This is to certify that the

dissertation entitled

Multicast Communication
in Multicomputer Networks

presented by

Xiaola Lin

has been accepted towards fulfillment
of the requirements for

Ph.D degree in Computer Science

Lionel M. Ni
Major professor

Date May 19, 1992

MULTICAST COMMUNICATION IN MULTICOMPUTER NETWORKS

By

Xiaola Lin

A DISSERTATION

Submitted to
Michigan State University
in partial fulfillment of the requirements
for the degree of

DOCTOR OF PHILOSOPHY

Department of Computer Science

1991

ABSTRACT

MULTICAST COMMUNICATION IN MULTICOMPUTER NETWORKS

By

Xiaola Lin

Efficient routing of messages is a key to the performance of multicomputers. Multicast communication service in multicomputers is to deliver the same message from a source node to an arbitrary number of destination nodes. Multicast communication is finding increased demand in many applications, including various simulations, image processing, and numerous parallel algorithms.

Providing support for multicast communication involves several, often conflicting, requirements. First, it is desirable that the message delay from the source to each of the destinations be as small as possible. Second, the amount of network traffic must be minimized. Third, the routing algorithm must not be computationally complex. Finally, the multicast communication protocol must be deadlock-free.

In this dissertation, the above issues are addressed. According to the different switching technologies and routing criteria, various multicast models, *multicast path, multicast cycle, Steiner tree,* and *multicast star*, have been proposed. We show that finding optimal route for each proposed model is NP-hard for the most common multi-

computer topologies, such as hypercube and mesh. Basic heuristic routing algorithms for these models are presented. Based on node labeling and network partitioning strategies, various deadlock-free routing schemes, *tree-like, dual-path, multi-path, and fixed path methods*, have been presented. These are the first deadlock-free multicast wormhole routing algorithms ever proposed. We also conduct a series of simulation studies to evaluate the performance of these multicast algorithms under both static and dynamic network traffic conditions. The programs are written in C and use an event-drive simulation package, CSIM.

In summary, this dissertation research focuses on the basic issues of multicast communication: multicast routing models, investigation of the optimal routing for the models, development of heuristic algorithms, deadlock avoidance, and performance study based on both static and dynamic network traffic.

To my parents, Xiao Ju-Xiang and Lin Yi.

ACKNOWLEDGEMENTS

I would like to thank Professor Lionel M. Ni for providing me with the opportunity to perform this research and continue my graduate studies under his direction. His suggestions and guidance were invaluable to the completion of this work. Professor Wen-Jin Hsu has been my co-advisor. His help in my studies and his advice on my research enabled me to overcome a host of obstacles.

Professor Philip K. McKinley's contributions are gratefully acknowledged. The discussions we had were always enlightening, and his recommendations helped to improve the quality of this research.

The careful review of this documentation by other Ph.D committee members, Professor Abdol H. Esfahanian, Professor Byron Drachman, and Professor Sakti Pramanik is greatly appreciated.

The graduate assistantship I received from the Case Center for CAEM of Engineering College was critical to the completion of this work. I wish to thank Professor Erik Goodman, Case Center director, for his financial support. Special thanks also go to Jackie Carlson, John Lees, Robert Raicsh, and Susan Smith for their kindness and friendship during my stay in Case Center.

I am grateful to my parents and my sisters for their love and encouragement. I am also thankful to Shiqi He, Liangsheng Cao, and Zhonggang Zeng for their assistance and friendship in these years.

TABLE OF CONTENTS

LIST OF TABLES ix

LIST OF FIGURES x

1 INTRODUCTION 1
- 1.1 Interprocessor Communication 4
- 1.2 Motivation and Problem Statements 6
- 1.3 Dissertation Organization 7

2 MULTICOMPUTER NETWORK AND MULTICAST COMMUNICATION 9
- 2.1 Topology of Multicomputer Network 10
  - 2.1.1 Hypercube Topology 11
  - 2.1.2 2D Mesh Topology 13
  - 2.1.3 Other Topologies 14
- 2.2 Switching Technology 15
  - 2.2.1 Store-And-Forward Switching 16
  - 2.2.2 Virtual Cut-Through Switching 16
  - 2.2.3 Circuit Switching 17
  - 2.2.4 Wormhole Routing 18
- 2.3 Message Routing and Flow Control 20
  - 2.3.1 Source Routing and Distributed Routing 20
  - 2.3.2 Deterministic Routing and Adaptive Routing 20
  - 2.3.3 Flow Control 21
  - 2.3.4 Deadlock Issues 22
- 2.4 Issues of Multicast Communication 26

3 MODELS FOR MULTICAST COMMUNICATION 28
- 3.1 Multicast Path and Cycle Problems 30
- 3.2 Steiner Tree Problem 32
- 3.3 Multicast Tree Problem 33

3.4 Multicast Star Problem 34

4 OPTIMAL MULTICAST IN MULTICOMPUTERS 36

4.1 Optimal Multicast in 2D Mesh Topology 36

4.2 Optimal Multicast in Hypercube Topology 41

4.3 Optimal Multicast in 3D Mesh Topology 49

5 BASIC HEURISTIC MULTICAST ROUTING ALGORITHMS 50

5.1 Heuristic Routing Algorithms for MP and MC 51

5.2 Heuristic Routing Algorithms for ST 55

5.3 Heuristic Routing Algorithms for MT 60

5.4 Illustrative Examples 64

6 DEADLOCK-FREE MULTICAST WORMHOLE ROUTING 73

6.1 Deadlock Issue in Multicast Wormhole Routing 73

6.2 Deadlock-Free Multicast in 2D Mesh 78

6.2.1 Tree-Like Deadlock-Free Multicast Routing Schemes 78

6.2.2 Path-Like Deadlock-Free Multicast Routing Schemes 83

6.3 Deadlock-Free Multicast in Hypercube 94

7 PERFORMANCE STUDY OF THE ROUTING SCHEMES 101

7.1 Performance Study under Static Network Traffic 101

7.2 Performance Study under Dynamic Network Traffic 106

8 CONCLUSIONS 113

8.1 Summary of Major Contributions 113

8.2 Direction for Future Research 115

BIBLIOGRAPHY 118

LIST OF TABLES

5.1 A Hamilton cycle and the corresponding mapping h of a 4×4 mesh. 65

5.2 The sorting key $f(x)$ and mapping $h(x)$ for each node x in a 4×4 mesh, where $u_0 = 9$. 65

5.3 A Hamilton cycle and the corresponding mapping h of a 4-cube. . . . 66

5.4 The sorting key $f(x)$ and mapping $h(x)$ for each node x in a 4-cube, where $u_0 = 9$. 67

LIST OF FIGURES

1.1 (a) A generic multicomputer architecture; (b) A generic node architecture. 2
1.2 A multicast communication pattern with three destinations. 5

2.1 Hypercube topology with 3 dimensions. 12
2.2 A 4 × 3 mesh. 14
2.3 A comparison of different switching technologies. 19
2.4 An example of deadlock involving four messages. 23
2.5 (a) A direct network I; (b) Channel dependency graph for I and X-first routing. 26

3.1 An example of multicast path in 2D mesh. 31
3.2 An example of Steiner tree in 2D mesh. 32
3.3 An example of multicast tree in 2D mesh. 33
3.4 An example of multicast star in 2D mesh. 35

4.1 The construction of G' from G. 39
4.2 A simple grid graph with 8 nodes. 43

5.1 The sorted MP algorithm for message preparation 52
5.2 The sorted MP algorithm for message routing 53
5.3 The greedy ST algorithm for message preparation 57
5.4 The greedy ST algorithm for message routing 58
5.5 X-first multicast algorithm 61
5.6 Divided greedy algorithm 63
5.7 A 4 × 4 mesh with node 9 as the source. 64
5.8 A 4-cube with node 9 as the source. 66
5.9 A complete ST routing pattern in an 8 × 8 mesh. 68
5.10 A complete ST routing pattern in a 6-cube. 69
5.11 The routing pattern of X-first algorithm for the example. 70
5.12 The routing pattern of divided greedy algorithm for the example. . . 71

6.1 Deadlock in a 3-cube multicomputer. 75
6.2 The detailed diagram of deadlock configuration. 75
6.3 An X-first multicast routing pattern. 76
6.4 A deadlock situation in a 3×4 mesh. 77
6.5 Network partitioning for 3×4 mesh. 79
6.6 Double-channel X-first routing algorithm. 80
6.7 The routing pattern of the tree algorithm. 81
6.8 The label assignment for $N_{+X,+Y}$ for channel ordering. 82
6.9 The labeling of a 4×3 mesh. 84
6.10 The other label assignment and channel partitioning for a 4×3 mesh. 85
6.11 Message preparation for the dual-path routing algorithm. 87
6.12 The dual-path routing algorithm. 88
6.13 An example of dual-path routing in a 6×6 mesh. 89
6.14 Message preparation for the multi-path routing algorithm in 2D mesh. 91
6.15 Multi-path destination address partitioning. 92
6.16 An example of multi-path routing in a 6×6 mesh. 93
6.17 An example of fixed-path routing in a 6×6 mesh. 94
6.18 The label assignment and the corresponding high-channel and low-channel networks for a 3-cube. 95
6.19 A 4-cube with node 1100 as the source for daul path routing. 98
6.20 Message preparation for the multi-path routing algorithm in hypercube. 99
6.21 A 4-cube with node 1100 as the source for multi-path routing. 100

7.1 Performance of the sorted MP algorithm on a 32×32 mesh. 102
7.2 Performance of the sorted MP algorithm on a 10-cube. 103
7.3 Performance of the greedy ST algorithm on a 32×32 mesh. 104
7.4 Performance of the greedy ST algorithm on a 10-cube. 104
7.5 Performance of the X-first and divided greedy algorithms on a 16×16 mesh. 105
7.6 Performance of different multicast methods on a 6-cube. 106
7.7 Performance of different number of destinations on 8×8 mesh. 107
7.8 Performance under different loads on a double-channel mesh. 108
7.9 Performance of different number of destinations on a double-channel mesh. 109
7.10 Performance under different loads. 110
7.11 Performance of different number of destinations. 111

CHAPTER 1

INTRODUCTION

Parallel processing is rapidly becoming a dominant theme in all areas of computer science and its applications. Some experts [1] believe that it is likely that virtually all developments in computer architecture, system programming, computer applications and the design of algorithms will be centered around the parallel computation within a decade.

