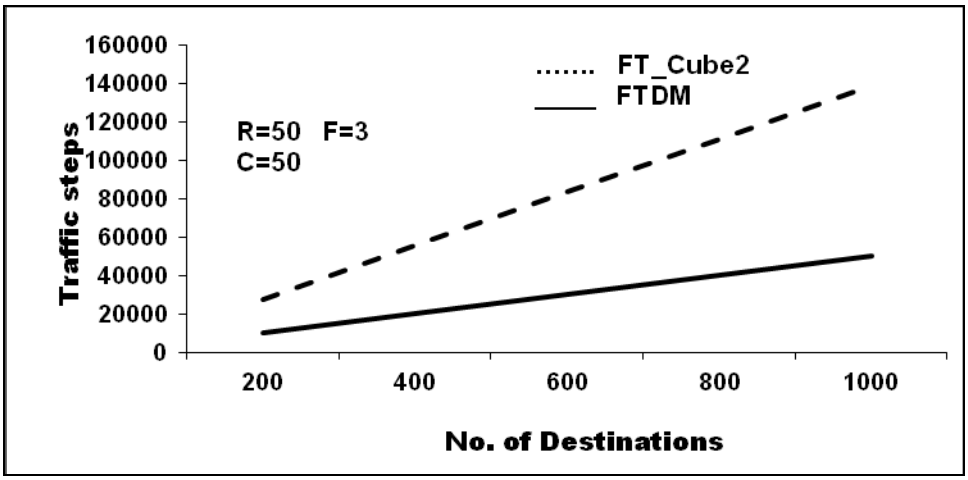



*Figure 47.* Validate comparison between FTDM and FT-cube2.

Figure 47 plots network traffic steps for various values of average number of destination nodes, ranging from 200 to 1000. The figure shows that the traffic steps computed by FTDM is nearly constant (slight increase) as number of destination nodes increases (the same behavior as in Figure 31). The increasing rate of network traffic steps computed by FT-cube2 is larger than FTDM (the same behavior as in Figure 31). The results shown on both Figure 47 and Figure 31 are similar; this validates results when the sample size of number of destinations is enlarged.

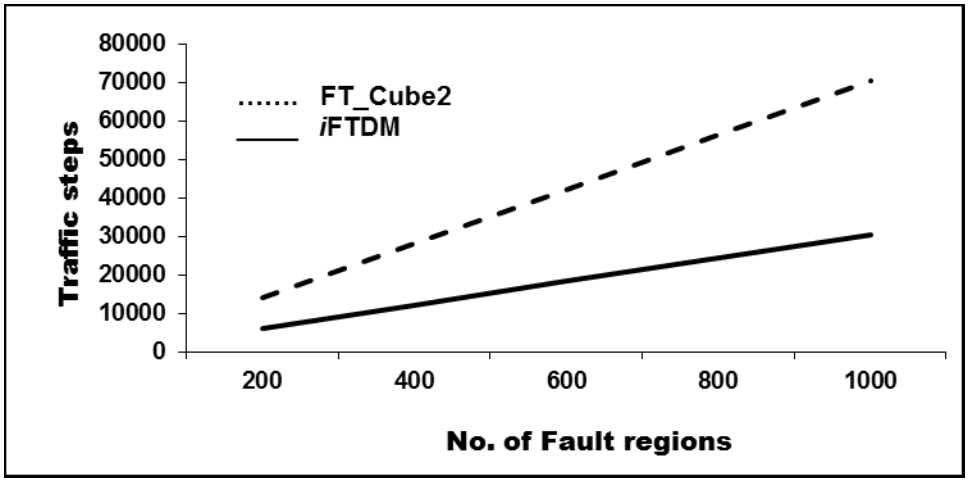



*Figure 48.* Validate comparison between *i*FTDM and FT-cube2.

Figure 48 plots network traffic steps for various values of number of fault regions ranging from 200 to 1000. From plotting our data as number of fault regions increases, network traffic steps computed by *i*FTDM routing algorithm is less than that computed by FT-cube2 routing algorithm. Using MATLAB results we can enlarge our data to 1000 destinations instead of 100 on the original simulations as shown in Figure 48.

Using data mining validation tools  the new fault tolerant multicast routing algorithms (FTDM and *i*FTDM) compared with competitive fault tolerant multicast routing algorithm (FT-cube2) are more effective in case of enlarge data set as well as in original simulations mentioned in Chapter IV.

# CHAPTER VI

# YASSIN ROUTING ALGORITHM

As processors in distributed-memory systems need to communicate with each other, effective communication is crucial to improve the performance of the system. In a distributed memory system with hundreds and maybe thousands of processors, fault tolerance is an important issue which is defined as the capability of the system to function in the occurrence of component (processor or connections) failures. The challenge is how to effectively accomplish routing in a faulty network, where each element fails with various probabilities. The current generation of these systems are very powerful and do not appear to fail very often in practice. However, even in some superior environments, fault tolerance ability must be addressed no matter how remote the probability of component failures is.

As stated in Chapter IV, mesh connected networks have been widely used in most distributed-memory systems. These computer systems generally use the e-cube routing algorithm with wormhole switching because of its simplicity. The main idea of e-cube algorithm is to route a message first along the row and then along the column in a 2D mesh. It is important to note that e-cube provides deadlock-free shortest path routing without needing virtual channels [55]. We introduced a multicast routing algorithm, Yassin, which depends on e-cube routing and uses no virtual channels to deliver a message. Distributed-memory systems are the most advantageous architectures in building a massively parallel computer system. These systems need switching techniques to broadcast messages among processors. The wormhole switching technique has been widely used in the design of parallel computer systems. The basic idea of wormhole routing is that a message is partitioned into flow control flits. The multicast pattern, in

which one processor (node) sends the same message to multiple processors (nodes), is the most fundamental communication pattern used on multicomputer. Fault tolerance is a central issue facing the design and implementation of interconnection networks for distributed-memory systems. This chapter will focus on studying the fault-tolerant multicast wormhole routings in a 2D mesh networks with concave fault regions.

As mentioned in previous chapter, a novel fault tolerant multicast routing algorithm, FTDM, for wormhole routed 2D mesh multicomputer proposed. This routing algorithm is a unicast/tree based multicast routing algorithm. The proposed routing algorithm works effectively for the most common faults in 2D mesh networks, f-rings and f-chains. This algorithm is proved to be deadlock-free in chapter four. As presented in chapter III, Park et al. [80] proposed a fault tolerant wormhole routing algorithm in mesh networks in the presence of concave fault regions, called F4. They propose a fault tolerant wormhole routing algorithms that deal with more restricted shapes of fault rings in the mesh networks. In their fault models, there might be several f-rings in a 2D mesh networks. Also, they divide each f-ring (convex or concave) fault region into four portions: the north, south, west and east sides. In this chapter, F4 compared with proposed routing algorithm to evaluate the performance of both algorithms.

In this chapter, an efficient fault tolerant multicast routing algorithm, Yassin, for wormhole routed 2D mesh multicomputer is presented. Yassin routing algorithm is deadlock-free in spite of the concave fault regions in mesh networks. Yassin routing algorithm works with minimum routing restrictions and exploits the advantages of the three multicast routing style, unicast, path and tree based. Since it takes routing decision with minimum numbers of nodes (neighbor and central nodes), the presented routing algorithm is applicable in interconnection networks. Four essential performance metrics

in mesh networks, network traffic steps, network latency steps, network traffic time and network latency time are evaluated.

## Yassin Fault Model

Numerous applications of network topologies require high reliability and availability. A large parallel computer requires that its interconnect network operates without packet loss for many hours. Thus, these networks must employ an error control mechanism to continue operation without interruption, and possibly without packet loss, despite the failure of a component. The failure of a processing element and its associated routers is referred to as a node failure, and the failure of any communication channel is referred to as a link failure. In our fault model, both node failures and link failures are considered. The fault model is the base for the fault tolerant routing algorithms. Types of faults, structures of fault regions and processes to component failures determine the approaches to design deadlock-free routing algorithm.

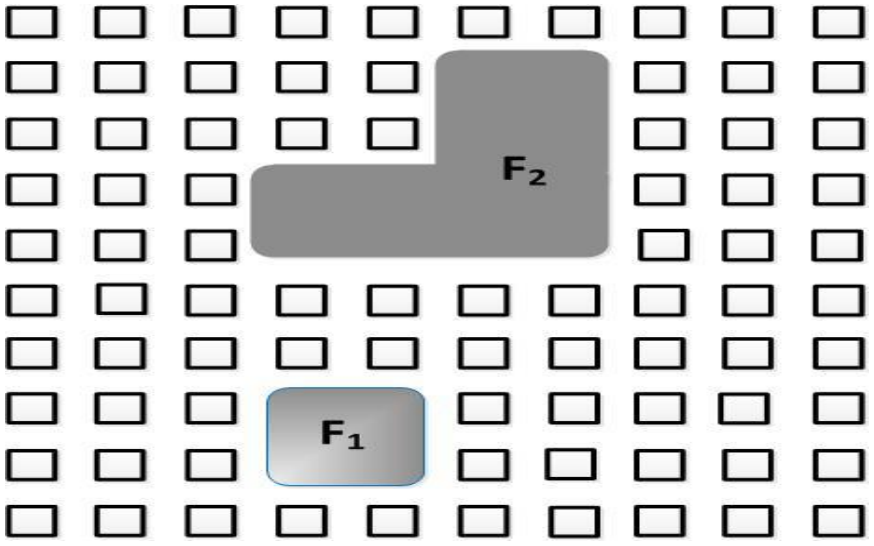



*Figure 49.* Fault model for Yassin.

