NOTE TO USERS

EFFICIENT MULTICAST ROUTING ALGORITHMS FOR MESH-CONNECTED MULTICOMPUTERS

SHENGJIAN WANG

A THESIS IN THE DEPARTMENT OF

COMPUTER SCIENCE AND SOFTWARE ENGINEERING

PRESENTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS

FOR THE DEGREE OF MASTER OF COMPUTER SCIENCE

CONCORDIA UNIVERSITY

MONTRÉAL, QUÉBEC, CANADA

AUGUST 2005

Your file Votre référence
ISBN: 0-494-10298-5
Our file Notre référence
ISBN: 0-494-10298-5

Abstract

Efficient Multicast Routing Algorithms for Mesh-connected Multicomputers

Shengjian Wang

Multicast is a collective communication method in which a message is sent from a source to an arbitrary number of distinct destinations. Multicomputers refer to massively parallel computers that consist of thousands of processors built to handle computation intensive applications. Mesh is a kind of network topology widely used in multicomputers.

Performance of multicomputers largely depends on that of the underlying network communications such as multicast, which is essential for processors to exchange data and messages. Two major parameters used to evaluate multicast routing are the time it takes to deliver the message to all destinations and the traffic which refers to the total number of links involved. Research indicated that these two parameters are normally not independent, but contradict each other. It has been proved that traffic optimal multicast problems such as *Optimal Multicast Path/Tree* in mesh-connected network are NP-complete. Hence, it is NP-hard to find multicast routing which is optimal on both time and traffic.

In this thesis, we proposed three efficient multicast routing algorithms for mesh-connected multicomputers: DIAG, DDS and XY-path, all of which have a small complexity of $O(KN)$ or less. The DIAG and DDS are two tree-based shortest path multicast routing algorithms designed for store-and-forward switched mesh network, which obtained near optimal time and reduced traffic significantly over its predecessor

VH algorithm. XY-path is a dual-path-based multicast routing algorithm intended for wormhole routed 2D mesh network, which reduced the time and traffic significantly over LIN's Hamiltonian path-based algorithm. Performance evaluations of these algorithms resulted from simulations are given at the end of the thesis.

Acknowledgement

I would like to express my sincere thanks to my advisor Dr. Hovhannes Harutyunyan without whom this work couldn't have been accomplished. I thank him for his mentorship to me of not only how to do research but also how to be a wonderful person, for his financial support to me, for the seminars he hosted which laid the solid foundation for my study in this field, for his patience to explain the problems to me and especially for his instructive advice to help me tackle some hard problems during the process of my thesis research. Thank you, Dr. Harutyunyan, for all the support I got from you.

I would also like to thank my parents, my brothers and sisters for their love and support, especially at tough times when I have doubt about myself.

Finally, I would like to extend my thanks to the faculty of the department of computer science and software engineering, some of whom instructed me in the major courses I took, for the knowledge and skills I learned from them and for providing the excellent research facilities and resources. I also deeply appreciate all kinds of assistance I've got from the staff and fellow graduate students of my department, especially our secretary of graduate program who is always kind to give me advice on my graduate study.

This Work is Dedicated to My Beloved Mom and Dad

Ms. Qiaoying Liu and Mr. Leyao Wang

Who are always Proud of My Success

Table of Contents

List of Figures

Chapter 1

Introduction

1.1 Motivation

Given the enormous amount of computation demanded by applications such as scientific simulations and 3D animations, computers with a few processors are not powerful enough to process the huge amount of data at a practical speed. High performance computers are highly desirable to tackle these challenges. Multicomputers [1], also known as MPC (Massively Parallel Computer) [2], are built to meet these demands. Multicomputers usually consist of thousands of processors which are connected to each other and have their own local memory. In a multicomputer, a time consuming task is usually divided into small tasks distributed to a set of nodes which compute concurrently. Due to the fact that nodes in a multicomputer do not share physical memory, they must communicate by passing messages through the network. Consequently, efficient routing of messages is critical to the performance of multicomputers [2].

Messages can be passed from one to one (unicast), one to all (broadcast), or one to many also known as multicast. In fact, unicast and broadcast are just special cases of multicast which is a very basic operation in multicomputer. For example, memory updates in distributed shared memory system and global notifications of events all need to be implemented through multicast operations [1, 11]. Hence, multicast communication

can become the bottleneck of the overall performance of a multicomputer and efficient multicast routing algorithms are much desired.

Time and traffic are two key parameters in evaluating the performance of multicast communications [1]. Mesh-connected network is a popular network topology for constructing multicomputers because it scales well [2]. Efficient multicast routing algorithms intended for mesh-connected networks, which try to minimize both time and traffic, will be especially practical and useful in the field of parallel and distributed computing. Many multicast routing algorithms have been proposed for mesh-connected multicomputers [4, 5, 10, 11, 26, 33, 34], but most of them are unicast-based which proved not to be very efficient neither on multicast time nor on multicast traffic. With the demand on the performance of multicomputers constantly increasing, more efficient hardware-supported tree-based multicast routing algorithms are yet to be developed. And this is why we conduct this thesis research.

1.2 Overview of the Problem

Multicast refers to a collective communication in which the same message is delivered from a source to an arbitrary number of distinct destinations [1]. It has been applied in many network-related domains such as multicomputers, telecommunication and the Internet for long time.

A network is essentially a set of objects which can communicate with each other through direct or indirect (through a sequence of other objects) connections. There are mainly two ways of connecting nodes in a network. One way is to connect all the nodes to a shared communication line called bus. The other is through point to point or direct

connection [2], in which there is an exclusive link between a node and each of its neighboring nodes. Networks using direct connection are called direct networks. Direct networks especially mesh-connected are the primary network topologies used in multicomputers because they scale well.

In a multicast within direct networks, the message is first sent from the source to its neighbors, and then disseminated further to other intermediate nodes until all the destination nodes receive the message. The efficiency of multicast in direct networks largely depends on the underlying network technologies such as topologies, routing algorithms, and switching technologies [1, 2].

Multicast in Multicomputer

Multicast exists as a basic operation in multicomputers to improve the efficiency of communications as mentioned in [1, 7, 11, 15]. First, it is often required to distribute large data array over system nodes in many scientific computations and to support data updating in distributed shared memory system. Second, it is required in control operations such as global synchronization in image and 3D graphics processing on parallel computers. Third and primarily, it is needed to support parallel applications, languages, and algorithms [1]. In parallel search algorithms including game-tree search and those for artificial intelligence problems, a set of processes collectively solve a decision or optimization problem. Processes in such applications typically search a global state space and may use multicast to efficiently inform one another concerning their findings.

Because multicast communication is a primitive operation in multicomputers, the performance of multicomputers depends not only on the speed of processors but also on the performance of communications. The primary metric for communication performance in direct network is the communication latency [6]. Hence, multicast communication latency incurred by a message traversing from a source node to destination nodes will significantly affect the overall performance of the multicomputer system and the granularity of parallelism as well. It is essential to develop efficient multicast routing algorithms for multicomputers.

Implementation of Multicast

There are several ways to implement multicast. A simple and naïve way to accomplish it is through multiple one to one communications (unicast), called unicast-based multicast [5]. Another way is to use broadcast in which every node forwards a copy of the message to its neighbors who have not received the message yet. Although these two ways are simple, they pay little attention to the efficiency and cost of the operation. In unicast-based multicast the message is sent from the source node to each destination node sequentially. In broadcast non-destination nodes are also to receive the message, which will obviously cause extra time and traffic.

A more efficient and effective way, called tree-based multicast [8, 27, 37, 39, 40], is to use a good strategy to disseminate the message from the source to multiple destinations concurrently through the smallest subset of links and nodes of the host network, in which each link and node occurs only once. Thus, nodes and links involved in the multicast form a tree rooted at the source node. In tree-based multicast, the message is first sent out

from the source node to some of its neighbors, and then nodes that have received the message relay it to some of their own neighbors, the dissemination keeps going until all destinations received the message. Tree-based multicast overcomes the drawbacks of unicast-based multicast, and broadcast. Therefore, tree-based multicast routing algorithms are desirable to improve the performance of multicomputers.

The Problem in Multicast

For tree-based multicast, the main challenge is to find the optimal multicast tree. There are mainly two parameters that are used to evaluate the performance of multicast, i.e. time and traffic. The problem is that multicast time and traffic do not seem to be independent from one another. Instead, they contradict each other. Experiments show that optimization dedicated to one parameter will usually deteriorate the performance of the other. As a matter of fact, Lin has proved in [1] that finding the optimal multicast tree in mesh-connected networks is NP-complete. So, the multicast problem seeking optimal results on both time and traffic is NP-hard. Heuristics are needed to solve the problem. A good approach is to minimize one parameter first, and try to reduce the cost of the other one as much as possible. Since our goal is to develop efficient multicast routing algorithms for time-critical multicomputers, we use the pro-time approach that minimizes the time first.

Mesh-connected network is essentially a multi-dimensional array of nodes connected to their neighbors in each dimension. Because of its geometric regularity, it scales very well and is by far the most used topology for multicomputers. Hence, multicast problems in multicomputers can often be treated as those in mesh-connected networks. In a mesh-

connected network, distance and routing between nodes are related to their positions in the array, which significantly reduces the complexity of routing algorithms. Therefore, the NP-complete optimal multicast tree problem can be much easier in a mesh-connected network. Time optimal multicast routing with near optimal traffic is likely achievable with reasonable computation complexity. Heuristics are the key to develop such algorithms.

1.3 Scope of the Study

Multicast in general is a very wide topic due to a large number of applications and a variety of underlying network technologies. Many problems such as traffic balancing, flow control and deadlock prevention need to be considered in multicast routing.

In our study, we focus on the efficiency of multicast routing algorithms designed for mesh-connected network (Hypercube excluded) under a theoretical model. We do not consider the details of any specific network architecture, and assume everything works perfectly in an ideal environment without unpredictable problems such as blocking and fault tolerance. Moreover, our algorithms are designed for multicasting of a single message without special optimizations on concurrent multicasting of multiple messages. Finally, due to the NP-completeness of the problem, our goal is not necessarily to find the optimal solution, but to design heuristics that achieve near optimal result.

1.4 Organization of the Thesis

The organization of the rest of the thesis is as follows. In Chapter 2 we review the preliminary knowledge, previous studies, and the latest development related to

multicasting in mesh-connected multicomputers. In Chapter 3 we present our methodology of solving the problem and propose three efficient multicast routing algorithms: the DIAG, DDS, and XY-path algorithms. In Chapter 4 we evaluate the performance of the three algorithms based on the data and statistics resulted from the simulations. The last chapter is a brief summary of the accomplishments and conclusions we achieved through our study, and of the future work that need to be done in this field.

Chapter 2

Literature Review

2.1 Preliminary

A network is often modeled as a graph in theory study. Hence, network related problems such as multicast problems become graph theory problems. To better understand the problem and our model which will be presented later, we first have a brief review of the basics of multicomputer network, graph theory, and multicast communication.

2.1.1 Network Basics

2.1.1.1 Communication Model

Depending on the number of destinations in a communication, there are three types of communication models: unicast (one to one), multicast (one to many), and broadcast (one to all) [3]. Unicast problem is usually to find the shortest path between a pair of nodes, while the problem of broadcast is more like constructing a minimum spanning tree. In our study we do not particularly address the unicast and broadcast problems, but only discuss multicast problems. In some cases, multicast can be implemented through multiple unicasts or broadcast. But such solutions will cause unnecessary large amount of traffic as shown in *Figure 2.1.1*. Our goal is to develop tree-based multicast routing algorithms that are more efficient than those implemented through unicast or broadcast.

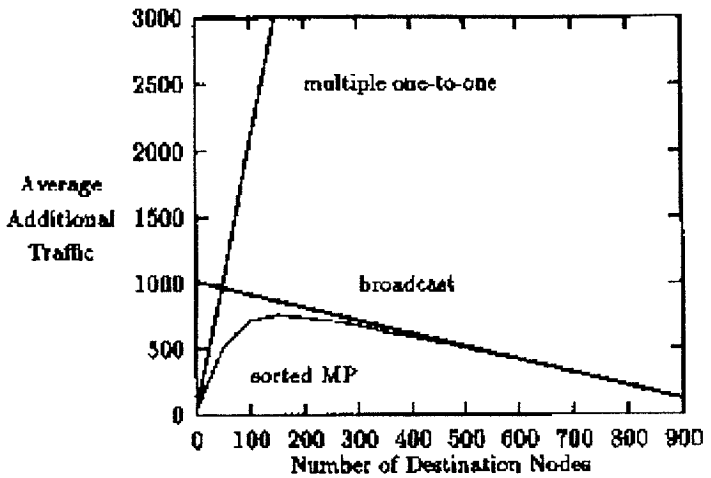


Figure 2.1.1: Performance of Multicast vs Broadcast

2.1.1.2 Network Topology

Network topology refers to the configuration of connections between nodes. It can be classified into two broad categories: direct networks and indirect networks [6].

Types of Network Topology

In direct networks each node has a point-to-point or direct connection to some of the other nodes called neighboring nodes. The ring, star, mesh, and tree are some examples of direct network topologies (*Figure 2.1.2* [21]). A direct network is called a mesh network if there are at least two nodes which can be connected through several paths.

In indirect networks nodes are connected to other nodes through one or more switching elements. Examples of indirect networks include the crossbar, bus, and multistage interconnection networks. Among them the bus network such as the Ethernet is the most commonly used. Bus networks are easy to set up (add or remove nodes). But the failure

of the bus will disable the whole network, and the performance degrades quickly as the number of nodes increases.

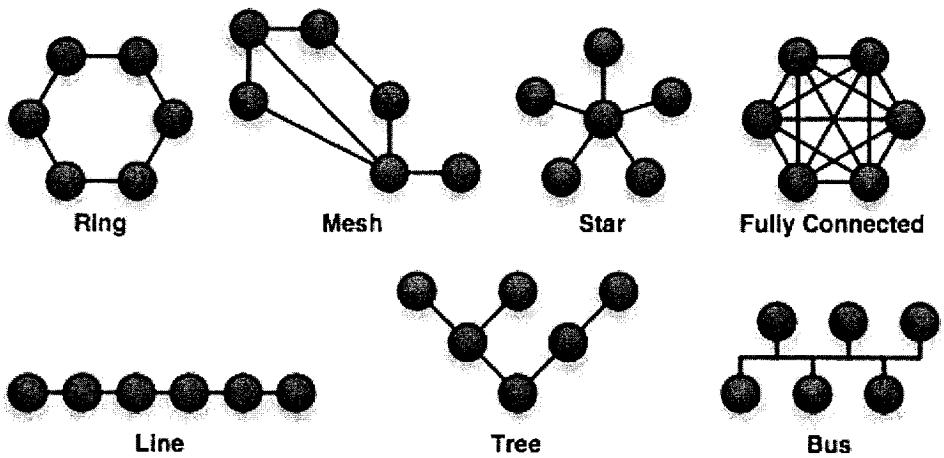


Figure 2.1.2: Common Network Topologies

Mesh networks, on the other hand, are more reliable and fault tolerant. The failure of one node or several links will not fail the whole network. More importantly they scale well. The total communication bandwidth, memory bandwidth, and processing capability of the system increase with the number of nodes. Therefore, mesh has emerged as a popular topology for multicomputers. In multicomputers nodes are usually arranged as an n-dimensional array and are connected to their neighbors in each dimension. Such topologies are called n-dimensional meshes. In what follows a mesh refers to an n-dimensional mesh unless otherwise specified. Here is its formal definition.

Definition 2.1.1 (n-Dimensional Mesh) An n-dimensional mesh is an interconnection structure that has $k_0 \times k_1 \times \ldots \times k_{n-1}$ nodes, where k_i denotes the number of nodes in the ith

dimension. Each node in the mesh is identified by an n-coordinate vector $(x_0, x_1, ..., x_{n-1})$, where $0 \leq x_i \leq k_i-1$. Two nodes $(x_0, x_1, ..., x_{n-1})$ and $(y_0, y_1, ..., y_{n-1})$ are connected if and only if there exists an i such that $x_i = (y_i \pm 1)$, and $x_j = y_j$ for all $j \neq i$. Thus the number of neighbors of a node ranges from n to $2n$, depending on its location in the mesh [6].

In a mesh there are no connections between the first node and the last node of each dimension. Hence, communications between them need to travel a long distance. But if we add a link between the first node and the last node, links along each dimension form a circle. Therefore, a message can travel in the two opposite directions of one dimension but reach the same destination, which significantly reduces the distances between nodes. Meshes with wraparound links are called tori. The following is the formal definition of an n-dimensional torus [2].

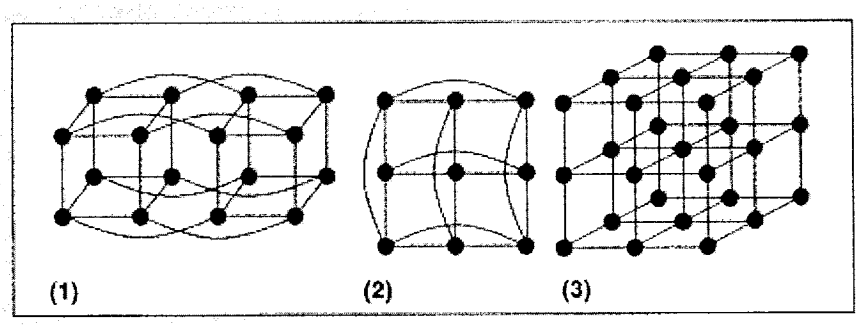
Definition 2.1.2 (n-Dimensional torus) An n-dimensional torus differs from an n-dimensional mesh only at the interconnection structure. Two nodes $(x_0, x_1, ..., x_{n-1})$ and $(y_0, y_1, ..., y_{n-1})$ in a torus are connected if and only if there exists an i such that $x_i = (y_i \pm 1) \bmod k_i$, and $x_j = y_j$ for all $j \neq i$. The wraparound links in an n-dimensional torus (specified by the use of modulus in the definition) are not present in n-dimensional meshes.

A torus is a symmetrical topology in which the degree of a node is the same regardless of its location in the network. This can balance the traffic load and simplify the routing algorithms in tori. The mesh, on the other hand, is an asymmetrical topology in which the node degree depends on its location. Links near the center of the mesh experience higher

traffic density than those near the boundaries. The diameter of a mesh is greater than that of the respective torus [6]. But the scalability of tori is not as good as that of meshes since wraparound links usually bear different weights (length, bandwidth etc.) from those of the regular links. Some special cases of n-dimensional meshes and tori have special names, e.g. K-ary n-cube and hypercube. Here are their definitions.

Definition 2.1.3 (K-ary n-cube) A K-ary n-cube is a special case of n-dimensional torus when the sizes of all n dimensions are the same, say, K.

Definition 2.1.4 (Hypercube) A hypercube is a special case of n-dimensional meshes when the sizes of all n dimensions are *2*. It is also a special case of K-ary n-cube when $K=2$, consequently called *2*-ary n-cube [2].



(1) 2-ary 4-cube (Hypercube); (2) 3×3 2D Torus; (3) 3×3×3 3D Mesh;

Figure 2.1.3: Mesh-connected Network Topologies

K-ary n-cube and hypercube are even more symmetrical and regular topologies than tori. They are popular for first generation multicomputers. For example, the hypercube was adopted by iPSC-2 and nCube-2/3. Whereas, low-dimensional meshes and tori

become more popular in the new generation multicomputers such as Paragon and Symult 2010 built with 2D meshes, Cray T3D and MIT J-Machine built with 3D meshes [2]. *Figure 2.1.3* [2] illustrates a *4*D hypercube, a 3D mesh, and a 2D torus.

Choosing a Topology for Multicomputers

Given the number of nodes in a multicomputer, how should we choose a topology that can achieve the best network performance? In general, low-dimensional meshes are preferred because they have low node degrees and fixed-length channel wires which make them more scalable than high-dimensional meshes and K-ary n-cubes. Low-dimensional meshes also have higher channel bandwidth per bisection density and lower blocking latencies, which result in lower communication latency [6]. On the other hand, high-dimensional meshes or K-ary n-cubes have smaller diameters and average distances. Therefore, for networks where communication latencies depend on the path length, high-dimensional meshes such as hypercubes are good. High-dimensional meshes also have more paths between pairs of nodes. They provide better adaptability and fault tolerance. In our study, the multicast algorithms are mainly designed for 2D or 3D meshes, but they can be easily applied to 2D/3D tori and higher dimensional meshes.

2.1.1.3 Node Architecture of Multicomputers

The nodes in a multicomputer communicate by passing messages through an interconnection network [6]. Hardware support is essential to handle the transmission of messages between nodes. In most systems, a router is associated with each node to handle communication-related tasks. *Figure 2.1.4* shows the architecture of a generic node

consisting of a processor, a local memory, a router, and some other I/O devices. The router is connected to the processor and local memory through internal channels. The internal input channel is used to absorb messages destined for the host processor. The internal output channel is used to send outgoing messages from the host processor to remote nodes. According to the number of internal channels, we have either all-port or *k*-port architecture. In the all-port architecture, every external channel has a corresponding internal channel, thus allowing the node to send or receive on all external channels simultaneously. A *k*-port architecture has *k* (less than the total number of external channels) internal channels, which are multiplexed by the external channels. When *k* is one, we call it one-port architecture.

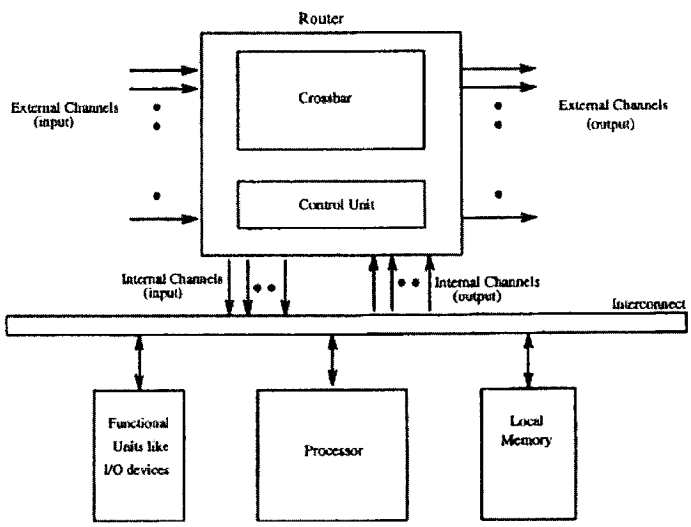


Figure 2.1.4: Node Architecture in Multicomputer

In practice one-port and all-port are the most used architectures. In one-port architecture, nodes can not send/receive messages to/from several different neighboring nodes simultaneously but sequentially due to the contention for the internal channel

which results in delay. But in all-port architecture, this kind of delay can be avoided since messages can be sent/received simultaneously.

2.1.1.4 Routing Techniques

Routing techniques can be classified with respect to different characteristics. They can be classified as source routing or distributed routing according to where routing decisions are made. In source routing the entire path for message routing is decided at the source node before the message is sent. Each message carries the complete routing information in its header, which increases the overall message size. In distributed routing, routing decisions are made at every intermediate node through which the message traverses [6].

Routing can also be classified as deterministic or adaptive based on the path selection process. In deterministic routing the path is determined by the source and destination addresses, and consequently is unique. Adaptive routing, on the other hand, provides multiple paths from the source to the destination, and the path taken by a particular message depends on network conditions and the routing algorithm.

Routing can be minimal or non-minimal. In minimal routing, the message is routed through one of the shortest paths between the source and the destination. In non-minimal routing, a message can take any path between the source and the destination, and thus might take a non-shortest path because of congestion or faults in the shortest paths.

In practice, a routing algorithm can combine characteristics from different routing techniques. Take multicast routing for example. If we use distributed routing, the complicated routing algorithm has to be executed at each node, which results in much overhead time and reduces overall performance dramatically when the number of

intermediate nodes is large. But if we use source routing, messages carry too much routing information in the header, which not only costs buffer space but also causes overhead transmission time. Semi-distributed routing is proposed as a compromised solution between source routing and distributed routing. In semi-distributed routing, an overall routing algorithm is executed at the source node generating the most important routing information (only information of the destination and replicate nodes) to be embedded in the message header. And then, at each intermediate node, only very simple routing decisions need to be made based on the routing information in the header. Semi-distributed routing successfully overcomes the pitfalls of source routing and distributed routing, and is widely used in multicast.

2.1.1.5 Switching Techniques

Nodes in a direct network communicate by passing messages from one node to another. A message may be divided into equal or variable-sized packets. A packet is the smallest unit of data that contains routing and sequencing information. In most multicomputers, a packet enters the network from a source node and is switched or routed toward its destination through a series of intermediate nodes. Four types of switching techniques are usually used for this purpose: circuit switching, store-and-forward, virtual cut-through, and wormhole.

In circuit switching, a dedicated path (physical circuit) is established between the source and the destination before the data transfer initiates. Once the data transfer is initiated, channels on the path are reserved exclusively, the message is never blocked and buffering of data is not needed. On the other hand, establishing the path requires

significant overhead time and reservation of all channels for the entire duration of message transmission degrades performance. Circuit switching was once used in old multicomputers such as iPSC/2 but is no longer used in new commercial multicomputers.

In store-and-forward switching, a message is divided into packets that are independently routed towards their destination. The entire packet is buffered at every intermediate node and then forwarded to the next node in its path. The advantage of packet switching is that the channel resources are occupied only when a packet is actually transferred. The drawback is that, since the packet is stored entirely at each intermediate node, the time to transmit a packet from the source to the destination is directly proportional to the number of hops in the path. Ncube-1 and Ametek 14 are multicomputers that adopted store-and-forward switching.

In virtual cut-through, while routing toward its destination, a packet is stored at an intermediate node only if the required next channel is occupied by another packet. Hence, the distance between the source and destination has little effect on communication latency. The disadvantage of the virtual cut-through technique is that each node must provide sufficient buffer space for all the messages passing through it.

Wormhole routing is a variant of the virtual cut-through technique that avoids the need for large buffer spaces. In wormhole routing, a packet is transmitted between the nodes in units of flits, the smallest unit of a message on which flow control can be performed. The header flit(s) of a message contains all the necessary routing information and guides the route. All the other flits contain the data elements and follow the header flit(s) contiguously in a pipelined fashion as indicated in *Figure 2.1.5*. The main advantage of wormhole routing is that its transmission latency is insensitive to the distance between

the source and destination. Moreover, since the message moves flit by flit across the network, each node needs to store only one flit. Wormhole routing is said to be the most promising switching technology for multicomputers and is used in Touchstone Delta, Symult 2010, and MIT J-Machine [6].

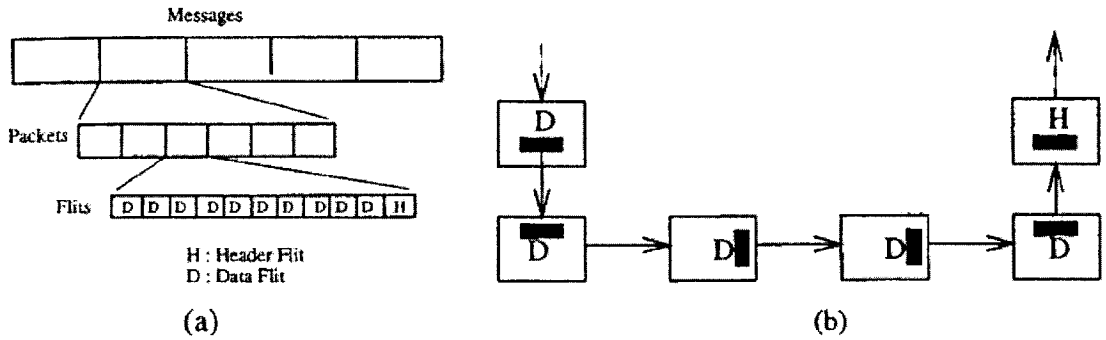


Figure 2.1.5: Message Format in Wormhole Routing

The drawback of wormhole routing is that it can easily cause blocking and deadlock. Since the header flit governs the routing, if it cannot advance in the network due to resource contentions, all the trailing flits are also blocked along the path and can further block other messages or cause deadlock. Prevention of deadlock is one of the main issues in wormhole switching, and is usually accomplished by a suitable choice of routing function (such as XY-routing) that selectively prohibits messages from taking some paths, thus prevents cycles in the network. Virtual channels can also be used in wormhole-routed networks to prevent deadlock and chained blocking [Dally 1992].