Parallel computers are those systems that emphasize parallel processing, including pipeline computers, array processors and multiprocessor systems [2].

Multicomputers are a class of MIMD multiprocessor systems composed of a large number of processors interconnected together by message-passing networks with some fixed topology [3]. Each processor or node is a programmable computer with its own CPU, local memory, and other supporting devices. Multicomputers are also referred to as *distributed-memory multiprocessors.* Figure 1.1(a) shows a generic multicomputer in which a set of nodes are connected through a multicomputer network. Figure 1.1(b) is the structure of a generic node. The router in the node is responsible for handling message passing for communication among the nodes. A dedicated router is required to allow computation and communication overlapped to support a high performance computing system.

The key characteristics of a multicomputer that distinguish it from other parallel

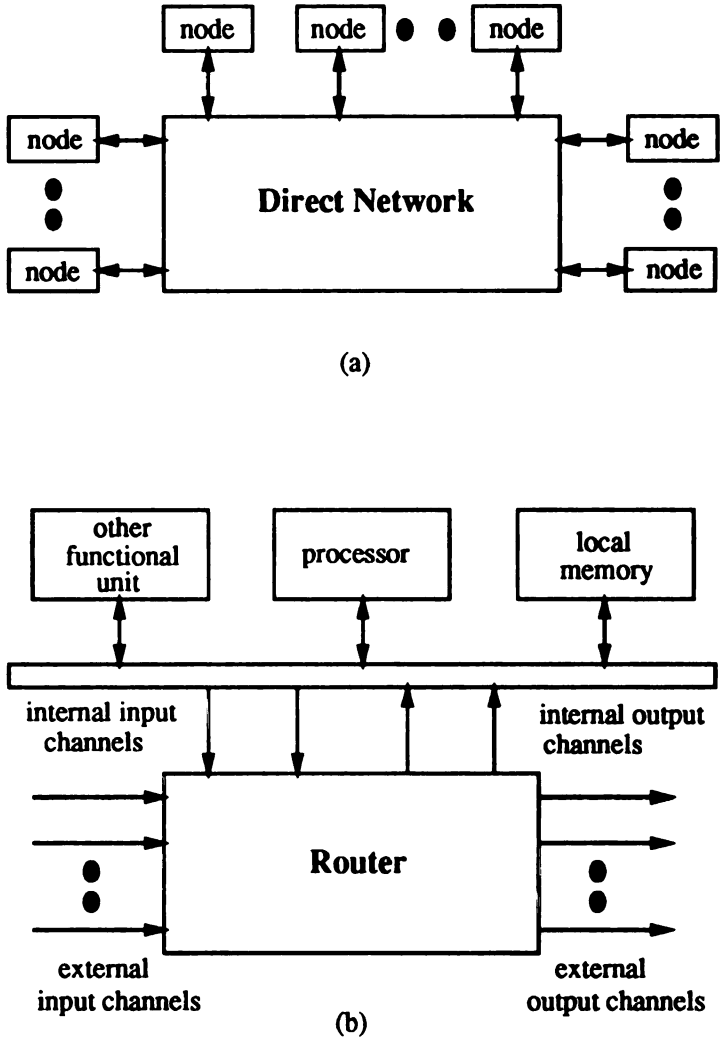


Figure 1.1. (a) A generic multicomputer architecture; (b) A generic node architecture.

processor systems are briefly summarized as follows [4]:

(1) *Large number of processors.* Because microprocessors are becoming inexpensive, easily packaged and consume little power, it is economically practical to construct systems with hundreds or thousands of nodes. Multicomputers are able to provide massive parallelism by having a large number of nodes. For example, FPS T-series [5] can have up to 2^{14} processors. nCUBE-1 has 1024 nodes, and the new nCUBE-2 can have up-to 8192 nodes [6]. A multicomputer of a few hundred nodes can have a peak performance exceeding that of today's supercomputers.

(2) *Message based communications.* There is no globally shared memory in multicomputer. Message passing is the only means for information exchange between nodes. It thus can reduce the software bottleneck.

(3) *Asynchronous execution.* Each node executes independently of all other nodes. Synchronization between nodes relies on message passing primitive. It is a MIMD machine.

(4) *Moderate communication overhead.* Delays in a well-designed multicomputer network should not exceed a few hundred microseconds, in contrast to milliseconds in loosely coupled distributed systems such as a network of workstations.

(5) *Medium grained computation.* A parallel program consists of a number of "tasks". The "grain" of a computation denotes the size of these tasks and it can be measured by the amount of computation between task interactions. In order to exploit the parallelism of the parallel system, the program should be decomposed into many small tasks. However, an extremely fine computation granularity will increase communication and task swapping overheads. Multicomputer are best suited for medium grain to balance the desire for massive parallelism against these overheads.

The appearance of multicomputers raises many challenging design and performance issues. Basically, two major factors affect the performance of a multicomputer: the computational power of each node and interprocessor communication of

the multicomputer network. With the development of VLSI technology, the nodes are becoming more powful and faster. An efficient communication mechanism should be able to pass message at the right time to keep all nodes busy to achieve a high parallelism. Interprocessor communication will dominate computing cost in both hardware and software [4].

1.1 Interprocessor Communication

At the system level, depending on the number of destinations, interprocessor communication can be classified into three types: unicast, broadcast, and multicast communication.

Unicast (one-to-one) communication is to send a message from a source node to one destination node. It is the basic type of communication and it has been directly supported by all current multicomputers. *Broadcast (one-to-all)* is a type of message delivery that a source node sends a message to all other nodes. This type of communication has also extensively studied in the past [7], it is directly supported in nCUBE-2 using wormhole routing [6].

Multicast (ono-to-many) communication refers to the delivery of the same message from a source node to an arbitrary number of destination nodes. Figure 1.2 shows a multicast communication pattern with three destinations, where process P_0 has to send the same message to three processes: P_1, P_2 and P_3 (here we assume each process resides in a separate node). Such a communication pattern is typical in many parallel and distributed simulations, such as circuit simulation, Petri net simulation, simulation of computer networks and queuing networks, and some other parallel algorithms. In these cases, the application of multicast is straightforward as the output of a gate (or server, place) may become the input of some connected gates. Parallel algorithms for such applications require proper partitioning and mapping strategies

to achieve a balance between granularity and communication overhead.

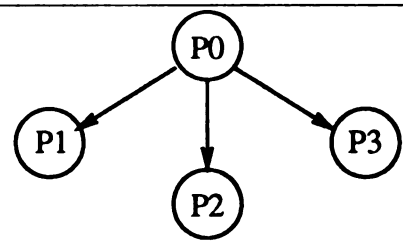


Figure 1.2. A multicast communication pattern with three destinations.

If multicast communication primitives are not supported, the program structure of the communication pattern in Fig. 1.2 is shown below.

```
P0: ......      P1: ......      P2: ......      P3: ......
    send(msg,P1)     ......          ......          ......
    send(msg,P2)    recv(msg,P0)   recv(msg,P0)   recv(msg,P0)
    send(msg,P3)     ......          ......          ......
```

Multicast communication, although highly demanded in the development of many parallel algorithms, has not received much attention. As shown above, multicast communication can be supported by multiple one-to-one communications. Assume that both **send** and **recv** are synchronous operations. If P_0 is executing **send(msg,P1)** and P_1 has not yet executed the **recv** statement, P_0 is blocked. In the mean time P_2 is executing the **recv** statement and is blocked because P_0 has not yet executed **send(msg,P2)**. Obviously, system resources are wasted due to the unnecessary blocking. Because of the non-determinism and asynchrony properties of multicomputers there is no way for P_0 to determine a proper message passing sequence *in priori*. Furthermore, some of these replicated messages may traverse the same communication channel, which creates much traffic than needed.

1.2 Motivation and Problem Statements

Efficient routing of messages is critical to the performance of multicomputers. Historically, commercial multicomputers have supported only single-destination, or *unicast*, message passing. More recently, multicomputers have begun to provide *multicast* communication services, in which the same message is delivered from a source node to an arbitrary number of destination nodes [6] [8]. The multicast primitive has been shown to be highly useful in many parallel algorithms [9] [10] [11]. A growing number of parallel applications can benefit from multicast services.