Yassin algorithm considers convex (also known as block faults) and concave faults, which are the most commonly encountered faults in mesh networks. A convex

fault is a fault region such that there is a rectangle whose interior contains all and only the faulty components of the fault region and all processors and links on its four boundaries are fault-free, F1, as shown in Figure 49. Convex faults can be f-chain if fault region includes boundary nodes otherwise f-ring. A concave fault is a fault region such that there is ┗, ┛, ┏ and ┓ shapes whose interior contains all and only the faulty components of the fault region and all processors and links on its four boundaries are fault-free, F2, as shown in Figure 49.

In Yassin as in *i*FTDM, we consider fault information of a fault is distributed to a limited number of nodes (0, $y_{bi}$) in case of odd rows or (m, $y_{bi}$) in case of even rows) in order to avoid the fault before reaching it. Because fault information is distributed to a limited number of nodes, Yassin is a limited-global-information-based multicasting which is a compromise of local-information-based approach and global-information-based approach.

## Yassin Routing Algorithm

Fault tolerance is an important concern for the design of interconnection networks for large scale parallel processing computers. Most of fault tolerant multicast routing algorithms which were proposed recently concentrate on unicast-based multicast algorithms to deliver a message from source node to destination nodes. Unicast-based multicast algorithms require a startup time for sending a message to each destination node, and this needs more work. As in *i*FTDM fault tolerant multicast routing, Yassin algorithm assigns a label for each node on a 2D mesh based on the position of that node in a Hamiltonian path. In the Hamiltonian path, for each node there is a unique label, label 0 for the first node in the path and N-1 for the last node on the path of routing. The two algorithms create the routing decision at each sending node and both are a limited-

global-information-based multicasting algorithm which is a compromise of local-information-based approach and global-information-based approach. *i*FTDM can tolerate convex faults without using virtual channels but cannot work with concave fault regions. However, Yassin routing algorithm can tolerate convex and concave faults without using virtual channels.

The routing technique used by Yassin minimizes the network traffic time involved between the source and the destination nodes. At the source node, Yassin algorithm divides the network into two sub networks. The high channel sub network contains the destination nodes and channels whose direction is from lower labeled nodes to higher labeled nodes, and the low channel sub network contains the destination nodes and channels whose direction is from higher labeled nodes to lower labeled nodes.

The message transmission in Yassin is delivered according to the following method:

Yassin is a unicast/tree-based multicast algorithm, which attempts to deliver the message to all destinations in two phases. In the first phase the message is delivered as a unicast-based (using separate addressing routing technique) to X-coordinate nodes (nodes (0, ybi) in case of odd rows or (m-1, ybi) in case of even rows) of each true fault regions at these nodes, central nodes. Yassin considers each node of them as a source node that has a message with header containing destination nodes in the three locations around the fault. In the second phase, the message is delivered from the central nodes in a tree-based fashion (using R, R′and R′′), which attempts to route the message to all destination nodes in a single multi- head worm that splits at some routers and replicates the data on multiple output ports.

Upon delivery of the message, each receiving node on the network decides whether it is the first destination node. In this case, it is removed from the set of destinations and receives the message. At this point, if there is a destination node on the sets of the destinations, Yassin algorithm continues according to the previous method. Yassin algorithm uses five functions and two procedures. The first function is R to route a message in each fault locations around the fault region. The second function is R′ to route a message on a boundary of a fault region in case of convex fault. The third function is R″ to route a message on a boundary of a fault regions in case of concave fault. The fourth function is SEND(s, *msg*, d) takes as input the source, the message itself and the destination node. It makes s send the message mess to its neighboring d. The fifth function is RECEIVE(d , *msg*, s) takes as input the destination, the message itself, and the source node. It makes d receive the message mess from s. The first procedure, *phase1*, to route a message in the first phase (unicast based multicast phase). The second procedure, *phase2*, to route a message in the second phase (tree based multicast phase).

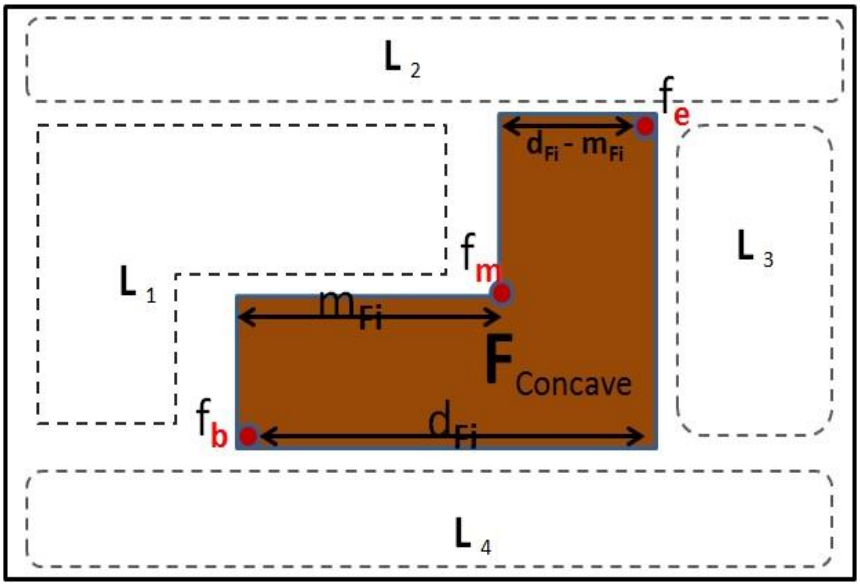



*Figure 50.* Locations around concave fault region.

*Routing Functions*

Yassin assigns a label for each node based on the position of that node in a Hamiltonian path [40].

The first routing function used in Yassin is defined as:

R (c, d) = w, where

$$Q(w) = \begin{cases} \max\{Q(z):\ Q(z) \le Q(u)\} & \text{if } Q(c) < Q(d) \\ \max\{Q(z):\ Q(z) \ge Q(u)\} & \text{if } Q(c) > Q(d) \end{cases}$$ and z is a neighboring node of c

It was proved in Lin and Ni [40] that for two arbitrary nodes *c* and *u* in a 2D mesh, the path selected by the routing function R is deadlock-free. The routing function R is used in each region does not contain any fault nodes.

The second routing function used in Yassin around convex fault region is defined as:

R′(c, d) = w, where

$$w = \begin{cases} (x_c, y_c-1) & \text{if } x_d = x_c \\ (x_c, y_c+1) & \text{if } x_d = x_c + d_x \times d_{Fi} \\ (x_c + d_x, y_c) & \text{otherwise} \end{cases}$$

It was proved in Shaheen and Abukmail [96], FTDM, this routing function is deadlock-free routing function.

The third routing function used in Yassin around concave fault region is defined as:

R′′(c, d) = w, where

$$W=\begin{cases} (x_c+ d_x, y_c) & \text{If} \quad x_d = x_c+ (d_x\times m_{Fi}), \\ & \text{Or} \quad x_d = x_c+ (d_x\times (d_{Fi} - m_{Fi})), \\ & \text{Or} \quad x_d = x_c+ (d_x\times m_{Fi}) \\ (x_c, y_c-1) & \text{If} \quad x_d= x_c \\ (x_c, y_c+1) & \text{otherwise} \end{cases}$$

This routing function is deadlock-free routing function because it works on five boundaries only for each concave fault region (which have six boundaries as in Figure 51), a message never visits any node around a concave fault region twice then, a cycle cannot exist within this path in the network and the path selected by the routing function R″ is the shortest path between the two nodes c and d.

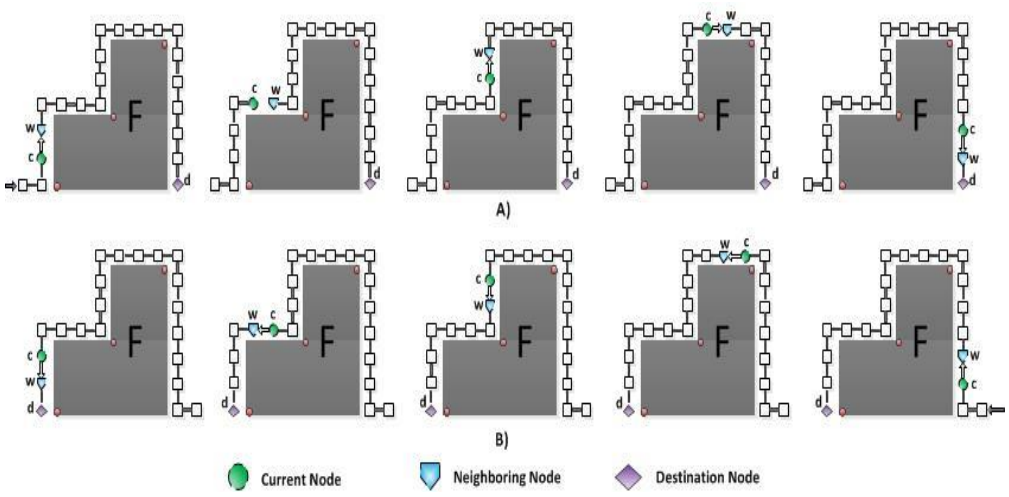



*Figure 51*. The routing path using R".

To define the path routing functions, which determines the next node for which the path of Yassin will be visited, some definitions are introduced:

1) Let $f_{bi}= (x_{bi}, y_{bi})$, $f_{mi}= (x_{mi}, y_{mi})$, and $f_{ei}= (x_{ei}, y_{ei})$ be the coordinates of each concave fault.

2) The fault region number *i*, $F_i$, is described by three nodes, $f_{bi}$, $f_{ei}$, and $f_{mi}$ where $f_{bi}$ is located in the southwest corner of the concave fault region, while $f_{ei}$ is located in the northeast corner of the concave fault region, and $f_{mi}$ is located in the angle of concave fault (at the middle of concave fault) as shown in Figure 50.

3) Width of a fault region $F_i$ is defined by: $d_{Fi}=| x_{ei} - x_{bi} |$ , $m_{Fi}=| x_{mi} - x_{bi} |$ as shown on Figure 50.