2.1.2 Multicast

2.1.2.1 Multicast Problems

Let the interconnection topology of a multicomputer be denoted by a host graph $G(V, E)$, where each vertex in V corresponds to a node and each edge in E corresponds to a communication link. For a multicast communication, let u_0 denote the source node and u_1, u_2, ..., u_k denote k destination nodes, where $k \geq 1$. The set $K = \{u_0, u_1, u_2, ..., u_k\}$, which is a subset of $V(G)$, is called a multicast set. Depending on the underlying communication paradigm and the routing method, the multicast communication problems can be formulated as four different graph theoretical problems: *Multicast Path (MP)* problem, *Multicast Cycle (MC)* problem, *Steiner Tree (ST)* problem, and *Multicast Tree (MT)* problem [1].

Multicast Path (MP) Problem

In some communication mechanisms, replication of an incoming message in order to be forwarded to multiple neighboring nodes involves too much overhead and is usually undesirable. Thus, some routing method does not allow each node to replicate the message passing by. In such case the multicast routing problem becomes finding a path starting from u_0 and visiting all k destination nodes. The optimization problem is to find an optimal multicast path (*OMP*) and is formally defined as follows.

Definition 2.1.5 (Optimal Multicast Path) A multicast path $MP(v_1, v_2, ..., v_n)$ for a multicast set K in G is a subgraph $P(V, E)$ of G, where $V(P) = \{ v_1, v_2, ..., v_n \}$ and $E(P) = \{ (v_i, v_{i+1}): 1 \leq i \leq n-1 \}$, such that $v_1 = u_0$ and $K \subseteq V(P)$. An *OMP* is an *MP* with the

shortest total length.

Multicast Cycle (MC) Problem

Reliable communication is essential to a message passing system. Usually, a separate acknowledgment message is sent from every destination node to the source node. One way to avoid sending $|K|$ separate acknowledgment messages is to have the source node itself receive a copy of the message it initiated after the last destination node being visited. Thus, the multicast communication problem is the problem of finding a shortest cycle, called *Optimal Multicast Cycle* (*OMC*) for *K*.

Definition 2.1.6 (Optimal Multicast Cycle) A multicast cycle $(v_1, v_2, ..., v_n, v_1)$ for K is a subgraph $C(V, E)$ of G, where $V(C) = \{ v_1, v_2, ..., v_n \}$ and $E(C) = \{(v_n, v_1), (v_i , v_{i+1}): 1 \leq i \leq n-1\}$ such that $K \subseteq V(C)$. An *OMC* is an *MC* with the shortest total length.

Steiner Tree (ST) Problem

Both *OMC* and *OMP* assume that the message will not be replicated at any node during transmission. However, message replication can be implemented by using some hardware approach. If the major concern is to minimize traffic, the multicast problem becomes the well-known *Steiner Tree* problem [1]. Formally, we restate the *ST* problem as follows.

Definition 2.1.7 (Minimal Steiner Tree) A Steiner tree $S(V, E)$ for a multicast set K is a subtree of G, such that $K \subseteq V(S)$. A *Minimal Steiner Tree* (*MST*) is a *ST* with the minimal total length.

Multicast Tree (MT) Problem

In the *ST* problem, we do not require the using of a shortest path from the source to a destination. If the distance of two nodes is not a major factor to the communication time, such as wormhole routing, the above optimization problem is appropriate. However, if the distance is a major factor to the communication time, such as the store-and-forward mechanism, then we may like to minimize time first then traffic. The multicast communication problem is then modeled as an *Optimal Multicast Tree* (*OMT*) problem which is defined as below.

Definition 2.1.8 (Optimal Multicast Tree) An *OMT*, $T(V,E)$, for K is a subtree of G, such that *a)* $K \subseteq V(T)$, *b)* $dT(u_0, u_i) = dG(u_0, u_i)$ for $1 \leq i \leq k$, and *c)* $|E(T)|$ is as small as possible.

Complexity of Multicast Problems

Essentially, in terms of computation complexity, the above graph optimization problems can be stated as: Given a host graph G, a multicast set K, and an integer l, does there exist an *OMP* (*OMC*, *MST*, *OMT*) for K with total length less than or equal to l? Apparently, the complexity of each of the above optimization problems is directly dependent on the underlying host graph. Lin *et al* [1] proved a number of results related to the complexity of multicast problems, which we treat as theorems here:

Theorem 2.1.1 The *OMC*, *OMP*, *MST* and *OMT* problems for 2D mesh graph are all NP-complete.

It is not difficult to see these results can also be applied to higher dimensional meshes. Since a multicast problem in a 2D-mesh can always be treated as a special case of that in an n-dimensional mesh ($n>2$), if multicast problems in n-dimensional meshes are not NP-complete, then they should not be NP-complete for 2D meshes either, which contradicts the theorem above. Hence, we have the following theorem.

Theorem 2.1.2 The *OMC*, *OMP*, *MST* and *OMT* problems for n-dimensional ($n>2$) mesh graphs are all NP-complete.

2.1.2.2 Multicast Models

We just studied the multicast problems merely at theoretical level in the previous section. In practice multicast in mesh-connected networks can be implemented in many different ways which are referred here as multicast models. In general, multicast models are divided into two categories: unicast-based and tree-based. In most of first generation multicoimputers, the hardware at each node only supports one to one communications. Therefore, multicast communications can only be implemented through multiple unicasts, named unicast-based multicast.

Unicast-based multicast

Unicast-based multicast can also be implemented in many ways. The naive way is just to send a separate copy of the message from the source node to every destination node, hence called separate addressing [4]. This is the simplest but most costly way both on

time and traffic since it is completely sequential and makes no use of any previous traffic. Another way is called chain tree, in which the source node only sends a copy of the message to a subset of destinations and then the destinations that have received the message further forward it to those that have not received yet. The process continues until all destinations receive the message, thus forming a logic message-passing tree composed of all destination nodes.

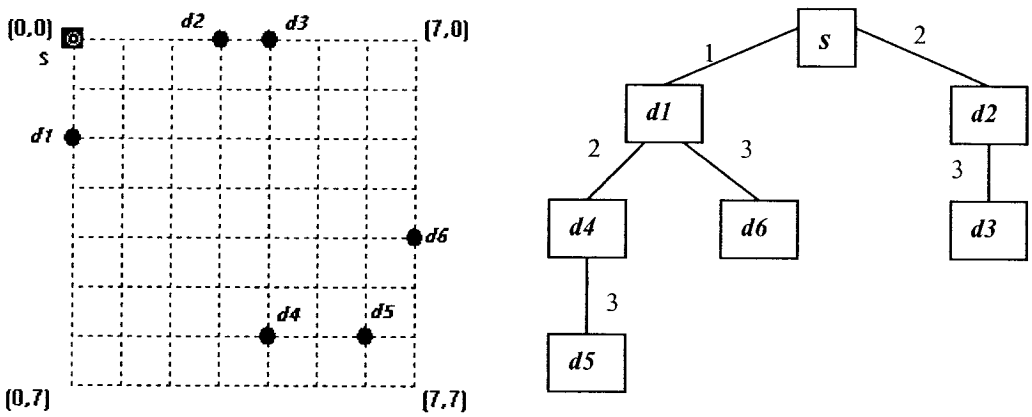


Figure 2.1.6: Example of Unicast-based Multicast

Figure 2.1.6 illustrates how chain tree multicast works. *Figure 2.1.6(a)* indicates the distribution of the destinations d_1, d_2, d_3, d_4, d_5, d_6 and the source node *s*. *Figure 2.1.6(b)* is a possible chain tree in which each link represents a unicast between the two nodes. Assume each node can be involved in one unicast at a time, the number besides the links indicates at which step the unicast is conducted.

Tree/Path-based multicast

In tree-based multicast, the hardware (router) at each node usually supports multicast routing. Routing information with the set of destinations is embedded in the header of a packet. The source node sends the packet to a selected set of its neighboring nodes, each of which is able to digest the routing information without the intervening of the *CPU* and decides what to do next. Depending on the routing information embedded in the packet, the receiving nodes may simply accept the packet (a leaf destination node) or may just pass the message to one of its own neighbors (a passerby node) or both (an intermediate destination node). In some cases the node (a replicate node) may need to replicate the message, modify the routing information in the header, and then pass it to several neighboring nodes. Thus, the message is disseminated from node to node in a certain order until all destinations receive the message. All nodes and links involved in the dissemination occur only once at a certain time, forming a tree in the host network rooted at the source node. In some special cases, the message is passed along one or several paths starting at the source node.

Tree-based multicast is recognized for its high efficiency on both time and traffic [8, 27, 37, 39, 40]. Hardware support and high parallelism in the message dissemination both significantly reduce the communication latency. In the multicast tree, destination nodes share as much common routing path as possible and no links occur twice. Hence, the total traffic tends to be small too. Tree-based multicast generally outperforms the unicast-based multicast and has become a more promising multicast model for the new generation multicomputers.

2.1.2.3 Multicast in Mesh

Mesh, due to its good properties, has become a popular network topology in multicomputers. Compared to multicast in an arbitrary network topology, multicast routing in mesh-connected networks is much easier to design. It can be implemented at a low cost as a result of the meshes' geometric regularity. For example, nodes of mesh are arranged like an array, connectivity and distance between them can be determined by comparing their coordinates. And the routing is simply to choose the direction (dimension) at each node. These good characteristics reduce the complexity of the algorithm significantly, which will otherwise involve a lot of graph based searching. Moreover, because of the regularity, the length and bandwidth of each link is almost the same. Hence, we can assign the same weight usually one to each link, which simplifies the calculation and achieves very good scalability.

2.1.2.4 Multicast Evaluation Criteria

The main process of multicast routing is to select paths to deliver a message from the source to all its destinations in a very efficient and economical way. Lin [1] proposed *traffic* and *time* as two major parameters to evaluate multicast communication. The parameter *traffic* refers to the number of links used to deliver the source message to all its destinations. The parameter *time* is the message transmission time starting from when the source sends out the message till the last destination receives it.

Good performance of a multicast routing algorithm involves several contradicting requirements. One requirement is that the message should be sent from the source to all destinations as soon as possible, i.e. to minimize the multicast time. Another requirement

is that the multicast operation should create as less traffic as possible. Heavy traffic not only costs resources but also causes contention, blocking, deadlock, and could increase the time. The third important requirement is the computational complexity of the routing algorithm should be small. Large computation complexity will increase the delay at each node and deteriorate the overall performance. However, these three parameters are not totally independent, and obtaining optimal result for one parameter may degrade the result for the other.

As a matter of fact, we have already learned that *OMP*, *OMC* and *MST* problems which try to obtain optimal multicast traffic are NP-complete. Hence, the problem of obtaining results optimal for both time and traffic will be NP-hard. A practical solution is to design heuristic algorithms with a reasonable complexity, which can obtain near optimal result of one parameter without seriously degrading the result of the other parameters.

2.1.2.4.1 Evaluation of Multicast Traffic

Calculation of multicast traffic is indifferent regardless of the underlying network topology, switching technology, and routing algorithms. For tree-based multicast, the traffic is just the total number of edges in the tree. Let *MT(V,E)* define the multicast tree for multicast from the source u_0 to K destinations where V is the set of nodes in the tree and E is the set of edges in the tree, then we have the formula.

$$\textit{Multicast Traffic} = |E(MT)| \qquad (2.1.1)$$

Besides the parameter *traffic,* another parameter called *Additional Traffic* is also widely used to evaluate the traffic [1]. *Additional Traffic* is essentially the total traffic subtracted by the total number of destinations [1, 7, 12]. Sending a message to K destinations needs at least K links, each of which delivers the message to one destination. So, the amount of traffic on top of the minimum necessary traffic is considered additional traffic. It reflects the efficiency of the traffic better than total traffic. In the multicast stated above, we have the following formula.

$$\textit{Additional Traffic} = |E(MT)| - K \qquad (2.1.2)$$

2.1.2.4.2 Evaluation of Network Latency

The performance of communication is often evaluated by the communication latency which is an accumulation of network latency. Network latency is defined as the time between the moment when the message head is injected into the network by the source node and the moment when the message tail is absorbed by the destination node. Calculation of network latency greatly depends on the underlying switching technologies. We will present the formulas to calculate the network latency for the four types of switching techniques used in multicomputer network.

In store-and-forward switched network, when a packet reaches an intermediate node, the entire packet has to be stored in the buffer before it is forwarded to a neighboring node. Hence, the message transmission time is linearly proportional to the number of hops (distance) between two nodes. Thus, the network latency is:

$$\textit{Network Latency}=(L/B)D \qquad (2.1.3)$$

Where L is the length of the packet, B is the bandwidth of the channel, and D is the length of the path between the two nodes, i.e. the number of links in the path.

In circuit switching, a control packet is first transmitted to construct a physical circuit between the source and the destination node. Once the circuit is established, the actual packet is transmitted along the circuit to the destination. Thus, the network latency is:

$$\textit{Network Latency}=(L_c/B)D + L/B \qquad (2.1.4)$$

Where L_c is the length of the control packet transmitted to establish the circuit and when $L_c \ll L$, the distance D has a negligible effect on the network latency.

In virtual cut-through switching, a packet is stored at an intermediate node only if the next required channel is busy; otherwise, it is forwarded immediately to next node without buffering. The network latency is:

$$\textit{Network Latency}= (L_h/B)D + L/B \qquad (2.1.5)$$

Where L_h is the length of the header field and when $L_h \ll L$, the distance D has a negligible effect on the network latency.

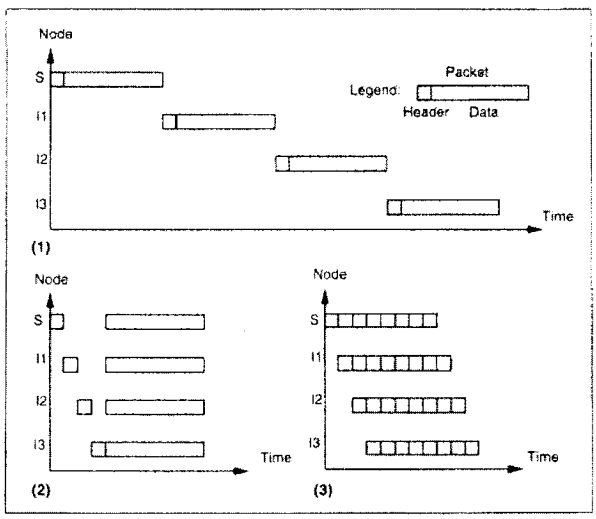
In wormhole routing, a packet is divided into a number of smaller units called flits for transmission. The header flit governs the route. As the header advances along the

specified route, the remaining flits follow in a pipeline fashion without delay. The network latency is:

$$\textit{Network Latency}=(L_f/B)D + L/B \qquad (2.1.6)$$

Where L_f is the length of each flit and when $L_f << L$, distance D has a negligible effect on the network latency.

Figure 2.1.7 illustrates the calculation of the network latency in networks using different switching techniques when a packet is transmitted from a source node *S* to the destination node *I3* through intermediate nodes *I1* and *I2*. Since the network latency of store-and-forward switching is the packet size times the hops, it is usually much larger than that of the other switching technologies. In general store-and-forward works well when the packets and network size are small.



(1) Store-and-forward (2) Circuit Switching (3) Wormhole routing

Figure 2.1.7: Comparison of Different Switching Techniques

Figure 2.1.8 compares the network latency at different distances in contention free networks using store-and-forward, circuit switching, and wormhole routing techniques. Although the network latency in wormhole-routed networks is distance-insensitive, it is still desirable to reduce the path lengths, which will reduce the overall traffic loads and channel contentions.

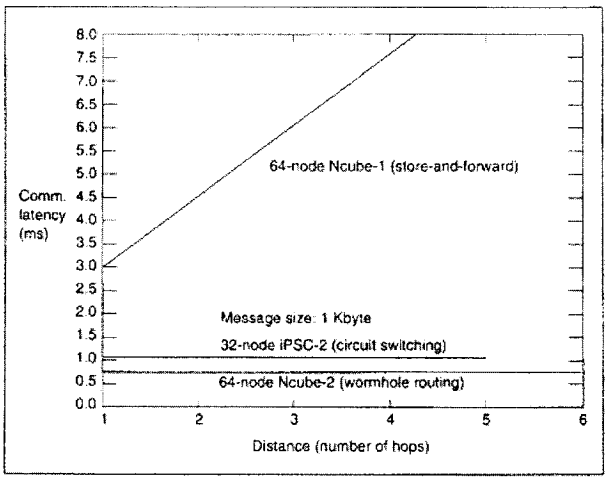


Figure 2.1.8: Comparison of Communication Latency

2.1.2.4.3 Evaluation of Multicast Time

Multicast time, also known as multicast communication latency, is the message transmission time of a multicast starting when the source sends out the message till the last destination receives the message. Communication latency depends on the underlying network technologies especially the switching techniques. Criterion for evaluating multicast time varies from one switching technology to another. In the following sections, we present the formulas to calculate the multicast time for the two most commonly used switching technologies: store-and-forward and wormhole routing.

For Unicast-based Multicast

Unicast-based multicast is accomplished through a series of unicasts between destinations in the order designated by a logical multicast tree. Since the network latency in wormhole routed networks is distance insensitive, each step of unicast takes approximately the same amount of time. Therefore, the unicast-based multicast latency is actually decided by the maximum number of message passing (unicast) steps needed to send the message to all destinations (i.e. depth of the logical multicast tree). Assume the number of unicasts needed to deliver the message from source node u_0 to destination nodes $v_1, v_2, \ldots, v_k$ is n_1, $n_2, \ldots, n_k$ respectively, then

$$\textbf{\textit{Multicast Time}}=max\{ n_1, n_2, \ldots, n_k \} \qquad (2.1.7)$$

An algorithm to construct the optimal multicast tree was proposed in [5] consisting of the minimum $\lceil log_2 m \rceil$ steps, where m is the total number of destinations. It is obtained when the number of destinations that receive the message doubles at each step.

For Tree/Path-based Multicast

Calculation of multicast latency for tree-based multicast is more complicated. Assume a multicast delivers a message to K destinations, the length of the path from the root/source node to the destination nodes $d_1, d_2, \ldots, d_{k-1}, d_k$ in the multicast tree are $l_1, l_2, \ldots, l_{k-1}, l_k$ respectively.

The network latency in store-and-forward switched networks is proportional to the distance, and can be counted in hops which represent the time for a message to be

transmitted from a node to its neighboring node. In a network with all-port architecture where the message can be transmitted from a node to all its neighbors simultaneously, the multicast time will simply be the length of the longest path to a destination, i.e.

$$\textit{Multicast Time} = max\{ l_1, l_2, \ldots, l_{k-1}, l_k \} \qquad (2.1.8)$$

But in networks with one-port architecture, hops waited at the branching nodes have to be counted. Assume on the path to destination node d_i, there are n branching nodes and the hops waited at each branching node is $h_1, h_2, \ldots, h_{n-1}, h_n$, then the total hops waited at branching nodes for d_i is w_i where

$$w_i = \sum_{j=1}^{n} h_j$$

Multicast time will be the maximum number of hops it takes for the message to reach all the destinations.

$$\textit{Multicast Time} = max\{ l_1+w_1, l_2+w_2, \ldots, l_{k-1}+w_{k-1}, l_k+w_k \} \qquad (2.1.9)$$

Figure 2.1.9 illustrates how the hops are counted in a multicast tree of one-ported network. The root is the source node, the filled circles stand for destination nodes, and the empty circles stand for intermediate nodes. The number beside each link designates the number of the hop when the message is transmitted at that link.

In wormhole-routed networks, things are quite different. The network latency (T) in wormhole routed networks is $T=(L_f/B)D+L/B$ as presented in *Formula (2.1.6).* Assuming

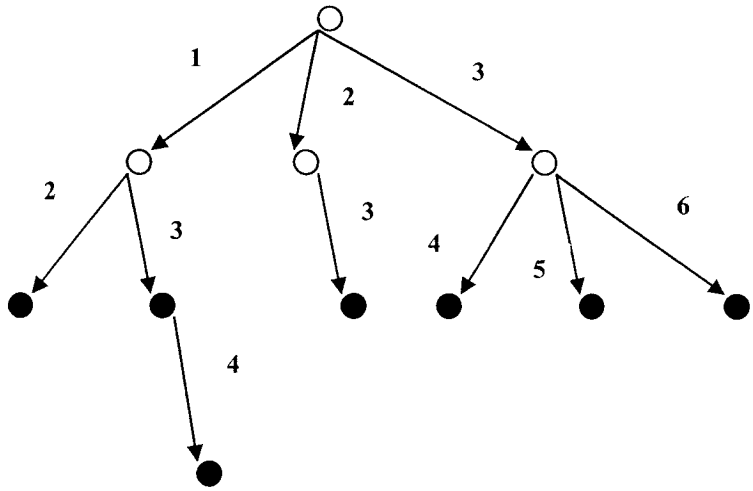


Figure 2.1.9: Calculation of Multicast Hops

that the time to transmit one flit from a node to its neighbors is one time unit, i.e. $L_f/B=1$, then the formula can be simplified as follows.

$$\textit{Network Latency}=D+L \qquad (2.1.10)$$

So, the network latency is the sum of the distance and the length of the message. In a multicast, the length (L) of the message is the same for all destination nodes. The multicast latency will be decided by the longest path to a destination in the multicast tree. Therefore, we have:

$$\textit{Multicast Time}=\max(l_1, l_2, \ldots, l_{k-1}, l_k)+L \qquad (2.1.11)$$

2.2 Review of Previous Studies

Many papers related to multicast in mesh-connected multicomputers have been published since the early 90s'. The following is a brief review of the history of the studies in this field.

Basic Theories and Early Studies

In 1993 Lin and Ni [1] introduced multicast communication in multicomputers in details and gave the formal definitions of the multicast problems. They further proved a series of results such as the NP-completeness of multicast problems, and defined the criteria for evaluating the performance of multicast routing. In addition they proposed several heuristic multicast algorithms in general such as the sorted *MP* algorithm, which we use as a guideline to design our algorithms.

Most of first generation multicomputers used store-and-forward switching and adopted the hypercube topology because it obtains a very short distance among any pair of nodes. In late 1980s, numerous papers [31, 32] were dedicated to multicast routing in multicomputers with hypercube topologies. But when the distance insensitive wormhole routing emerged as a more popular switching technology, hypercube was gradually replaced by low dimensional meshes.

In 1993 Ni and Mckinley conducted a very thorough survey [2] on wormhole routing technology. Performance evaluation of wormhole routed networks is discussed in details. Due to the fact that wormhole routing is particularly susceptible to deadlock, several approaches to ensure deadlock-free routing are proposed. Some of the techniques such as dimensional ordered routing (XY-routing) are adopted by our algorithms. Later in 1998,

Mohapatra further discussed various techniques for enhancing the performance and reliability of the wormhole routing [6]. These deadlock-preventing and fault-tolerant techniques [2, 6] can be easily incorporated into our algorithms upon implementation.

On Unicast-based Multicast

In early wormhole-routed multicomputers, hardware used to only support unicast. Hence, a multicast communication has to be done through a sequence of unicasts. Mckinley [5] presented two optimal efficient algorithms to implement multicast communication in wormhole-routed n-dimensional meshes which delivers a multicast message to m-1 destinations in $\lceil log_2 m \rceil$ message passing steps. In 1995 Robinson et al presented the version of the algorithms for wormhole-routed torus networks called U-torus [11], which also achieved the $\lceil log_2 m \rceil$ optimal result.

Unicast-based multicast proved not to be very efficient due to its low parallelism, lack of optimization on traffic, and too much software involvement. Hardware supported tree-based multicast was needed to improve the performance of multicomputers.

On Tree-based Multicast

In 2003 Liu [3] proposed two tree-based multicasting algorithms for mesh and torus networks: the *VH* algorithm with a time complexity of O(KD), and the *DIST* algorithm with a time complexity of O(KDN). Simulations showed that the *DIST* algorithm generates less traffic than *VH* algorithm, but at the price of much larger multicast time and computation complexity. *VH* algorithm, on the other hand, obtained very low computation complexity and optimal multicast time. But simulations indicated that it creates a very large amount of traffic. Harutyunyan later proposed the *MIN* algorithm

[22], which was designed to reduce the multicast traffic. Simulations proved that the *MIN* algorithm produces the least traffic among all the algorithms without significantly degrading the multicast time, but its computation complexity is higher than others.

Hence, each of these algorithms needs some improvement on one aspect or the other. Fortunately, due to the good properties of mesh topology, multicast routing that obtains optimal time is achievable at a low computation complexity. How to reduce the traffic as much as possible for the time optimal multicast algorithms becomes a new challenge in this field, which leads to the development of our new algorithms: the *DIAG* and *DDS*.

On Path-based Multicast

Tree-based multicast routing may seem to be efficient on both time and traffic. But simulations showed they only perform well in store-and-forward switched networks, not in wormhole routed networks. Lin proved in details in [7] that path-based multicast routing outperforms tree-based multicast routing in wormhole routed networks. With wormhole emerging as the most promising switching technology [7], efficient path-based multicast routing algorithms are also very desirable.

Lin and Ni proposed the dual-path and multi-path multicast routing algorithms for wormhole-routed multicomputers with 2D-mesh or hypercube topology [7]. Both algorithms used a Hamiltonian path to guide the routing of the message along as short paths as possible. The drawback of such algorithms is that they create large excessive traffic because routing has to observe the order of the Hamiltonian path. Hence, how to reduce the total traffic or the length of routing paths remains a challenge and leads to the development of our XY-path algorithm featuring two guiding paths for message routing.

Chapter 3

Proposed Multicast Algorithms

3.1 Modeling of the Problem

Following the convention, we model the mesh network in multicomputers as a graph, and the multicast problem as finding the optimal multicast tree in the host graph. We will introduce the model in details in the following sections.

3.1.1 Problem and the Model

In our model every node of the multicomputer usually consisting of a processor and a router is deemed as a vertex, and connections between pairs of nodes are deemed as edges (links) in the host graph. Depending on the underlying network topology, the deployment of the nodes and connectivity between them will result in very different host graphs. In our study, the mesh-connected multicomputer network is modeled as an n-dimensional mesh graph.

We can just imagine that the n-dimensional mesh is fit into the first (positive) section of an n-dimensional coordinate space with their origins overlapped. Each node of the mesh is located at a point which is identified by the node's corresponding n-coordinate vector, and each pair of neighboring nodes are connected with a line that represents the

link between them. Distance between neighboring nodes is one which stands for the weight of the edge.

The Problem

As we mentioned in Chapter 2, the two parameters *time* and *traffic* used to evaluate multicast contradict each other. It is hard to minimize both. So the common strategy is to minimize one parameter first and try to reduce the value of the other parameter as much as possible [3]. If an algorithm tries to obtain minimal traffic, the problem becomes a *Minimal Steiner Tree (MST)* or *Optimal Multicast Path (OMP)* problem which is NP-complete in general. If it minimizes the time first and tries to reduce the traffic as much as possible, the problem becomes *Optimal Multicast Tree (OMT)* problem which is NP-compete in general as well. Fortunately, *OMT* which is often reduced to shortest path problems is not very hard in a mesh graph. In our study, we choose the second approach to design efficient multicast routing algorithms for mesh-connected multicomputers. Hence, our problem becomes the *Optimal Multicast Tree (OMT)* problem in mesh graphs. Here is the formal definition.

Definition 3.1.1 (OMT Problem In Meshes) Given the host mesh network M, and a multicast from the source node u_0 to k destination nodes set $K=\{ u_1, ..., u_k \}$, find the *Optimal Multicast Tree, T(V,E)*, such that *a)* T is a subgraph of M; *b)* $K \subseteq V(T)$; *c)* $dT(u0, ui) = dM(u0, ui)$ for $1 \leq i \leq k$, (Shortest Path); *d)* $|E(T)|$ is as small as possible.

The Computational Model

In reality any node in the mesh could be the source node u_0 that sends a message to a set of arbitrary destination nodes. In order to simplify the mathematical model and simulations, we assume the source node is fixed at the origin of the mesh or torus.

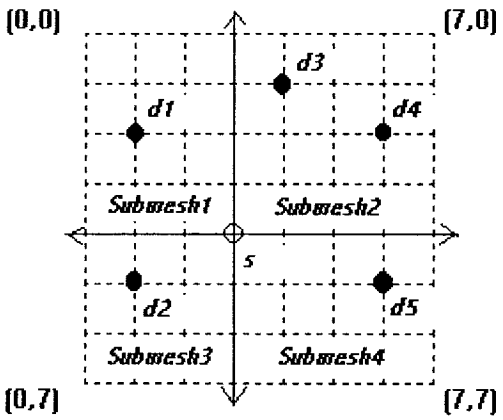


Figure 3.1.1: Model of Multicast in Mesh

The legitimacy of this assumption is obvious in the case of the symmetrical torus network. An arbitrary instance of multicast in tori can be reduced to an equivalent case, whose source node is located at the origin, by simply rotating some distance along each dimension. In the case of mesh, an arbitrary instance of multicast can also be reduced to several such multicasts in submeshes, which have the source node at their respective origins and a subset of the destination nodes. Consider the multicast in 2D mesh as an example illustrated in *Figure 3.1.1*. An arbitrary multicast is launched in the mesh from a source node $s(3,4)$ to a set of destination nodes $d_1(1,2)$, $d_2(1,5)$, $d_3(4,1)$, $d_4(6,2)$, $d_5(6,5)$. We can split the mesh into *4* submeshes by dividing it along the X and Y dimensions of node s, i.e. $submesh_1$, $submesh_2$, $submesh_3$, and $submesh_4$. Meanwhile, the original destination set was also divided into *4* subsets, one for each submesh. Now, assume that

the source node s is the origin of each submesh. Then the original multicast can be reduced to four multicasts from the origin s to its respective subset of destinations in each submesh. The computation complexity, value of performance evaluating parameters, and simulation results of the original multicast can be reflected by those in the four submeshes.