- In parallel simulation, an event modeled at one processor may cause a chain reaction of events that are to be modeled at other processors. The causal information must be communicated to those other processes [12].

- In parallel search algorithms, a set of processes collectively solve a decision or optimization problem. Examples include parallel alpha-beta search and parallel algorithms for artificial intelligence problems [13]. Processes in such applications typically search some global state space and must inform one another concerning their findings as well as pruning information.

- In image processing and pattern recognition, parallel processes operate on different areas of an image and must exchange information in order to identify complex objects and identify changes in the image since a previous image was generated [14] [10].

- In parallel graph algorithms, such as finding multiple shortest paths [15], information concerning the characteristics of the graph discovered by one process affects the behavior of other processes.

- In numerical algorithms, such as finding steady-state solutions of power flow equations [16], it is common for iterations of a loop to be done in parallel. If

some steps in an iteration depend on results from previous iterations, then all of the processes must synchronize after executing earlier steps. That is, none of the other processes will proceed with the dependent step until all executed earlier steps. This "barrier synchronization" can be efficiently implemented using multicast communication [17].

A software approach for multicast communication in hypercube multicomputers was proposed in [18]. The Cosmic Environment, a popular message-based parallel programming language, developed at Caltech also supports multicast communication [8]. A better heuristic multicast algorithm for the hypercube was proposed in [19] [20]. In [19], it was conjectured that finding an optimal multicast communication tree in terms of time and traffic is NP-hard in a hypercube graph. Furthermore, a VLSI router to handle multicast communication based on the virtual cut-through communication mechanism has been successfully prototyped [21].

Realizing the importance of multicast communication to multicomputers, the objective of this dissertation research is to study the major issues of multicast communication, including the modeling of the multicast patterns, investigation of the complexity properties of the optimal routing problems, development of efficient distributed routing algorithms, deadlock issues and performance study. We will state these issues in detail in the following chapter.

1.3 Dissertation Organization

This chapter introduces the demand of high performance parallel computers, especially the use of multicast communication in multicomputers. The motivation of this dissertation research is also presented.

The next chapter gives a brief review for the basic properties of multicomputer networks. The main issues of multicast communication will be addressed.

Based on various routing evaluation criteria and switching technologies, various multicast models are proposed in Chapter 3.

Chapter 4 examines the computational complexities of the proposed optimal multicast problems for 2D mesh, hypercube, and 3D mesh topologies. It shows that all of the optimal multicast problems are NP-complete.

In Chapter 5, basic heuristic algorithms are proposed for the multicast models in 2D mesh and hypercube multicomputers.

Chapter 6 focus on deadlock-free multicast wormhole routing. It is shown that the broadcast wormhole routing adopted in the nCUBE-2 is not deadlock-free, a property that is essential to wormhole routed networks. Several multicast wormhole routing strategies for 2D mesh and hypercube multicomputers are proposed and studied. All of the algorithms are shown to be deadlock-free.

A simulation study has been conducted that compares the performance of these multicast algorithms under both static and dynamic network traffic conditions. The performance study results are presented in Chapter 7.

The last chapter summarizes the dissertation research work and proposes the direction of the related future research.

CHAPTER 2

MULTICOMPUTER NETWORK AND MULTICAST COMMUNICATION

The critical component of a multicomputer is its communication network. Multicomputer networks are used to pass messages between the nodes in multicomputers. An interconnection network is characterized by its topology, switching technology, routing scheme and flow control mechanism. The *topology* defines the way the nodes are interconnected by channels, which is usually modeled as graph. A *switching* mechanism transfers data from an input channel to an output channel during the message passing process. *Routing* determines the path(s) selected for messages to reach the destination(s). *Flow control* deals with the allocation of channels and buffers to a message as it travels along the path.

Multicomputer networks have become a popular architecture for building large-scale multiprocessors to offer massive parallelism. The issues of interprocessor communication are basically related to the structure of a multicomputer network. In this chapter, we discuss the general properties of multicomputer networks. Multicast communication issues will also be addressed.

2.1 Topology of Multicomputer Network

A key decision in design of a multicomputer is the choice of the topology of its interconnection network. A good network topology should be able to accommodate a large number of nodes while minimizing the message transmission time. It is desirable that it could connect each node to every other node like the connection in a complete graph. Message would be delivered directly without passing through any intermediate node. The number of physical connections, however, would be extremely large, especially when the number of the nodes increases. The hardware constrains such as the number of the available pins and pads on the router as well as the communication area preclude such a completed connected network. Therefore, all of the multicomputers with relatively large size adopt a fixed topology.

The following factors should be considered in selecting and evaluating multicomputer network topology:

number of the connection: the number of channels used to connect the nodes, to get a reasonable wire complexity, it should be not massive;

regularity: to make the design easy, the network looks alike from every node, all of the nodes have the same or nearly the same degree;

diameter: the maximum distance between all of the pairs of nodes, a smaller diameter has smaller worst case delay caused by passing message through the intermediate nodes;

scalability: the system size can be increased by easily adding new nodes to the network;

routing: the easy of designing of deadlock-free and efficient routing algorithm;

robustness: the fault-tolerant ability, there should exist alternate paths between pairs of nodes;

throughput: the total number of messages the network can handle per unit time, the network should be able to efficiently handle various traffic distributions.

Some of the factors are related, others may be conflicting. Therefore, trade-off must be taken for selecting the network topology. In our study for multicast communication, we will focus on two popular topologies: n-cube and 2D mesh. Various network topologies have been proposed and studied. We discuss the properties of these topologies in more detail as follows.

2.1.1 Hypercube Topology

All first generation multicomputers adopt n-cube or *hypercube* topology due to its rich topological properties [22]. Most second generation multicomputers also adopt the hypercube topology, such as iPSC-2 [23] and nCUBE-2 [6].

An n-dimensional hypercube, also known as an n-cube, is a multicomputer with N, $N=2^n$ nodes interconnected as an n-dimensional binary cube. There are 2^n distinct n-bit binary addresses or labels that may be assigned to each node. A node's address differs from that of each of its n neighbors in exactly one bit position. Figure 2.1 illustrates a hypercube with 3 dimensions.

It has been known that hypercube topology has many nice features that make it suitable for the multicomputer network [24]. One of the most important advantages of hypercube topology is that meshes of all dimensions and trees can be embedded in a hypercube so that neighboring nodes are mapped to neighbors in the hypercube. The communication structures used in the fast Fourier transform and bitonic sort algorithm can be embedded easily in the hypercube. As we know, a great many scientific applications use mesh, tree, FFT, or sorting interconnection structures, the

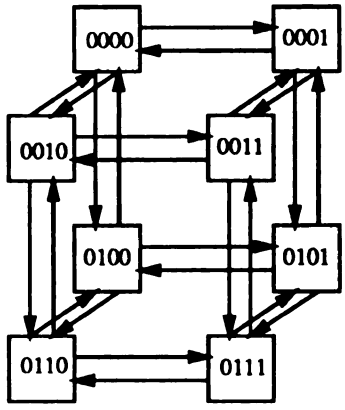


Figure 2.1. Hypercube topology with 3 dimensions.

hypercube is a good choice for general-purpose multicomputer networks. The other major advantage is that a hypercube's maximum internode distance (diameter) is only n, $n=\log_2 N$, N is the size of the hypercube, so that any two nodes can communicate fairly rapidly. Although the diameter is greater than the unit diameter of a complete connected topology, it is achieved with nodes having a degree of $n=\log_2 N$ compared with the $N-1$ degree of nodes in complete connection. Another important feature is there are $d!$ distinct paths of length d between two nodes of distance d in the hypercube.

There are additional features of hypercube topology that are useful in designing multicomputers. For example, it is homogeneous in that all nodes look the same, an I/O channel can be added to each node to get an extremely high I/O rates. Also, there are many ways to partition a hypercube into subcubes to support multiprocessing. Different users can work on different subcube in the same hypercube. This feature makes a system more tolerant faults since the operating system can allocate subcubes that contain no faulty processors or faulty channels. In summary, hypercube topology balances node connectivity, communication diameter, algorithm embeddability, and programming easy. This balance makes it suitable for broad class of computational

problems.

However, it was shown in [25] that, in wormhole networks, low-dimension networks (e.g., 2D mesh) have lower latency and higher hot-spot throughput than high-dimension networks such as hypercubes with the same bisection width. Further, low dimensional networks are more scalable than high dimensional networks. For instance, to increase the size of hypercube with 2^k nodes, at least 2^k nodes must be added. The size of a hypercube is limited by the degree of each node that is usually fixed. For 2D mesh the degree of each node is a constant, 4 or 2. The system can be easily upgraded with additional rows or columns, and there is no size limitation imposed by the degree of the node. We will examine the topology properties of 2D mesh topology that is low dimensional network.

2.1.2 2D Mesh Topology

As shown in Fig. 2.2, a node in a 2D mesh has connections to at most four neighbors. Some second generation multicomputers such as Ametek 2010 uses 2D mesh topology [26]. Intel's "touchstone" next generation multicomputer will also adopt 2D mesh topology due to its desirable properties of regularity, low cross-section bandwidth, low fixed degree of channels, and scalability.