4) The variable $d_x$ is equal to 1 if the direction of the message path is from west to east or -1 if it is from east to west.

5) *LN* is the label of last node, $(x_{ei}, y_{bi})$, of a fault region which the message path visits. The value of LN is zero if the message path is in a non-fault region, while it is non zero if the message path is in a fault region.

6) Let $L_1$, $L_2$ and $L_3$ are three locations around each true fault regions as in Figure 50, and $L_4$ is a location in case if the 1st fault region is on a south boundary of 2D mesh.

7) *True fault regions* are the main fault regions which have three locations around them and may have other fault regions on locations, $L_3$ or $L_1$, with $f_{bi}= (x_{bi}, y_{bi})$, and $f_{ei}= (x_{ei}, y_{ei})$ less than it.

8) *Central nodes* are the nodes which the source node sends a copy of a message in the first phase in a unicast fashion and Y-coordinate of them is $y_{bi}$ for each true fault region; a source node is one of the central nodes if the first convex and concave fault region is not on a south boundary of 2D mesh (inside 2D mesh).

9) $D_i = \{(x, y) : (x, y) \in D \wedge x > x_{bni} \wedge y < y_{ei} \}$

10) $D = D_a + D_b$ if the 1st fault region is on a south boundary of 2D mesh, where $D_a$ is the destination nodes on $L_1$ and $L_2$, $D_b$ is the destination nodes on $L_3$ plus the destination

nodes around fault region, where is *a* equal *k* and *b* equal *k* (k is number of central nodes). Otherwise, $D = D_a + D_b + D_4$, where $D_4$ is the destination nodes on $L_4$

11) $Lf_i$ is the distance around fault regions (on the boundary of the fault) which equal $d_{Fi}$ plus $2*|(y_{ei}-y_{bi})|$

**Algorithm Yassin**
**Input**: The message *msg***,** Label node LN, Central nodes $CN_k$, k is the number of central nodes, Source node $s=(x_s, y_s)$, Destination set D, and Fault region $F_i$, the address of the current node of *msg* is $c=(x_c, y_c)$.
**Output**: $\forall\ d_j \in D$, RECEIVE ($d_j$, *msg*, s)
***BEGIN***
[1] IF $y_s$ is even THEN $d_x = 1$ ELSE $d_x = -1$
[2] IF $c = d_1$ THEN
    2. 1) $D = D-\{c\}$
    2. 2) RECEIVE ($d_,$ *msg*, *c*)
[3] IF $D = \phi$ THEN stop
[4] *Procedure Phase 1*
[5] *Procedure Phase 2*
[6] Repeat the above steps until each destination in the message header is reached.
***End of Yassin***

***Procedure Phase 1***
*Begin*
/* *Send copies of message to $CN_k$* */
[a] SEND (s, *msg*, $CN_k$) using separate addressing routing.
[b] Modify header of messages, *msg*, and put in each header $D_k$ destinations.
[c] Let each $CN_k$ as a new source node
[d] IF $CN_k = d$ THEN
    $D = D-\{CN_k\}$
    RECEIVE ($d_,$ *msg*, $CN_k$)
[e] IF $D = \phi$ THEN stop
*End of Procedure Phase 1*

***Procedure Phase 2***
*Begin*
/* *Send msg using R, R' and R'' functions* */
// At each central node, send two copies of message, $msg_1$ and $msg_2$
a) SEND(s, $msg_1$, $D_a$) using R
    IF $c = d_a$ THEN
    $D_a = D_a -\{c\}$
    RECEIVE ($d_a$, $msg_1$, *c*)
b) SEND(s, $msg_2$, $D_b$) using R' (convex) or R'' (Concave) to route a message around the fault region until the message reach to LN, and then use R.
    IF $c = d_b$ THEN

$D_b = D_b - \{c\}$
RECEIVE ($d_b$, $msg_2$, $c$)
c) IF $L_3$ have another faults then recursively apply Yassin.
*End of Procedure Phase 2*

*Pseudocode 3*. Yassin Routing Algorithm

Yassin uses the routing function R" in concave fault regions only. Figure 51 illustrates the different cases of the routing function R" and the way of its work around the fault region. The direction of the message path may be from west to east, Figure 51(A) or from east to west, Figure 51(B).

Yassin can be considered as a general form of FTDM (in case of convex fault regions only by applying R' function) and *i*FTDM (using the same main idea to overcome overlap problem of convex fault regions by use column path routing) that tolerate regular (convex and concave) fault region with overlap of convex faults.

***Lemma****:* Yassin algorithm is deadlock-free

*Proof:* Because Yassin use R, R′, R" and separate addressing routing to route a message on a 2D mesh with convex and concave faults and all of these are deadlock-free (no cyclic dependency can be created among the channels) as mentioned. Then Yassin is deadlock-free.

*Results and Discussions*

A simulation study has been conducted to evaluate Yassin performance and to compare it with F4 routing algorithms. The simulations were conducted on a 50×50 2D mesh and double channels were used. The two algorithms were written using C++ language and were implemented on a PC. In this section, we present the simulation results and analysis. In the simulation, wormhole routing is chosen as the switching technique and the routing algorithm is also applicable with other switching techniques.

This configuration creates different networks with a number of processors ranging from 100 to 1080. The average number of destinations is ranging from 10 to 100 and using three fault regions one convex (f-ring) and two concave, ⌟ .

In this subsection, four essential performance metrics in direct networks, network latency steps, network traffic steps, network traffic time and network latency time are calculated. The network latency step is the greatest number of channels which the message takes to reach its destinations. The network traffic step is the total number of channels used to deliver the message to all destinations. The network latency time is the longest message transmission time involved. The network traffic time is the overall time required to deliver the message to all destinations. They affect the overall performance of the distributed memory system and the granularity of parallelism that can be exploited from the system [30]. Network latency time depends on network latency steps, while network traffic time depends on network traffic steps. The startup time also affects the value of the network latency and network traffic times. The startup time is the time acquired by the system in preparing the message at the source node to deliver the message to the network and at the destination node to receive the message from the network. It depends on the design of system software within the nodes and the interface between nodes and routers.

In this subsection, the network latency steps, network traffic steps, network latency time and network traffic time are calculated for Yassin and F4 routing algorithms. The formula that can be used to calculate the four performance metric for Yassin routing technique (tolerate concave fault regions) can be derived from FTDM routing technique (tolerate convex fault regions), if we consider a concave fault region as multiple convex

fault regions (two convex) as shown in Figure 52. The following formulas can be used to calculate the performance metrics for Yassin and F4 algorithms.

Our partitioning of the 2D mesh around each fault regions into $L_{ih}$ and $L_4$, will result in partitioning the destinations D into $D_{ih}$ and $D_4$ respectively where i is ranging from 1 to F, h is ranging from 1 to 3 and F is number of fault regions. In addition, ($c_X$, $c_Y$) is the coordinate of central node.

- Distance ($d_{i+1}$, $d_i$) = $|x_{di+1} - x_{di}| + |y_{di+1} - y_{di}|$
- Latency (D) = $\sum_{i=1}^{|D|}$ Distance ($d_{i+1}$, $d_i$)

Which is dependent on the start coordinates and end coordinates for each location.

- $L_4$ =Latency ($D_{i4}$) + | ($S_X$-$x_{di}$) | + | ($S_Y$-$y_{di}$)|
- $L_{ih}$ =Latency ($D_{ih}$), where is h = 1, 2, 3
- $Traffic_{(i)} = L_{ih} + Lf_i + y_{bi} + x_{bi} + 2$
- $Left_{(i)} = L_{ih} + y_{bi} + 1$, where h = 1 and 2
- $Right_{(i)} = L_{ih} + Lf_i + y_{bi} + x_{bi} + 2$, where h = 3