3.1.2 Definitions and Notations

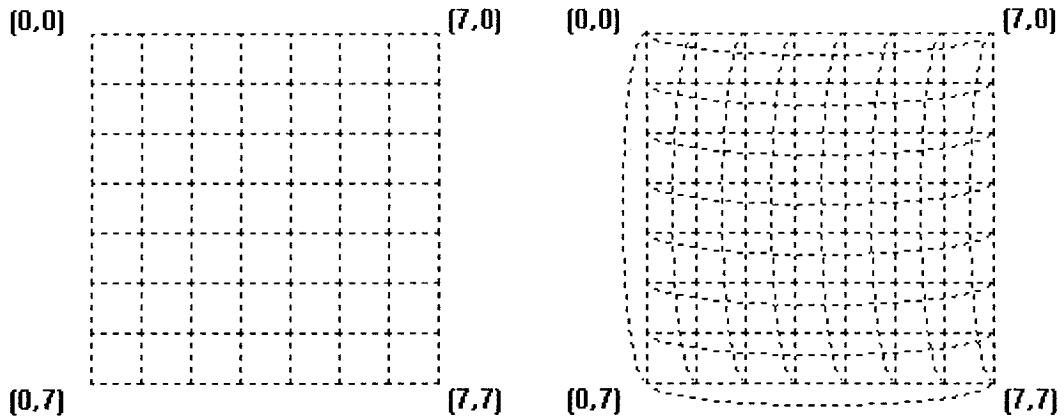
Definitions and notations of a graph, mesh and path in general were introduced in Chapter 2. Here we present more specific notations related to our study of multicast in 2D/3D meshes/tori.

On Mesh Network

In an n-dimensional mesh network, the distance between nodes $(x_0, x_1, ..., x_{n-1})$ and $(y_0, y_1, ..., y_{n-1})$ is $\sum| y_i - x_i |$, $0 \le i \le n-1$. The maximum degree of the node in the n-dimensional mesh is $2n$, and the diameter of the n-dimensional mesh is $\sum(k_i-1)$, $0 \le i \le n-1$.

A 2D mesh is usually denoted as $2DM(m \times n)$. Each node is identified by coordinate (x, y), where $0 \le x \le m-1$ and $0 \le y \le n-1$. The total number of nodes in $2DM$ is $N = m \times n$. Every node in $2DM$ has at least 2 and at most 4 neighbors. And the diameter of $2DM$ is $m + n - 2$. *Figure 3.1.2 (a)* shows an 8×8 2D mesh.

A 3D mesh network is usually denoted as $3DM(m \times n \times p)$, which contains $N = m \times n \times p$ nodes. Each node is identified by coordinate (x, y, z), where $0 \le x \le m-1$, $0 \le y \le n-1$, and $0 \le z \le p-1$. The node of 3D mesh has maximum degree 6, and the diameter of $3DM$ is $m + n + p - 3$.



(a) 8 × 8 2D mesh (b) 8 × 8 2D torus

Figure 3.1.2: 2D Mesh and Torus

Definition 3.1.2 (Zone) A zone Z of an n-dimensional mesh $M(k_0 \times k_1 \times \ldots \times k_{n-1})$, where $k_i \geq 2$ $(0 \leq i \leq n-1)$, is a submesh within M. Assume $x(x_0, x_1, \ldots, x_{n-1})$ and $y(y_0, y_1, \ldots, y_{n-1})$ are the two end points of the diagonal of the zone, then for any node in the zone $z(z_0, z_1, \ldots, z_{n-1})$, z_i is between x_i and y_i $(0 \leq i \leq n-1)$. We denote a zone with the two diagonal nodes x and y as $\{x \Leftrightarrow y\}$.

Definition 3.1.3 (Multicast Zone) In an n-dimensional mesh or torus $M(k_0 \times k_1 \times \ldots \times k_{n-1})$, where $k_i \geq 2$ $(0 \leq i \leq n-1)$, if a multicast sends a message from the source node s to a set of destination nodes $D=\{ d_1, d_2, \ldots, d_k \}$. The multicast zone is the minimal zone that covers the source node s and all the destination nodes in D.

On Torus Network

In general a torus network is a mesh with wraparound links that connect the two end nodes ($x_0, ..., x_{i-1}, k_i-1, x_{i+1}, ..., x_{n-1}$) and ($x_0, ..., x_{i-1}, 0, x_{i+1}, ..., x_{n-1}$) in the ith dimension. All of the links are bi-directional. A transfer is in the positive direction if it is from node ($x_0, ..., x_{i-1}, x_i, x_{i+1}, ..., x_{n-1}$) to node ($x_0, ..., x_{i-1}, (x_i+1) \bmod k_i, x_{i+1}, ..., x_{n-1}$) and vice verse it is called negative direction.

In an n-dimensional torus network, the distance between node ($x_0, x_1, ..., x_{n-1}$) and node ($y_0, y_1, ..., y_{n-1}$) is $\Sigma min(|y_i - x_i|, k_i-|y_i - x_i|), 0 \le i \le n-1$. The degree of a node in the n-dimensional torus is $2n$ and its diameter is $\Sigma \lfloor k_i/2 \rfloor, 0 \le i \le n-1$. We will use the same set of notations of 2D/3D meshes for 2D/3D tori.

On Multicast

Definition 3.1.4 (Multicast in 2DM/2DT) In a 2D mesh/torus network $M(m \times n)$, given a source node s$(0,0)$ and a set of destination nodes $D=\{ (x_1, y_1), (x_2, y_2), ..., (x_i, y_i) \}$, where $0 \le x_i \le m-1$ and $0 \le y_i \le n-1$. Multicast sends a message from s to all nodes in D. Define: $X_{max} = max\{x_1, ..., x_i\}$; $Y_{max} = max\{y_1, ..., y_i\}$; The total number of nodes in the mesh $N = m \times n$; The diagonal node of s in the multicast zone $d = (X_{max}, Y_{max})$; $D_{max} = max\{D_1, ..., D_i\}$ where D_i is the distance between the source and the destination (x_i, y_i).

Definition 3.1.5 (Multicast in 3DM/3DT) In a 3D mesh/torus network $M(m \times n \times p)$, given a source node s$(0,0,0)$ and a set of destination nodes $D=\{ (x_1, y_1, z_1), (x_2, y_2, z_2), ..., (x_i, y_i, z_i) \}$, where $0 \le x_i \le m-1$, $0 \le y_i \le n-1$ and $0 \le z_i \le p-1$. Multicast sends a message from s to all nodes in D. Define: $X_{max} = max\{x_1, ..., x_i\}$; $Y_{max} = max\{y_1, ..., y_i\}$; $Z_{max} = max\{z_1, ..., z_i\}$; The total number of nodes in the mesh $N = m \times n \times p$; The diagonal node

of s in the multicast zone $d = (X_{max}, Y_{max}, Z_{max})$; $D_{max} = max\{D_1, ..., D_i\}$ where D_i is the distance between s and the destination (x_i, y_i, z_i).

Definition 3.1.6 (Multicast in n-D Mesh) In a n-dimensional mesh network $M(k_1 \times k_2 \times ... \times k_n)$, given a source node $s(0, 0, ..., 0)$ and a set of destination nodes $D=\{d_1, d_2, ..., d_i\}$ Multicast sends a message from s to all nodes in D.

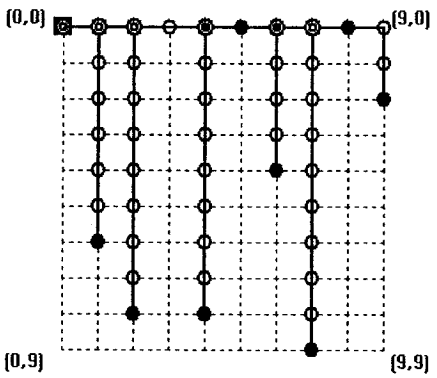


Figure 3.1.3: Legend for a Multicast Tree

According to the kind of role they play, nodes in a multicast tree can be classified into the following categories: *Source node, Destination node, Leaf node, Intermediate node, Replicate node, and Passerby node.* Any node between the source node and a leaf node is called *Intermediate node.* If an *Intermediate node* has to forward the message to more than one node, it is called *Replicate node.* An *Intermediate node* can also be a *Destination node.* If an *Intermediate node* is neither a *Destination node* nor a *Replicate node*, it is called a *Passerby node.*

Figure 3.1.3 illustrates the legend for different type of nodes in a multicast tree. A double circle in a filled square is the source node. An empty double circle is a non-

destination replicate node and a filled double circle is a destination replicate node. An empty circle is a passerby node. A filled circle is a non-replicate destination node.

3.1.3 Solution to Multicast in Torus

Multicast problems in a torus are very similar to those in a mesh. We present here in details how the multicast problem in tori can be reduced to those in meshes.

2D Torus Network

In the multicast given by *Definition 3.1.4,* while delivery of the message to the destinations in *2DM* always follows the positive directions, that of *2DT* may follow in either positive or negative direction, depending on the position of these nodes. The *2DT* graph can be divided into four zones shown below and each node (x_i, y_i) belongs to one of them.

$zone_1$: $\{ (0, 0) \Leftrightarrow (\lceil m/2 \rceil - 1, \lceil n/2 \rceil - 1) \}$

$zone_2$: $\{ (m-1, 0) \Leftrightarrow (\lceil m/2 \rceil, \lceil n/2 \rceil - 1) \}$

$zone_3$: $\{ (0, n-1) \Leftrightarrow (\lceil m/2 \rceil - 1, \lceil n/2 \rceil) \}$

$zone_4$: $\{ (m-1, n-1) \Leftrightarrow (\lceil m/2 \rceil, \lceil n/2 \rceil) \}$

After dividing the *2DT* into four zones, the multicast in *2DT* becomes multicast in four submeshes. In $zone_1$, message is routed from source node $(0, 0)$ to all the destinations in it. In $zone_2$, node $(m-1, 0)$ first gets the message from node $(0, 0)$ through wraparound links and then is treated as a source node in its zone. Similarly, in $zone_3$, node $(0, n-1)$ is treated as a source node. And node $(m-1, n-1)$ is treated as the source node in $zone_4$. *Figure 3.1.4* illustrates how it works in an 8×8 2D mesh.

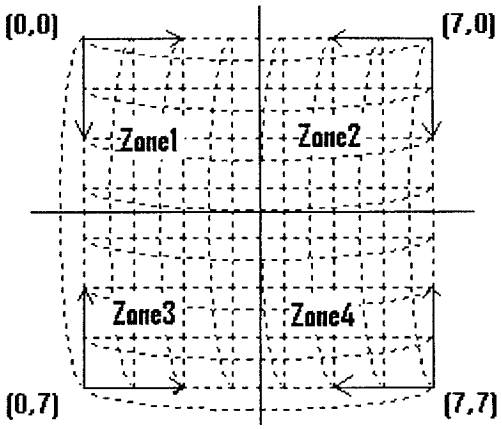


Figure 3.1.4: Division of 2D Torus Network

3D Torus Network

Multicast in 3D torus can be solved in a similar way as in the 2D torus. Unlike the *2DT* network, which consists of four 2D submeshes, the *3DT* network can be divided into eight zones as shown below and each node (x_i, y_i, z_i) belongs to one of them. Hence, multicast in the 3D torus given by *Definition 3.1.5* is reduced to multicasts in eight 3D submeshes.

$zone_1$: $\{ (0, 0, 0) \Leftrightarrow (\lceil m/2 \rceil - 1, \lceil n/2 \rceil - 1, \lceil p/2 \rceil - 1) \}$

$zone_2$: $\{ (m-1, 0, 0) \Leftrightarrow (\lceil m/2 \rceil, \lceil n/2 \rceil - 1, \lceil p/2 \rceil - 1) \}$

$zone_3$: $\{ (m-1, 0, p-1) \Leftrightarrow (\lceil m/2 \rceil, \lceil n/2 \rceil - 1, \lceil p/2 \rceil) \}$

$zone_4$: $\{ (0, 0, p-1) \Leftrightarrow (\lceil m/2 \rceil - 1, \lceil n/2 \rceil - 1, \lceil p/2 \rceil) \}$

$zone_5$: $\{ (0, n-1, 0) \Leftrightarrow (\lceil m/2 \rceil - 1, \lceil n/2 \rceil, \lceil p/2 \rceil - 1) \}$

$zone_6$: $\{ (m-1, n-1, 0) \Leftrightarrow (\lceil m/2 \rceil, \lceil n/2 \rceil, \lceil p/2 \rceil - 1) \}$

$zone_7$: $\{ (m-1, n-1, p-1) \Leftrightarrow (\lceil m/2 \rceil, \lceil n/2 \rceil, \lceil p/2 \rceil) \}$

$zone_8$: $\{ (0, n-1, p-1) \Leftrightarrow (\lceil m/2 \rceil - 1, \lceil n/2 \rceil, \lceil p/2 \rceil) \}$

For n-dimensional tori, in the same way we deal with 2D and 3D tori, we can transform a multicast in an n-dimensional torus into multicasts in 2^n n-dimensional submeshes by dividing the original mesh at the middle of each dimension.

3.2 Tree-based Multicast Algorithms

An efficient multicast routing algorithm should transmit the source message to each destination node with as small transmission time and less communication channels (links) as possible. For a set of given destination nodes, the tree-based multicast routing delivers the message along common paths as much as possible, then branches the message to each destination node which results in a tree-like routes. One multicast can be accomplished through many multicast trees. Our goal is to find the multicast tree that is optimal on both time and traffic. However, the problem is NP-hard, and only heuristics are realistic solutions. Here, we use a pro-time strategy to design the tree-based multicast algorithms that minimize the time first and try to reduce the traffic as much as possible.

Network technologies such as switching techniques significantly affect the performance of multicast algorithms. Research indicated that the general tree-based multicast is good for store-and-forward switching but not for wormhole routing. On the other hand, the path-based multicast in which all destinations on the path can receive and absorb the message worm simultaneously works well in wormhole routed network [1]. We will study on the path-based multicast routing algorithms later.

Here, we first propose two tree-based shortest path multicast algorithms for store-and-forward switched mesh networks: the DIAG algorithm and the DDS algorithm, which obtain near optimal multicast time while keep the multicast traffic as low as possible.

3.2.1 DIAG Multicast Algorithm

DIAG is designed to reduce the large amount of multicast traffic of the previous tree-based time optimal VH algorithm (See details below). The algorithm first finds a path that approximates the diagonal of the multicast zone from the source node to the opposite end. It then uses the diagonal as the main path or stem of the multicast tree and connects all the other destinations through the shortest paths possible. Thus, in the multicast tree, the path between the source node and each destination node is a shortest one which results in near optimal time and modest amount of traffic.

In the following sections, we will discuss in details the motivation and heuristics of designing this algorithm, then present the formal description and analysis of the algorithm in 2D/3D meshes and tori.

3.2.1.1 Motivation of Designing DIAG

Liu [3] designed a time-optimal multicast tree algorithm called VH algorithm in which the message is routed strictly in dimensional order to each destination node. So, the message is transmitted from the source node first along the lowest dimension, then turns to the next lowest dimension, the process goes on until the message arrives at the destination. Thus, we can find the routing path for every destination and then construct the multicast tree by merging the common parts of all the routing paths. *Figure 3.2.1.1* shows the multicast tree constructed through VH algorithm for a multicast from the source node *(0, 0)* to destinations *(2, 3), (3, 5), (4, 6)* and *(6, 9)* in a 2D mesh.

VH algorithm delivers the message to each destination along a shortest path obtaining the optimal multicast time. The upper bound of multicast time of VH algorithm in 2D mesh is $D_{max}+1$, where D_{max} is the distance from the source to the farthest destination node and the minimum possible multicast time in theory. VH algorithm also has a very low computational complexity as low as O(*KD*) compared to O(*KDN*) of some other algorithms such as the MIN algorithm.

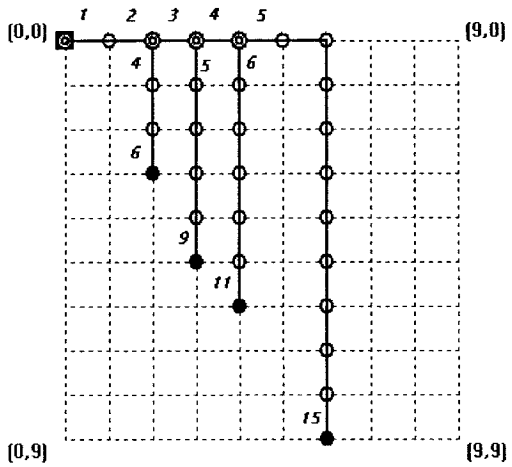


Figure 3.2.1.1: Example of VH Multicast Tree in 2D Mesh

Unfortunately, VH algorithm pays little attention to reduce the multicast traffic. As a result, VH algorithm generates a very large amount of traffic. *Figure 3.2.1.2* [22] shows that the additional traffic of VH algorithm almost doubles that of the pro-traffic MIN algorithm.

The question is: Is the large amount of traffic created by VH algorithm necessary for obtaining the shortest paths? After a close look at the multicast example, we find out why VH generates so much traffic. In fact, if we send a copy of the message to node *(6, 9)* from

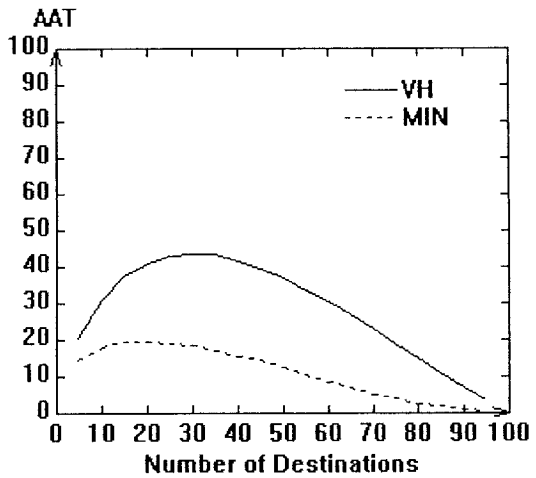


Figure 3.2.1.2: AAT of VH versus MIN

node *(4, 6)* instead of from the far away node *(4, 0)*, much traffic will be saved. So, the excessive traffic is most likely a result of the rigid dimensional order. If we ignore the dimensional order, the message can be routed along the tree shown in *Figure 3.2.1.3*, which also obtains minimum multicast time with only *15* links of traffic compared to *29* links in the VH multicast tree.

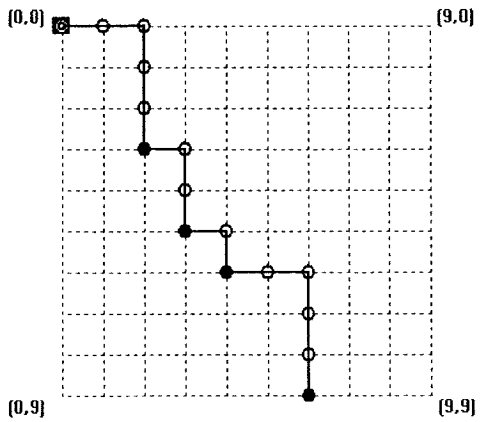


Figure 3.2.1.3: The Diagonal Routing

Our question now becomes: can we develop a new algorithm that can obtain near optimal multicast time and reduce the large amount of excessive traffic of VH algorithm. DIAG (diagonal) multicast algorithm is designed just for this purpose. Next, we will discuss the heuristics of DIAG algorithm.

3.2.1.2 Heuristics of DIAG Algorithm

As DIAG algorithm is designed to reduce the traffic of VH algorithm, the main strategy will be to eliminate bad scenarios in VH algorithm which create large excessive traffic.

One major reason why VH enforces the dimensional order is to ensure that messages reach each destination along a shortest path. However, the dimension-ordered routing is not a prerequisite for a shortest path. *Figure 3.2.1.4* shows that the dimension-ordered routing results in route *(1)*. But if we take route *(2),* i.e. alternatively routing along X and Y dimensions, we also get a shortest path.

One of the bad scenarios for VH routing in 2D mesh is when the destinations are located at the bottom of each column as shown in *Figure 3.2.1.5*. The message has to be routed from top of a column all the way through to the bottom only to one or two destinations, which is not efficient on traffic. This is mainly caused by the strict dimensional order in routing. As a result, the major routing path (X dimension route) is located at the top which is far away from most of the destinations at the bottom.

A natural solution to such a bad case is to let the X dimension routing path intercept the columns at the middle instead of at the top, which can save almost half of the total traffic. In order to balance the two dimensions, we may need to apply the same trick to Y

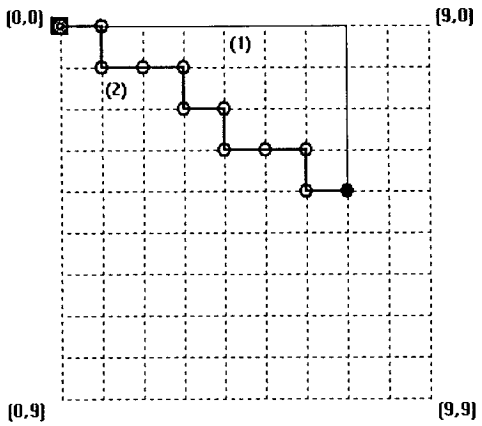


Figure 3.2.1.4: Example of Non-dimensional Shortest Path

dimension. Hence, the major routing path needs to travel along both dimensions concurrently, instead of in just one dimension. To achieve this, we can let the major routing path advance along either dimension alternatively at small steps (instead of to the maximum coordinate at once) until both dimensions reach their maximum coordinates.

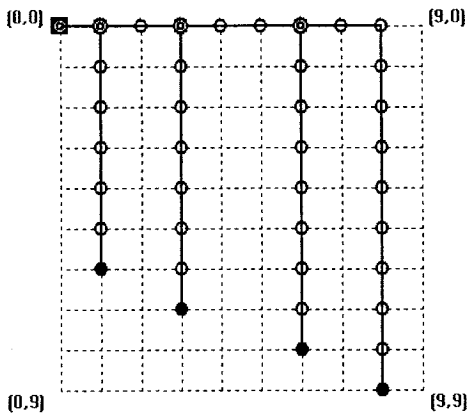


Figure 3.2.1.5: Example of Bad Case VH Multicast Tree

To keep the path balanced in both dimensions, the step length for each dimension should be proportional to the maximum coordinate in the respective dimension. Thus, the

resulted path approximates the diagonal line of the multicast zone. If we route the message from some point at the diagonal major path to each destination node, long distance travel along one dimension to just one or two destinations can be avoided.

Based on this idea, we propose a so-called DIAG (Diagonal) multicast algorithm. Here are some principles or heuristics that we use to design it.

- To maintain the minimized multicast time obtained by VH algorithm, we demand that the message be routed from the source to each destination along a shortest path, in which case, the total traffic will not be very bad either.
- Since the major path (stem of the tree) is the diagonal of the multicasting zone, it is likely that most of the destinations will be around the diagonal not too far away.
- If we first send the message to the destination nodes closer to the source node, then they can forward the message to farther destinations with just a little extra distance.
- If we connect a destination to the existing multicast tree through a shortest path, the new added traffic is likely small, which will result in a small total traffic.

The main procedures of DIAG multicast routing algorithm are roughly described as follows.

Step 1. Find the multicast zone and the diagonal node d which has the maximum coordinate in every dimension of all destination nodes.

Step 2. Form a shortest path that approximates the diagonal line between the source node s and the diagonal node d.

Step 3. Sort all the destination nodes according to their distance to the source node and put them in a queue.

Step 4. Remove the first node q from the queue and connect it to the closest node c in the multicast tree and within the zone $\{ s \Leftrightarrow q \}$.

Step 5. Repeat *step 4* until the queue is empty

3.2.1.3 DIAG Algorithm in 2D Mesh

This section presents the formal description and analysis of the DIAG multicast algorithm in 2D meshes under the multicast defined by *Definition 3.1.4.* We divide the algorithm into two parts. Part one is to find the diagonal path and called DIAG multicast preparation algorithm. Part two is to construct the complete multicast tree and called DIAG multicast routing algorithm.

Algorithm 3.2.1.1(Multicast preparation algorithm of DIAG in 2DM)

Let us denote the diagonal path from the source node s to its diagonal node d as DP. Denote the diagonal line from s to d as sd. $\sigma_x(u)$ denotes the x coordinate of node u, and $\sigma_y(u)$ denotes the y coordinate of node u. $D(u,l)$ denotes the distance from node u to line l.

Input: M, D, s
Output: DP
Variable: u, u', u'' (temporary nodes)

(1) $DP=\{ s \}$; $u=s$; Find the diagonal node d
(2) $\sigma_x(u')= \sigma_x(u)+1$; $\sigma_y(u')= \sigma_y(u)$
$\sigma_x(u'')= \sigma_x(u)$; $\sigma_y(u'')= \sigma_y(u)+1$
IF ($D(u', sd) \leq D(u'', sd)$) THEN $u=u'$
ELSE $u=u''$
$DP=DP+\{ u \}$
(3) Repeat *Step (2)* until $u=d$

Proposition 3.2.1.1 The time complexity of the preparation algorithm of DIAG multicast in 2D meshes is O(m+n).

Proof. The preparation algorithm is to find a path which approximates the diagonal line from the source node s to the diagonal node d of the multicast zone as much as possible. The process starts from the source node s and keeps routing from the current node toward node d to one of its neighboring nodes which is closest to the diagonal line. The result is a shortest path from the source node s to the diagonal node d. The length of the path is the times that this process is repeated, i.e. the time complexity, and less than the diameter of the mesh which is m+n-2.

Algorithm 3.2.1.2 (DIAG multicast routing algorithm in 2DM)

Let *DP* denote the diagonal path from the source node *s(0, 0)* to its diagonal node *d*. The multicast tree constructed through the routing algorithm is denoted as *MT*

Input: *M, D, s, DP*
Output: *MT*
Variables: *u, c* (temporary nodes)

(1) Sort the destination nodes in *D* according to their distance to the source node *s*
(2) *MT*=*DP* (Initialize the multicast tree with the diagonal path)
(3) - *u*=first node in *D*
    - Find a node *c* in *MT* and in zone $\{s \Leftrightarrow u\}$, that is closest to node *u*
    - Add node *u* to *MT* through a shortest path from *c* through *XY* routing
    - *D*=*D* - $\{u\}$
(4) Repeat *Step (3)* until $D=\Phi$
(5) Cut the tail part of *DP* which has neither destination nodes nor replicate nodes

Proposition 3.2.1.2 The time complexity of the routing algorithm of DIAG multicast in 2D meshes is O(KN) where K is the total number of destinations and N is the total number of nodes in the mesh.

Proof. We know that *step (1)* which sorts the destinations will take O($KlogK$), *step (3)* is actually to add the K destinations to the existing multicasting tree. For each destination, it needs to traverse the multicast tree which contains no more than N nodes. So *step (3)* will take O(KN). All together, it will take O($KN+KlogK$) which is at the same order with O(KN).

Proposition 3.2.1.3 In the multicast tree constructed by DIAG routing algorithm, the path from the source node s to any destination node is a shortest one.

Proof. The diagonal path found by the preparation algorithm is a shortest path from source node s to the diagonal node d. Since the diagonal path is used as the stem of the initial multicast tree, the path between the source node and every node on the current tree is shortest. Any new destination d' is connected to a node c on the shortest path tree and in zone $\{s \Leftrightarrow d'\}$. Routing between d' and c is dimensional-ordered which results in the shortest path between them. Path sc and cd' are both shortest and in zone $\{s \Leftrightarrow d'\}$. So, path sd' is also shortest.

Proposition 3.2.1.4 DIAG multicast routing in mesh or torus networks is deadlock free.

Proof. First, since the message is routed along a multicast tree, it is obvious there is no deadlock within a single multicast. Second, for several concurrent multicasts, DIAG algorithm uses dimensional-ordered *XY* routing to transmit message between any two nodes, which will guarantee no deadlock cycle within the mesh or torus networks [2].