VLSI systems are wire limited [25]. The complexity of the connection is limited by the wire density and the performance is also limited by the delay of interconnection. *Bisection density* is used to account for wire density, which is the product of the width of a channel and bisection width that is in turn defined as the minimum number of channels required to divided network into two equal halves. It is not hard to see that low-dimensional networks have a wider channel bandwidth, and thus a higher channel bandwidth, assuming that all the networks have the same bisection density. 2D mesh is low-dimensional network. For wormhole networks, it has been shown by theoretical analysis and simulation [25] that low-dimensional networks reduce contention

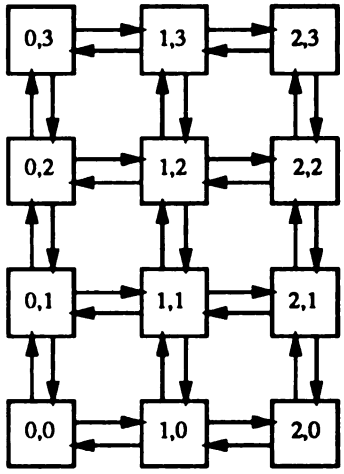


Figure 2.2. A 4 × 3 mesh.

because having a few high-bandwidth channels has more resource sharing than having many low-bandwidth channels in high-dimensional networks. It also makes low-dimensional networks higher hot-spot throughput than higher-dimensional networks. Low-dimensional networks have a higher maximum throughput and lower average blocking latency.

2D mesh is asymmetric network. The channels in the center of the mesh are likely having higher traffic than those in boundary. Another disadvantage is its greater diameter. For a pair of source and destination nodes, the average number of channels that a message traverses in a $\sqrt{N} \times \sqrt{N}$ mesh is $\sqrt{N}/2$ while it is $\log N/2$ in an n-cube, n=$\log N$.

2.1.3 Other Topologies

Many other interconnection topologies have been proposed for multicomputers. For example, 3D mesh is becoming popular recently. MIT J-machine [30] and Caltech MOSAIC [3] adopt 3D mesh topology. Other topologies are ring, tree, etc. A general topology is the k-ary n-cube. It has n dimensions and k nodes at each dimension

that connected as a ring. Both hypercube and 2D mesh topologies discussed before are special classes of k-ary n-cube topology.

2.2 Switching Technology

The network performance is usually measured in terms of *communication latency.* It is defined as the elapsed time from the instance a "message send" is initiated by a sending process to the instance the message is received by the receiving process. The communication latency is the summation of three values: ***start-up latency, network latency, and blocking time*** [27]. The start-up latency is the time for message framing/unframing, memory/buffer copying, validation and etc., at both source and destination node. It is dependent on the software techniques used in each node. The blocking time is the time delay during the message passing mainly due to the contention of communication resources such as waiting for a busy channel to become idle or waiting for the available buffers. It reflects the dynamic behavior of the network when there is a contention for network resources. Network latency is the time when the head of message is sent out at a source node until the message tail enters the destination node. Obviously, the switching technologies have a great impact on network latency.

Four switching technologies have been proposed to handle the switching tasks at intermediate nodes: store-and forward switching, virtual cut-through, circuit switching, and wormhole routing. We will describe these switching technologies in more detail.

A message is usually divided into a number of ***packets*** for transmission in order to use network resources efficiently. Packet is the information unit handled by the router.

2.2.1 Store-And-Forward Switching

All first generation hypercube multicomputers adopt the *store-and-forward* mechanism, in which each packet is stored completely in each intermediate node before forwarding it to next node. If we define the network latency as the time from when the head of a message enters the network at the source until the tail emerges at the destination. The network latency of store-and-forward switching can roughly be expressed as [3]

$$T_p D + L/B = (L/B)D + L/B = (L/B)(D+1),$$

where T_p is the delay of the individual routing nodes on the path, and D is the number of nodes on the path (distance). L/B is the time required for the message of length L to pass through the channels of bandwidth B. In this approach, the message transmission time is linearly proportional to the number of hops (links) between two nodes. Thus, the parameter time is usually represented by the number of hops.

2.2.2 Virtual Cut-Through Switching

Virtual cut-through switching [28] was originally proposed for computer network. In this method, the routing decision is made as soon as the header with destination information is available. The node then immediately forwards the package to the next node if the corresponding outgoing channels is available. The network latency without considering the block time is

$$T_p D + L/B = (L_h/B)D + L/B,$$

where L_h is the length of the header field. When the length of message L is much greater than L_h, $(L_h/B)D$ is negligible. Therefore, the distance D is no longer an

important factor affecting the network latency.

As indicated in [28], however, when the outgoing channel is unavailable, the package has to be buffered at the intermediate node. If the traffic is heavy in the network, the package is buffered at each intermediate node, virtual cut-through switching acts just like store-and-forward switching.

2.2.3 Circuit Switching

Circuit switching is similar to the way of signal transmission in telephone networks. In circuit switching, a physical circuit between the source and destination nodes must be established before the source node can start sending message. The circuit is torn down after the tail of message is delivered.

In the circuit establishment phase, the source node sends a short control package with routing information such as the destination address to set up a connection to the destination node. The control package can be stored at intermediate nodes. If a circuit cannot be set up due to the contention for channels, various protocols can be used to reestablish the circuit [27]. After the circuit being established, the source node sends the message through the circuit that is exclusively reserved for the message transmission. No message buffer is needed for the message transmission.

The network latency for circuit switching is

$$T_p D + L/B = (L_c/B)D + L/B,$$

where L_c is the length of the control packet to be sent for establishing the circuit. Again, if $L_c << L$, the distance D has little impact on the network latency.

The major advantage of circuit switching is that routing cost is paid only once at the circuit establishment phase. After the circuit is set, the message can be transmitted with very little delay and no buffered is required.

2.2.4 Wormhole Routing

Wormhole routing was proposed by Dally and Seitz in their torus routing chip design [29]. In wormhole routing, the message is divided into flits (flow control digits). Similar to the virtual cut-through, when the header flit that contains routing information arrives at an intermediate node, it is sent out immediately through an available outgoing channel. The remaining flits of the message follow in a pipeline fashion. However, the blocked messages remain in the network in the wormhole routing, while virtual cut-through buffers blocked messages and thus removes them from the network.

The network latency for the new switching mechanism is

$$T_p D + L/B = (L_f/B)D + L/B,$$

where L_f is the length of its smallest information unit, *flit*. If $L >> L_f$, the message transmission time is almost independent of the number of hops between two nodes.

Because wormhole routing has low network latency and requires small amount of dedicated buffers at each node, it becomes the most promising switching technology and has been adopted in Symult 2010, nCUBE-2, iWARP, and Intel's Touchstone project. It is also being used in some fine-grained multicomputers, such as MIT's J-machine [30] and Caltech's MOSAIC [3]. Detailed studies of communication issues and wormhole routing method can be found in [27] [31].

Figure 2.3 illustrates the transmission of a packet from source node S to destination node D for store-and-forward switching, circuit switching, and wormhole routing in a contention-free network. A good survey and comparison study of various communication paradigms can be found in [32] [27].

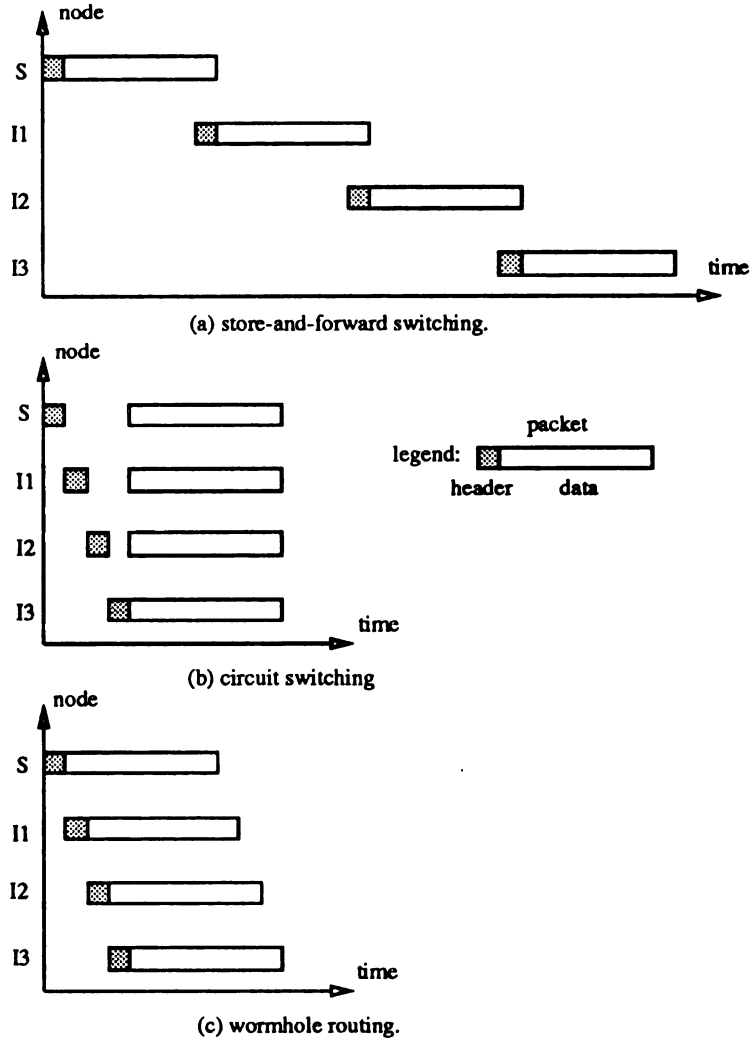


Figure 2.3. A comparison of different switching technologies.