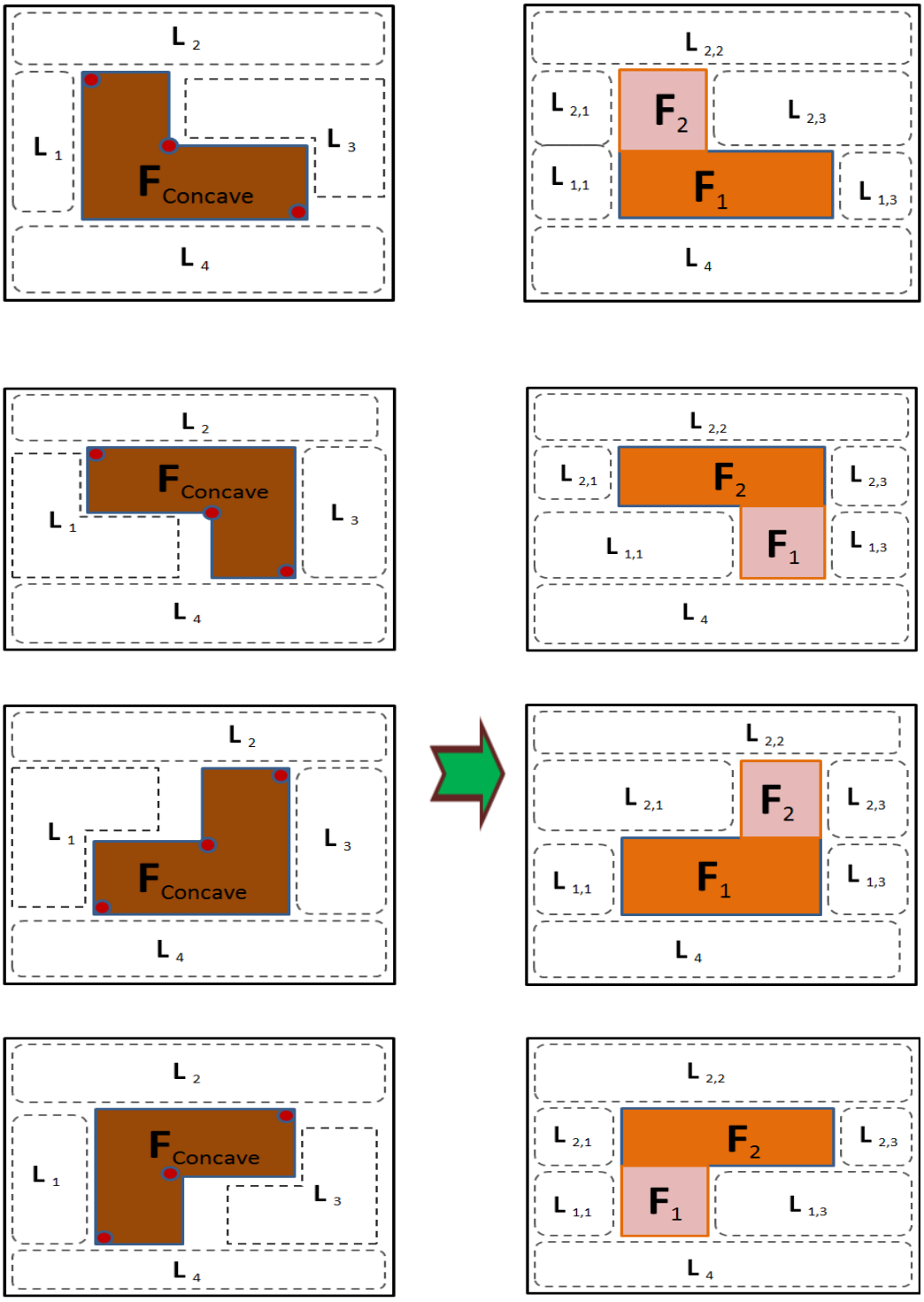



*Figure 52.* The Relationship between Yassin and FTDM Algorithms.

The network latency steps of Yassin is given by:

$$Yassin\_Latency=Max(Left_{(i)}, Right_{(i)}, L_4) \quad (1)$$

The network traffic steps of Yassin is given by:

$$Yassin\_Traffic = Traffic_{(i)} + L_4 \quad (2)$$

The network latency steps, F4_Latency, of F4 is given by:

$$Width_{(ij)}= \begin{cases} |y_{ej} - y_{bj}| & \text{if } x_{di} > x_{ej} \\ |y_{di} - y_{ej}| & \text{if } y_{bj} < y_{di} < y_{ej} \\ |x_{di} - y_{bj}| & \text{if } y_{di} < y_{ej} \ \& \ x_{di} < x_{ej} \\ 0 & \text{otherwise} \end{cases}$$

Where *width* is number of steps around fault region and dependence on destination position

$$F4\_Latency = Max\{ |x_{di} - S_x| + |y_{di} - S_y| + 2* Width_{(ij)}, 1 \le j \le T, 1 \le i \le |D|\} \quad (3)$$

Where T is number of fault regions participate between source node and destination node

The network traffic steps, F4_Traffic, of F4 is given by:

$$F4\_Traffic = \sum_{i=1}^{|D|} (|x_{di} - S_x| + |y_{di} - S_y| + 2* Width_{(ij)}) \quad (4)$$

The worst case of network latency time of Yassin algorithm can be calculated by:

$$Yassin\_Latency\_Time = t_{header} * D_{latency\_steps} + t_{copy} * F_{latency\_steps} + t_{channel} * Yassin\_Latency + t_{startup} * (cn+1) \quad (5)$$

The worst case of network latency time of Yassin algorithm can be calculated by:

$$Yassin\_Traffic\_Time = t_{header} * |D| + t_{copy} * |F| + t_{startup} * (cn+1) + t_{channel} * Yassin\_Traffic \quad (6)$$

The worst case of network latency time of F4 algorithm can be calculated by:

$$F4\_Latency\_Time = t_{startup} * |D| + t_{channel} * F4\_Latency \quad (7)$$

The worst case of network traffic time of F4 algorithm can be calculated by:

$$F4\_Traffic\_Time = t_{startup} * |D| + t_{channel} * F4\_Traffic \quad (8)$$

Where: $t_{channel,}$ channel time, is the time taken between two neighbor nodes. The channel time, $t_{channel}$, equals the sum of the router latency time, $t_r$, and the channel propagation time, $t_p$. The time, $t_{startup}$ is the startup time. The time, $t_{header}$, is the time taken to modify the message header at each destination. $D_{latency\_steps}$ is a set of destinations participating in the longest path. The time, $t_{copy}$, is the time taken to copy the message at each fault region participates in the longest path. $F_{latency\_steps}$, is a number of fault regions participate in the longest path.

*Comparative study.* As an example, to demonstrate the difference between Yassin algorithm and F4 algorithm, a 15×15 2D mesh is considered, Figure 53. The source node is (0, 0) and the destination nodes equals 11. We assume that the channel time, $t_{channel}$, equals 25 nanoseconds, the time, $t_{copy}$, equals 15 nanoseconds, the time, $t_{header}$, equals 5 nanoseconds, and the startup time, $t_{startup}$, equals 33 nanoseconds. The number of fault regions equals 2, $D_{latency\_steps}$, equals 2, $F_{latency\_steps}$, equals one, number of central nodes equals 3.

By using Yassin algorithm, Figure 53(a), network latency steps is 27 channels. Then, by applying equation (5), the latency time computed by Yassin equals 799 nanoseconds. Using F4 algorithm, Figure 53(b), network latency steps is 20 channels. Then, by applying equation (7), network latency time computed by F4 algorithm equals 863 nanoseconds. It is clear that, network latency steps of Yassin algorithm is larger than that of F4 algorithm while network latency time of Yassin algorithm is less than that of F4 algorithm. Since F4 algorithm is a unicast-based technique, it produces low network latency steps and high network latency time.

By using Yassin algorithm, Figure 53(a), network traffic steps is 70. Then, by applying equation (6), network traffic time computed by Yassin equals 1919

nanoseconds. Using F4 algorithm, Figure 53(b), network traffic steps is 123. Then, by applying equation (8), network traffic time computed by F4 algorithm equals 3438 nanoseconds. It is clear that, the network traffic steps and network traffic time of Yassin algorithm are very less than those of F4 algorithm.

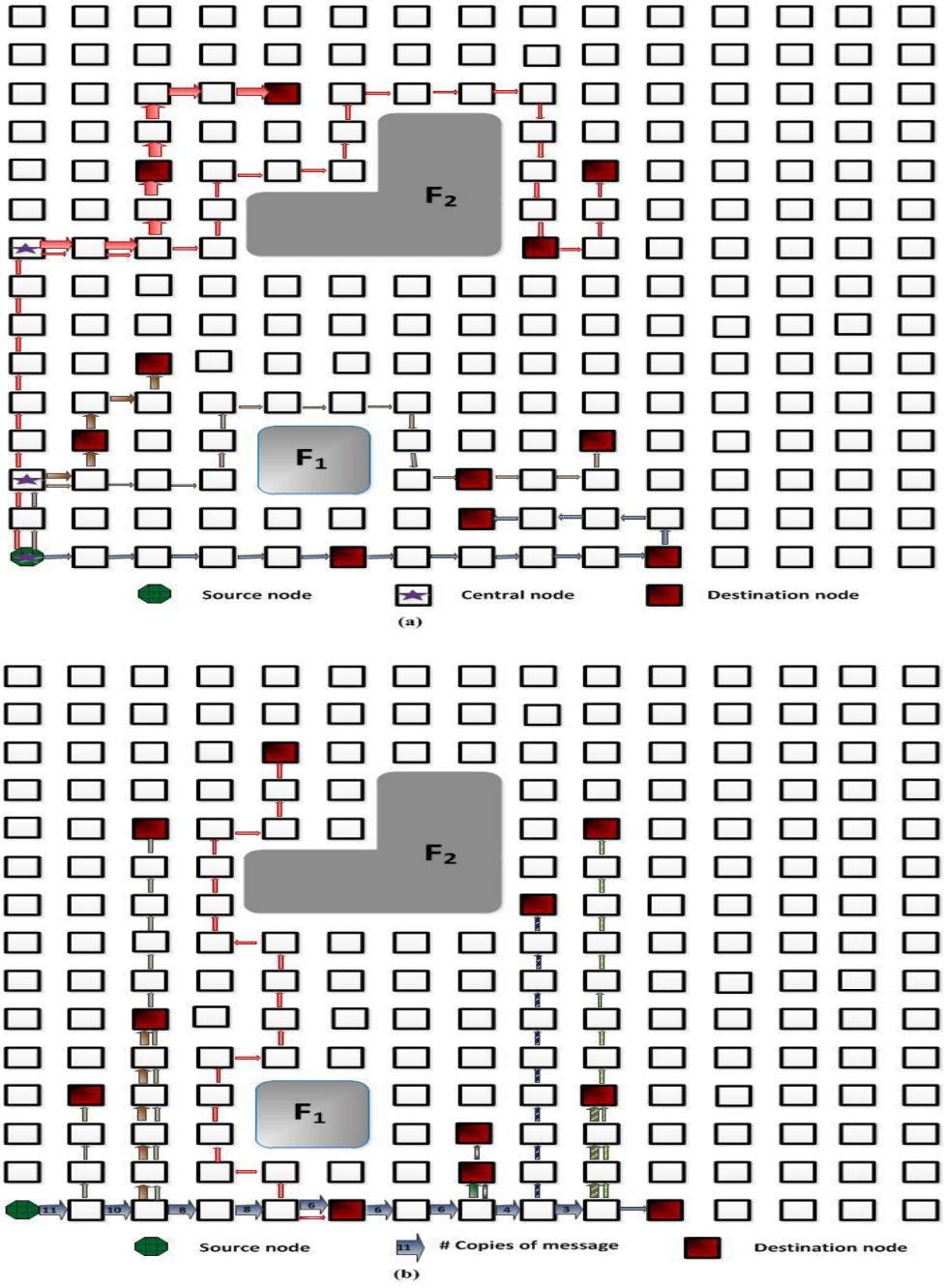



*Figure 53.* Network traffic steps computed by (a) Yassin algorithm, (b) F4 algorithm.