Example 3.2.1.1 In a 2D mesh network $M(8 \times 8)$, construct the multicast tree using DIAG algorithm for a multicast that sends a message from the source node *s(0, 0)* to a set of destination nodes $D=\{ (0, 2), (3, 0), (4, 0), (4, 6), (6, 6), (7, 4) \}$.

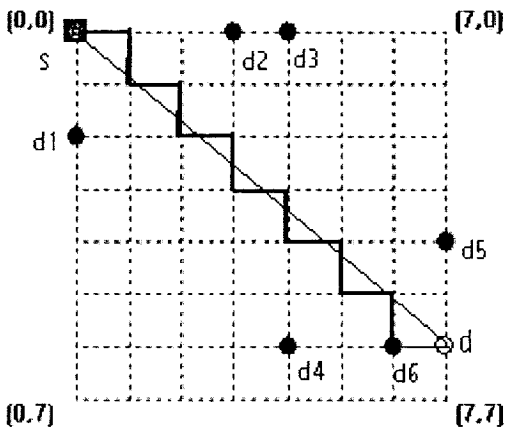


Figure 3.2.1.6: Example of Diagonal Path of DIAG Multicast

In this example, the multicast zone is between the source node *s(0, 0)* and the diagonal node *d(7, 6)* as shown in *Figure 3.2.1.6.* We now run the preparation algorithm to find the diagonal path between *s* and *d.* Starting from the source node *s(0, 0),* we noticed that node *(1, 0)* is the one closer to the diagonal line *sd* among the two neighbors of node *s*, so the diagonal path routed from *s* to node *(1, 0).* Then, we check node *(1, 0)'s* neighboring nodes toward the diagonal node *d* and find node *(1, 1)* is closer to the diagonal line *sd* than the

other node *(2, 0)*, and the path routed to node *(1, 1)*. The routing process proceeds in this way until the path reaches the diagonal node *d*. The final diagonal path between *s* and *d* is illustrated with dark zigzag line in *Figure 3.2.1.6*.

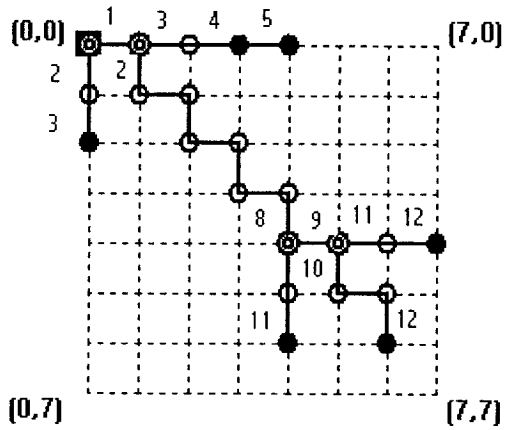


Figure 3.2.1.7: Example of DIAG Multicast in 8×8 Mesh

Now, we need to use the diagonal path as the stem to construct the multicast tree. First, we sort all the destination nodes according to their distances to the source node, and get d_1, d_2, ..., d_6 as indicated in *Figure 3.2.1.6*. Then, we connect each destination node to the tree through dimension-ordered *XY* routing as illustrated in *Figure 3.2.1.7*. At the beginning, the multicast tree consists of only the diagonal path. For the first destination node $d_1(0, 2)$, the closest node in the already existing tree and within zone $\{s \Leftrightarrow d_1\}$ is $s(0, 0)$, so we connect d_1 to *s* through *XY* routing. Likewise, we connect $d_2(3, 0)$ to node $(1, 0)$, $d_3(4, 0)$ to $d_2(3, 0)$, d_4 to node $(4, 4)$, and $d_5(7, 4)$ to node $(5, 4)$ through *XY* routing. Destination node $d_6(6, 6)$ is already on the diagonal path. Finally, we need to cut off the unused part of the diagonal path after $d_6(6, 6)$. Here are the steps of constructing the tree: $s(0, 0) \Rightarrow d(7, 6)$

(along the diagonal path), $s(0, 0) \Rightarrow d_1(0, 2)$, $(1, 0) \Rightarrow d_2(3, 0)$, $d_2(3, 0) \Rightarrow d_3(4, 0)$, $(4, 4) \Rightarrow d_4(4, 6)$, $(5, 4) \Rightarrow d_5(7, 4)$.

The completed multicast tree is shown in *Figure 3.2.1.7*. The multicast traffic is *21* links, and the multicast time is *12* hops. The same example using VH algorithm for 2D mesh will result in *25* links and *13* hops (*Figure 3.2.1.8*), which shows that DIAG generates less traffic than VH but does not sacrifice the multicast time.

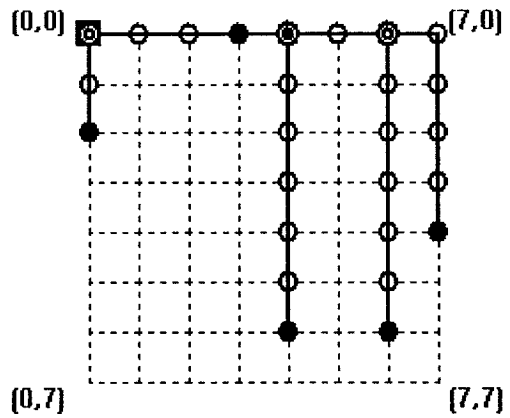


Figure 3.2.1.8: Example of VH Multicast in 8×8 Mesh

Proposition 3.2.1.5 The bound of DIAG multicast traffic in 2D meshes is $D_{max} \leq Traffic \leq \sum |x_i+y_i|$, $1 \leq i \leq K$.

Proof. It is obvious that the total traffic can be no less than the distance from source node to the farthest destination node which is D_{max}. This best case result of multicast traffic is obtained when all the destination nodes are located on a shortest path from the source to the farthest destination node. Since the path from the source $s(0,0)$ to each destination node (x_i, y_i) in the multicast tree is a shortest path and less than $|x_i+y_i|$, the

total links of the tree is less than the sum of all these shortest paths, which is $\sum| x_i+y_i |$, $1 \leq i \leq K$.

Proposition 3.2.1.6 The bound of DIAG multicast time in 2D meshes is $D_{max} \leq Time \leq D_{max}+K-1$.

Proof. In all-port architectures, a message can be sent from a node to all its neighbors simultaneously, and the number of hops needed to transmit a message to a destination node is simply the distance to the source node. Hence, the multicast time in any mesh networks is always D_{max}, which is the distance from the source node to the farthest destination node.

In one-port node architectures, the message can only be sent to one neighbor at a time (hop). If a node needs to send the message to several neighbors, it does so sequentially which results in delay hops. Assume that at a branching node the message is always sent to a neighbor on the diagonal path (stem) first, if there is one, and then the other neighbors. Like in all-port architecture, the multicast time can be no less D_{max}. This best case multicast time is obtained when the path from the source node to the farthest destination node always takes the direction with higher transmission priority at a branching node. The worst case though is that the path from the source node to the farthest destination node takes the direction with lower transmission priority at branching nodes, which results in one delay hop at each branching node. For a multicast with K destination nodes, the branching nodes on the path to a destination node can be no more

than $K-1$. Consequently, the delay hops for the farthest destination node is less than $K-1$ and the multicast time is less than $D_{max}+K-1$.

3.2.1.4 DIAG Algorithm in 2D Torus

This section discusses how to apply the DIAG algorithm to solve the multicast problems in 2D tori, and give the formal description and analysis of the algorithm.

Algorithm 3.2.1.3 (DIAG algorithm in 2D tori)

Input: T, D, s
Output: MT (Multicast Tree)

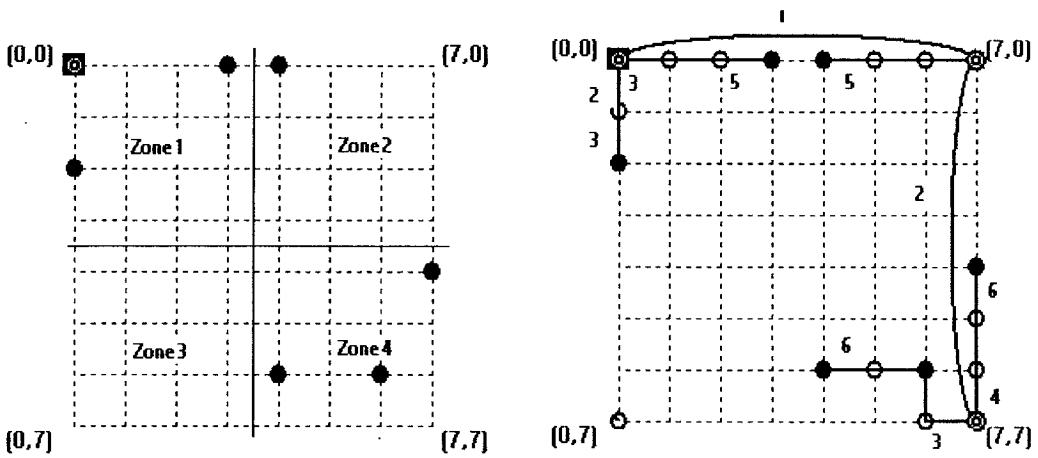
(1) Divide the torus T into four submeshes M_1, M_2, M_3 and M_4, each of which is defined by a zone as following:

M_1: $\{ (0, 0) \Leftrightarrow (\lceil m/2 \rceil-1, \lceil n/2 \rceil-1) \}$
M_2: $\{ (m-1, 0) \Leftrightarrow (\lceil m/2 \rceil, \lceil n/2 \rceil-1) \}$
M_3: $\{ (0, n-1) \Leftrightarrow (\lceil m/2 \rceil-1, \lceil n/2 \rceil) \}$
M_4: $\{ (m-1, n-1) \Leftrightarrow (\lceil m/2 \rceil, \lceil n/2 \rceil) \}$

(2) Partition the destination set D into four subsets D_1, D_2, D_3 and D_4 which contain the destination nodes in submesh M_1, M_2, M_3 and M_4 respectively
(3) Assume $s_1(0, 0)$ as the source node and origin of submesh M_1, D_1 as the set of destination nodes, apply DIAG algorithm in submesh M_1 and get the multicast subtree MT_1
(4) Assume $s_2(m-1, 0)$ as the source node and origin of submesh M_2, D_2 as the set of destination nodes, apply DIAG algorithm in submesh M_2 and get the multicast subtree MT_2
(5) Assume $s_3(0, n-1)$ as the source node and origin of submesh M_3, D_3 as the set of destination nodes, apply DIAG algorithm in submesh M_3 and get the multicast subtree MT_3
(6) Assume $s_4(m-1, n-1)$ as the source node and origin of submesh M_4, D_4 as the set of destination nodes, apply DIAG algorithm in submesh M_4 and get the multicast subtree MT_4
(7) Assemble the multicast subtrees MT_1, MT_2, MT_3 and MT_4 into the overall multucast tree MT that rooted at the source node $s(0, 0)$ by connecting s_2, s_3 to s and connecting s_4 to s_2

Proposition 3.2.1.7 The time complexity of DIAG algorithm in 2D tori is O(*KN*) where *N* is the total number of nodes in the torus and *K* is the total number of destinations.

Proof. It is obvious that *step (1)* and *(2)* will take O(*K*). *Step (3)~step (6)* is actually to execute the DIAG algorithm in a mesh of size (*m/2*) × (*n/2*) with less than *K* destinations. So each step will take O(*Kmn/4*)=O(*KN/4*), all together the four steps will take O(*KN*). *Step (7)* takes O(*1*). So the total time complexity will be O(*K+KN+1*) which is at the same order with O(*KN*).



(a) Distribution of destinations (b) Multicast tree

Figure 3.2.1.9: Example of DIAG Multicast in 8×8 Torus

We use the same multicast in *Example 3.2.1.1* to show how the DIAG algorithm in 2D tori works. As illustrated in *Figure 3.2.1.9 (a),* the torus is first divided into four submeshes at the middle of each dimension: $zone_1$, $zone_2$, $zone_3$ and $zone_4$. The original destination set is divided into four subsets respectively. Second, we treat node *(0, 0)*, node *(7, 0)*, node *(0, 7)* and node *(7, 7)* as the origin and source node of $zone_1$, $zone_2$, $zone_3$ and

zone$_4$, and then apply the DIAG algorithm for 2D mesh in each zone with the respective destination nodes. Thus, each zone will have its own subtree (consisting of the straight links in the figure), and we can connect each subtree to the original source node *s(0, 0)* through wraparound links (arches in the figure). Here are the exact steps to construct the multicast tree. Subtree of *zone*$_1$: *(0, 0) ⇒ (0, 2), (0, 0) ⇒ (3, 0)*; Subtree of *zone*$_2$: *(7, 0) ⇒ (4, 0)*; Subtree of *zone*$_4$: *(7, 7) ⇒ (6, 6), (7, 7) ⇒ (7, 4), (6, 6) ⇒ (4, 6)*; Combine the subtrees: *(0, 0) ⇒ (7, 0), (7, 0) ⇒ (7, 7)*.

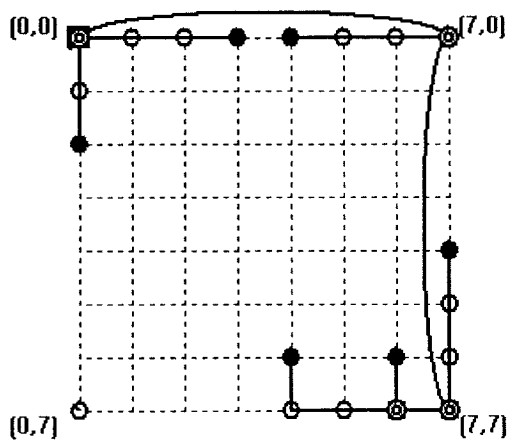


Figure 3.2.1.10: Example of VH Multicast in 8×8 Torus

The completed multicast tree is shown in *Figure 3.2.19 (b)*. The total multicast traffic is *17* links. If we assume the wraparound links have a higher transmission priority than other links, then the multicast time is *6* hops. Compared with DIAG for 2D meshes in the same example, the multicast time is reduced significantly by a half from *12* to *6*, but the traffic is not reduced much, only *4* links. Compared with VH for 2D tori in the same example (*Figure 3.2.1.10*), the multicast time is the same, and there is only one link difference on total traffic. So the traffic reduction effect of DIAG over VH in 2D tori is

not as obvious as in 2D meshes. This is probably due to the fact that the 2D torus is divided into *4* smaller meshes.

Definition 3.2.1.1 The 2D torus is divided into four roughly equal submeshes, each of which has its own origin (source) node and destination set. K_i is the number of destinations in $zone_i$ and D_{maxi} is the maximum distance from a destination in $zone_i$ to its origin ($1 \le i \le 4$). Then, we have:

$K = K_1 + K_2 + K_3 + K_4$

$D_{max} = max\{D_{max1}, D_{max2}, D_{max3}, D_{max4}\}$

Proposition 3.2.1.8 The bound of multicast time of DIAG in 2D tori is $D_{max} \le Time \le D_{max}+K+1$.

Proof. The lower bound of multicast time in $zone_i$ is D_{maxi}. If $D_{max}=D_{max1}$, there are no wraparound links for $zone_1$, so D_{max} will also be the lower bound of the multicast time in the whole torus. The upper bound of the multicast time in $zone_i$ is $D_{maxi}+K_i-1$, the multicast time for the whole torus also includes the number of wraparound links which is at most two of $zone_4$. It is not hard to see that when $D_{max}=D_{max4}$ and $K=K_4$, the multicast time obtains the maximum value: $D_{max4}+K_4-1+2$, *i.e.* $D_{max}+K+1$.

Proposition 3.2.1.9 The bound of the multicast traffic of DIAG in 2D tori is reduced to those of DIAG in its four submeshes.

3.2.1.5 DIAG Algorithm in 3D Mesh

In this section, we will give the formal description and analysis of the DIAG multicast algorithm in 3D meshes under the multicast defined by *Definition 3.1.5.* The algorithm is divided into two parts. Part one is to find the diagonal path, and called DIAG multicast preparation algorithm. Part two is to construct the complete multicast tree, and called DIAG multicast routing algorithm.

Algorithm 3.2.1.4(Multicast preparation algorithm of DIAG in 3DM)

DP denotes the diagonal path from the source node *s* to its diagonal node *d*. *sd* denotes the diagonal line from *s* to *d*. $\sigma_x(u)$ denotes the *x* coordinate of node *u*, $\sigma_y(u)$ denotes the *y* coordinate of node *u* and $\sigma_z(u)$ denotes the *z* coordinate of node *u*. $D(u,l)$ denotes the distance from node *u* to line *l*.

Input: *M, D, s*
Output: *DP*
Variable: u, u_x, u_y, u_z (temporary nodes)
D_{min} (minimal distance from nodes to *sd*)

(1) $DP=\{s\}$; $u=s$; Find the diagonal node *d*
(2) $\sigma_x(u_x)=\sigma_x(u)+1$; $\sigma_y(u_x)=\sigma_y(u)$; $\sigma_z(u_x)=\sigma_z(u)$;
$\sigma_x(u_y)=\sigma_x(u)$; $\sigma_y(u_y)=\sigma_y(u)+1$; $\sigma_z(u_y)=\sigma_z(u)$;
$\sigma_x(u_z)=\sigma_x(u)$; $\sigma_y(u_z)=\sigma_y(u)$; $\sigma_z(u_z)=\sigma_z(u)+1$;

$D_{min}=min\{D(u_x, sd), D(u_y, sd), D(u_z, sd)\}$;
IF ($D(u_x, sd)==D_{min}$) THEN $u=u_x$;
ELSE IF ($D(u_y, sd)==D_{min}$) THEN $u=u_y$;
ELSE IF ($D(u_z, sd)==D_{min}$) THEN $u=u_z$;

$DP=DP+\{u\}$;
(3) Repeat *step (2)* until $u=d$

Proposition 3.2.1.10 The time complexity of the preparation algorithm of DIAG multicast in *3DM* is $O(m+n+p)$.

Proof. Like it was explained for DIAG multicast in 2D meshes, the result of the algorithm is a shortest path from the source node s to the diagonal node d. The length of the path is the times that *step (2)* is repeated. Hence, the time complexity is less than the diameter of the mesh, which is $m+n+p-3$.

Algorithm 3.2.1.5 (DIAG multicast routing algorithm in 3DM)

Let DP denote the diagonal path from the source node $s(0, 0, 0)$ to its diagonal node d, MT denote the multicast tree constructed through the routing algorithm.

Input: M, D, s, DP
Output: MT
Variables: u, c (temporary nodes)

(1) Sort the destination nodes in D according to their distance to the source node s
(2) $MT=DP$ {Initialize the multicast tree with the diagonal path}
(3) - u=first node in D
- Find a node c in MT and in zone $\{s \Leftrightarrow u\}$ that is closest to node u
- Add node u to MT through a shortest path from c through XYZ routing
- $D=D - \{u\}$
(4) Repeat *step (3)* until $D=\Phi$
(5) Cut the tail part of DP which has neither destination nodes nor replicate nodes

Proposition 3.2.1.11 The time complexity of the routing algorithm of DIAG multicast in 3D meshes is $O(KN)$ where K is the total number of destinations and N is the total number of nodes in the mesh.

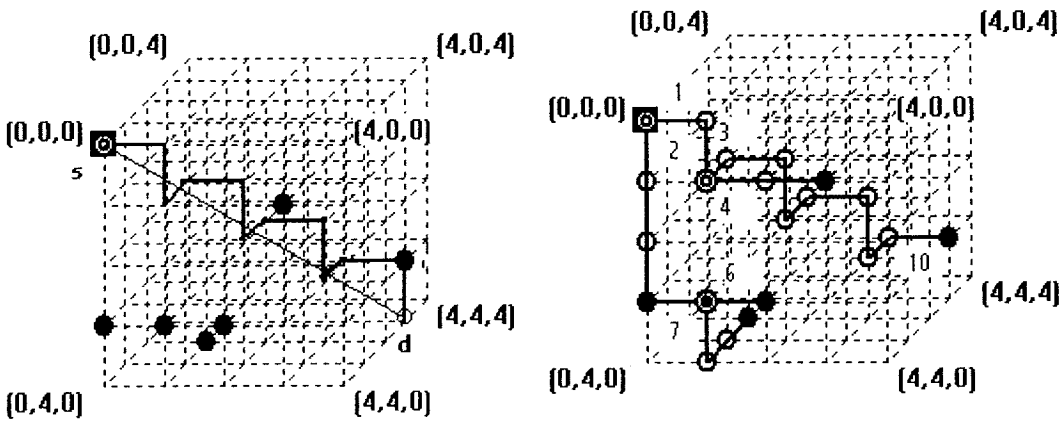
Proof. The proof is similar to that of the DIAG multicast in *2DM*.

Proposition 3.2.1.12 In the multicast tree constructed by DIAG algorithm in *3DM*, the path from the source node *s* to any destination node is a shortest path.

Proof. The proof is similar to that of DIAG multicast in *2DM*.

Example 3.2.1.2 In a 3D mesh network $M(5 \times 5 \times 5)$, construct the multicast tree using DIAG algorithm for a multicast that sends a message from the source node $s(0, 0, 0)$ to a set of destination nodes $D=\{ (0, 3, 0), (1, 3, 0), (1, 4, 2), (2, 3, 0), (3, 1, 0), (4, 3, 3) \}$.

The process of solving this example is similar to that of 2D mesh. The diagonal path is drawn in dark line illustrated in *Figure 3.2.1.11 (a)*. The completed multicast tree is shown in *Figure 3.2.1.11 (b)*. The multicast traffic is *20* links and multicast time is *10* hops.



(a) Diagonal path (b) Multicast tree

Figure 3.2.1.11: Example of DIAG Multicast in 5×5×5 Mesh

Proposition 3.2.1.13 The bound of DIAG multicast time in 3D meshes is $D_{max} \leq Time \leq D_{max}+K-1$.

Proof. The proof is similar to that of the DIAG algorithm in 2D meshes.

Proposition 3.2.1.14 The bound of DIAG multicast traffic in 3D meshes is $D_{max} \leq Traffic \leq \sum| x_i+y_i+z_i |$, $1 \leq i \leq K$.

Proof. The proof is similar to that of the DIAG algorithm in 2D meshes.

3.2.1.6 DIAG Algorithm in 3D Torus

In this section we will discuss how to apply the DIAG algorithm for 3D meshes to solve the multicast problem in 3D tori, and give the formal description and analysis of the DIAG multicast algorithm in 3D tori under the multicast defined by *Definition 3.1.5*.

Algorithm 3.2.1.6 (DIAG multicast algorithm in 3D Tori)

Input: T, D, s
Output: MT

(1) Divide the 3D torus T into eight 3D submeshes M_1, M_2, M_3, M_4, M_5, M_6, M_7 and M_8, each of which is defined by a zone as follows:

M_1: $\{ (0, 0, 0) \Leftrightarrow (\lceil m/2 \rceil-1, \lceil n/2 \rceil-1, \lceil p/2 \rceil-1) \}$
M_2: $\{ (m-1, 0, 0) \Leftrightarrow (\lceil m/2 \rceil, \lceil n/2 \rceil-1, \lceil p/2 \rceil-1) \}$
M_3: $\{ (m-1, 0, p-1) \Leftrightarrow (\lceil m/2 \rceil, \lceil n/2 \rceil-1, \lceil p/2 \rceil) \}$
M_4: $\{ (0, 0, p-1) \Leftrightarrow (\lceil m/2 \rceil-1, \lceil n/2 \rceil-1, \lceil p/2 \rceil) \}$
M_5: $\{ (0, n-1, 0) \Leftrightarrow (\lceil m/2 \rceil-1, \lceil n/2 \rceil, \lceil p/2 \rceil-1) \}$
M_6: $\{ (m-1, n-1, 0) \Leftrightarrow (\lceil m/2 \rceil, \lceil n/2 \rceil, \lceil p/2 \rceil-1) \}$
M_7: $\{ (m-1, n-1, p-1) \Leftrightarrow (\lceil m/2 \rceil, \lceil n/2 \rceil, \lceil p/2 \rceil) \}$
M_8: $\{ (0, n-1, p-1) \Leftrightarrow (\lceil m/2 \rceil-1, \lceil n/2 \rceil, \lceil p/2 \rceil) \}$

(2) Partition the destination set D into eight subsets D_1, D_2, D_3, D_4, D_5, D_6, D_7 and D_8 which contain the destination nodes in submesh M_1, M_2, M_3, M_4, M_5, M_6, M_7 and M_8 respectively.

(3) Assume $s_1(0, 0, 0)$ as the source node and origin of submesh M_1, D_1 as the set of destination nodes, apply DIAG algorithm in submesh M_1 and get the multicast subtree MT_1

(4) Assume $s_2(m-1, 0, 0)$ as the source node and origin of submesh M_2, D_2 as the set of destination nodes, apply DIAG algorithm in submesh M_2 and get the multicast subtree MT_2

(5) Assume $s_3(m-1, 0, p-1)$ as the source node and origin of submesh M_3, D_3 as the set of destination nodes, apply DIAG algorithm in submesh M_3 and get the multicast subtree MT_3

(6) Assume $s_4(0, 0, p-1)$ as the source node and origin of submesh M_4, D_4 as the set of destination nodes, apply DIAG algorithm in sub mesh M_4 and get the multicast subtree MT_4

(7) Assume $s_5(0, n-1, 0)$ as the source node and origin of submesh M_5, D_5 as the set of destination nodes, apply DIAG algorithm in submesh M_5 and get the multicast subtree MT_5

(8) Assume $s_6(m-1, n-1, 0)$ as the source node and origin of submesh M_6, D_6 as the set of destination nodes, apply DIAG algorithm in submesh M_6 and get the multicast subtree MT_6

(9) Assume $s_7(m-1, n-1, p-1)$ as the source node and origin of submesh M_7, D_7 as the set of destination nodes, apply DIAG algorithm in submesh M_7 and get the multicast subtree MT_7

(10) Assume $s_8(0, n-1, p-1)$ as the source node and origin of submesh M_8, D_8 as the set of destination nodes, apply DIAG algorithm in submesh M_8 and get the multicast subtree MT_8

(11) Assemble the multicast subtrees M_1, M_2, M_3, M_4, M_5, M_6, M_7 and M_8 into the overall multucast tree that is rooted at the source node $s(0, 0, 0)$ by connecting s_2, s_4 and s_5 to s, connecting s_3 and s_6 to s_2, connecting s_4 to s_8, and connecting s_7 to s_3

Proposition 3.2.1.15 The time complexity of DIAG algorithm in 3D tori is $O(KN)$ where N is the total number of nodes in the torus and K is the total number of destinations.

Proof. It is obvious that *step (1)* and *(2)* will take $O(K)$. *Steps (3)* to *(10)* are actually to execute the DIAG algorithm in a mesh of size $(m/2) \times (n/2) \times (p/2)$ with less than K destinations. So each step will take $O(Kmnp/8)=O(KN/8)$, all together the eight steps will

take O(KN). *Step (11)* takes O(1). So the total time complexity will be O($K+KN+1$) which is at the same order with O(KN).