2.3 Message Routing and Flow Control

Message routing is an important issue in parallel processing. Since there are potentially many paths joining pairs of nodes in a multicomputer, different routes can be found depending on the criteria employed. The other important and practical issue is the avoidance of the deadlock in message passing.

2.3.1 Source Routing and Distributed Routing

Depending on which node(s) determines the routing path, either *source routing* or *distributed routing* may be adopted. In source routing, the source node determines all the nodes that will be involved in delivering its message to all the destinations. Thus, the routing information will be carried in the message itself. With distributed routing, the source node determines only to which of its neighboring nodes the source message will be sent. These nodes will repeat the procedure in turn. In this approach, the destination field in the message only carries the destination addresses.

The main drawback of source routing is that the addresses of all the intermediate nodes must be carried in the source message, which will create extra message overhead. In distributed routing, on the other hand, since the routing decision is made in each intermediate node, the routing algorithm should be simple enough to achieve a low overall message transmission time. All commercially available multicomputers support distributed routing for one-to-one communications. However, obtaining an efficient multicast routing algorithm is not an easy task in a distributed manner [33].

2.3.2 Deterministic Routing and Adaptive Routing

The routing scheme can also be classified as deterministic routing and adaptive routing. With deterministic routing, the path from source node to destination node is determined solely by the addresses of the source and the destination node. In adap-

tive routing, however, it can use more than one path to deliver the message for a pair of source and destination nodes. The choice of the path to be taken by a message may depend on the network traffic. Minimal adaptive routing refers to the adaptive routing that always takes a shortest path from source node to the destination node. Nonminimal adaptive routing can take any path between source and destination.

The deterministic routing strategy is simple and easy to implement in hardware. Since the path from a source to a destination is determined *a priori*, some restriction can be imposed to the selection of the path so that deadlock can be avoided. For example, the E-Cube routing for hypercube and XY routing for 2D mesh are the well-known deterministic deadlock-free routing schemes that have been used in many multicomputers [34] [35]. However, in a multicomputer, the network usually has many paths between a pair of nodes, only one is regularly used. The load may not evenly be distributed over the channels and it is not fault-tolerant.

On the other hand, adaptive routing has the advantage of being able to provide more flexibility to select different channels based on the network condition and the network throughput is increased. Also, it can avoid the fault channels to achieve fault-tolerant. The main issue in providing adaptive routing is to avoid deadlock while having relatively simple routing algorithm. By introducing virtual channels, some adaptive routing schemes have been proposed [36]. Partially adaptive routing scheme is presented in [37] without introducing virtual channels.

2.3.3 Flow Control

Flow control deals with the allocation of communication resources such as channels and buffers to messages as they are passing through the paths, as well as the resolution of resource collisions. A good flow control policy should be able to reduce the communication latency.

A resource collision occurs when more than one message want to use a same

resource such as communication channel. Usually, the one that requests first gets the resource, the rest will be blocked until the resource is available. Depending on the switching technology and routing scheme, several methods can be used to resolve the collisions.

For the channel collision, the message is buffered at an intermediate node where the collision occurred. In virtual cut-through switching, it buffers the blocked message until the relevant outgoing channel is available.

The message is blocked in place until the requested resource becomes available. In wormhole routing, the blocked message stops progress and remains in the network until the relevant outgoing channel becomes idle.

The block message is dropped. The source node then waits for a random time before trying to send the same message again.

The message is derouted through another channels. In adaptive routing, message can be routed in different paths according to the usage of the channels.

When several messages are requesting the same resource, the *resource selection policy* decides which message may use the resource. Possible selection policies include the first come first serve, round-robin, and fixed resource priority, etc.

2.3.4 Deadlock Issues

In multicomputer networks, communication channels and message buffers constitute the set of *permanent reusable resources* [27]. The processors that send or receive the messages compete for these resources. In short, messages are the entities that compete for these resources. Deadlock refers to the situation in which a set of messages each is blocked forever because each message in the set holds some resources also needed

by the other. Figure 2.4 shows a deadlock configuration in which the four messages each acquire some channels required by the other messages.

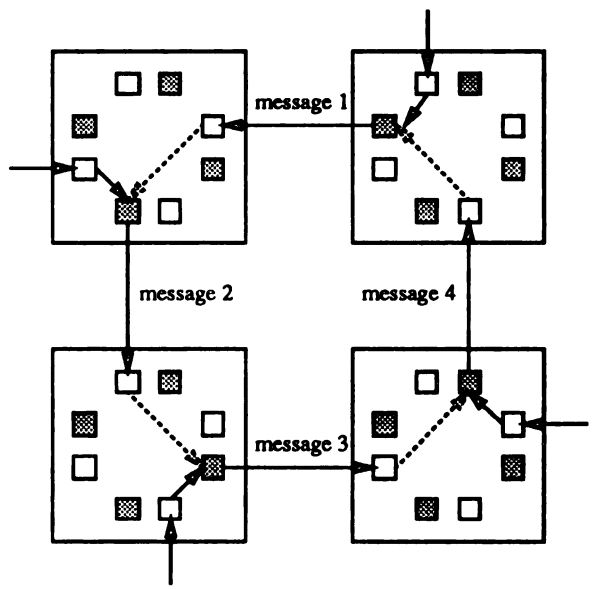


Figure 2.4. An example of deadlock involving four messages.

In store-and-forward and virtual cut-through switching, buffers are the resources used in the message passing, while in circuit switching and wormhole routing, channels are the critical resources if each physical (virtual) channel has its own set of buffers.

Buffer Deadlock

Buffers are used in multicomputer to reduce the effect of traffic fluctuations and increase the total bandwidth. In message transmission, buffers hold both incoming and outgoing packets and they are released when the packets are sent out or consumed. The size of the buffer at each node is an important issue. If the size of the buffer were unlimited, deadlock would never occur.

A large number of deadlock-free routing algorithms have been developed for store-and-forward networks [38] [39] [40] [41] [42]. These algorithms are mostly based on the concept of *structure buffer pool.* In this method, the message buffers at each node are divided into $C + 1$ classes numbered from 0 to C, C is the length of the longest route in the network. The assignment of buffers to message packets is restricted to define a partial order on buffer classes. A packet is ready to send out only if the buffer pool of class 0 is not empty and it is put to the buffer of class 0. The queued packet can be sent to next node only if buffers of class 1 at that node are available, and so on until the packet arrives the destination node. The buffers of class i only hold the packets that have traversed at least i hops. The packets are dropped if the buffers are not available and will be retransmitted later.

The other method of the same idea is to divide the buffers into C classes. A packet with i hops remaining to the destination can only be put in the buffers of class i.

It is not difficult to prove that the structure buffer pool algorithm is deadlock free since it assigns a partial order to resources. In the second version of the algorithm, for example, after a destination node consumes the previous message packet, the buffers of class 1 will be available, a packet with 1 hops remaining to the destination can be forwarded and the buffer be released. Then the buffers of class 1 at that node are not empty, and the packet with 2 hops remaining to the destination can be sent out, and so on until all the packets are reached to their destination nodes.

There are two major drawbacks of this method. The buffer utilization is low due to their restricted usage. Many buffers are barely used in the message passing. The other problem is that if the diameter of the network is large, the buffers have to be divided into many classes and the amount of the buffers must also be huge.

As indicated earlier, more advanced switching technologies such as circuit switching and wormhole routing have been used in new multicomputers. We need to study the channel deadlock problem in the networks using these switching technologies.

Channel Deadlock

If we suppose that each physical (virtual) channel has its own small set of buffers as in the wormhole routing and circuit switching, channels are critical resources. We consider here the channel deadlock problem in wormhole routing. The solution can also be applied to circuit switching. Figure 2.5 shows an example of channel deadlock configuration.

In [44], a graph model, *channel dependency graph* (CDG), for a direct network is proposed to establish the necessary and sufficient condition for deadlock-free routing. A channel dependency graph for a given network I and routing function f, is a directed graph, $D = G(C, E)$. The nodes of D are channels of I. The edges of D are the pairs of channels connected by R:

$E = \{ (c_i, c_j) | R(c_i, n) = c_j \text{ for some } n \in N \}$,

where $R(c_i, n) = c_j$ means that if a message with destination n is entered from channel c_i, the routing function R forwards it to channel c_j toward destination n.

It has been proved in [44] that a routing algorithm is deadlock-free only and if only there is no cycle in its CDG. Therefore, we need to restrict routing to avoid cycle in the channel dependency graph. Figure 2.5 gives an application of channel dependency graph. The routing function in this example always first forwards message horizontally then vertically, which is called X-first routing.