*Results.* The equations from 1 to 8 are used to calculate network latency steps, network traffic steps, network latency time and network traffic time for both algorithms in 2D mesh. Figures from 54 to 57 show the results of the two algorithms. The continuous line represents results of Yassin, while the dotted line represents results of F4.

Figure 54 plots network latency steps for various values of the average number of destination nodes, ranging from 10 to 100. The figure shows that, network latency steps computed by Yassin increases as number of destination nodes increases. The increase is not affected by type of the fault region (convex and concave). The latency steps computed by F4 is nearly constant as number of destination nodes increases. This is because Yassin is a unicast/tree-based multicast routing algorithm while F4 is unicast-based multicast routing algorithm.

Figure 55 plots network traffic steps for various values of average number of destination nodes, ranging from 10 to 100. The figure shows that network traffic steps computed by Yassin are nearly constant (slight increase) as number of destinations increases. Network traffic steps computed by F4 are increase as the number of destination nodes increases. The increasing rate of network traffic steps computed by F4 is large because each destination node needs a separate message path.

Figure 56 plots network latency time for various values of the average number of destination nodes, ranging from 10 to 100. The figure shows that the network latency time computed by the two algorithms increases as number of destination nodes increases. Clearly, at a small average number of destination nodes, Yassin algorithm outperforms F4 algorithm, while at a large average number of destination nodes, F4 algorithm outperforms Yassin algorithm. This is because the network latency steps are the dominant

factor of network latency time computed by Yassin algorithm, while the startup time is the dominant factor of network latency time computed by F4 algorithm.

Figure 57 plots network traffic time for the various values of average number of destination nodes, ranging from 10 to 100. The figure shows that network traffic time computed by Yassin algorithm is nearly constant as the number of destination nodes increases, while the traffic time computed by F4 algorithm increases. This is because the traffic time values depend on network traffic steps values.

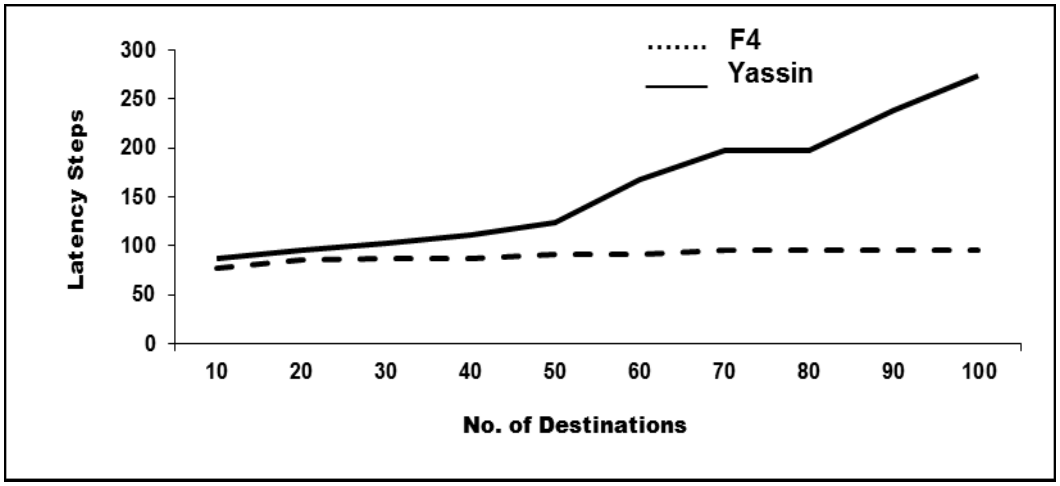



*Figure 54.* Network Latency Steps Vs. No. of Destinations.

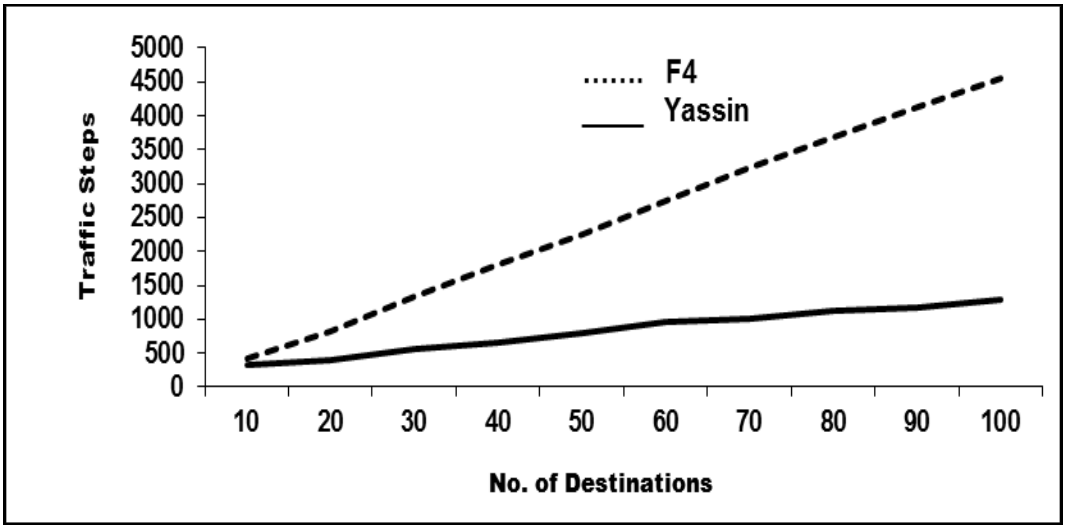



*Figure 55.* Network Traffic Steps Vs. No. of Destinations.

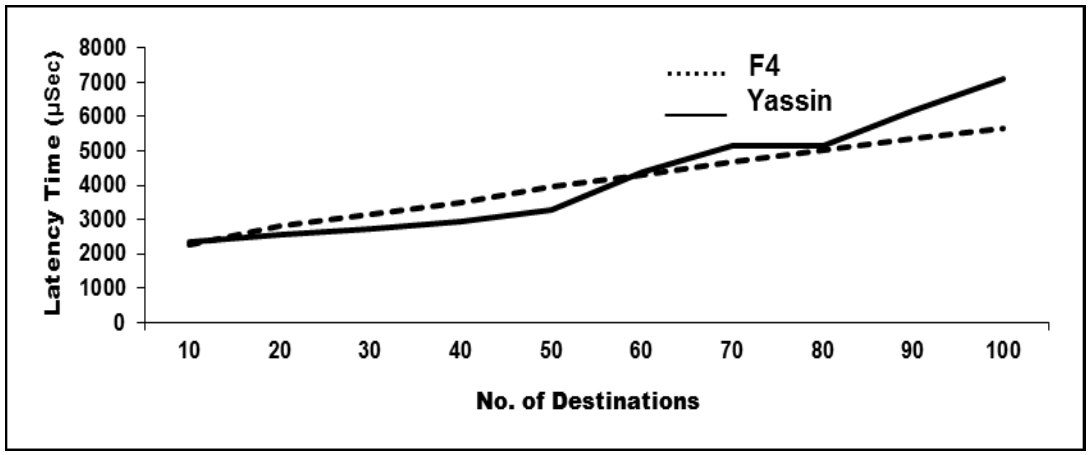



*Figure 56.* Network Latency Time Vs. No. of Destinations.

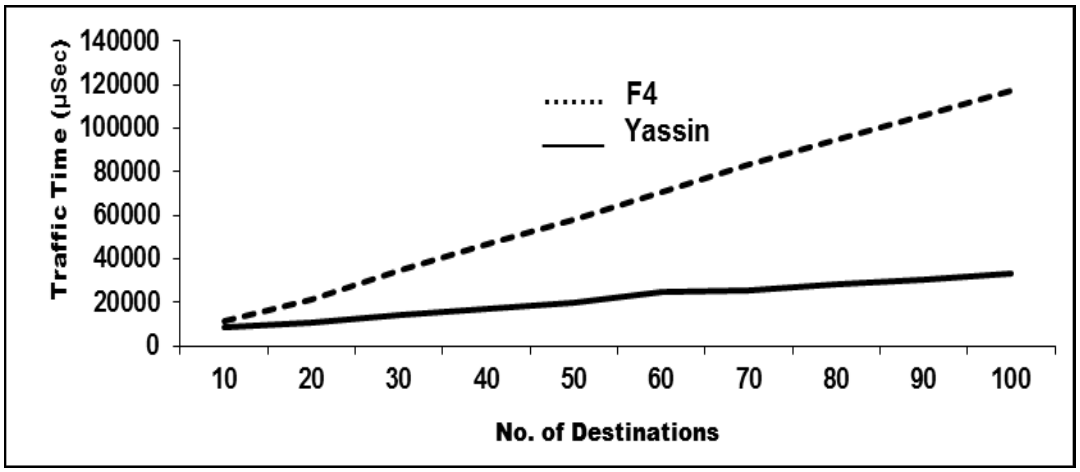



*Figure 57.* Network Traffic Time Vs. No. of Destinations.