Next, we use the same multicast in *Example 3.2.1.2* to show how the DIAG algorithm in 3D tori works. We first divide the original mesh into eight zones (a.k.a. submesh) at the middle of each dimension. The eight zones are $zone_1$ { *(0, 0, 0)* ⇔ *(2, 2, 2)* }, $zone_2$ { *(4, 0, 0)* ⇔ *(3, 2, 2)* }, $zone_3$ { *(4, 0, 4)* ⇔ *(3, 2, 3)* }, $zone_4$ { *(0, 0, 4)* ⇔ *(2, 2, 3)* }, $zone_5$ { *(0, 4, 0)* ⇔ *(2, 3, 2)* }, $zone_6$ { *(4, 4, 0)* ⇔ *(3, 3, 2)* }, $zone_7$ { *(4, 4, 4)* ⇔ *(3, 3, 3)* }, and $zone_8$ { *(0, 4, 4)* ⇔ *(2, 3, 3)* }. $Zone_1$, $zone_3$, $zone_4$, $zone_6$, $zone_8$ have no destination nodes. $Zone_5$ has destination nodes *(0, 3, 0), (1, 3, 0), (1, 4, 2),* and *(2, 3, 0).* $Zone_2$ has destination node *(3, 1, 0),* and $zone_7$ has destination node *(4, 3, 3).*

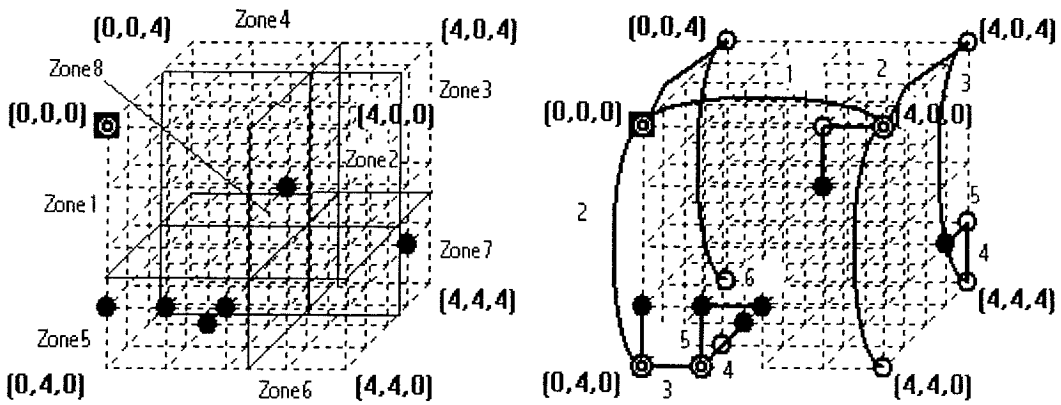


Figure 3.2.1.12: Example of DIAG Multicast in 5×5×5 Torus

Then, we treat nodes *(0, 0, 0), (4, 0, 0), (4, 0, 4), (0, 0, 4), (0, 4, 0), (4, 4, 0), (4, 4, 4),* and *(0, 4, 4)* as the origin and source node of $zone_1$ ~ $zone_8$ respectively, and apply the DIAG algorithm for 3D meshes in each zone with their respective destination nodes. Thus, each zone will have its own multicast subtree (consisting of the straight links in the figure), and we then

connect each subtree to the original source node *s(0, 0, 0)* through wraparound links(arches in the figure) to build the final multicast tree. Here are the exact steps to build the multicast tree. Subtree of $zone_2$: *(4, 0, 0)* ⇒ *(3, 0, 0)* ⇒ *(3, 1, 0)*; Subtree of $zone_5$: *(0, 4, 0)* ⇒ *(1, 4, 0)* ⇒ *(1, 4, 2)*, *(0, 4, 0)* ⇒ *(0, 3, 0)*, *(1, 4, 0)* ⇒ *(1, 3, 0)*, *(1, 3, 0)* ⇒ *(2, 3, 0)*; Subtree of $zone_7$: *(4, 4, 4)* ⇒ *(4, 3, 4)* ⇒ *(4, 3, 3)*; Combine the subtrees: *(0, 0, 0)* ⇒ *(4, 0, 0)* ⇒ *(4, 0, 4)* ⇒ *(4, 4, 4)*, *(0, 0, 0)* ⇒ *(0, 4, 0)*.

The completed multicast tree is shown in *Figure 3.2.1.12 (b)*. The total multicast traffic is *14* links, and the multicast time is *6* hops. Those wraparound links that did not send message to any destinations such as *(0, 0, 0)* ⇒ *(0, 0, 4)* ⇒ *(0, 4, 4)* are not counted. We assume the wraparound links have a higher transmission priority than other links.

Definition 3.2.1.2 The 3D torus is divided into eight roughly equal submeshes, each of which has its own origin (source) node and destination set. K_i is the number of destinations in $zone_i$ and D_{maxi} is the maximum distance from a destination in $zone_i$ to its origin ($1 \le i \le 8$), then we have:

$$K = \sum_{i=1}^{8} K_i$$

$$D_{max} = max\{D_{maxi,} \mid 1 \le i \le 8\}$$

Proposition 3.2.1.16 The bound of the multicast time of DIAG in 3D tori is $D_{max} \le Time \le D_{max}+K+2$.

Proof. The lower bound of multicast time in $zone_i$ is D_{maxi}. If $D_{max}=D_{max1}$, there are no wraparound links for $zone_1$, so D_{max} is the lower bound of multicast time for the whole

torus. The upper bound of multicast time in $zone_i$ is $D_{max_i}+ K_i-1$, the multicast time for the whole torus needs to add the number of wraparound links which are at most three of $zone_5 \sim zone_8$ (In lower zone first transmission priority). It is not hard to find that when $D_{max}=D_{max_7}$ and $K=K_7$ (the same with $zone_5$, $zone_6$ and $zone_8$), the multicast time obtains the largest possible value which is $D_{max_7}+ K_7-1+3$, *i.e.* $D_{max}+ K+2$.

Proposition 3.2.1.17 The bound of the multicast traffic of DIAG in 3D tori is reduced to those of the DIAG in its eight submeshes.

3.2.1.7 DIAG in *n*-D Mesh and Torus

We can extend the DIAG algorithm to n-dimensional meshes and tori very easily. We will not give the precise description of the algorithm here but rather the rough idea of how it works under the multicast defined by *Definition 3.1.6*. Assume *MZ* is the multicast zone, source node s will be one of the diagonal nodes of the zone, let us denote the other diagonal node as $d(x_1, \ldots, x_n)$ and the diagonal line as *sd*.

Like all the previous DIAG algorithms, the algorithm consists of two parts. Part one forms the diagonal path by starting from the source node s, and then traveling along each dimension i alternatively at small steps y_i which is proportional to x_i, i.e. it satisfies $y_1/x_1 = y_2/x_2 = \ldots = y_n/x_n$ ($y_i \leq x_i$). This way it will finally reach the diagonal node d. Part two is actually the same as the DIAG multicast routing algorithm in 2D and 3D meshes, i.e. to construct the multicast tree by connecting all destination nodes.

For n-dimensional tori, we can use the similar way as handling the 2D and 3D tori. First, divide the torus into 2^n submeshes by dividing the original torus at the middle of

each dimension. The destination set D will be partitioned into 2^n subsets too. And then apply the n-D DIAG algorithm for meshes in each submesh with the respective destination subset. Finally assemble the multicast subtrees in each submesh into a single multicast tree that is rooted at the source node s.

3.2.2 DDS Multicast Algorithm

DDS (Dimensional Distance Sorted) is a tree-based multicast routing algorithm for store-and-forward switched mesh networks. It is designed to obtain near optimal multicast time and further reduce the large multicast traffic of previous shortest path multicast algorithms such as DIAG. First, here is the definition of dimensional distance.

Definition 3.2.2.1 (Dimensional distance) The dimensional distance between two nodes is the distance along one dimension between them. For example, for any two nodes $x(x_1, \ldots, x_i, \ldots, x_n)$ and $y(y_1, \ldots, y_i, \ldots, y_n)$, the ith dimensional distance $d_i = | x_i - y_i |$. The minimum dimensional distance between x and y denoted as $d_{min} = min\{d_1, \ldots, d_i, \ldots, d_n\}$. If one of the two nodes is the source node $s(0, \ldots, 0)$, the other node is $x(x_1, \ldots, x_n)$ then $d_i = | x_i |$, $d_{min} = min\{ | x_1 |, \ldots, | x_n | \}$.

DDS algorithm sorts all the destinations in dimensional distance so that those closer to the source node in one or more dimensions can receive the message first and hopefully forward it to destinations behind them through an extra short distance. Thus, the path or distance previously traveled in each dimension can be shared as much as possible by nodes coming behind. When constructing the multicast tree, we keep connecting the

sorted destination nodes to the tree through an as short as possible path which is part of a shortest path from the source node to the respective destination.

In the following sections, we will discuss in details the motivation and heuristics of designing this algorithm, and the formal description and analysis of the DDS algorithm in 2D/3D meshes and tori.

3.2.2.1 Motivation of Designing DDS

The DIAG multicast algorithm maintains the near optimal multicast time while reduces the multicast traffic significantly over the time optimal VH algorithm. Now, the questions are: Does DIAG algorithm generate the least traffic among all shortest-path multicast algorithms, can we further reduce the traffic, and how close is it to the optimal multicast traffic?

The answer is not obvious. But first, let us compare its traffic with the previously known pro-traffic multicast algorithm: MIN. The MIN algorithm constructs the multicast tree by alternatively choosing the node with the smallest coordinate in each dimension from the remaining destination nodes and connecting it to the tree through a shortest path possible.

Figure 3.2.2.1 illustrates the average total traffic curve for both DIAG and MIN algorithm in a 10×10 mesh. There is a notable difference between the traffic of the DIAG algorithm and the MIN algorithm, which means that there is still potential space to further reduce the traffic resulted from the DIAG algorithm. Although the MIN algorithm generates less traffic than DIAG algorithm, it is not a shortest path multicast algorithm and its multicast time is not near optimal.

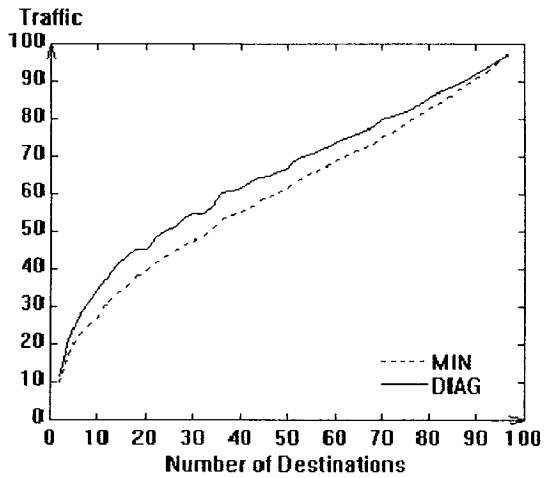


Figure 3.2.2.1: Comparison of Traffic of DIAG and MIN

Therefore, our new challenge will be to develop a new multicast routing algorithm which obtains near optimal time and as good performance on traffic as the MIN algorithm. The DDS algorithm is designed for such purposes.

3.2.2.2 Heuristics of DDS Algorithm

As DDS algorithm is designed to reduce the traffic of previous shortest path multicast algorithms such as VH and DIAG, the main strategy will be to eliminate bad scenarios which create large excessive traffic.

We know that both VH and DIAG use a major routing path (the X path in VH and the diagonal path in DIAG) in the multicast tree and then connect the destination nodes to the major path. The bad thing is, in some cases, destinations may be far away from the major path. Hence, to connect these destinations to the major path will create large traffic. Moreover, the major path itself can become excessive traffic when very few destinations are connected to it.

In DDS we discard the use of a major routing path, but just construct the multicast tree from scratch by connecting the destinations to current multicast tree. In order to ensure the shortest path routing and share as much common path as possible with previously connected nodes, we propose the following heuristics:

- If all the destination nodes, which are closer to the source node in some dimension than one destination node, receive the message earlier, very likely it will need to travel less distance in each dimension to get the message from a preceding node.
- If the message is routed along a shortest path between any two nodes in a multicast tree, it is likely that the total links of the tree will be low, and the transmission time between the source node and the destination nodes will also be minimized.
- If a shortest path travels along different dimensions alternatively rather than in dimension order, then there are more routing choices and the chances of receiving the message from a nearby node are increased which can save traffic.

Based on these heuristics, we designed the DDS multicast algorithm which we will discuss in details next.

3.2.2.3 DDS in *n*-Dimensional Mesh

This section presents the formal description and analysis of the DDS multicast algorithm in *n*-D meshes under the multicast defined by *Definition 3.1.6.*

DDS multicast algorithm can be divided into two parts. Part one is to sort the destinations according to their dimensional distance and is called DDS multicast

preparation algorithm. Part two is to construct the multicast tree that covers the sorted destinations and is called DDS multicast routing algorithm.

Algorithm 3.2.2.1 (DDS multicast preparation algorithm)

The main process of this algorithm is to sort the destination nodes in D according to their dimensional distance to the source node $s(0, \ldots, 0)$. We can use any commonly used sorting algorithm such as the merge sort to deal with the sorting. So the major problem is how to compare the dimensional distance of any two nodes. Here is the algorithm to just do that.

Assume the two different nodes to be compared are $x(x_1, \ldots, x_i, \ldots, x_n)$ and $y(y_1, \ldots, y_i, \ldots, y_n)$. Let us denote the minimum dimensional distance of x as $D_{min}(x)$, the first dimension which bears the minimum distance as $N_{min}(x)$.

Input: x, y
Output: the order between x and y

(1) IF $D_{min}(x) < D_{min}(y)$ then $x<y$
IF $D_{min}(x) > D_{min}(y)$ then $x>y$
IF $D_{min}(x) == D_{min}(y)$
{
$i = N_{min}(x)$
$j = N_{min}(y)$
IF $i<j$ then $x<y$
IF $i>j$ then $x>y$
IF $i==j$ then $x = x - \{x_i\}$; $y = y - \{y_j\}$;
}
(2) Repeat *step (1)* until $x<y$ or $x>y$

Proposition 3.2.2.1 The time complexity of the preparation algorithm of DDS multicast in n-D mesh is $O(n^2KlogK)$ where K is the total number of destinations and n is the number of dimensions.

Proof. The calculation of the minimum dimensional distance of an n-D node takes $O(n)$. In the algorithm of comparing the dimensional distance of two nodes, the worst

case will need to do n times minimum dimensional distance calculation which takes $O(n^2)$. To merge sort K destinations, $O(KlogK)$ comparisons of nodes are done which will take $O(n^2KlogK)$.

Algorithm 3.2.2.2 (DDS multicast routing algorithm)

The main process of the routing algorithm is to construct a multicast tree by connecting the sorted destination nodes in D sequentially. Let us denote the multicast tree as MT.

Input: M, D, s
Output: MT
Variables: u, c (temporary nodes)

(1) $MT=\{ s \}$ (Initialize it with only source node)
(2) - u=first node in D
- Find a node c in MT and in zone $\{s \Leftrightarrow u\}$ that is closest to node u
- Connect node u to MT through a shortest path from c through dimension-ordered routing
- $D=D - \{u\}$
(3) Repeat *step (2)* until $D=\Phi$

Proposition 3.2.2.2 The time complexity of the routing algorithm of DDS multicast in n-D meshes is $O(KN)$ where K is the total number of destinations and N is the total number of nodes in the mesh.

Proof. The algorithm connects each destination node to the multicast tree MT. For each destination, finding the shortest path to be connected to MT needs a traversing of the tree, which takes $O(N)$. K destinations will take $O(KN)$.

3.2.2.4 DDS Algorithm in 2D Mesh

DDS multicast algorithm in 2D meshes is just a special case of DDS algorithm in *n*-dimensional meshes. This section presents a more specific description and analysis for it under the multicast defined by *Definition 3.1.4.*

Algorithm 3.2.2.3 (DDS multicast algorithm in 2DM)

Let the multicast tree constructed through the routing algorithm be denoted as *MT.*

Input: *M, D, s*
Output: *MT*
Variables: *x, y* (*x, y* coordinate)
d (dimensional distance)

(1) *MT*={ *s* } (Initialize the multicast tree and set the root of the tree at *s*)
(2) *d=0* (Initialize the dimensional distance)
(3) Scan the column *x=d*, starting from the point (*d, d*) down to (*d, n-1*)
(Here is the pseudo code for this step)

```
x=d;
FOR( y=d; y<n; y++)
{
    IF (x, y) is a destination node
    {
        - Find a node c in MT and in zone {s ⇔ (x, y)}, which is closest to
          node (x, y)
        - Connect node (x, y) to MT through a shortest path from c through XY
          routing
    }
}
```

(4) Scan the row *y=d*, starting from the point (*d, d*) to (*m-1, d*)
(Here is the pseudo code for this step)

```
y=d;
FOR( x=d; x<m; x++)
{
    IF (x, y) is a destination node
    {
        - Find a node c in MT and in zone {s ⇔ (x, y)} that is closest to node
          (x, y)
        - Connect node (x, y) to MT through a shortest path from c through XY
          routing
```

}
}
(5) IF $d=m-1$ or $d=n-1$ then it is done,
ELSE $d=d+1$ and repeat *step (3)* and *(4)*

Proposition 3.2.2.3 The time complexity of the routing algorithm of DDS multicast in *2DM* is O(KN) where K is the total number of destinations and N is the total number of nodes in the mesh.

Proof. The process of scanning the mesh will take O(mn), i.e. O(N). Similar to the case of DDS in n-D meshes, constructing the tree will take O(KN). The whole process will take O($N+KN$) which is at the same order with O(KN).

Proposition 3.2.2.4 In the multicast tree constructed by DDS algorithm in 2D meshes, the path from the source node s to any destination node is a shortest path.

Proof. The proof is similar to that of DIAG algorithm in 2D meshes.

Proposition 3.2.2.5 DDS Multicast routing algorithm in mesh or torus networks is deadlock free.

Proof. The proof is similar to that of DIAG algorithm in 2D meshes.

Let us use the same multicast in *Example 3.2.1.1,* which sends a message from the source node *s(0, 0)* to a set of destination nodes $D=\{ (0, 2), (3, 0), (4, 0), (4, 6), (6, 6), (7, 4) \}$, to show how *DDS* algorithm in 2D meshes works. As illustrated in *Figure 3.2.2.2*, the algorithm starts by using the source node *s(0, 0)* as an aligning point to scan the column and then the row where it is located. Then, it moves to the next aligning point *(1, 1)* increasing both coordinates of the previous aligning point by *1*, and do the same. Repeat the process until all the nodes in the mesh have been scanned. During the scanning, whenever we come cross a destination node, add it to a queue. At the end, the sorted destination nodes in the queue are $d_1(0, 2)$, $d_2(3, 0)$, $d_3(4, 0)$, $d_4(4, 6)$, $d_5(7, 4)$, $d_6(6, 6)$ sequentially.

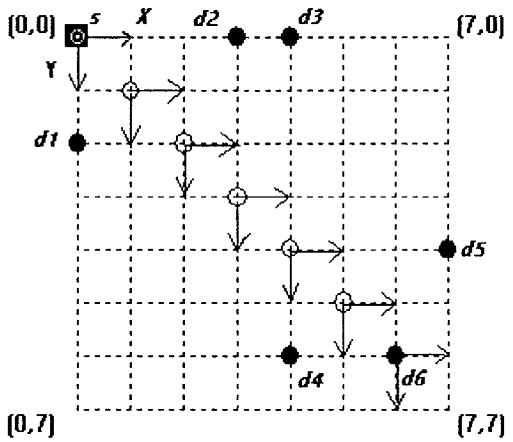


Figure 3.2.2.2: Scanning and Sorting of DDS in 8×8 2DM

After the sorting, we will construct the multicast tree by connecting destination nodes to it. At the beginning, the multicast tree consists of only the source node itself and we connect the first destination node $d_1(0, 2)$ to source node *s(0, 0)* through *XY* routing. Then for the second destination node $d_2(3, 0)$, the closest node in the already existing tree and

within zone $\{s \Leftrightarrow d_2\}$ is source node s, so we connect d_2 to s through XY routing. Likewise, we connect $d_3(4, 0)$ to $d_2(3, 0)$, $d_4(4, 6)$ to $d_3(4, 0)$ through XY routing. For destination node $d_5(7, 4)$, the closest node in the already existing tree and within zone $\{s \Leftrightarrow d_5\}$ is an intermediate node $(4, 4)$, so we connect node $d_5(7, 4)$ to node $(4, 4)$ through XY routing. Likewise, we connect the last destination node $d_6(6, 6)$ to an intermediate node $(6, 4)$. Here are the exact steps of constructing the tree: $(0, 0) \Rightarrow (0, 2)$, $(0, 0) \Rightarrow (3, 0)$, $(3, 0) \Rightarrow (4, 0)$, $(4, 0) \Rightarrow (4, 6)$, $(4, 4) \Rightarrow (7, 4)$, $(6, 4) \Rightarrow (6, 6)$.

The completed multicast tree is shown in the *Figure 3.2.2.3*. The multicast traffic is *17* links and the multicast time is *13* hops. The same example using DIAG results in *21* links and *12* hops, which shows that DDS generates less traffic than DIAG but does not sacrifice much on multicast time.

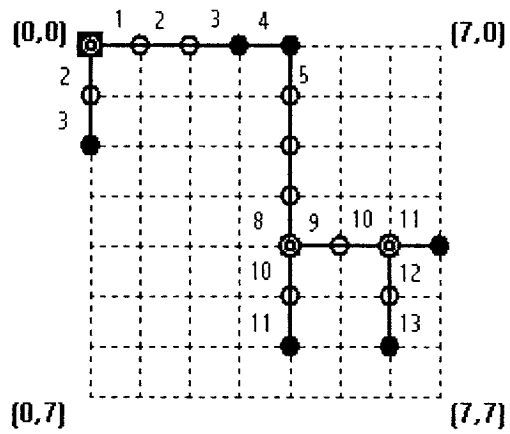


Figure 3.2.2.3: Example of DDS Multicast in 8×8 Mesh

Proposition 3.2.2.6 The bound of DDS multicast time in 2D meshes is $D_{max} \leq Time \leq D_{max}+K-1$.

Proof. Assume that at a branching node the message is sent to neighbors in X dimension first and then those in Y dimension. The rest of the proof is similar to that of DIAG algorithm in 2D meshes.

Proposition 3.2.2.7 The bound of DDS multicast traffic in 2D meshes is $D_{max} \leq Traffic \leq \sum| x_i+y_i |$, $1 \leq i \leq K$.

Proof. The proof is similar to that of DIAG algorithm in 2D meshes.

3.2.2.5 DDS Algorithm in 2D Torus

In this section, we will discuss how to apply the DDS algorithm to solve the multicast problem in 2D tori and give the formal description and analysis of the DDS multicast algorithm in 2D tori.

Algorithm 3.2.2.4 (DDS algorithm in 2DT)

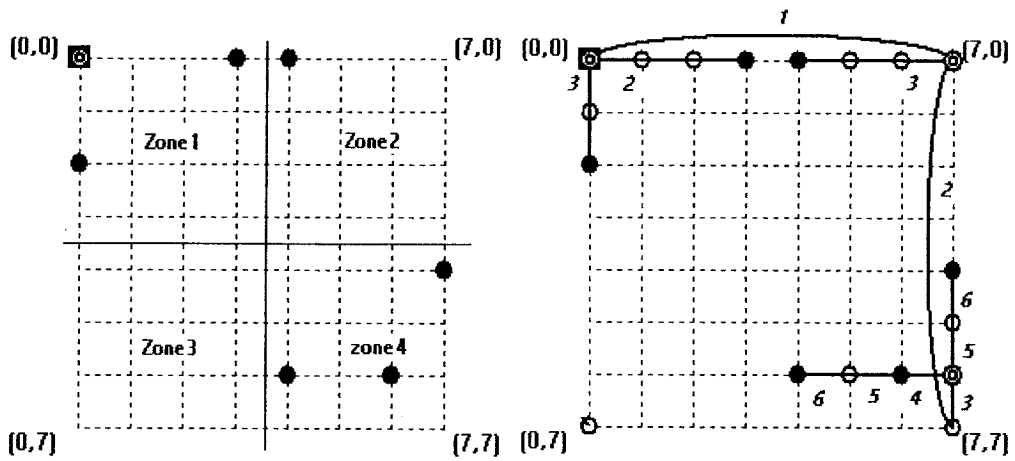
Input: T, D, s
Output: MT (Multicast Tree)

The main procedures are similar to those of DIAG algorithm in *2DT* as described in *Algorithm 3.2.1.3,* except that we apply DDS for 2D meshes in each submesh here instead of DIAG for 2D meshes.

Proposition 3.2.2.8 The time complexity of DDS algorithm in 2D tori is $O(KN)$ where K is the total number of destinations and N is the total number of nodes in the torus.

Proof. The proof is similar to that of DIAG algorithm in 2D meshes.

To show how *DDS* algorithm in 2D tori works, we give an example where the source node *s(0, 0)* sends a message to a set of destination nodes *D*={ *(0, 2), (3, 0), (4, 0), (4, 6), (6, 6), (7, 4)* }. The main procedures are similar to those of the DIAG algorithm in *2DT* except that we apply DDS for 2D meshes in each zone instead of DIAG for 2D meshes. Here are the exact steps to construct the multicast tree. Subtree of $zone_1$: *(0, 0) ⇒ (0, 2), (0, 0) ⇒ (3, 0);* Subtree of $zone_2$: *(7, 0) ⇒ (4, 0);* Subtree of $zone_4$: *(7, 7) ⇒ (7, 4), (7, 6) ⇒ (6, 6), (6, 6) ⇒ (4, 6);* Combine the subtrees: *(0, 0) ⇒ (7, 0), (7, 0) ⇒ (7, 7).*



(a) Distribution of destinations (b) Multicast tree

Figure 3.2.2.4: Example of DDS Multicast in 8×8 Torus

The completed multicast tree is shown in *Figure 3.2.2.4*. The total multicast traffic is *16* links and the multicast time is *6* hops. Compared with DDS for 2D meshes in the same example, the multicast time is reduced significantly by half from *13* to *6*, but the traffic is not reduced much.

Proposition 3.2.2.9 The bound of multicast time of DDS in 2D tori is $D_{max} \leq Time \leq D_{max}+K+1$.

Proof. The proof is similar to that of DIAG in 2D tori.

Proposition 3.2.2.10 The bound of multicast traffic of DDS in 2D tori is reduced to those of DDS in its four submeshes.

3.2.2.6 DDS Algorithm in 3D Mesh

In this section, we will have the formal description and analysis of the DDS multicast algorithm in 3D meshes under the multicast defined by *Definition 3.1.5*. Let the multicast tree constructed through the routing algorithm be denoted as *MT*.

Algorithm 3.2.2.5 (DDS multicast routing algorithm in 3DM)

Input: *M, D, s*
Output: *MT*
Variables: *x, y, z* (*x, y, z* coordinate)
d (dimensional distance)

(1) *MT*={ *s* } (Initialize *MT* with only the root *s*)
(2) *d=0* (Starting from dimensional distance *0*)
(3) Scan the surface *x=d*, starting from the point (*d, d, d*) toward (*d, n-1, p-1*)
- Treat zone {(*d, d, d*)⇔(*d, n-1, p-1*)} as a 2D mesh whose origin is (*d, d, d*)
- Apply the DDS algorithm for 2D meshes in zone {(*d, d, d*)⇔(*d, n-1, p-1*)} to connect all the destinations in this zone to *MT*

(4) Scan the surface *y=d*, starting from the point (*d, d, d*) toward (*m-1, d, p-1*)
- Treat zone {(*d, d, d*)⇔(*m-1, d, p-1*)} as a 2D mesh whose origin is (*d, d, d*)
- Apply the DDS algorithm for 2D meshes in zone {(*d, d, d*)⇔(*m-1, d, p-1*)} to connect all the destinations in this zone to *MT*

(5) Scan the surface $z=d$, starting from the point (d, d, d) toward $(m-1, n-1, d)$
- Treat zone $\{(d, d, d) \Leftrightarrow (m-1, n-1, d)\}$ as a 2D mesh whose origin is (d, d, d)
- Apply the DDS algorithm for 2D meshes in zone $\{(d, d, d) \Leftrightarrow (m-1, n-1, d)\}$ to connect all the destinations in this zone to *MT*

(6) IF $d=m-1$ or $d=n-1$ or $d=p-1$ then it is done
ELSE $d=d+1$ and repeat *step (3), (4)* and *(5)*

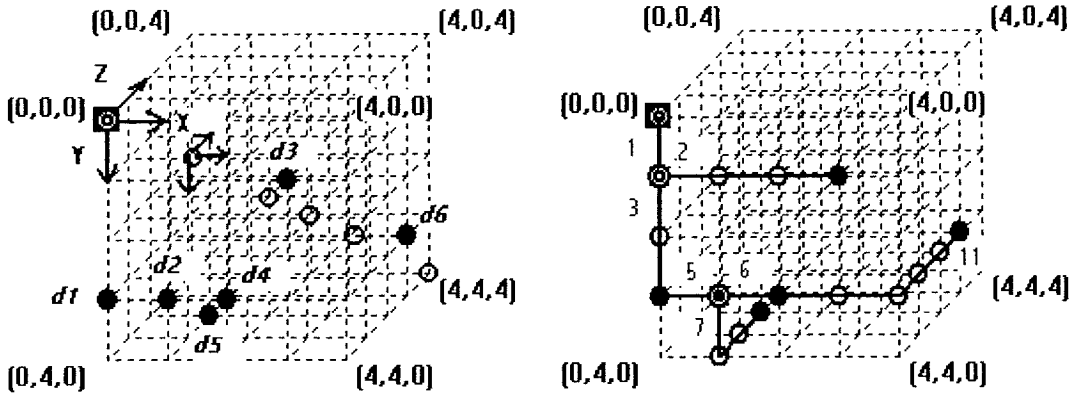
Proposition 3.2.2.11 The time complexity of the routing algorithm of DDS multicast in *3DM* is $O(KN)$ where K is the total number of destinations and N is the total number of nodes in the mesh.

Proof. The proof is similar to that of DDS multicast routing algorithm in 2D meshes.

Proposition 3.2.2.12 In the multicast tree constructed by DDS algorithm in 3D meshes, the path from the source node s to any destination node is a shortest path.