The other similar solution is to divide the network into several acyclic subnetworks, messages will be routed in each subnetwork independently. For unicast communication, the routing scheme cannot let deadlock since each subnetwork is acyclic and no cycle can be formed in its channel dependency graph.

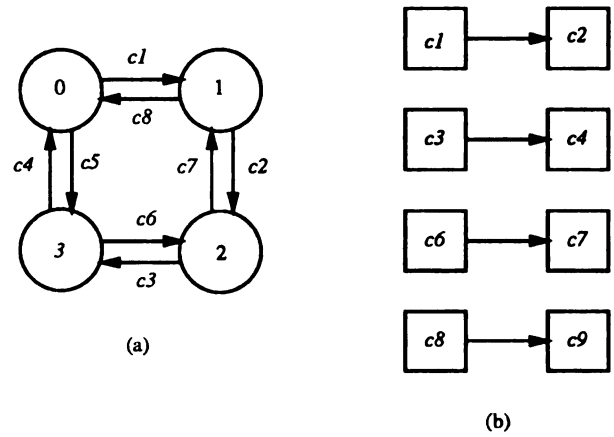


Figure 2.5. (a) A direct network I; (b) Channel dependency graph for I and X-first routing.

2.4 Issues of Multicast Communication

Providing support for multicast communication involves several, often conflicting, requirements. First, it is desirable that the message delay from the source to each of the destinations be as small as possible. Sending a separate copy of the message to each destination along shortest paths appears to be a logical solution, but the increased traffic load resulting from these copies may actually hinder the progress of messages. Hence arises the second requirement, that the amount of network traffic be minimized. Unfortunately, as will be shown later, finding optimal multicast routes, in terms of delay and traffic, has been shown to be NP-hard for the most common multicomputer topologies. The third requirement, then, is that the routing algorithm not be computationally complex. Heuristic algorithms must be employed, but are constrained by the final requirement, that the multicast communication protocol be deadlock-free.

Designing multicast protocols and routing algorithms to meet these requirements naturally depends on the network topology and the underlying switching mechanism

used in the multicomputer.

Apparently, the underlying switching techniques affect the criteria in evaluating multicast communication schemes. Based on the different criteria of the multicast communication, various multicast routing models should be proposed.

Theoretically, we need to examine the computational complexity of the optimal routing problem under each proposed model. As will be shown later, all the routing problems for the proposed various models are NP-complete for 2D mesh, hypercube, and 3D mesh topologies. These results make the existence of any polynomial time routing algorithms for optimal routing unlikely, especially in a distributed manner.

Having established theoretical foundation of the routing problem, we need to develop heuristic multicast routing algorithms for each proposed model. In designing the routing algorithms, there are three issues which are of practical importance. First, the heuristic routing should be in distributed manner. Second, the routing algorithm should be deadlock-free or capable of dealing with deadlock situation. Finally, such routing algorithms should be simple enough for efficient hardware implementation.

Wormhole routing is becoming the most promising switch technique used in multicomputer networks. We will study new routing models and network partitioning strategies for multicast wormhole routing. Based on the routing models and network partitioning strategies, new multicast wormhole routing algorithms must be developed. These routing algorithms should be deadlock-free and have low probability a message is blocked.

Performance study is necessary to evaluate the multicast routing schemes. The performance of a multicast routing algorithm can be measured by average traffic it takes. More importantly, we need to study the network latency under the dynamic network condition because it mainly depends on the interaction of the different messages in the network.

We will address these issues in more detail in the following chapters.

CHAPTER 3

MODELS FOR MULTICAST COMMUNICATION

Two major routing design parameters are *traffic* and *network latency*. For a given multicast communication, the parameter traffic is quantified in the number of communication links used to deliver the source message to all its destinations. The parameter network latency is the message transmission time. In an asynchronous multiprocessing environment, the message transmission time should be considered from the destination node point of view because the receiving process can continue its execution as soon as the message is received. Obviously, the time required to transmit a message from a source to any destination should be minimized. It is desirable to develop a routing mechanism that completes communication while minimizing both traffic and network latency. However, these two parameters are not, in general, totally independent and achieving lower bound for one may prevent us from achieving the other.

The network latency is dependent on the underlying switching technology. As shown in Chapter 2, in *store-and-forward* mechanism, the network latency is linearly proportional to the number of hops (links) between two nodes. Thus, the parameter network latency is usually represented by the number of hops. By minimizing

network latency, in this case, it implies that for each destination node, the message should be delivered through a shortest path to that destination. In second generation multicomputers, more advanced switching mechanisms have been adopted. For example, iPSC-2 [43] adopts *circuit switching* and Ametek 2010 (Symult) adopts *wormhole routing* [26] [44]. In these new switching techniques, the network latency is almost independent of the number of hops between two nodes. Apparently, the underlying switching technique will affect the criterion in evaluating multicast communication schemes.

Graph will be used to model the underlying topology of multicomputer. We will closely follow the graph theoretical terminology and notation of [45]; terms not defined here can be found in that book. Let graph $G(V, E)$ denote a graph with node set V and edge set E. When G is known from context, the sets $V(G)$ and $E(G)$ will be referred to as V and E, respectively. A *path* with length n is a sequence of edges $e_1, e_2, \ldots, e_n$ such that

1. e_i and e_{i+1} have a common end-node, and
2. if e_i is not the first or last edge, then it shares one of its end-nodes with e_{i-1} and the other with e_{i+1}.

Suppose $e_i = (v_i, v_{i+1})$ for $1 \leq i \leq n$. In the following discussion, this path with length n will be represented by its node visiting sequence $(v_1, v_2, \ldots, v_n, v_{n+1})$. A *cycle* is a path whose starting and ending nodes are the same, *i.e.*,$v_1 = v_{n+1}$. Furthermore, we assume that for every pair of nodes in the path except v_1 and v_{n+1} are different. A graph is said to be *connected* if every pair of its nodes are joined by a path. A *tree* is a connected graph which contains no *cycles*. A graph $F(V, E)$ is a *subgraph* of another graph $G(V, E)$, if $V(F) \subseteq V(G)$ and $E(F) \subseteq E(G)$. A subgraph which is a tree is referred to as a *subtree*. For a pair of nodes u, v in $V(G)$, $d_G(u, v)$ denotes the length (the number of edges) of a shortest path from u to v in G.

The interconnection topology of a multicomputer is denoted by a host graph $G(V, E)$, where each node in V corresponds to a processor and each edge in E corresponds to a communication link. For a multicast communication, let u_0 denote the source node and $u_1, u_2, \ldots, u_k$ denote k destination nodes, where $k \geq 1$. The set $K = \{u_0, u_1, \ldots, u_k\}$, which is subset of $V(G)$, is called a *multicast set.* Depending on the underlying switching technique and the routing method, the multicast communication problem in a multicomputer can be formulated as different graph theoretical problems.

3.1 Multicast Path and Cycle Problems

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, the routing method does not allow each processor to replicate the message passing by. Also, as indicated in [46], a multicast path model provides better performance than the tree model when there is a contention in the network. From communication technology point of view, the multicast path model is more suitable for the new communication mechanism such as wormhole routing.

The multicast communication problem becomes the problem of finding a shortest path starting from u_0 and visiting all k destination nodes. This optimization problem is the finding of an *optimal multicast path* (OMP) and is formally defined below.

Definition 3.1 *A multicast path (MP)* $(v_1, v_2, \ldots, v_n)$ *for a multicast set* K *in* G *is a subgraph* $P(V, E)$ *of* G, *where* $V(P) = \{v_1, v_2, \ldots, v_n\}$ *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.*

Figure 3.1 shows a example of multicast path in 2D mesh graph.

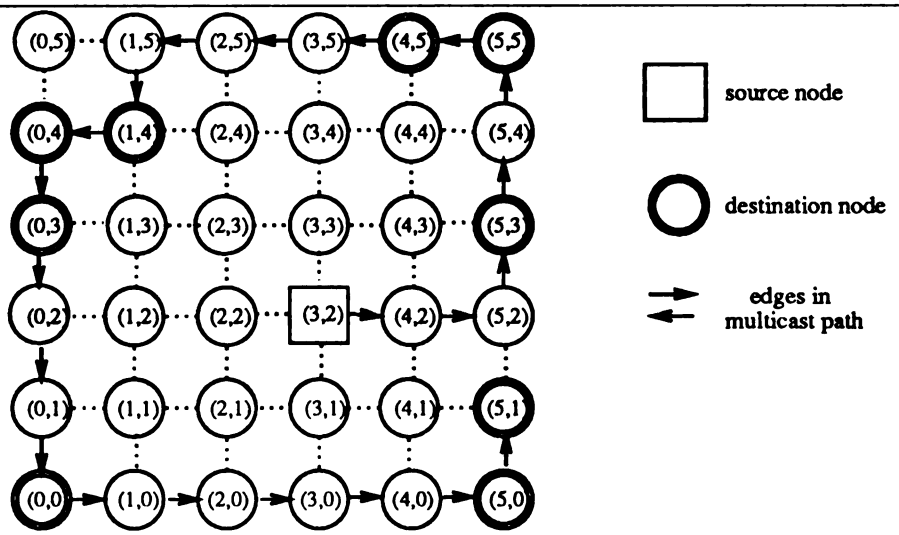


Figure 3.1. An example of multicast path in 2D mesh.