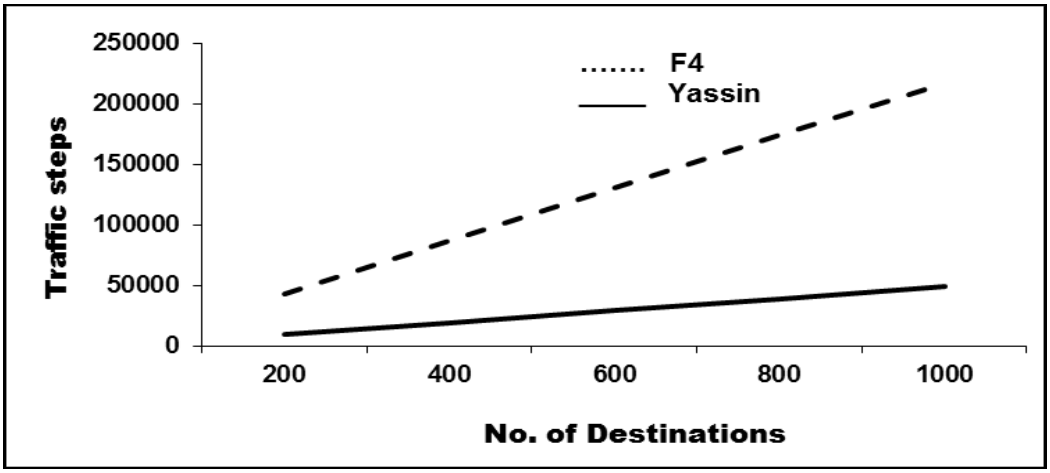



*Figure 58.* Validate comparison between Yassin and F4.

Figure 58 plots network traffic steps for various values of the average number of destination nodes, ranging from 200 to 1000 using data mining regression tools. The figure shows that the traffic steps computed by Yassin is nearly constant (slight increase) as number of destination nodes increases (the same behavior as in figure 55). The increasing rate of network traffic steps computed by F4 is larger than Yassin (the same behavior as in Figure 55). When compared, the results shown in both Figure 58 and Figure 55 are the same, and then this validates our results and enlarges the sample size of number of destinations.

Generally, from the previous figures, the following notes can be observed:

- As number of destinations network traffic steps and network traffic time computed by Yassin algorithm is nearly constant, that computed by F4 algorithm increases and Yassin algorithm is very effective than F4 algorithm.
- As number of destinations increases, network latency steps and network latency time computed by Yassin algorithm increases, while that computed by F4 algorithm is nearly constant. In most tested cases, network latency steps and network latency time computed by Yassin algorithm is larger than that computed by F4 algorithm.

As explained in Chapter IV, network traffic steps and network traffic time are more significant criteria of measuring the efficiency of fault tolerant multicast routing algorithms. Hence, Yassin is more efficient than F4.

# CHAPTER VII

# SUMMARY AND FUTURE WORK

This chapter summarizes the research of this dissertation and introduces some suggestions for future research.

## Summary of This Dissertation

This dissertation presented new fault tolerant multicast routing techniques for distributed-memory systems performance.

Parallel computer systems, which emphasize *parallel processing*, are the most favorable architectures to increase the computing power. A parallel computer system consists of several powerful processors connected together into a single system. These connected processors cooperate to solve a single problem that exceeds the ability of any one of those processors. Parallel computer systems are constructed by connecting a number of powerful computer processors together into a single system, which cooperates to solve grand-challenge problems. They provide cost-effective mechanisms to achieve high system performance through concurrent activities.

In Chapter II, multiprocessors systems, distributed-memory, shared-memory, and distributed-shared-memory, which are currently the most promising parallel systems, were considered. Popular network topologies, such as hypercube, tori, and mesh networks for distributed-memory systems were discussed. In distributed-memory systems, a switching technique is used to transmit a message between two nodes. The most common switching techniques for direct networks, including circuit, store-and-forward, virtual cut-through switching, and wormhole switching, were considered. Multicast routing techniques can be classified as unicast-based, path-based and tree-based

algorithms. Multicast routing algorithms for direct networks were surveyed. In this chapter, we have demonstrated that:

- Distributed-memory systems have become the popular architectures for massively parallel computers.
- Lower-dimension mesh networks are the suited topologies for current distributed-memory systems.
- The wormhole routing has emerged as the most widely used in distributed-memory systems

In Chapter III, an overview of fault tolerant multicast routing algorithms was presented. In distributed-memory systems, packets usually travel across several intermediate nodes before reaching the destination node. Deadlock occurs when some packets cannot advance toward their destination because the buffers requested by them are full. Also, some components such as processors, routers, and communication channels may fail. Fault tolerance refers to the ability of the system to operate correctly in the presence of faults. According to number of parameters, faults are classified into different types. Fault tolerance is an important issue facing the design of distributed-memory systems. A brief introduction to deadlock, Fault model and fault tolerance was given. Fault tolerant multicast routing algorithms for regular and irregular fault regions were studied. In the same Chapter, we have demonstrated that designing fault tolerant and deadlock-free multicast routing algorithms are the important problems in distributed-memory systems.

In Chapter IV, FTDM and *i*FTDM fault tolerant multicast routing algorithms were proposed. Fault tolerant multicast routing algorithm that uses less network traffic steps, network traffic time, network latency steps and network latency time is one of the most

important issues that deals with the implementation of interconnection networks for large-scale parallel computer systems. The proposed fault tolerant routing algorithm, FTDM, can tolerate convex faults without using virtual channels. Also, FTDM tolerates f-chains in mesh networks and does not lead to deadlock with any number of non-overlapping f-regions. The proposed routing algorithm, FTDM, can tolerate convex faults with presence of a large number of destination nodes. In addition, an efficient fault tolerant multicast routing algorithm for 2D mesh, *i*FTDM, which is an improvement of FTDM algorithm, is presented. The proposed routing algorithm, *i*FTDM, can tolerate convex faults with the presence of a large number of fault regions and large fault region size. The routing algorithm *i*FTDM tolerates f-chains in meshes with the overlapping of convex fault regions. This algorithm is a dead lock free. Because fault information is distributed to a limited number of nodes, both FTDM and *i*FTDM are limited-global-information-based multicasting algorithms, which is a compromise of local-information-based approach and global-information-based approach. The simulation results show that FTDM and *i*FTDM routing algorithms has better performance than FT-cube2 in most significant criteria to measure the efficiency.

In Chapter V, data mining for FTDM and *i*FTDM routing algorithms were presented to validate the results. Data mining has enormous applications in different areas like agriculture, marketing, biology, computational research etc. Regression analysis is one of the most important tools in data mining. There are many tools available to do data mining analysis such as WEKA. Results are obtained using the tools WEKA, EXCEL and MATLAB. Using data mining concept, network traffic steps are calculated for two different algorithms by changing the number of destinations.

In Chapter VI, an efficient fault tolerant multicast routing algorithm, Yassin, for wormhole routed 2D mesh multicomputer is presented. The routing algorithm is deadlock-free in spite of the concave fault regions in mesh networks. The presented routing algorithm works with minimum routing restrictions and exploits the advantages of the three multicast routing style, unicast, path and tree based. Since it takes a routing decision with a minimum numbers of nodes (neighbor and central nodes), the presented routing algorithm is applicable in interconnection networks. Four essential performance metrics in mesh networks, network traffic steps, network latency steps, network traffic time and network latency time are evaluated.

## Future Work

Several issues related to the study of fault tolerant multicast routing techniques for distributed-memory systems have been covered in this dissertation. Conversely, they are worthy of further study. They include

- To extend this fault tolerant multicast routing techniques to other high level mesh networks (generalizing the use of this routing to n-D mesh networks).
- To extend this fault tolerant multicast routing techniques to other network topologies, such as tours (as it is a mesh with wraparound) and hypercube.
- Applying these fault tolerant routing algorithms, (FTDM, *i*FTDM and Yassin), for irregular fault regions with complicated ship (the proposed algorithms was applied for regular fault regions – convex and concave – it can be enhanced to achieve better effective results).
- Using another path routing functions to determine the next node to which the message will be forwarded, (the proposed fault tolerant multicast routing algorithms was used in three functions R, R′and R″).

- To extend this fault tolerant unicast/tree based multicast routing techniques to other types of combination (unicast, path and tree based multicast routing).
- Using another switching technique (such as store and forward, virtual cut-through and circuit switching), the proposed fault tolerant routing algorithms used wormhole switching.

# REFERENCES

[1] E. Alotaibi and B. Mukherjee,"A survey on routing algorithms for wireless ad-Hoc and mesh networks," *Computer Networks*, vol. 56, no.2, 2012, pp. 940–965.

[2] K. Hwang and D. DeGroot, *Parallel Processing for Super-Computers and Artificial Intelligence*, McGraw Hill, 1989.

[3] A. Trew and G. Wilson, *Past, Present, Parallel: A Survey of Available Parallel Computer Systems*, Springer-Verlag, 1991.

[4] G. Hager and G. Wellein, *Introduction to High Performance Computing for Scientists and Engineers*, Chapman & Hall/CRC Press, 2010.

[5] R. Esser and R. Knecht, *Intel Paragon XP/S - Architecture and Software Enviroment*, Anwendungen, Architekturen, Trends, Seminar, June 24-26, 1993, pp. 121-141.

[6] Thinking Machine Corporation, CM5 Technical Summary, October 1991.

[7] Meiko Limited, Meiko CS-2 System Overview, 1994.

[8] K. Hwang, *Advanced Computer Architecture: Parallelism, Scalability, Programmability*, McGraw-Hill, 1993.

[9] C.L. Seitz, "Mosaic C: An Experimental Fine-Grain Multicomputer," *Tech. Rep., California Institute of Technology*, Pasadena, CA 91125, 1992.

[10] W.J. Dally, et al., "The Message-Driven Processor: A Multicomputer Processing Node with Efficient Mechanisms," *IEEE Micro*, vol. 12, no. 2, Apr. 1992, pp. 23-39.

[11] H. Stephen Morse, *Practical Parallel Computing*, Academic Press Limited, 1994.

[12] W.J. Dally, *Network and Processor Architecture for Message-Driven Computers*, Morgan Kaufmann, VLSI and Parallel Computation, 1990.