Proof. The proof is similar to that of DDS algorithm in 2D meshes.

Figure 3.2.2.5 shows how *DDS* algorithm for 3D meshes works in *Example 3.2.1.2*. The aligning points *(0, 0, 0), (1, 1, 1), (2, 2, 2), (3, 3, 3)* and *(4, 4, 4),* the *XY*, *XZ* and *YZ* scanning surfaces, and the sorted destinations from d_1 to d_6 are indicated by *Figure 3.2.2.5 (a)*. The completed multicast tree is shown in *Figure 3.2.2.5 (b)*. The multicast traffic is *16* links and the multicast time is *11* hops. The same example using DIAG for 3D meshes results in *20* links and *10* hops.



(a) (b)

Figure 3.2.2.5: Example of DDS Multicast in 5×5×5 Mesh

Proposition 3.2.2.13 The bound of DDS multicast time in 3D meshes is $D_{max} \leq Time \leq D_{max}+K-1$.

Proof. The proof is similar to that of DDS multicast in 2D meshes.

Proposition 3.2.2.14 The bound of DDS multicast traffic in 3D meshes is $D_{max} \leq Traffic \leq \sum|x_i+y_i+z_i|$, $1 \leq i \leq K$.

Proof. The proof is similar to that of DDS algorithm in 2D meshes.

3.2.2.7 DDS Algorithm in 3D Torus

In this section we will discuss how to apply the DDS algorithm to solve the multicast problems in 3D tori, and give the formal description and analysis of the DDS multicast algorithm in 3D tori under the multicast defined by *Definition 3.1.5.*

Algorithm 3.2.2.6 (DDS multicast algorithm in 3D torus)

Input: *T, D, s*
Output: *MT*

The main procedures are similar to those of DIAG algorithm in *3DT* as described in *Algorithm 3.2.1.6,* except that we apply DDS for 3D meshes in each submesh instead of DIAG for 3D meshes.

Proposition 3.2.2.15 The time complexity of DDS algorithm in 3D tori is O(*KN*) where *K* is the total number of destinations and *N* is the total number of nodes in the torus.

Proof. The proof is similar to that of the DIAG algorithm in 3D tori.

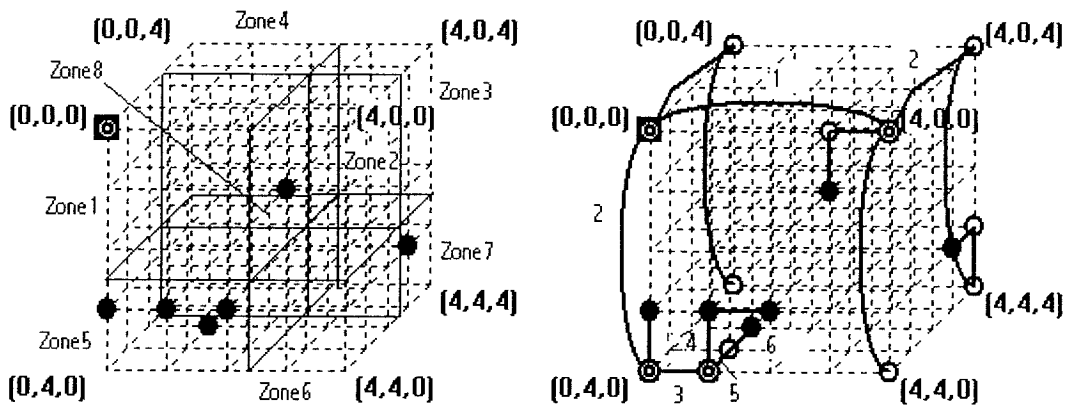


Figure 3.2.2.6: Example of DDS Multicast in 5×5×5 Torus

Now, let us use *DDS* algorithm in 3D tori to solve the multicast in *Example 3.2.1.2.* The main procedures are similar to the example of DIAG algorithm in *3DT* except that

we apply DDS for 3D meshes in each zone instead of DIAG for 3D meshes. The completed multicast tree is shown in *Figure 3.2.2.6 (b)*. The total multicast traffic is *14* links and the multicast time is *6* hops.

Proposition 3.2.2.16 The bound of multicast time of DDS in 3D tori is $D_{max} \leq Time \leq D_{max}+K+2$.

Proof. The proof is similar to that of DIAG in 3D tori.

Proposition 3.2.2.17 The bound of multicast traffic of DDS in 3D tori is reduced to those of DDS in its eight submeshes.

3.2.3 Comparison of DIAG, DDS, VH and MIN

Optimal multicast solution that minimizes both time and traffic is NP-hard or even does not exist at all. The algorithms we have discussed so far including DDS, DIAG, MIN and VH are all heuristic algorithms. That means none of them can obtain the best results at all cases. Each of them has pros and cons. In this thesis, we will discuss the strengths and drawbacks of DDS, DIAG, MIN and VH based on the results we get in 2D meshes. The relationships of these algorithms in 3D meshes or tori are very similar.

The table (*Figure 3.2.3.1*) compares the multicast traffic, time and computation complexity between DDS, DIAG and VH multicast algorithms in 2D meshes.

As illustrated in the table, DDS and DIAG algorithms have the same computation complexity of O(KN). VH algorithm has the best complexity since $KD \leq KN$ ($D=m+n-1$, $N=m\times n$), while MIN has the worst complexity which is D times that of DDS.

	DDS	DIAG	MIN	VH
Complexity	$O(KN)$	$O(KN)$	$O(KND)$	$O(KD)$
Traffic (Best case)	D_{max}	D_{max}	D_{max}	D_{max}
Time (Best case)	D_{max}	D_{max}	D_{max}	D_{max}
Time (Worst case)	$D_{max}+K-1$	$D_{max}+K-1$	$\sum(\lvert x_i-x_{i-1}\rvert+\lvert y_i-y_{i-1}\rvert),\ 1 \le i \le K$	$D_{max}+1$

Figure 3.2.3.1: Comparisons between Algorithms in 2DM

All four algorithms have the same minimum multicast time which is D_{max}. But VH has a much better worst case multicast time than all the others, which is $D_{max}+1$. Therefore, VH will almost always have the near optimal multicast time which is either D_{max} or $D_{max}+1$, and is very predicable unlike the other three which have a wide range. Although both DDS and DIAG are also shortest path multicast algorithms and their multicast time is optimal in all-port node architectures, they cause more delay hops in one-port node architectures. Hence, in general multicast time of DDS and DIAG can be no better than VH, which can be as high as $D_{max}+K-1$ in the worst case. MIN algorithm will normally need more time to do the same multicast than DDS and DIAG since it is not shortest path based. In the worst case, when all destination nodes are connected through a non-shortest path, multicast time of MIN is the total number of links (total traffic).

We also notice that all the algorithms have the same minimum multicast traffic which is D_{max}. However, we have learned from the heuristics and examples that, DDS and DIAG usually create less traffic than VH for the same multicast. The worst case multicast traffic of VH is actually the maximum x coordinate plus the sum of the y coordinates of

all destination nodes, which could be much worse than that of all the other three algorithms. MIN is believed to be the one that normally creates least traffic since it is optimized for traffic only but at a cost of highest computation complexity.

3.3. Path-based Multicast Algorithm

3.3.1 About Path-based Multicast

In path-based multicast, a message is passed along a path that starts from the source node and goes through all the destinations.

Problems of Tree-based Multicast

It is believed that tree-based multicast is efficient on both time and traffic due to the fact that the message is routed along common paths as much as possible and then branches to different destinations. This is true especially for store-and-forward switched networks since the network latency is proportional to the distance [1]. In tree-based algorithm, the message is being passed concurrently from node to node, so the distance of the path to the farthest node is likely much less than that of a path-based multicast which passes message from node to node sequentially.

However, the story is a little bit different in wormhole routed networks. In tree-based multicast, when we launch only one worm from the source node to be routed along the whole tree and once blocking occurs at any node on the routes, the flits will remain in contiguous channels of the network and the whole wormhole routing tree is frozen. Such scenarios in turn block the transmission of other messages in need of the occupied channels, and are very vulnerable to deadlocks and congestions. Lin and Ni have

discussed in [7, 13] that tree-based multicast routing is not suitable for wormhole routed network, and that channel congestion and deadlock becomes an important factor affecting the network performance when the traffic is high.

Why Path-based Multicast?

In wormhole routed networks, the network latency is insensitive to the distance especially when the message is very long. So a reasonable solution will be to pass the message to as many destinations as possible in a single worm and prohibit the branching at any intermediate nodes, which leads to multicast path pattern [7]. A multicast path for a multicast is a set of contiguous channels, starting from the source node and reaching each destination node in a certain order.

Since path-based routing does not need to replicate messages at each intermediate node, it can save much overhead time. An analysis in [1, 7] proved that the probability that a message is blocked in path-like routing is lower than that in tree-like routing. Experiments and simulations in [1, 7] further show that path-based model provides much better performance than the tree-based model when there are contentions in the network.

Strategies to Design Path-based Multicast

Here is a typical way to develop a path-based multicast algorithm in 2D meshes. First, find a Hamiltonian path of the mesh and label the nodes with their order on the path. Then, sort the destinations according to their labels. And finally develop a routing function that routes the message from one destination to another through a sequence of nodes whose labels are in the same order (ascending or descending) as on the base path.

The Hamiltonian path guarantees the feasibility of forming a message-passing path and defines the order of the nodes, which has to be observed when passing the message. Almost all topologies currently used in multicomputer networks, including 2D meshes and hypercubes, have Hamiltonian paths [7]. The Hamiltonian path is usually used as a guide to form the message-passing path. Message is generally passing along the base path, but can take shortcuts from one destination toward the next destination down the base path (i.e. Can skip some parts of the base path between the two destinations) as long as the order of nodes on the passing path is consistent with that on the base path.

The key of designing efficient path-based multicast routing is to reduce the distance between the destinations and reduce the total length of the path, which consequently reduces the total traffic and the transmission time. Another trick is: if the number of destinations in each path is small, the length of the path and the average distance between destinations will also tend to be small.

3.3.2 Review of LIN's Algorithm

In [7], Lin proposed a path-based multicast algorithm for 2D meshes, which uses a Hamiltonian path (*Figure 3.3.1 (a)*) to govern the routing. The algorithm, starting from the source node, labels each node with their order along the path and sorts the set of destination nodes according to their label. It then passes the message from one destination to another by routing from current node to one of its neighbor that is closest to next destination down the Hamiltonian path. *Figure 3.3.1 (b)* illustrates the message passing path from LIN's algorithm based on the Hamiltonian path in *Figure 3.3.1 (a)* for a set of multicast destinations nodes. The main advantages of LIN's algorithm are as follows.

(1) It is very easy to form the Hamiltonian path, and computation complexity is small.

(2) The routing function and supporting hardware are simple since messages are always passed along a single path.

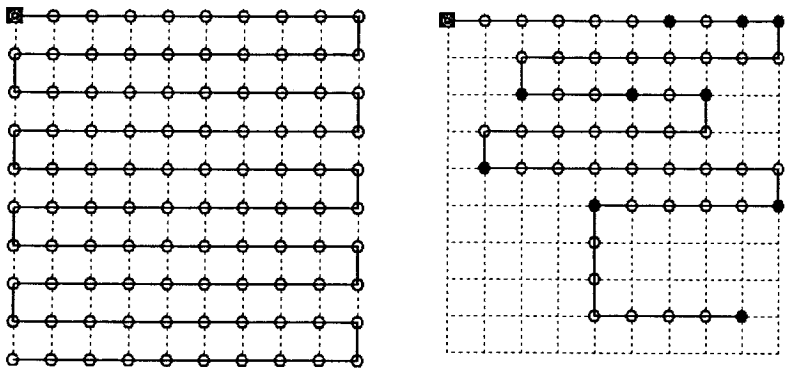


Figure 3.3.1: A Multicast by LIN's Algorithm in 10×10 Mesh

The drawbacks of this algorithm are:

(1) Low parallelism. Message is passed along one single path.

(2) Very large maximum message passing distance which is the length of the path.

(3) Very large amount of traffic due to the fact that messages have to be passed in a certain order in accordance with the base path.

(4) Very large amount of inefficient traffic. Such as row *4* and row *5* in *Figure 3.3.1 (b)* exist only to send the message to one destination node because the Hamiltonian path travels back and forth through the whole row which is a very long distance.

As a result, LIN's algorithm produces a lot of excessive traffic because messages have to be passed strictly following the order of nodes in a single Hamiltonian path. Furthermore, the time that it takes for the last node to receive the message is related to the

total length of the message-passing path, which is likely to be a large number. To overcome these disadvantages of LIN's algorithm, we develop a dual-path-based multicast routing algorithm called XY-path algorithm.

3.3.3 XY-path Multicast Algorithm

XY-path is a dual-path-based multicast routing algorithm designed for wormhole routed 2D-mesh multicomputer networks. First, we will discuss the heuristics used to design it.

3.3.3.1 Heuristics of XY-path Algorithm

Two basic facts about multicast path problems were stated in [1], which we cite as two theorems for our study.

Theorem 3.3.1 There is a Hamiltonian path in any 2D mesh.

Theorem 3.3.2 The OMP (*Optimal Multicast Path*) problem in 2D meshes is NP-complete.

It is unrealistic for us to design an optimal path-based multicast algorithm with polynomial time, but only heuristic algorithms that try to reduce the maximum path distance and total traffic as much as possible. Our goal is to overcome those pitfalls of LIN's algorithm. The following are some heuristics we used.

(1) Increase the parallelism of the message passing. If we increase the number of paths, then messages can be passed concurrently along several paths.

(2) Reduce inefficient traffic. If we reduce the back and forth traveling distance of the base path, then that of the routing path will also be reduced and some unnecessary traffic will be saved.

(3) Reduce the total traffic. If a message is passed along a shortest path between destinations, the total traffic and maximum message passing distance will also tend to be small.

(4) Reduce the maximum message passing distance. If we try to partition the network into several disjoint sets and form a path for each partition, the length of each path is also likely much smaller than that of a single path.

In a 2D mesh we notice that the source node, which is located at the left-top corner, is connected to its neighbors through two separate channels. This provides the possibility of passing the message along two different paths simultaneously to the destinations. Based on these heuristics, we developed the so-called XY-path multicast routing algorithm, in which two message-passing paths start from the source node, one travels back and forth along the X dimension and the other along Y dimension. Together they cover all the destination nodes in the mesh. XY-path algorithm can well overcome the pitfalls of LIN's algorithm. First, since messages were passed through two paths, the length of each path will be likely less than that of a single. Second, the mesh network is partitioned to two small parts, which reduces the chances of long back and forth traveling and the average distance between two destinations.

3.3.3.2 Formal Description of XY-path Algorithm

The XY-path multicast routing algorithm actually consists of two phases. First we need to split the nodes of the mesh into two subsets joined only at the source node, and form a Hamiltonian path for each subset, i.e. X path and Y path. Then we route the message in X path and Y path concurrently from one destination to another in the order as they occur in X path or Y path. Hence, XY-path algorithm contains two parts: the partition algorithm and the routing algorithm.

The partition algorithm is critical since it forms the base paths which very much determine the overall performance of the multicast. Basically the partition algorithm splits all the nodes into two paths joined at the source. We accomplish it by alternatively running one path (X) along rows back and forth and the other path (Y) along the columns exclusively.

Algorithm 3.3.1 (XY-path multicast partition algorithm in 2DM)

Denote the set of nodes in X path as X and the set of nodes in Y path as Y. *Label(u)* denotes the label of node u, *Length(X)* denotes the length of path X. *x(v)* denotes the x coordinate of node v, and *y(v)* denotes the y coordinate of node v.

Input: M
Output: X, Y
Variables: u, v (temporary nodes)

(1) $X=Y=\{s\}$; (Initialization)
Done(X)=Done(Y)=FALSE ;
(2) IF NOT *Done(X)*
(a) v=last node in X
IF *x(v)==m-1* and v is the last node of its row and *Length(X)>Length(Y)*
THEN go to *step (3)*
ELSE find next node u by traveling along rows, make U turn to next row if it reaches the end of a row or a node belonging to Y
(b) IF u is founded THEN $X=X+\{u\}$, *Label(u)*=its order in X
ELSE *Done(X)=TRUE* (It either visited last row or can not make U turn)

(c) Continue *step (2)*

(3) IF NOT *Done(Y)*

(a) v=last node in Y

IF $y(v)==n-1$ and v is the last node of its column and *Length(Y)>Length(X)*

THEN go to *step (4)*

ELSE find next node u by traveling along columns, make U turn to next row if it reaches the end of a column or a node belonging to X

(b) IF u is founded THEN $Y=Y+\{u\}$, *Label(u)*=its order in Y

ELSE *Done(Y)=TRUE* (It either visited last column or can not make U turn)

(c) Continue *step (3)*

(4) REPEAT *step (2)* and *step (3)* UNTIL *Done(X)* and *Done(Y)*

Proposition 3.3.1 Using the partition algorithm, we can always find a pair of X path and Y path joined only at the source node $s(0, 0)$, which cover all nodes in a 2D mesh.

Proof. According to the partition algorithm, the switching point can only be a node at the boundary. Therefore, whenever we switch the path, the unoccupied area is a mesh. This process will continue until only one row or one column is left. And this row or column can be easily added to X path or Y path depending on the final scenario.

Proposition 3.3.2 The time complexity of XY-path partition algorithm for 2D meshes is $O(N)$ where N is the total number of nodes in the mesh.

Proof. It is easy to see the process of partitioning the 2D mesh is to check each node in the mesh and add it to either X or Y path according to the status of the node. The total number of nodes in the mesh is N. So the complexity is $O(N)$.

The XY-path partition algorithm is very dynamic. The switching point is decided by comparing current length of the paths and can only be at the boundary of the mesh not at

the middle of a row or column. When one path comes across the other, it must make a *U*-turn. Hence, it is not known which node belongs to which path until the algorithm is executed. Here is our goal of making it dynamic: First, to balance the length of the two paths so the traffic load and maximum traveling distance on both paths will also be balanced; Second, to avoid too many long runs in the back and forth direction. As indicated in *Figure 3.3.2*, in an asymmetric 16×8 mesh, Y path needs to travel back and forth several rounds to match the distance traveled by X in just one round. Thus, the length of runs of X path decreases faster than that of Y path, and the number of very long runs is limited.

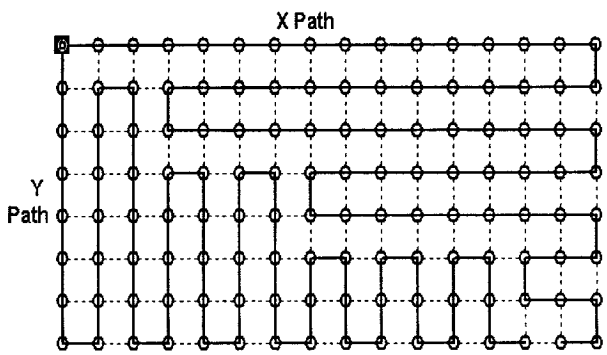


Figure 3.3.2: *X* path and *Y* path in 8×16 Mesh

We got X path and Y path that can be used as base paths to govern the message routing. A basic rule for path-based routing is that the message must be passed from one destination to next through a sequence of nodes whose order is in accordance with that of the base path [1]. Messages need not move exactly along the base path, and can take a shortcut from one destination to next, but the ascending or descending order of the nodes must be observed. The XY-path routing algorithm is to find the message-passing paths that pass message to the destinations in X path and Y path.

Algorithm 3.3.2 (XY-path multicast routing algorithm in 2DM)

Let X' denote the message-passing path for X path, and Y' denote the message-passing path for Y path. $R(u,v)$ is the routing function to route from node u to node v on a Hamiltonian path of a mesh.

Input: M, D, X, Y
Output: X', Y'

(1) $X'= Y'=\{s\}$; (Initialization)
(2) Get the message passing path X' for destinations in X path
  (a) Get the last node u in path X'
  (b) Get next destination node v on X path
  (c) Using $R(u,v)$ based on X path to find the path $P(u...v)$ which connect u to v
  (d) $X'=X'+P$
  (e) Repeat *step (2)* until all destinations on path X are exhausted
(3) Similar to step *(2)*, we can Get the message passing path Y' for destinations in path Y

The routing function $R(u,v)$ is derived from a general routing function in [1] refereed to as sorted *MP* message routing algorithm . Its main idea is to choose next node in the path that is a neighbor of current node and closest to next destination down the base path. When we apply this function in our algorithm, we get our $R(u,v)$. Let u be the current node, v be the next destination node down X path, and $L(x)$ be the label of node x on X path, then we have the definition of $R(u,v)$ based on X path:

$$R(u,v)=w \qquad (3.3.1)$$

Where w satisfies $L(w)=max\{ L(p),\ p$ is a neighbor of u and $L(v)\geq L(p)>L(u) \}$. Likewise we can have the $R(u,v)$ based on Y path.

Proposition 3.3.3 The time complexity of XY-path routing algorithm for 2D meshes is $O(N)$ where N is the total number of nodes in the mesh.

Proof. The time cost to find the multicast paths actually involves executing the routing function R at each node along the paths. Time complexity of R is $O(1)$. The total number of nodes in both X path and Y path is less than that of the mesh. Hence, the routing algorithm will cost $O(N)$ time.

Proposition 3.3.4 Time complexity of XY-path multicast algorithm for 2D meshes is $O(N)$ where N is the total number of nodes in the mesh.

Proof. Both the XY partition algorithm and the routing algorithm cost $O(N)$. The over all time of XY-path multicast algorithm is also $O(N)$, which is at the same order of $O(KlogK)$ of the LIN's algorithm.

Proposition 3.3.5 For two destination nodes u and v on the X path or Y path in a 2D mesh, the path selected by routing function $R(u,v)$ is a shortest path from u to v.

Proof: Consider path Y for example. According to the routing function $R(u,v)$, the message will be passing through a sequence of nodes whose labels are in ascending order. Assume u and v are two destination nodes on column C_u and C_v, $u<v$ (i.e. u precedes v), hence $C_u \leq C_v$. If $C_u = C_v$, it is obvious that the path between u and v will be along the same column and it is shortest. Now $C_u < C_v$, according to $R(u,v)$, there will be no links on the path that travel from a larger column to a small column. So we have *a)* the total distance traveled along X dimension is just the distance between u and v in X dimension,

i.e. $|x(u)-x(v)|$. Then we need to prove *b)* the distance traveled along Y dimension is $|y(u)-y(v)|$. The following is the proof for *b)*. If $y(u) \leq y(v)$, then we can prove that the path travels downward from the node with smaller y to node with larger y. Assume w is a node on the path from which the path travels along a column upward and reaches node w' at the same column, then we have $y(v) \geq y(w) > y(w')$. We notice w's neighbor on next column, say w'', is obviously closer to destination v than node w' in both dimension X and Y since $y(w'')=y(w)$. According to $R(u,v)$, it should be routed from w to w'' not w', which shows the assumption is wrong. Hence all routing along dimension Y is downward. Similarly we can prove when $y(u) \geq y(v)$ all routing along Y dimension is upward. So we can draw the conclusion of *b)*. Combine *a)* and *b)*, the distance traveled by the routing path between u and v is $|x(u)-x(v)|+|y(u)-y(v)|$, which is shortest.

Proposition 3.3.6 The multicast message-passing path obtained through the routing function is optimal for the given base path X or Y.

Proof. Since message is routed along a shortest path from one destination to another, the length of the multicast path, which is essentially the sum of the distance between each pair of destinations along the path, is also minimized.

Example 3.3.1 In a 2D mesh network $M(10 \times 10)$, construct the multicast path using XY-path algorithm for a multicast that sends a message from the given source node $s(0, 0)$ to a set of destination nodes $D=\{ (2, 0), (3, 1), (7, 2), (7, 5), (0, 4), (1, 3) \}$.

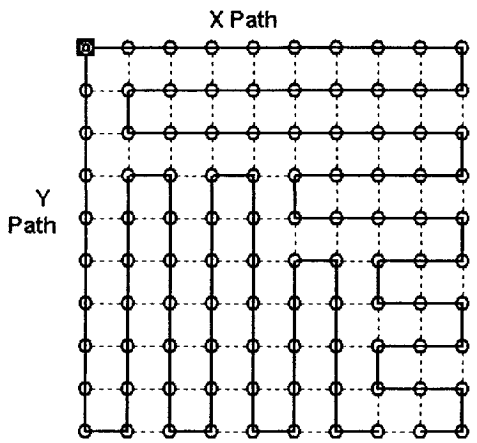
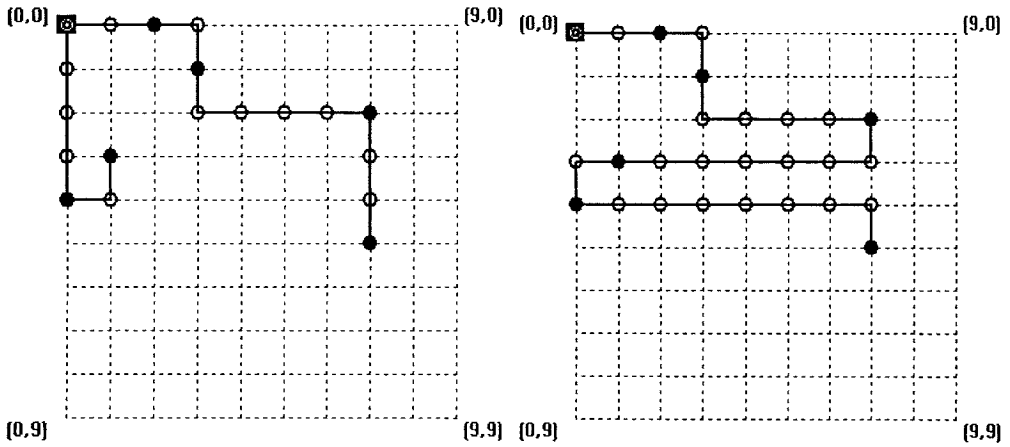


Figure 3.3.3: Example of XY-path in 10×10 Mesh

First, we partition the nodes of the mesh into two independent paths joined at the source node. In this example, the algorithm starts advancing path *Y* from the source node along column *0* till the boundary node *(0, 9)*, then compares the length of the two paths and finds that path *X* is shorter than path *Y* (at this point, length of path *X* is zero), so the turn is switched to path *X*. Then, it advances path *X* from source node along row *0* till boundary node *(9, 0)* where the length of path *X* is still not longer than that of path *Y*, so instead of switching to path *Y*, it continues advancing path *X*. It turns to node *(9, 1)* on row *1* and advances along row *1*, then makes *U* turn to row *2* when it hits path *Y* and continues on row *2* till it comes back to the boundary node *(9, 2)* where we notice that path *X* is already longer than path *Y*. So it is time to advance path *Y*. Thus, whenever a path arrives at a boundary node, compare their length and give the turn to the shorter path. We keep alternatively advancing path *X* along rows and path *Y* along columns back and forth until both paths can not advance any more. The final path *X* and *Y* are illustrated in *Figure 3.3.3*. The partition also divides the destination nodes into two subsets { *(0, 4)*, *(1, 3)* } on path *Y* and { *(2, 0)*, *(3, 1)*, *(7, 2)*, *(7, 5)* } on path *X*.



(a) XY-path (b) LIN's

Figure 3.3.4: Example of XY-path Multicast in 10×10 Mesh

After the partition, we will need to find out the message-passing paths through the XY-path routing algorithm. Take path *X* for example. Starting from the source node *s(0, 0),* it is very obvious that the shortest route between *s(0, 0)* and destination node *(2, 0)* is on the base path *X*. But for the route between node *(2, 0)* and node *(3, 1),* simply following the base path is not a good choice, so we need to find a shortcut. We execute routing function *R(u, v)* at node *(2, 0),* which actually checks the distances to node *(3, 1)* from all node *(2, 0)*'s neighbors behind node *(2, 0)* but before node *(3, 1)*, and routes to the one closest to node *(3, 1)* which is *(3, 0)*. We then execute routing function *R(u, v)* at node *(3, 0),* which finally routes the message to destination node *(3, 1)*. Afterwards, we can route the message from destination node *(3, 1)* to destination node *(7, 2)* and to destination node *(7, 5)* using the same routing function *R(u, v)*. Likewise, we can find out the message-passing path for destinations on path *Y*. The complete message-passing paths are illustrated in *Figure 3.3.4 (a)*. The total traffic is *18* links and the maximum distance of the path is *12*.

The same multicast using LIN's algorithm will generate total traffic of *26* links which is also the maximum distance as shown in *Figure 3.3.4 (b).*

Proposition 3.3.7 The bound of multicast traffic for XY-path multicast is $K \leq Traffic \leq KD$, where K is the total number of destinations, D is the diameter of the multicast zone.