Reliable communication is essential to a message passing system. Usually, a separate acknowledgement message is sent from every destination node to the source node upon receipt of a message. One way to avoid the sending of $|K|$ separate acknowledgement messages is to have the source node itself receive a copy of the message it initiated after all destination nodes have been visited. Acknowledgements are provided in the form of error bits flagged by intermediate nodes when a transmission error is detected. Thus, the multicast communication problem is the problem of finding a shortest cycle, called *optimal multicast cycle* (OMC) for K.

Definition 3.2 *A multicast cycle* $(v_1, v_2, \ldots, v_n, v_1)$ *for* K *is a subgraph* $C(V, E)$ *of* G, *where* $V(C) = \{v_1, v_2, \ldots, v_n\}$ *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.*

3.2 Steiner Tree Problem

Both OMC and OMP assume that the message will not be replicated by any node during transmission. However, message replication can be implemented by using some hardware approach [21]. If our major concern is to minimize traffic, the multicast problem becomes the well-known *Steiner tree* (ST) problem [47]. An example of Steiner tree in 2D mesh is shown in Fig. 3.2 Formally, we restate the ST problem as follows.

Definition 3.3 *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 a minimal total length.*

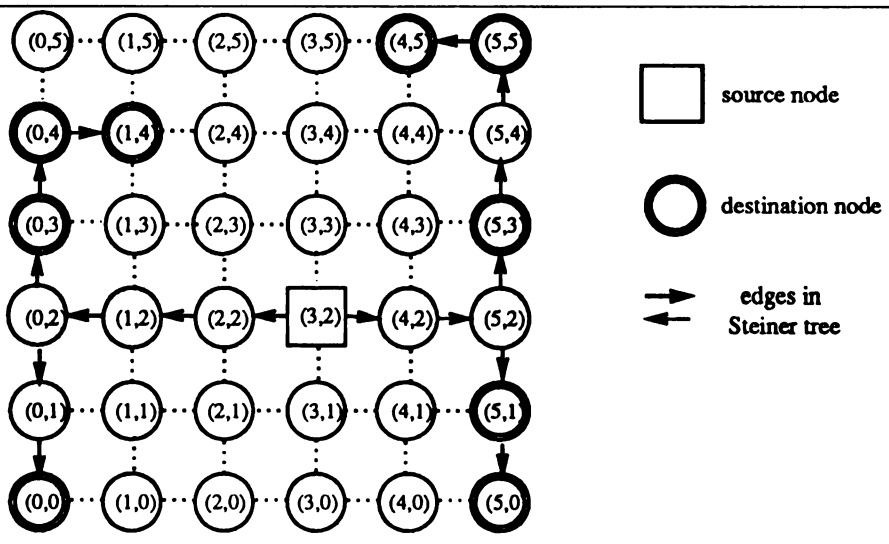


Figure 3.2. An example of Steiner tree in 2D mesh.

3.3 Multicast Tree Problem

In the ST problem, we don't 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 network latency, such as virtual cut-through, wormhole routing, and circuit switching, 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), as shown in Fig. 3.3. The OMT problem was originally defined by [19].

Definition 3.4 *An OMT, $T(V, E)$, for K is a subtree of G, such that (a) $K \subseteq V(T)$, (b) $d_T(u_0, u_i) = d_G(u_0, u_i)$, for $1 \leq i \leq k$, and (c) $|E(T)|$ is as small as possible.*

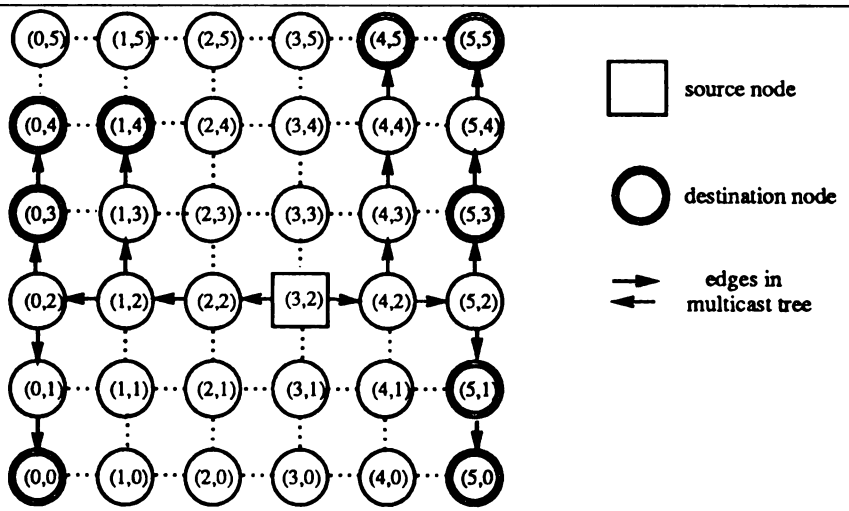


Figure 3.3. An example of multicast tree in 2D mesh.

3.4 Multicast Star Problem

In new switching technology such as wormhole routing, the tree-like multicast models are not suitable because the progress of the message passing of the entire tree is blocked if any one of its branches is blocked due to the contention of the communication resources [48] [49]. Further, the deadlock properties of wormhole routing are different from those of store-and-forward and virtual cut-through. As will be shown later, More than one path are needed for deadlock-free multicast wormhole routing. A practical multicast routing scheme must be deadlock-free, Also it should transmit the source message to each destination node with as small transmission time and communication channels as possible. The optimal routing problem in our routing scheme described above becomes the problem of finding an optimal multicast star in a given host graph.

Definition 3.5 *A multicast star (MS) $S(V, E)$ of a host graph $G(V, E)$ for a multicast set K is a collection of several multicast paths. The i-th path starts from source node and reaches each destination in D_i, where $(\bigcup_{\forall j} D_j) \cup \{u_0\} = K$ and $D_i \cap D_j = \emptyset$ if $i \neq j$. An optimal MS (OMS) is a MS with a minimum length.*

Figure 3.4 is an example of multicast star in 2D mesh graph.

The graph optimization problems can be stated as follows:

> *Given a host graph G, a multicast set K, and an integer ℓ, does there exist an OMP (OMC, MST, OMT, OMS) for K with total length less than or equal to ℓ?*

We shall investigate the computational complexities of the above optimal problems for 2D mesh, hypercube, and 3D mesh topologies in next chapter.

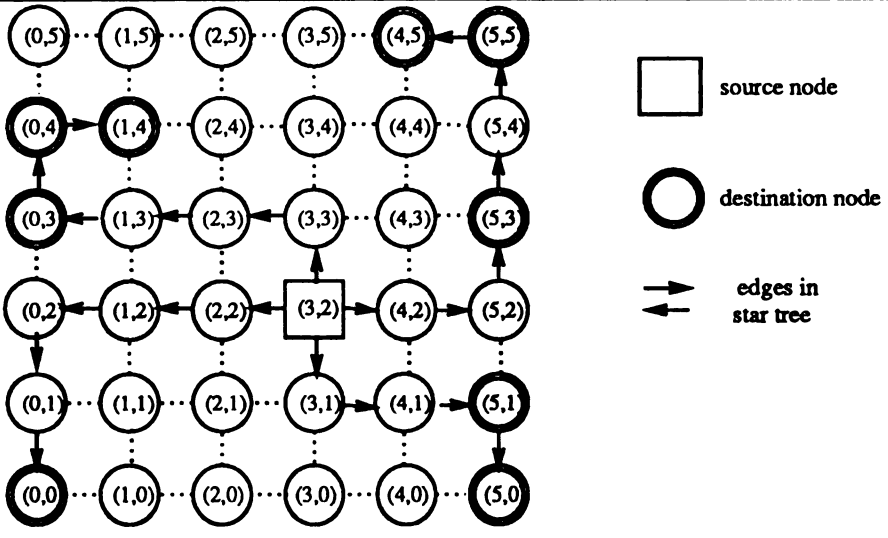


Figure 3.4. An example of multicast star in 2D mesh.

CHAPTER 4

OPTIMAL MULTICAST IN MULTICOMPUTERS

Apparently, the complexity of each of the above optimization problems is directly dependent on the underlying host graph. We will focus on two popular host graphs: n-cube and 2D mesh. Results obtained for 2D mesh can be easily extended to 3D mesh topology. In this chapter, we will examine the computational complexities of the MP, MC, ST and MS optimization problems for these topologies. The OMT problem was originally studied in [19] and has been proven to be NP-complete for n-cube graphs [50]. It is still open if OMT problem is NP-complete or not for 2D mesh graph. The OMT problem will not be discussed here.

4.1 Optimal Multicast in 2D Mesh Topology

We first consider the case in which the host graph is a 2D mesh (non-wraparound) as adopted in Ametek 2010 [26]. Before discussing the 2D mesh graph, we have to briefly review a more general class of graphs, called *grid graphs* [51].

Let G^{∞} be the infinite graph whose vertex set consists of all points of the plane with integer coordinates and in which two vertices are connected if and only if the

Euclidean distance between them is equal to 1 [51]. A grid graph, $G(V, E)$, is a finite, node-induced subgraph of G^{∞}. A grid graph is completely specified by its vertex set $V(G)$. Let v_x and v_y be the coordinates of the vertex $v \in V(G)$. A 2D mesh, also named *rectangular graph*, is a special case of grid graphs as defined below.