[13] F. Gebali, *Algorithms and Parallel Computers*, New York: John Wiley, 2011.

[14] J. Laudon and D. Lenoski, "The SGI Origin: A ccNUMA Highly Scalable Server," *Proc. of the 24th IEEE/ACM Ann. Intl. Symp. on Computer Architecture (ISCA-24)*, June 1997, pp. 241-251.

[15] D. Lenoski, et al., "The DASH Prototype: Logic Overhead and Performance," *IEEE Trans. On Parallel and Distr. Systems*, vol. 4, no. 1, Jan. 1993, pp. 41-61.

[16] M. Heinrich, et al., "The Performance Impact of Flexibility in the Stanford Flash Multiprocessor," *6th Intl. Conf. On Architecture Support for Programming Language and Operating Systems (ASPLOS-VI)*, San Jose, CA, Oct. 1994, pp. 274-285.

[17] J. Duato, S. Yalamanchili, and L. M. Ni, *Interconnection Networks: An Engineering Approach,* Morgan Kaufmann, 2002.

[18] C.L. Seitz. "The Cosmic Cube." *Communication ACM*, vol. 28, 1985, pp. 22-33.

[19] S.A. Felperin, et al., "Routing Techniques for Massively Parallel Communication," *Proc. of the IEEE*, vol. 79, 1991, pp. 488-503.

[20] P. Kermani and L. Kleinrock, "Virtual Cut-Through: A new Computer Communication Switching Technique," *Commut. Network* vol. 3, no. 4, 1979, pp. 267-286.

[21] S. Konstantinidou and L. Snyder, "Chaos Router: Architecture and Performance," *Proc. of the 18th International Symposium on Computer Architecture*, June 1991, pp. 212-221.

[22] S. Nugent, "The iPSC/2 Direct Connect Communications Technology," *Proc. of the Third Conference on Hypercube, Concurrent Computers and Applications*, Jan. 1988, pp. 51-59.

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

[24] C.L. Seitz, et al., "The Architecture and Programming of the Ametek Series 2010 Multicomputer," *Proc. Conf. Hypercube Computers and Concurrent Applications*, Pasadena, CA, Jan. 1988, pp. 33-36.

[25] W.J. Dally, "The J-machine: System Support for Actors," *Actors: Knowledge-Based Concurrent Computing* (Hewitt and Agha, eds.), MIT Press, 1989.

[26] D.F. Robinson, P.K. McKinley, and B.C. Cheng, "Path-Based Multicast Communication in Wormhole-Routed Unidirectional Torus Networks," *J. of Parallel and Distributed Computing*, vol. 45, Sep. 1997, pp. 104-121.

[27] S. Chittor and R. Enbody, "Performance Evaluation of Mesh-Connected Wormhole-Routed Networks for Interprocessor Communication in Multicomputers," *Proc. of Supercomputing'90*, Nov. 1990, pp. 647-656.

[28] S. Chen, et al., *Reconfigurable Networks-on-Chip*, Springer, 2012.

[29] L.M. Ni and P.K. McKinley, "A Survey of Wormhole Routing Techniques in Direct Networks," *IEEE Computing.*, vol. 26, Feb. 1993, pp. 62-76.

[30] M. E. Shaheen, "High Distributed-Memory Systems Performance," master thesis, Computer Scince Dept., Cairo University (Fayoum Branch), 2005.

[31] D.F. Robinson, P.K. McKinley, and B.C. Cheng, "Optimal Multicast Communication in Wormhole-Routed Torus Networks," *IEEE Trans. on Parallel and Distributed Systems*, vol. 6, no. 10, Oct. 1995.

[32] P. Mckinley, et al., "Unicast-Based Multicast Communication in Wormhole-Routed Networks," *IEEE Trans. on Parallel and Distributed Systems*, vol. 5, no. 12, Dec. 1994, pp. 1252-1265.

[33] R.V. Boppana, S. Chalasani, and C.S. Raghvendra, "Resource Deadlocks and Performance of Multicast Routing Algorithms," *IEEE Trans. on Parallel and Distributed Systems*, vol. 9, no. 6, June 1998.

[34] A. Agarwal et al., "The MIT Alewife Machine: Architecture and Performance," *In ISCA '98: 25 years of the International Symposia on Computer Architecture*, New York, NY, USA, 1998, pp. 509-520.

[35] Oliveira C and Pardalos P. *Mathematical Aspects of Network Routing Optimization*, Springer Verlag, vol 53, 2011.

[36] Baijian Yang and Prasant Mohapatra. "DiffServ- Aware Multicasting." *Journal of High Speed Networks*, vol 13, 2004, pp. 37–57.

[37] C.T. Ho and M.-Y. Kao, "Optimal Broadcast in All-Port Wormhole-Routed Hypercubes," *IEEE Trans. Parallel Distributed Systems*, vol. 6, 1995, pp. 200 – 204.

[38] E. Rosenberg, *A Primer of Multicast Routing*, Springer, 2012.

[39] Y.-C. Tseng, S.-Y. Wang, and C-W. Ho, "Efficient Broadcasting in Wormhole-Routed Multicomputers: A Network-Partitioning Approach," *IEEE Trans. on Parallel and Distributed Systems*, vol. 10, no. 1, Jun. 1999.

[40] X. Lin and L.M. Ni, "Deadlock-Free Multicast Wormhole routing in Multicomputer Networks," *Proc. of the Intl. Symp. On Computer Architecture*, 1991, pp. 116-124.

[41] J. Duato, "On the Design of Deadlock-Free Adaptive Multicast Routing," *Parallel Processing Letters*, vol. 3, 1993, pp. 321-334.

[42] X. Lin, P. K. McKinley, and L. M. Ni, "Deadlock-Free Multicast Wormhole Routing in 2-D Mesh Multicomputers," *IEEE Trans. on Parallel and Distributed Systems,* vol. 5, no. 8, Aug. 1994.

[43] R.V. Boppana, S. Chalasani, and C.S. Raghvendra, "On Multicast Wormhole Routing in Multicomputer Networks," *Proc. 6th IEEE Symp. On Parallel and Distributed Processing*, Oct. 1994, pp. 722-729.

[44] M. A. Abd El-Baky. "New Routing Techniques for High Message-Passing Systems Performance." Doctoral dissertation, Computer Science Dept. Cairo University ( Fayoum Branch), 2000.

[45] M.P. Malumbres, J. Duato, and J. Torrellas, "An Efficient Implementation of Tree-Based Multicast Routing for Distributed Shared-Memory Multiprocessors," *Proc. Of the 8th IEEE Symp. On Parallel and Distributed Processing*, Oct. 1996, pp. 186-189.

[46] H. Wang and D. Blough, "Tree-Based Fault-Tolerant Multicast in Multicomputer Networks Using Pipelined Circuit Switching," *Tech. Rep. ECE-97-05-01, Dept. of Electrical and Computer Engineering, Univ. of California at Irvine*, May 1997.

[47] J. Wu and L. Sheng, "Deadlock-Free Multicasting in Irregular Networks Using Prefix Routing," *Journal of Supercomputing*, vol. 31, no. 1, 2005, pp. 63-78.

[48] Nen-Chung Wang and Chih-Ping Chu, "An Efficient Tree-Based Multicasting Algorithm on Wormhole-Routed Star Graph Interconnection Networks Embedded with Hamiltonian Path," *Journal of Supercomputing*, vol. 34, no. 1, 2005, pp. 5-26.

[49] D.K. Panda and R. Sivaram, "Fast Broadcast and Multicast in Wormhole Multistage Networks with Multidestination Worms," *Tech. Rep. OSU-CISRC-4/95-TR21, Dept. of Computer and Information Science, The Ohio State Univ*., Apr. 1995.

[50] H. Moharam, M.A. Abd El-Baky, and S.M.M. Nassar, " YOMNA: An Efficient Deadlock–Free Multicast Wormhole Algorithm in 2-D Mesh Multicomputers," *J. of Systems Architecture*, vol. 46, no. 12, pp. 1073-1091 Oct. 2000.

[51] J. Wu and X. Chen, "A Fault-tolerant Tree-Based Multicasting in Mesh Multicomputers," *Department of Computer Science and Engineering, Florida Atlantic University, Boca Raton*, FL 33431, Apr. 1999.

[52] P. T. Gaughan and S. Yalamanchili, "A family of Fault-Tolerant Routing Protocols for Direct Multiprocessor Networks," *IEEE Transactions on Parallel and Distributed Systems*, vol. 6, no. 5, May 1995, pp. 482-495.

[53] R.E. Kessler and J.L. Schwarzmeier, "CRAY T3D: A New Dimension for Cray Research," *Compcon*, Spring 1993, pp. 176-182.

[54] M. Singhal, "Deadlock Detection in Distributed Systems," *IEEE Computer*, vol. 22, no. 11, 1989, pp.37-48.

[55] R. V. Boppana and S. Chalasani, "Fault-Tolerant Wormhole Routing Algorithms for Mesh Networks," *IEEE Transactions on Computers*, vol. 44, no. 7, July 1995, pp. 848-864.

[56] S. Gunasekran and K. Duraiswamy," Robust Multicast Communication in Heterogeneous Network with Fast Recovery Scheme," *J. of Mathematics & Technology*, no. 2, 2010, pp. 48-53.

[57] S. Chalasani and R.V. Boppana,"Communication in Multicomputers with Nonconvex Faults," *IEEE Trans. Comput*. vol. 46, no. 5, May 1997, pp. 616-622.

[58] Nitin and Durg Chauhan, "Comparative analysis of Traffic Patterns on k-ary n-tree using adaptive algorithms based on Burton Normal Form," *J. of Supercomputing*, vol. 59, no. 2, 2012, pp. 569-588.