Proof. Assume there are p destinations (v_1, v_2, ..., v_p) in X path, q destinations (u_1, u_2, ..., u_q) in Y path . $D(u,v)$ is the distance between node u and v, s is the source node. K is the total number of destinations, i.e. $K=p+q$. The multicast traffic of XY-path algorithm is:

$$Traffic = D(s,v_1) + \sum_{i=1}^{p-1} D(v_i, v_{i+1}) + D(s,u_1) + \sum_{i=1}^{q-1} D(u_i, u_{i+1})$$

It has been proven that the minimum total links is K for any kind of multicast [1], i.e. the message reaches a destination every time it passes through a link. In case of XY-path algorithm, traffic reaches the low bound K when the K destinations line up in just one path and adjacent to one after another. Since the path between each pair of destinations is shortest, which is less than D, the total distance of the K pairs of destinations will be less than KD.

Proposition 3.3.8 The bound of multicast time for XY-path multicast is $K+L \leq Time \leq KD+L$, where L is the length of the message.

Proof. According to *Formula (2.1.11),* in wormhole routed networks *Multicast Time=max(l_1, l_2, ..., l_{k-1}, l_k)+L.* In the case of XY-path multicast, assume the length of *X* path and *Y* path is *dx* and *dy* respectively, then we have *Multicast Time=max(d_x, d_y)+L.*

So, when *max(d_x, d_y)<<L,* the effect of distance is negligible and length of the message dominates the multicast time. When *max(d_x, d_y)>>L,* the distance of the path dominates the multicast time. But since for a certain message to be multicasted, its length *L* is constant, the distance of the multicast path becomes the decisive factor. The bound of *max(d_x, d_y)* is *K~KD* according to *Proposition 3.3.7,* so the bound of multicast time will be *K+L~KD+L.*

Chapter 4

Simulations and Discussions

4.1 The Simulation Model

The purpose of this chapter is to evaluate the performance of these new algorithms and verify whether we have achieved our goals of designing them. We developed software to simulate the multicast communications in mesh-connected multicomputer networks.

4.1.1 Model of Simulation Program

The generic multicast simulation program is developed not only for multicast simulation in mesh-connected networks but also for any kind of network topology, any kind of multicast algorithm, and any kind of node architecture. Hence, we choose the object-oriented design methodology featuring good hierarchy and flexibility. *Figure 4.1.1* illustrated the model and class diagram of this program.

In order to maintain a good generality, flexibility, and extensibility of the program, we introduce a set of abstract classes: *Graph, Node, Processor* and *MultiAlgorithm*. They form the core of the model and define the interfaces. In addition to the abstract classes, we defined corresponding concrete classes that implement the interfaces and add specific features. *Mesh, MeshNode* and *MeshAlgorithm* are all concrete classes for meshes.

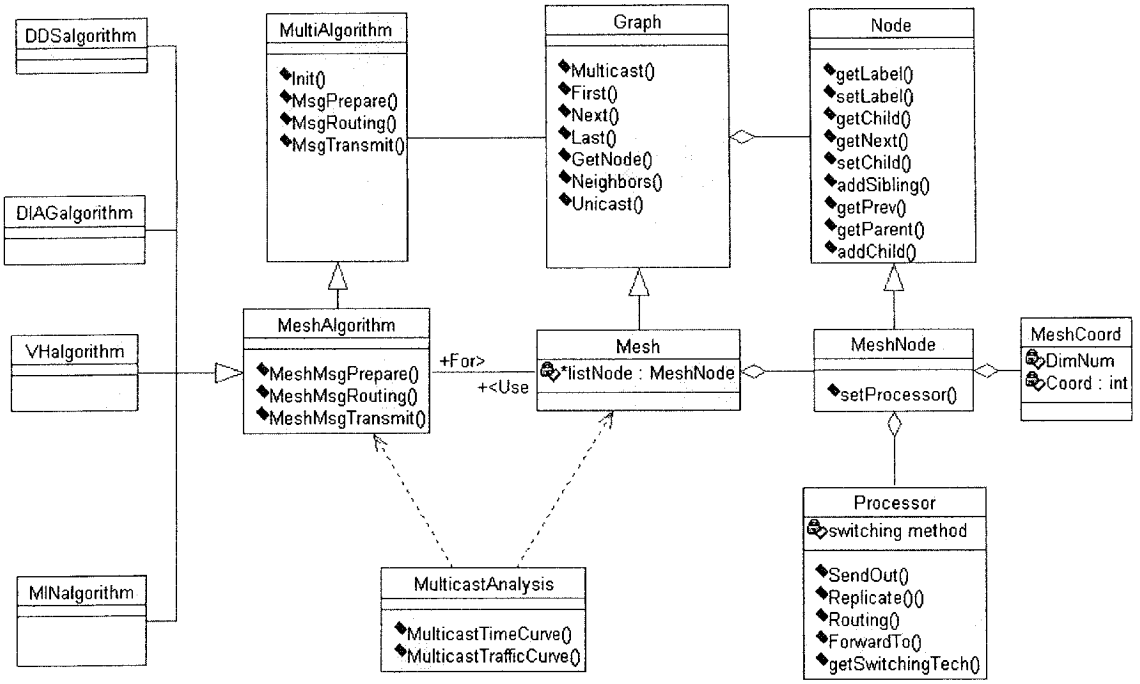


Figure 4.1.1: Class Diagram of Simulation Program

4.1.2 Implementation of Simulation System

The simulation program we have just introduced can be implemented in any object oriented programming language such as *C++* or Java. In our case, since the simulation of multicast in multicomputer network is very computation intensive, we chose *C++* which is a very efficient programming language. More specifically we developed the simulation software using *Visual C++* and *MFC* within *Microsoft Visual Studio 6.0* under *Windows XP* environment. The program can run under any Windows operating system with the *Win32* and *MFC* libraries installed. To ensure the speed and efficiency of the simulation, we recommend computer systems with at least *2GHz* CPU clock speed and *1GB* RAM.

4.1.3 Performance Evaluation Model

Since the multicast routing algorithms that we proposed are heuristic algorithms, none of them is optimal, i.e. no algorithm is best at all scenarios. One algorithm performs better than the other in some cases, but worse in other cases. So, we can only use the average value of a large number of randomly generated multicasts to evaluate the performance of each algorithm on average.

There are two major parameters used to evaluate the performance of multicast operation: time and traffic. Here we use several types of charts related to these two parameters to evaluate and analyze the performance of the multicast algorithms. They are the *multicast time* curve, the *multicast traffic* curve and the *average additional traffic (AAT)* curve. The *time* curve reflects the changing trend of the average multicast time with respect to the number of destination nodes. The *traffic* curve reflects the changing trend of the average multicast traffic with respect to the number of destination nodes. The *AAT* curve reflects the changing trend of the average additional traffic with respect to the number of destination nodes. Both *traffic* and *AAT* curves reflect the multicast traffic of an algorithm, but *AAT* is a better parameter to indicate the efficiency of the multicast traffic. The *traffic* curve indicates just the total traffic that a multicast creates.

To compare the performance of several algorithms in one chart, we will draw a curve for each algorithm, resulted from exactly the same set of samples. Every point on the curve is averaged over *1000* runs of multicasts with the same number of destination nodes [1, 7]. The set of multicast destination nodes and their distribution are generated randomly through pseudo random number.

We also introduce another parameter *mean* to indicate the average performance of an algorithm in a certain mesh network regardless of the number of destinations. This method is derived from the ideas introduced in [18] for evaluating the efficiency and resource consumption of multicast in an arbitrary network.

Definition 4.1.1 (Mean of a Parameter) Given a mesh or torus M and a multicast algorithm A, the average value of a parameter P from all the multicast samples, regardless of the number of its destinations, in a statistical experiment is called the *mean* of parameter P for multicast algorithm A in mesh or torus M.

Mean roughly indicates the overall performance of algorithm A on parameter P in the mesh M. It is actually the average line for the curve of parameter P. We will use *multicast time mean*, *multicast traffic mean* and *AAT mean* to reflect the multicast time, traffic and *AAT* of a certain multicast algorithm in a given mesh in general.

4.2. Performance Evaluation of DIAG and DDS

4.2.1 Simulation Assumptions

The following are the assumptions and conditions under which the simulations are done for the performance evaluation of the tree-based multicast algorithms.

- All the simulations are done in a store-and-forward switched mesh or torus network, since both algorithms are best for store-and-forward technology.
- The nodes in the network have one-port architecture which is the mostly used architecture in multicomputers.

- To simplify the calculation, we assume the size of each dimension of an *n*-dimensional mesh is the same. The unit of time is hop and the unit of traffic is link.
- The sampling resolution for the curves is *10* destination nodes per sample point. The value at each sample point is averaged over *1000* runs of multicasts with the same number of destination nodes.
- Use dimension-ordered routing as the routing function between pairs of nodes, which results in a shortest path and prevents deadlock.
- Use semi-distributed routing scheme, which has been introduced in Chapter 2, for message transmission along the multicast tree.
- Formulas to calculate multicast time, traffic, and additional traffic in our simulations are those for store-and-forward switched networks with one-port architecture which are introduced in earlier sections.

$$\textit{Multicast Traffic} = |E(MT)|$$

$$\textit{Additional Traffic} = |E(MT)| - K$$

$$\textit{Multicast Time} = max\{ l_1+w_1, l_2+w_2, \ldots, l_{k-1}+w_{k-1}, l_k+w_k \}$$

4.2.2 Simulation of DDS and DIAG in 2D Mesh

We study the performance of the algorithms in 2D meshes through simulations done in a 20×20 mesh in this section.

In *Figure 4.2.1* multicast time curves of *DIAG* and *VH* are almost overlapped, which indicates that on average the multicast time resulted from *DIAG* algorithm is very close to

that of *VH* algorithm. This is in accordance with the fact that they are both shortest path multicast algorithms, which result in near optimal multicast time. In this example, the multicast time mean of *DIAG* is *35.76*, which is *0.41%* less than *VH's 35.91*. We can also notice that the multicast time curve is pretty flat, which means it does not relate very much to the number of destinations. This is because the multicast time mainly depends on the distance of the farthest destination node but not the number of the destination nodes. The multicast time generally increases with the number of destinations.

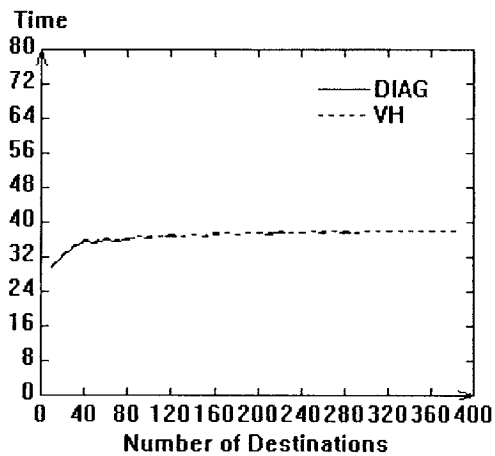


Figure 4.2.1: Multicast Time of DIAG versus VH in 2D Mesh

Figure 4.2.2 indicates that multicast traffic generally increases with the number of destinations, but *AAT* first increases with the number of destinations and reaches its peak at about *20%* percent of the total nodes, and then it decreases and finally reaches *0*. This means that when the number of destination nodes is very small or very large, the multicast traffic is most efficient. In between it will generate more inefficient traffic. The figure also indicates that both traffic and *AAT* of *DIAG* algorithm is much less than that

of *VH* algorithm. In this example, the multicast traffic mean of *DIAG* is *247.28* which is *25%* less than *VH's 333.69*. The *AAT* mean of *DIAG* algorithm is *52.28* which is *62%* less than *VH's 138.69*.

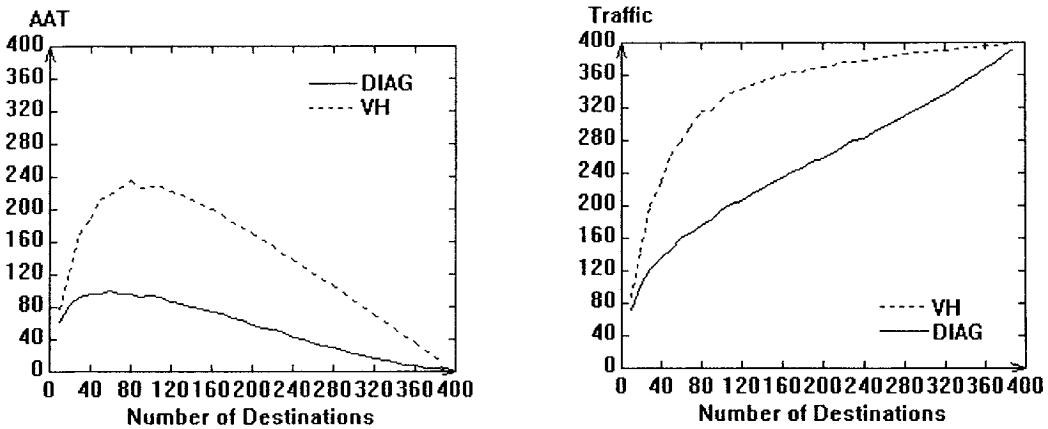


Figure 4.2.2: Multicast traffic of DIAG versus VH in 2D Mesh

From the simulation results of *DIAG* and *VH* algorithms in this 2D mesh, we can notice that *DIAG* algorithm reduces the multicast traffic by *25%* and increase traffic efficiency by *62%* over *VH* algorithm without sacrificing the near optimal multicast time of *VH* algorithm. The *DIAG* algorithm even somehow performs slightly better than *VH* in terms of multicast time.

In *Figure 4.2.3* multicast time curves of *DDS*, *DIAG* and *MIN* show that, on average, *DIAG* has the best multicast time, *DDS* has a little bit larger multicast time than *DIAG*, *MIN* has the worst multicast time. This is due to the fact that both *DDS* and *DIAG* are shortest path multicast algorithm, *MIN* is not. In this example, the multicast time mean of *DDS* is *37.88*, which is *8.5%* more than *DIAG's 34.89*, but *5%* less than *MIN's 39.86*.

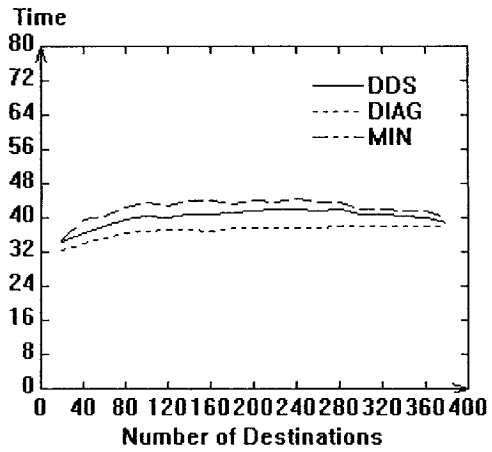


Figure 4.2.3: Multicast Time of DDS versus MIN in 2D Mesh

Figure 4.2.4 indicates that the multicast traffics of *DDS* and *MIN* are very close and obviously better than *DIAG*. In this example, the multicast traffic mean of *DDS* is *230.85* which is *1.3%* more than *MIN's 227.77* but *4.3%* less than *DIAG's 241.41*. The *AAT* mean of *DDS* algorithm is *40.85* which is only *8%* more than *MIN's 37.77* and *20.5%* less than *DIAG's 51.41*. The trend of these curves is similar to those of *DIAG* we mentioned in the previous section.

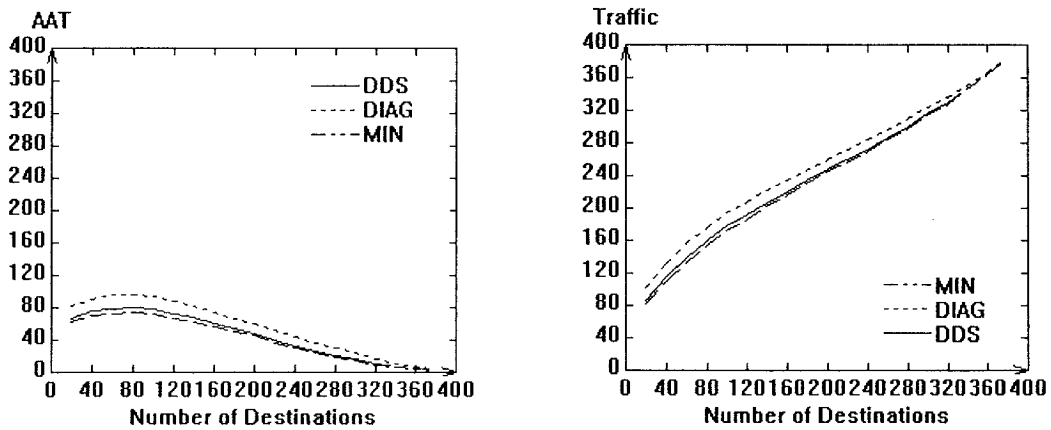


Figure 4.2.4: Multicast Traffic of DDS versus MIN in 2D Mesh

From the simulation results of *DDS*, *DIAG* and *MIN* algorithms in this 2D mesh, we can notice that *DDS* algorithm reduces the average additional traffic by *20%* over *DIAG* algorithm and reaches the neighborhood of the pro-traffic *MIN* algorithm. It improves the multicast traffic efficiency significantly while keeping the multicast time near that of *DIAG* algorithm.

4.2.3 Simulation of DDS and DIAG in 2D Torus

In this section we study the performance of the algorithms in 2D tori through simulations done in a 20×20 torus.

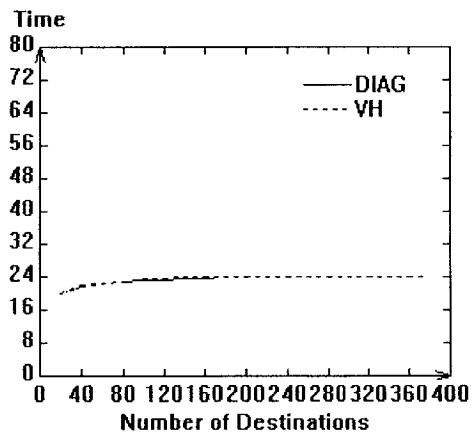


Figure 4.2.5: Multicast Time of DIAG versus VH in 2D Torus

Figure 4.2.5 shows the multicast time of *DIAG* and *VH* in 2D tori is also very close. The difference is that the multicast time of both algorithms is reduced significantly by almost *40%* over their counterpart in the 2D mesh.

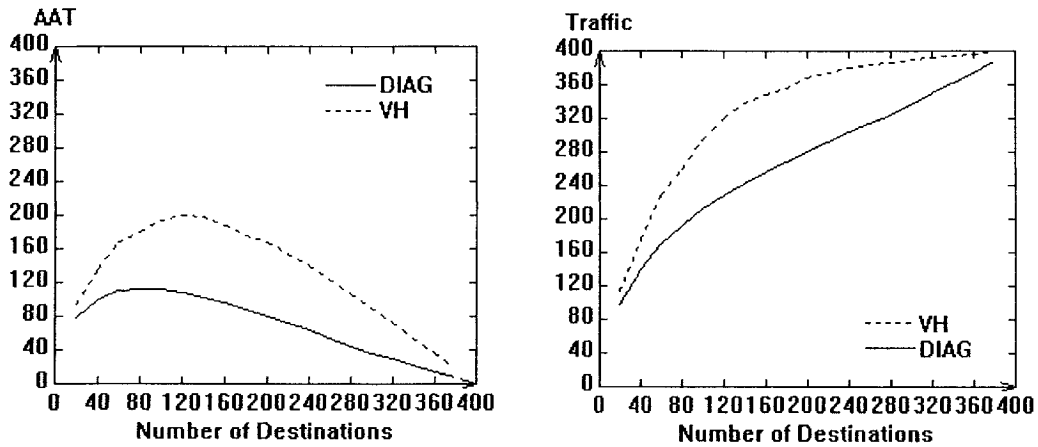


Figure 4.2.6: Multicast Traffic of DIAG versus VH in 2D Torus

In *Figure 4.2.6* the trend of the multicast traffic curves of *DIAG* and *VH* in the 2D torus is similar to those in the 2D mesh. The multicast traffic mean of *DIAG* is about *255.63*, decreased by *18.54* over *VH's 313.84*, and the *AAT* mean of *DIAG* is about *65.63* decreased by *47%* over *VH's 123.84*. These simulation results lead to the similar conclusion as in the 2D mesh.

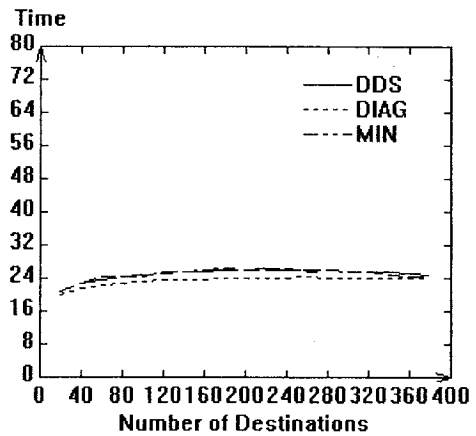


Figure 4.2.7: Multicast Time of DDS versus MIN in 2D Torus

Figure 4.2.7 shows that the relations between the multicast time of *DDS*, *DIAG* and *MIN* in the 2D torus is similar to that in the 2D mesh, except that the difference between them is smaller, with *DDS* at *23.7*, *DIAG* at *22.25* and *MIN* at *23.69*. This is due to the fact that their multicast time over all is reduced by almost *40%* over their counterparts in the 2D mesh.

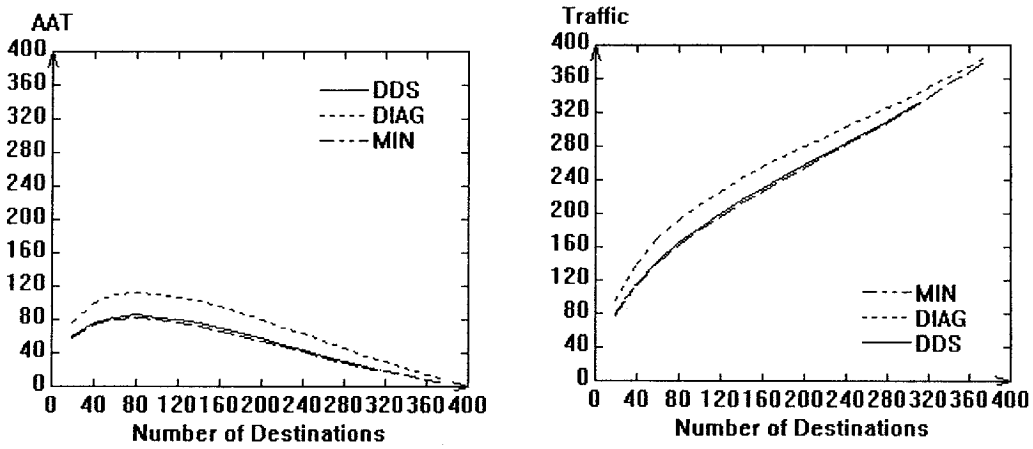


Figure 4.2.8: Multicast Traffic of DDS versus MIN in 2D Torus

Figure 4.2.8 shows that the trend and relations of multicast traffic curves of *DDS*, *DIAG* and *MIN* in the 2D torus are similar to those in the 2D mesh. The multicast traffic mean of *DDS*, *DIAG* and *MIN* is *237.30*, *255.39* and *235.29* respectively. The *AAT* mean of *DDS* is *47.30*, which is *28%* less than *DIAG's 65.39* and only *4%* more than *MIN's 45.29*.

After comparing the performance of different multicast algorithms in the 2D mesh and torus, we now study the performance of the same algorithm in the 2D mesh over 2D torus. *Figure 4.2.9* indicates the relation of *DIAG* algorithm in the 2D mesh and 2D torus. Multicast time mean of *DIAG* in the 2D torus is *22.27* which is about *36%* less than that

in the 2D mesh. This is understandable given the fact that the diameter of a torus is half of that of the respective mesh.

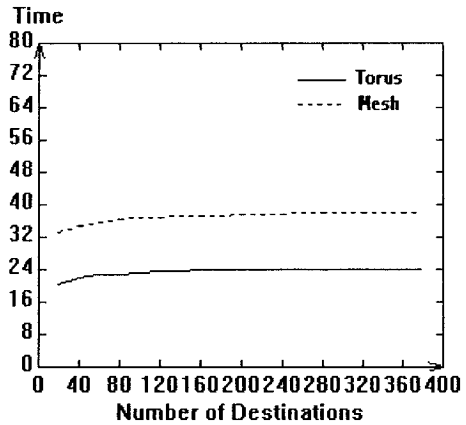


Figure 4.2.9: Multicast Time of DIAG in 2D Torus versus 2D Mesh

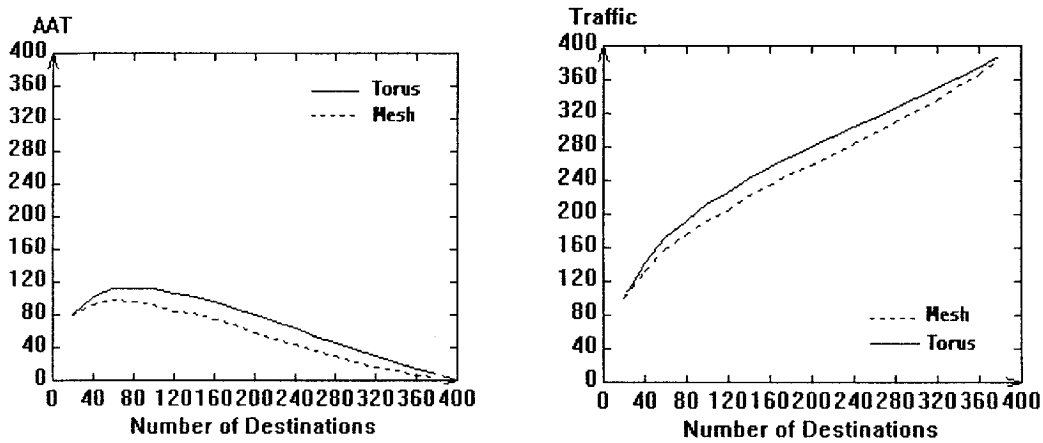


Figure 4.2.10: Multicast Traffic of DIAG in 2D Torus versus 2D Mesh

While the multicast time of *DIAG* in the 2D torus is less than that of the 2D mesh, the multicast traffic increases slightly. As we can see from *Figure 4.2.10,* the multicast traffic mean in the 2D torus is *255.99*, increased by *6%* over the 2D mesh's *241.38*. The *AAT* in the 2D torus is increased by about *22%* from *51.38* of the 2D mesh to *65.99*. This is probably because the torus is divided into four submeshes each of which forms a separate

multicast subtree of its own. Hence, destination nodes in one submesh can not receive messages from a nearby node in other submeshes but from the source node at the corner of its submesh which can be a long way away and creates a lot of inefficient traffic.

From the simulation results of *DIAG* algorithm in the 2D mesh and the 2D torus, we can notice that the *DIAG* algorithm in the 2D torus reduces the multicast time significantly by *40%* over that of the 2D mesh but at the cost of slightly more traffic.

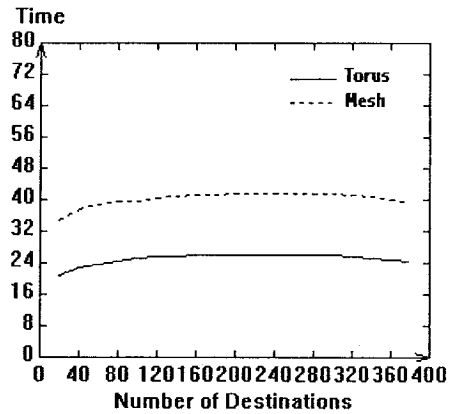


Figure 4.2.11: Multicast Time of DDS in 2D Torus versus 2D Mesh

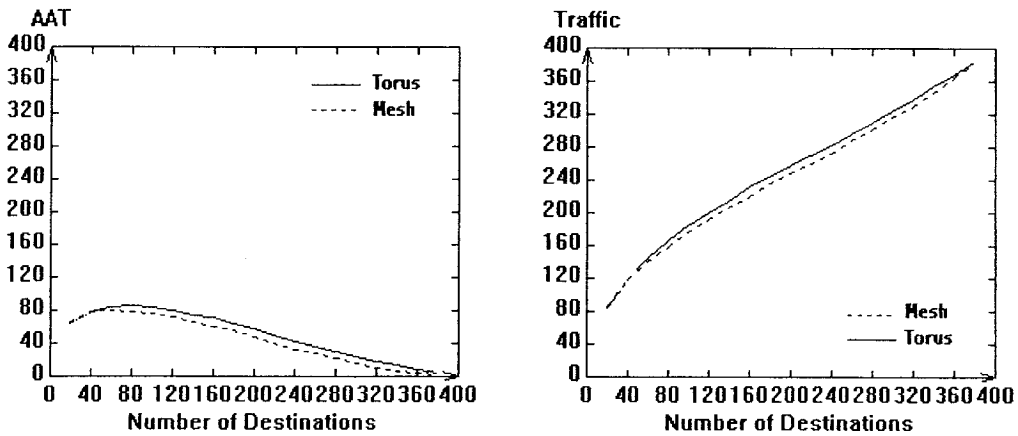


Figure 4.2.12: Multicast Traffic of DDS in 2D Torus versus 2D Mesh

Simulation results of *DDS* algorithm in the 2D mesh and 2D torus illustrated in *Figures 4.2.11* and *4.2.12* resemble those of *DIAG* algorithm.