[59] P.-H. Sui and S.-D. Wang, "Fault-Tolerant Wormhole Routing in Two-Dimensional Mesh Networks with Convex Faults," *Information Sciences*, vol. 121, 1999, pp. 217-231.

[60] H.-H. Chang and G.-M. Chiu, "An Improved Fault-Tolerant Routing Algorithm in Meshes with Convex Faults," *Parallel Computing*, vol. 28, 2002, pp. 133-149.

[61] C.J. Glass and L.M. Ni, "The Turn Model for Adaptive Routing," *J. of the ACM*, vol. 41, Sept. 1994, pp. 874-902.

[62] C.J. Glass and L.M. Ni, "Fault-Tolerant Wormhole Routing in Meshes Without Virtual Channels," *IEEE Trans. Parallel Distrib. Syst*. vol. 7, no. 6, 1996, pp. 620-636.

[63] C. J. Glass and L. M. Ni, "Fault-Tolerant Wormhole Routing in Meshes," *Proc. of the 23rd International Symposium on Fault-Tolerant Computing*, June 1993, pp. 240-249.

[64] J. Wu, "Fault-Tolerant Adaptive and Minimal Routing in Mesh-Connected Multicomputers using Extended Safety Levels," *IEEE Trans. Parallel Distrib. Syst.* vol. 11, no. 2, 2000, pp. 149-159.

[65] J. Wu, "A Fault-Tolerant and Deadlock-Free Routing in 2D Meshes Based on Odd-Even Turn Model," *IEEE Trans. Comput*. vol. 52, no. 9, 2003, pp. 1154-1169.

[66] A. Rezazadeh, M. Fathy and A. Hassanzadeh, "A Performance-Enhancing Fault-Tolerant Routing Algorithm for Network-on-Chip in Uniform Traffic," *Asia International Conference on Modeling and Simulation*, 2009, pp. 614-619.

[67] A. Rezazadeh, M. Fathy and Gh. Rahnavard, "An Enhanced Fault-Tolerant Routing Algorithm for Mesh Network-on-Chip," *Int. Conf. on Embedded Software and Systems (ICESS 2009)*, 2009, pp. 505-510.

[68] Mohtashamzadeh, M., Momeni, L. and Rezazadeh, A. "An Innovative Fault-Tolerant Method for 2-D Mesh-Based Network-on-Chip Routing," *EMS*, 2011, pp. 339-343.

[69] Xie, L., et al., "The Two-Level-Turn-Model Fault-tolerant Routing Scheme in Tori with Convex Faults," *Proc. of the 2008 international Conference on Computer Science and information Technology (August 29 - September 02, 2008),* 2008, pp. 379-387.

[70] Xie, L., et al., "A New Fault-Tolerant Wormhole Routing Scheme in Tori with Convex Faults," *Proc. of the 2008 11th IEEE High Assurance Systems Engineering Symposium (December 03 - 05, 2008),* 2008, pp. 467-470.

[71] F. Safaei, et al., "Evaluating the Performance of Adaptive Fault-Tolerant Routing Algorithms for Wormhole-Switched Mesh Interconnect Networks," *IPDPS,* 2007, pp.1-8.

[72] J. Wu and X. Chen, "Fault-Tolerant Tree-Based Multicasting in Mesh Multicomputers," *Journal of Computer Science and Technology,* vol. 16, no. 5, 2001, pp. 393-408.

[73] H. Gu, et al., "Enhanced fault tolerant routing algorithms using a concept of 'balanced ring'," *J. of Systems Architecture,* vol. 53, pp. 902–912, 2007.

[74] J. Zhou and F.C.M. Lau, "Fault-Tolerant Wormhole Routing in 2D Meshes," *Proceedings of the Fifth International Symposium on Parallel Architectures, Algorithms, and Networks*, 2000, pp. 94-101.

[75] J. Zhou and F.C.M. Lau, "Adaptive Fault-Tolerant Routing with Two Virtual Channels in 2D Meshes," *Proc. Seventh Int'l Symp. Parallel Architectures, Algorithms and Networks,* 2004, pp. 142-148.

[76] J. Zhou and F.C.M. Lau, "Multi-phase minimal fault-tolerant wormhole routing in meshes," *J. parallel computing*, vol. 30, 2004, pp. 423-442.

[77] J. Zhou, "Fault-tolerant wormhole routing with 2 virtual channels in meshes," *Journal of Computer Science and Technology*, vol. 20, 2005, pp. 822-830.

[78] S. Chalasani and R. V. Boppana, "Fault-Tolerant Wormhole Routing in Tori," *Proc. of the 8$^{th}$ International Conference on Supercomputing*, July 1994.

[79] L. Xie, and D. Xu," The Two-Level-Turn-Model Fault-Tolerant Routing Scheme in Tori with Convex and Concave Faults," *Proc. of the 2009 Sixth international Conference on information Technology: New Generations (April 27 - 29, 2009),* 2009, pp. 107-113.

[80] S. Park, J.-H. Youn and B. Bose, "Fault-Tolerant Wormhole Routing Algorithms in Meshes in the Presence of Concave Faults," *IPDPS, 14th International Parallel and Distributed Processing Symposium*, 2000, pp.633-638.

[81] Y. Fukushima, et al., "A Hardware-Oriented Fault-Tolerant Routing Algorithm for Irregular 2D-Mesh Network-on-Chip without Virtual Channels," *Proc. of International Symposium on Defect and Fault Tolerance in VLSI Systems (DFT'10),* 6-8 Oct., Kyoto, Japan, 2010, pp. 52-59.

[82] A. Mejia, et al., "Segment-Based Routing: An Efficient Fault-Tolerant Routing Algorithm for Meshes and Tori," *IEEE International Parallel & Distributed Processing Symposium (IPDPS),* April 2006.

[83] J. Wu and D. Wang, "Fault-tolerant and deadlock-free routing in 2-D meshes using rectilinear-monotone polygonal fault blocks," *The Int'l J. of Parallel, Emergent and Distributed Systems,* vol. 20, no. 2, 2005, pp. 99-111.

[84] M. Stojmenovic and A. Nayak, "Broadcasting and routing in faulty mesh networks," *IPDPS,* 2006.

[85] J-D Shih, "Fault-tolerant routing in hypercube networks without virtual channels," *IEE Proc. Comput Digit Tech,* vol. 5, 2006, pp. 377-384.

[86] D. Xiang, et al., "Fault-Tolerant Routing in Meshes/Tori Using Planarly Constructed Fault Blocks," *Proc. 34th Int'l Conf. Parallel Processing (ICPP '05),* 2005, pp. 577-584.

[87] D. Xiang, Y. L. Zhang, and Y. Pan, "Practical Deadlock-Free Fault-Tolerant Routing in Meshes Based on the Planar Network Fault Model," *IEEE Transactions on Computers,* vol. 58, no. 5, pp. 620-633, 2009.

[88] Dong Xiang, "Deadlock-Free Adaptive Routing in Meshes with Fault-Tolerance Ability Based on Channel Overlapping," *IEEE Transactions on Dependable and Secure Computing,* vol. 8, no. 1, pp. 74-88, 2011.

[89] F. Safaei, et al., "Performance analysis of fault-tolerant routing algorithm in wormhole-switched interconnections," *The Journal of Supercomputing, Springer US,* vol. 41, no. 3, 2007, pp. 215-245.

[90] F. Safaei, and A. Mortazavi, "A Novel Routing Algorithm for Achieving Static Fault-Tolerance in 2-D Meshes," *10th IEEE International Conference on Computer and Information Technology(CIT 2010),* 2010, pp. 2621-2627.

[91] J.-H Youn, B. Bose and S. Park, "Fault-Tolerant Routing Algorithm in Meshes with Solid Faults," *The J. of Supercomputing*, vol. 37, 2006, pp.161–177.

[92] G. Wang, J. Chen and C. Lin, "A New Fault-Tolerant Broadcast Routing Algorithm on Mesh Networks," *J. of Interconnection Networks* , vol. 11, nos. 3 & 4, 2010, pp. 175–187.

[93] X. Duan, D. Zhang and X. Sun, "Fault-Tolerant Routing Schemes for Wormhole Mesh," *Int'l Symposium on Parallel and Distributed Processing with Applications, ISPA*, 2009, pp. 298-301.

[94] Z. Jiang, J. Wu, and D. Wang, "A New Fault Information Model for Fault-Tolerant Adaptive and Minimal Routing in 3-D Meshes," *Proc. 34th Int'l Conf. Parallel Processing*, 2005, pp. 500-507.

[95] C. Chen and G. Chiu, "A Fault-Tolerant Routing Scheme for Meshes with Nonconvex Faults," *IEEE Transactions on Parallel and Distributed* Systems, vol. 12, no. 5, May 2001, pp. 467-475.

[96] M. E. Shaheen and A. Abukmail, "A Fault Tolerant Deadlock-Free Multicast Algorithm for 2D Mesh Multicomputers," *The Journal of management and Engineering Integration (JMEI)*, vol. 5, no. 2, May 2012, pp. 1-9.

[97] J. Han and M. Kamber, *Data Mining: Concepts and Techniuqes*, Morgan Kaufmann Publishers, 2001.

[98] Lee J. L. and Siau K, "A review of data mining techniques," *Industrial Management and Data Systems*, 2001, pp. 41–46.

[99] Li Y.-H., Sun L.-Y, "Study and applications of data mining to the structure risk analysis of customs declaration cargo," *Proc. of the IEEE International Conference on e-Business Engineering*, October 2005, pp. 761–764.

[100] I. H. Witten and E. Frank. *Data Mining: Practical Machine Learning Tools and Techniques*, Morgan Kaufmann, 2nd ed. 2005.

[101] G. Harvey, *Excel 2007 For Dummies*, Wiley, 2006.

[102] Amos Gilat, *MATLAB: An Introduction with Applications 2nd Edition*, John Wiley & Sons, 2004.