4.2.4 Simulation of DDS and DIAG in 3D Mesh

In this section, we study the performance of the algorithms in 3D meshes through simulations done in a $10 \times 10 \times 10$ mesh.

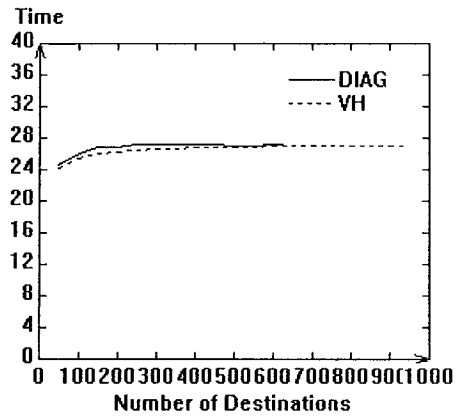


Figure 4.2.13: Multicast Time of DIAG versus VH in 3D Mesh

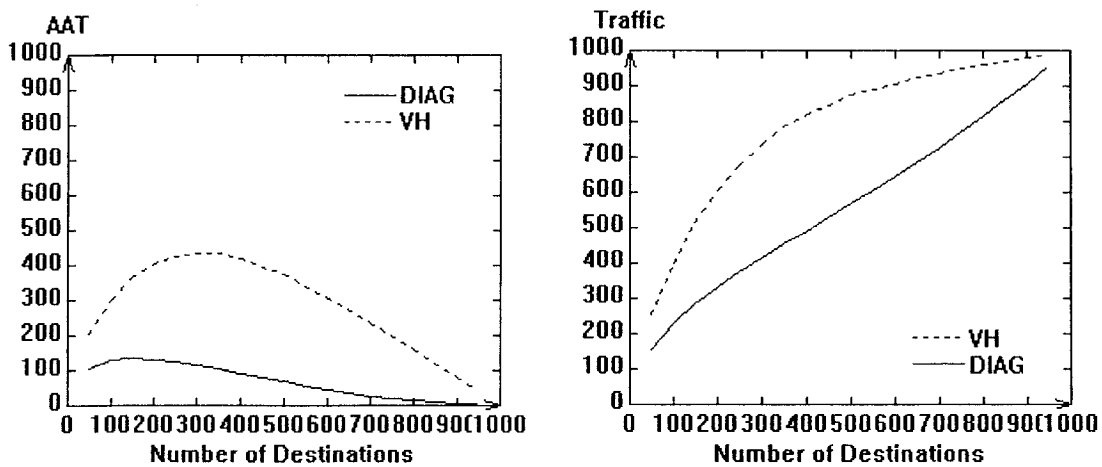


Figure 4.2.14: Multicast Traffic of DIAG versus VH in 3D Mesh

Figures 4.2.13 and *4.2.14* indicate that the results of *DIAG* and *VH* algorithms in 3D meshes are consistent with those in 2D meshes. The *AAT* mean of *DIAG* algorithm is *62.75*, which is *77%* less than *VH's 273.62*, an even bigger difference than in the 2D mesh.

Figure 4.2.15 shows that the multicast time mean of *DDS* in the 3D mesh is *28.24* which is *10%* more than *DIAG's 25.34* but *12%* less than *MIN's 32.05*, a larger difference than in the 2D mesh. We can also notice that the curve is not as flat as in the 2D mesh, and it is more like an arch. This is probably because when the number of destinations is very large, there may be fewer branching nodes in the tree, which causes less delay hops.

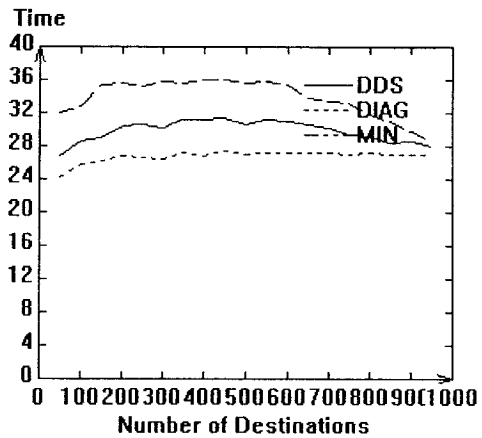


Figure 4.2.15: Multicast Time of DDS versus MIN in 3D Mesh

Figure 4.2.16 illustrates the simulation results on the multicast traffic of *DDS, DIAG* and *MIN* in the 3D Mesh. It seems that the difference between these algorithms in 3D meshes is not as big as that in 2D meshes.

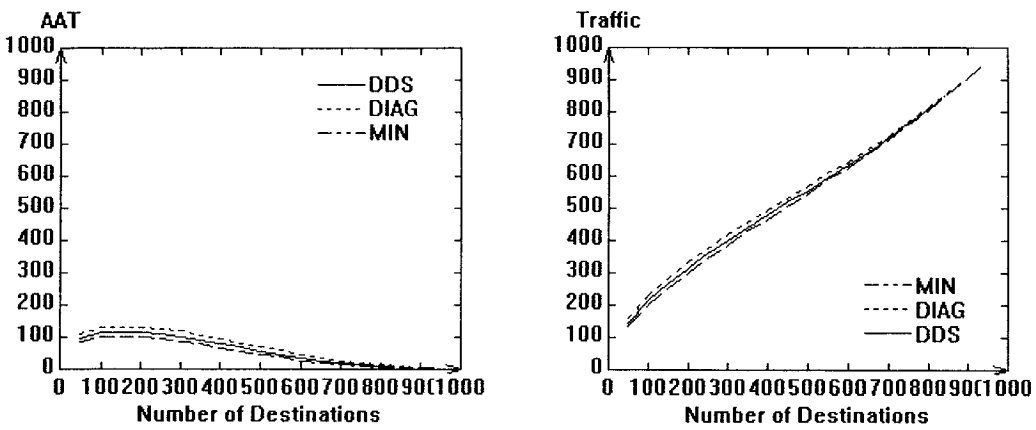


Figure 4.2.16: Multicast Traffic of DDS versus MIN in 3D Mesh

4.2.5 Simulation of DDS and DIAG in 3D Torus

In this section we study the performance of the algorithms in 3D tori through simulations done in a $10 \times 10 \times 10$ torus.

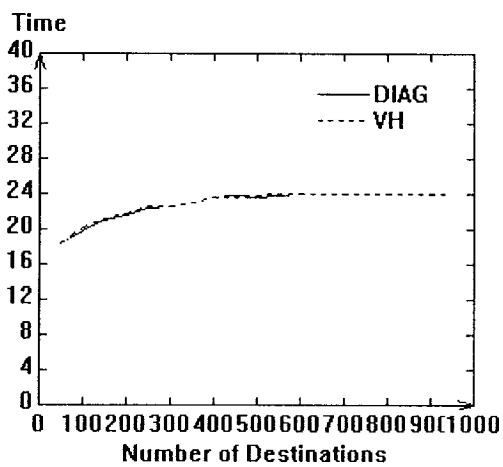


Figure 4.2.17: Multicast Time of DIAG versus VH in 3D Torus

Figure 4.2.17 and *Figure 4.2.18* show that the simulation results and conclusions of *DIAG* and *VH* in the 3D torus are similar to those in the 2D torus and/or 3D mesh.

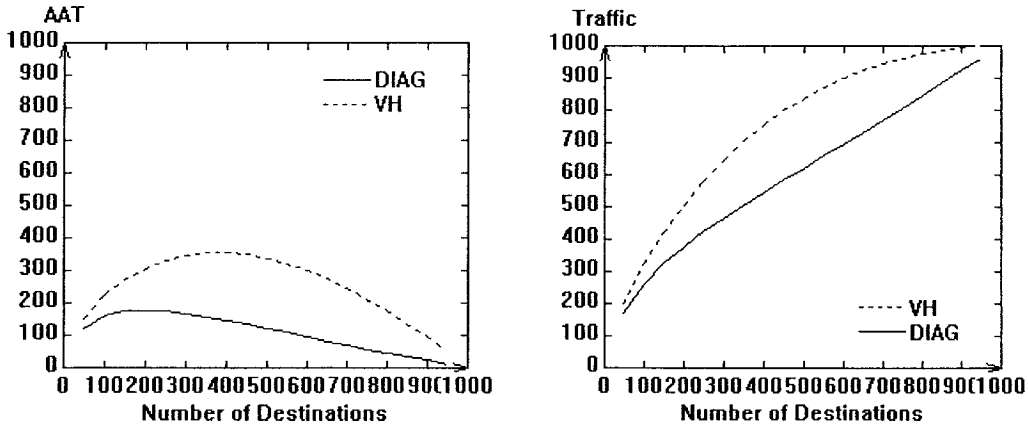


Figure 4.2.18: Multicast Traffic of DIAG versus VH in 3D Torus

Figure 4.2.19 and *Figure 4.2.20* show that the simulation results and conclusions of *DDS*, *DIAG* and *MIN* in the 3D torus are similar to those in the 2D torus and/or 3D mesh.

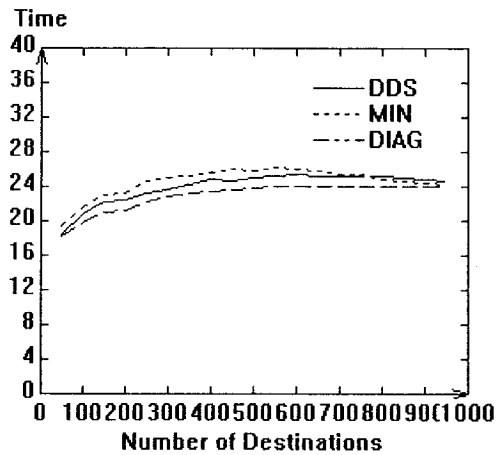


Figure 4.2.19: Multicast Time of DDS versus MIN in 3D Torus

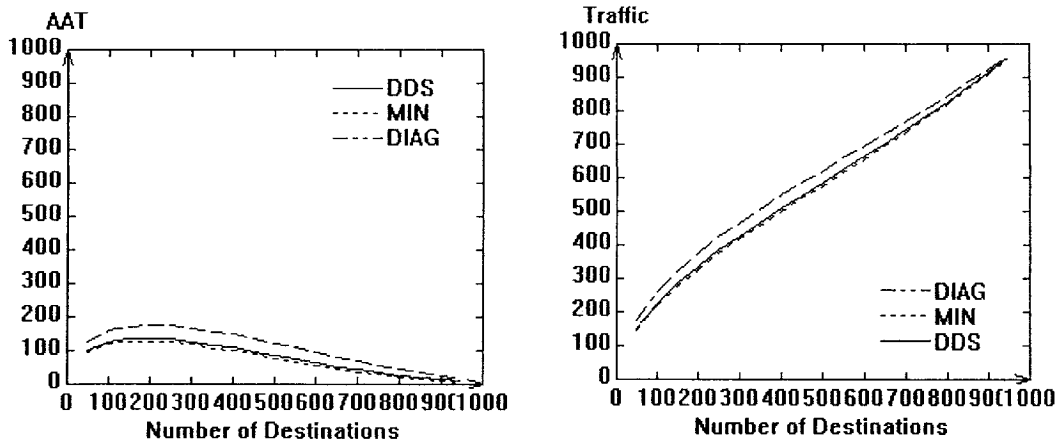


Figure 4.2.20: Multicast Traffic of DDS versus MIN in 3D Torus

4.3 Performance Evaluation of XY-path

4.3.1 Simulation Assumptions

In this section we will analyze the performance of the *XY-path* multicast algorithm. Since it is path-based and suitable for wormhole routed network, the simulation conditions are different from those of *DDS* and *DIAG* algorithms. Here are the assumptions:

- Simulations are done in a wormhole routed 2D-mesh network. To simplify the calculation, we assume the sizes of the two dimensions are the same.
- Nodes in the network have all-port architecture in order to increase the parallelism and obtain the best performance of the algorithm.
- Sampling resolution is *10* destination nodes per sample point. The value of each sample point is averaged over *1000* runs of multicasts with the same number of destination nodes [1, 7].
- The unit of time (a hop) is the time to transmit a flit through a link and the unit of traffic is link.

- The length of the message is fixed at *20* flits, which tends to be the average message length being delivered in a multicomputer network.
- Use distributed routing scheme, which has been introduced in Chapter 2, for message transmission along the multicast path.
- *R(u,v)*, which is introduced in *XY-path* algorithm, is the routing function executed at each node on the paths.
- Formulas to calculate the multicast time, traffic and additional traffic in our simulations are those for wormhole routed networks with all-port nodes which were introduced in earlier sections. Assume d_x and d_y are the length of the X and Y message passing paths respectively, then:

$$\textbf{\textit{Multicast Time}} = \max(d_x, d_y) + L \qquad (4.3.1)$$

$$\textbf{\textit{Multicast traffic}} = d_x + d_y \qquad (4.3.2)$$

$$\textbf{\textit{Additional traffic}} = d_x + d_y - K \qquad (4.3.3)$$

4.3.2 Simulation of XY-path in 2D Mesh

In this section we study the performance of *XY-path* algorithm compared to *LIN's* algorithm through simulations done in a 20×20 mesh.

Figure 4.3.1 indicates that the multicast time of *XY-path* algorithm is much less than that of *LIN's* algorithm. In this example the multicast time mean of *XY-path* is *185.83*, which is *48%* less than *LIN's 356.39*. The time curves of path-based algorithm climb quickly at the beginning, after the point of about half the total number of nodes, it

becomes very flat which means once the number of destinations are more than *50%*, the length of the path does not differ much.

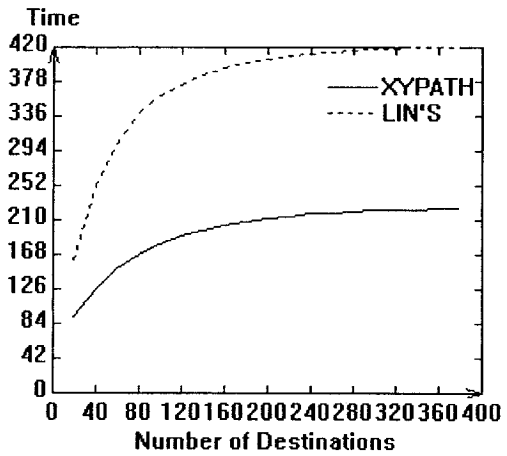


Figure 4.3.1: Multicast Time of XY-path versus LIN's in 2D Mesh

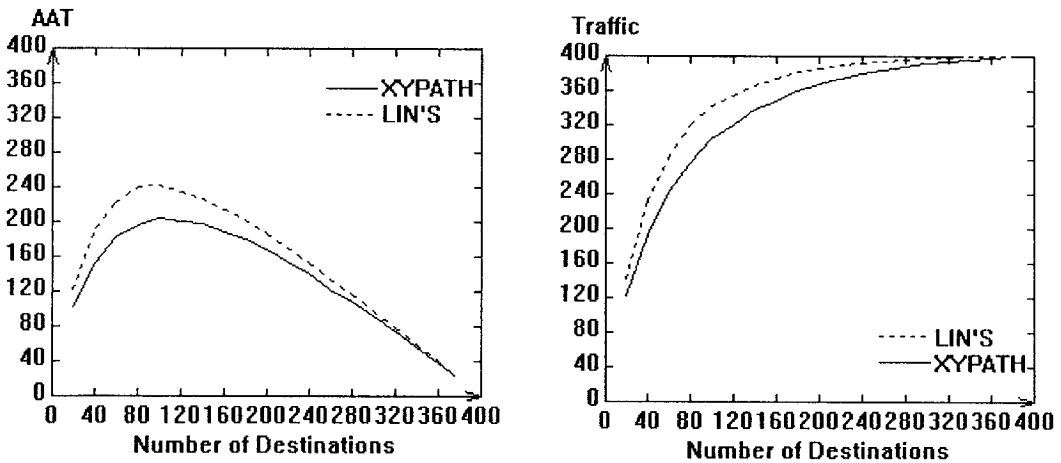


Figure 4.3.2: Multicast Traffic of XY-path versus LIN's in 2D Mesh

Figure 4.3.2 indicates that multicast traffic generally increases with the number of destinations, but *AAT* first increases with the number of destinations and reaches its peak at about *20%* percent of the total nodes. Then, it decreases with the number of

destinations. This means that when the number of destination nodes is very small or very large, the multicast traffic is more efficient. In between it will generate more inefficient traffic. In this example, the multicasts traffic mean of *XY-path* and *LIN's* algorithm are *317.59* and *336.39* respectively. The *AAT* mean of *XY-path* algorithm is *127.59*, which is *13%* less than *LIN's 146.39*.

From these simulation results of *XY-path* and *LIN's* algorithms in the 2D mesh, we can notice that *XY-path* algorithm reduces the multicast time by almost *50%* and reduces the additional multicast traffic by *13%* over *LIN's* algorithm without sacrificing the computation complexity.

Chapter 5

Conclusion and Future Work

Multicast problems in mesh-connected networks are NP-complete in general. Efficient multicast routing algorithms with good heuristics, that can minimize the time and reduce the traffic as much as possible, are needed to improve the performance of mesh-connected multicomputers under different circumstances. To tackle this challenge, we conducted a study in this field and achieved the following accomplishments and results.

- Designed the pro-time tree-based shortest path multicast routing algorithm called *DIAG* for store-and-forward switched mesh networks which significantly reduced the multicast traffic (by about *50%* in 2D meshes) over the previous *VH* algorithm without sacrificing the multicast time and complexity.
- Designed the pro-time tree-based shortest path multicast algorithm called *DDS* for store-and-forward switched mesh networks which further reduced the multicast traffic (by about *20%* in 2D meshes) over *DIAG* with only a slight increase (by *8%* in 2D mesh) on multicast time.
- Designed and implemented a generic program to simulate multicasting in mesh-connected multicomputers for performance evaluation, which can be easily customized for any size of meshes or tori, different switching techniques, and node architectures.

- Analyzed the performance of *DIAG* and *DDS* algorithms through simulation results with comparisons to previous ones in 2D and 3D meshes. We also presented the pros and cons of each algorithm under different circumstances.
- Designed the dual-path-based *XY-path* multicast routing algorithm for wormhole routed 2D mesh networks which significantly reduced the multicast time (by about *50%* in 2D meshes) and slightly decreased (by *15%* in 2D meshes) the multicast traffic over *LIN's* Hamiltonian path-based algorithm.
- Analyzed the performance of *XY-path* algorithm through simulations with comparison to *LIN's* algorithm in 2D meshes. We also presented the pros and cons of the algorithms under different circumstances.

Experiments and simulations in our research showed that each of these algorithms including *DIAG*, *DDS*, *VH*, *MIN* and *XY-path* has its strengths and weaknesses. None of them performs best at all cases. Hence, more attention and research will need to be directed to this field to develop good multicast routing heuristics for different network architecture, applications and performance requirements. The following are some of the work that can be done in the future.

- Simulate these algorithms in higher dimensional meshes and evaluate their performance with respect to the size and dimensions of meshes.
- Incorporate these multicast algorithms with deadlock preventing, fault tolerant, and traffic balancing heuristics, and get it ready for real applications.
- Develop algorithms optimized for concurrent multiple message multicasting.
- Develop pro-traffic multicast algorithms that target to minimize traffic first.

- Develop more multicast algorithms that are specially optimized for tori and/or wormhole routed networks since they are the more promising multicomputer technologies.

Bibliography

[1] X. Lin and L. M. Ni, *Multicast communication in multicomputer networks*, IEEE Trans. Parallel and Distributed Systems, vol. 4, pp. 1105--1117, October 1993.

[2] L. Ni and P.K. McKinley, *A survey of wormhole routing techniques in direct networks*, IEEE Computer 26 (2), pp. 62-76, 1993.

[3] X. Liu, *Multicasting algorithms for mesh and torus networks,* Call no. TK 5105.887 L58 2003, Concordia University Library, 2003.

[4] J.-Y. L. Park, H.-A. Choi, N. Nupairoj, and L. M. Ni. *Construction of Optimal Multicast Trees Based on the Parameterized Communication Model.* In Proc. Int. Conference on Parallel Processing (ICPP), Volume I, pp. 180--187, 1996.

[5] P. K. McKinley, H. Xu, A.-H. Esfahanian, and L. M. Ni. *Unicast-based Multicast Communication in Wormhole-routed Networks.* IEEE Transactions on Parallel and Distributed Systems, 5 (12), pp.1252--1265, Dec. 1994.

[6] P. Mohapatra, *Wormhole Routing Techniques for Directly Connected Multicomputer Systems*, ACM Computing Surveys (CSUR), Volume 30, Issue 3, pp. 374 – 410, 1998.

[7] X. Lin and L. M. Ni, *Deadlock-free multicast wormhole routing in multicomputer networks*, In Int. Symp. on Computer Architecture, pp. 116--125, 1991.

[8] J.-S. Yang and C.-T. King, *Efficient Tree-based Multicast in Wormhole-Routed 2D Meshes,* ISPAN, p. 494, 1997.

[9] P. K. McKinley, Y. Tsai, and D. F. Robinson, *Collective Communication in Wormhole Routed Massively Parallel Computers,* Computer, vol. 28, no. 12, pp. 39-50, De-

cember 1995.

[10] X. Lin, P.K. McKinley, and L.M. Ni, *Deadlock-Free Multicast Wormhole Routing in 2-D Mesh Multicomputers,* IEEE Transactions on Parallel and Distributed Systems, vol. 05, no. 8, pp. 793-804, August 1994.

[11] D. F. Robinson, P. K. McKinley, and B. H. C. Cheng, *Optimal Multicast Communication in Wormhole-Routed Torus Networks*, IEEE Transactions on Parallel and Distributed Systems, Volume 6 , Issue 10, pp. 1029 – 1042, October 1995.

[12] C. S. Yang, Y. M. Tsai, and C. Y. Liu, *Performance Evaluation of Multicast Wormhole Routing in 2D-Torus Multicomputers*, ICCI 1992, pp.173-178, 1992.

[13] K.-P. Fan and C.-T. King, *Optimal Multicast Communication in Wormhole-Routed Torus Networks*, Proceedings of the 6th Symposium on the Frontiers of Massively Parallel Computation, p. 50, 1996.

[14] E. Fleury and P. Fraigniaud, *Analysis of deadlock-free path-based wormhole multicasting in meshes in case of contentions,* Frontiers (6th), p. 34, 1996.

[15] A. Al-Dubai, M. Ould-Khaoua, and L. M. Mackenzie, *An efficient path-based multicast algorithm for mesh networks*, Proc. 17th Int. Parallel and Distributed Processing Symposium (IPDPS) Nice, France, IEEE Computer Society Press , pp. 283-290, 22 –26 April, 2003.

[16] H. Xu, P. K. McKinley, and L. M. Ni, *A Scalable Multicast Service in 2D Mesh Networks*, Frontiers'92: The 4th Symposium on the Frontiers of Massively Parallel Computation , pp.156--163, Oct. 1992.

[17] W. Jia, L. Cheng, and G. Xu, *Efficient Multicast Routing Algorithms on Mesh Networks,* Proceedings of the Fifth International Conference on Algorithms and Archite-

ctures for Parallel Processing (ICA3PP.02), 2002

[18] P. V. Mieghem, G. Hooghiemstra, and R. V. Hofstad, *On the Efficiency of Multicast*, IEEE/ACM Transactions on Networking, vol. 9, no. 6, pp. 719-732, December 2001.

[19] Motorola Inc., *Mobile Mesh Networks Technology*, http://www.motorola.com/,2004

[20] National Institute of Standards and Technology, *Dictionary of Algorithms and Data Structures,* http://www.nist.gov/dads, last updated 16:43:26, Mon. Jan. 3, 2005.

[21] Free Software Foundation, Inc, *Wikipedia, the free encyclopedia,* http://en.wikipedia.org/wiki/Main_Page, last modified 14:33, April 9, 2005.

[22] H. A. Harutyunyan and X. Liu, *New Multicast Algorithms in Mesh-connected networks*, International Symposium on Performance Evaluation of Computer and Telecommunication Systems (SPECTS)' 2003, pp. 284-291, Montreal, Canada, 2003.

[23] T.-S. Chen, C.-Y. Chang, and J.-P. Sheu, *Efficient path-based multicast in wormhole-routed mesh networks Source*, Journal of Systems Architecture, Volume 46 , Issue 10, pp. 919 – 930, 2000.

[24] E. Fleury and P. Fraigniaud, *Strategies for path-based multicasting in wormhole-routed meshes*, Journal of Parallel and Distributed Computing, Volume 53, Issue 1, pp. 26-62, August 1998.

[25] R. Cypher and L. Gravano. *Storage-efficient deadlock-free packet routing algorithms for torus networks*, IEEE Trans. on Computers, vol. C-43, no. 12, pp. 1376-1385, 1994.

[26] S.-Y. Wang, Y.-C. Tseng, and C.-W. Ho, *Efficient Multicast in Wormhole-Routed 2D Mesh/Torus Multicomputers: A Network-Partitioning Approach,* Symposium on

the Frontiers of Massively Parallel Computation, pp. 42-49, 1996.

[27] P. Mohapatra and V. Varavithya, *A Hardware Multicast Routing Algorithm for Two-Dimensional Meshes,* SPDP (8th), p. 198, 1996.

[28] J. Wu, *Maximum-Shortest-Path (MSP): An Optimal Routing Policy for Mesh-Connected Multicomputers,* IEEE Transactions on Reliability, 48 (3), pp. 247-255, Sept. 1999.

[29] B. D. Birchler, A.-H. Esfahanian, and E. Torng, *Sufficient Conditions for Optimal Multicast Communication*, ICPP'97, 1997.

[30] K.-P. Fan and C.-T. King. *Turn grouping for efficient multicast in wormhole mesh networks,* Frontiers (6th), p. 50, 1996.

[31] S.-H. Sheu and C.-B. Yang, *Multicast algorithms for hypercube multiprocessors Source*, Journal of Parallel and Distributed Computing, Volume 61, Issue 1, pp. 137–149, 2001.

[32] Y. Lan, A.-H. Esfahanian, and L. M. Ni, *Multicast in hypercube multiprocessors*, Journal of Parallel and Distributed Computing, Volume 8 , Issue 1, pp. 30– 41, 1990.

[33] N. Nupairoj, L.M. Ni, J. Park, and H. Choi, *Architecture-Dependent Tuning of the Parameterized Communication Model for Optimal Multicasting,* Proceedings of the 1997 International Parallel Processing Symposium *(IPPS'97),* University of Geneva, Switzerland, April 1997.

[34] Z. Liu and J. Duato, *Adaptive Unicast and Multicast in 3D Mesh Networks*, Proc. 27th Hawaii Int'l Conf. System Sciences, pp.173-82, Jan. 1994.

[35] A. Shaikh, S. Lu, and K. Shin, *Localized multicast routing*, Proceedings of Globecom'95, pp. 1352-6, 1996.

[36] J.H. Park, H.G. Kim, S.T. Hwang, J. Kim, I. Jang, and H. Yoon, *An Efficient Unicast-Based Multicast Algorithm in Two-Port Wormhole-Routed 2D Mesh Networks*, Proceedings of IEEE Second International Conference on Algorithms And Architectures for Parallel Processing (ICA3PP'96), 1996.

[37] H. Wang and D. M. Blough, *Tree-Based Multicast in Wormhole-Routed Torus Networks*, Technical Report ECE-98-03-01, University of California, Irvine, March 1998.

[38] S. Oral and A. George. *Multicast Performance Analysis for High-Speed Torus Networks,* LCN (27th), p. 0619, 2002.

[39] H. Wang and D. M. Blough, *Tree-Based Fault-Tolerant Multicast in Multicomputer Networks*, Proceedings of the 1998 IEEE International Symposium on Modeling and Analysis of Computer and Telecommunication Systems, pp. 44-49, 1998.

[40] D. Kumar, W. A. Najjar, and P. K. Srimani, *A New Adaptive Hardware Tree-Based Multicast Routing in K-ary N-cubes*, IEEE Transactions on Computers, vol. 50, no. 7, pp. 647-659, July 2001.