Definition 4.1 *Let $M(V, E)$ represent a 2D $N_1 \times N_2$ mesh graph, which is a grid graph whose vertex set is*

$$V(M) = \{v : x_0 \leq v_x \leq N_1 + x_0 - 1 \text{ and } y_0 \leq v_y \leq N_2 + y_0 - 1\},$$

where (x_0, y_0) is the lower left coordinate of the mesh.

In [51], the authors have proved a number of related and interesting results as stated below.

G1 *The Hamilton cycle (circuit) problem for grid graphs is NP-complete.*

G2 *The Hamilton path problem, (G, s, t), for grid graphs is NP-complete.*

G3 *The Hamilton cycle problem for rectangular subgrid graphs is NP-complete.*

G4 *The Hamilton path problem, (G, s, t), for rectangular subgrid graphs is NP-complete.*

Our problem is different from these in grid graphs in which we are looking for a multicast cycle or path for a given multicast set, K, which is a subset of the 2D mesh, M.

Theorem 4.1 *The OMC problem for 2D mesh graphs is NP-complete.*

Proof: In the proof of NP-completeness of this theorem, we shall use the above known NP-completeness results. First, we have to show that we can construct a 2D

mesh graph $M(V,E)$ from $G(V,E)$ in polynomial time. Then we will show that we can select a multicast set K in $M(V,E)$ in polynomial time. Finally, we have to prove that there is an OMC for K in $M(V,E)$ with total length less than or equal to some number if and only if $G(V,E)$ has a Hamilton cycle. From G1, we know the Hamilton cycle problem is NP-complete for a grid graph $G(V,E)$. Thus, the OMC problem for 2D mesh graphs is NP-complete. This technique will also be used in the proof of other NP-complete theorems.

In this case, given a grid graph $G(V,E)$, we can easily construct a 2D mesh graph $M(V,E)$ such that $V(G) \subseteq V(M)$ in polynomial time. Then, we select K=$V(G)$. It is straightforward to see that G has a Hamilton cycle if and only if there exists an OMC for K in M with total length $|V(G)|$. □

The Hamilton path problem (G,s,t) in G2 has a solution if there exists a Hamilton path from s to t in G. To prove the OMP problem is NP-complete, we need to prove the following lemma which is more general than G2, in which there is no restriction on which node should be the termination node in the Hamilton path.

Lemma 4.1 *The Hamilton path problem with a given starting node for grid graphs is NP-complete.*

Proof: For a given grid graph $G(V,E)$, we construct a grid graph $G'(V,E)$ such that G has a Hamilton cycle if and only if G' has a Hamilton path starting from a given node s.

First, we select a corner node $u = (u_x, u_y)$ such that $u_x = \min_{v \in V(G)}\{v_x\}$ and $u_y = \min_{v \in V(G) \text{ and } v_x = u_x}\{v_y\}$.

We define the following four points

$$p = (u_x - 1, u_y), q = (u_x - 1, u_y + 1), t = (u_x - 2, u_y + 1), s = (u_x - 1, u_y - 1).$$

Then, we can construct a grid graph $G'(V, E)$ such that $V(G') = V(G) \cup \{p, q, t, s\}$ (see Fig. 4.1).

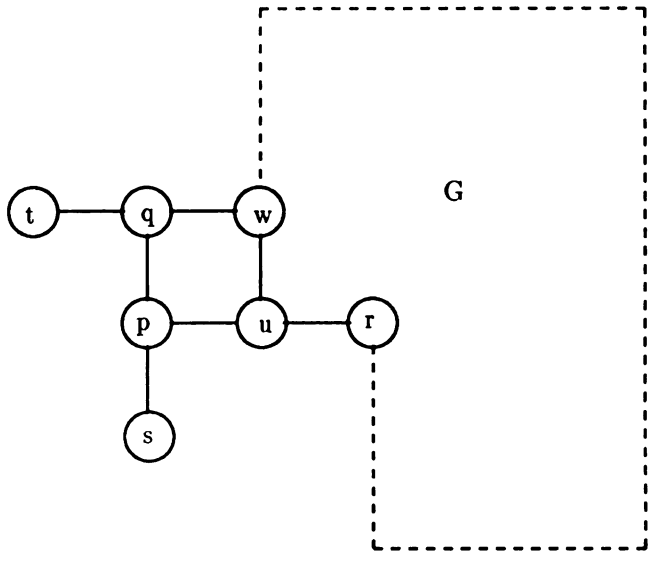


Figure 4.1. The construction of G' from G.

Note that the degree of each node in a grid graph is at most 4 and for a given node v of a grid graph, the coordinates of its four possible adjacent nodes are $(v_x - 1, v_y)$, $(v_x + 1, v_y)$, $(v_x, v_y - 1)$ and $(v_x, v_y + 1)$, respectively. Suppose there exists a Hamilton cycle in G, then the degree of each node in $V(G)$ is at least 2. Now, considering the node u, since its two possible adjacent nodes with coordinates $(u_x - 1, u_y)$ and $(u_x, u_y - 1)$ are not in $V(G)$ due to the selection of u, the other two possible adjacent nodes with coordinates $(u_x, u_y + 1)$ and $(u_x + 1, u_y)$ must be in $V(G)$. Let $w = (u_x, u_y + 1)$ and $r = (u_x + 1, u_y)$ (see Fig. 4.1). Obviously, both edges (u, r) and (w, u) should be in this Hamilton cycle. Suppose this Hamilton cycle is $(u, r, \cdots, w, u)$. It is easy to see that G' has a Hamilton path $(s, p, u, r, \ldots, w, q, t)$.

Conversely, if G' has a Hamilton path starting from s, by the selection of u, we know p, q, t and s are not in $V(G)$, and

$$E(G') = E(G) \cup \{(q, w), (t, q), (q, p), (p, s), (p, u)\}.$$

The Hamilton path should end with t since the degree of t is 1. It is straightforward to check that such a Hamilton path must have the form $(s, p, u, r, \ldots, w, q, t)$. Therefore, $(u, r, \cdots, w, u)$ is a Hamilton cycle in G. □

The transformation from the Hamilton path problem, stated in Lemma 4.1, into the OMP problem for 2D mesh graphs is similar to the one in the proof of Theorem 4.1. Thus, we have the following theorem.

Theorem 4.2 *The OMP problem is NP-complete for 2D mesh graphs.*

For the multicast star problem, again we have the following result.

Theorem 4.3 *The OMS problem is NP-complete for 2D mesh graphs [52].*

Proof: To show that the MS problem for 2D mesh graphs is NP-complete, we will reduce the known NP-complete problem of finding a Hamilton cycle [51] for grid graphs to this problem.

Given a grid graph G, as in the proof of Theorem 4.1, a 2D mesh graph $M(V, E)$ can be constructed in polynomial time such that $V(G) \subseteq V(M)$ (Fig. 4.1). Now select $K = V(G')$ and $u_0 = s$. We need to prove that

G1 G has a Hamilton cycle if and only if G' has a Hamilton path starting from s;

G2 G' has a Hamilton path starting from s if and only if $M(V, E)$ has a MS with s as the root total length $|V(G')| - 1$, where $|V(G')| = |V(G)|+4$.

The proof of G1 is the same as the one for Theorem 4.1 and here we only prove G2, which is straightforward. Suppose G' has a Hamilton path starting from s. Since G' is a subgraph of $M(V,E)$, the Hamilton path is also a MS for K ($K = V(G')$) in $M(V,E)$ with s as the root, which has the minimum length $|V(G')|-1$. On the other hand, if $M(V,E)$ has a MS with s as the root and with length $|V(G')|-1$, the MS can only contain the nodes in K since K=$V(G')$. Because the only neighbor node of s in K is p, the MS consists of only one path starting from s to p, otherwise at least one node in MS would not be in K. Obviously, the MS is a Hamilton path in G'. The proof of this theorem is completed by combining G1 and G2. □

Garey and Johnson [52] have proved that the *rectilinear Steiner tree* (RST) problem is NP-complete. The RST problem for a set A is a tree structure, composed solely of horizontal and vertical line segments, which interconnect all the points in A. Replacing A by the multicast set K, this problem is the same as the MST problem for 2D mesh graphs. The following theorem can be derived directly from [52].

Theorem 4.4 *The MST problem is NP-complete for 2D mesh graphs [52].*

4.2 Optimal Multicast in Hypercube Topology

We first formally define a hypercube, or n-cube, graph.

Definition 4.2 *Let $H(V,E)$ represent an n-dimensional hypercube graph, $|V(H)|$=2^n and each node in $V(H)$ has a unique n-bit binary address. For each $v \in V(H)$, let $b(v)$ denote its n-bit binary address and $\| b(v) \|$ represent the number of 1's in $b(v)$. An edge $e = (u,v) \in E(H)$ if and only if $\| b(u) \oplus b(v) \| = 1$, where $\oplus$ is the bitwise exclusive OR operation on binary numbers.*

The shortest distance between any two nodes $u, v \in V(H)$ is $d_H(u,v) = \| b(u) \oplus b(v) \|$.

To show that the OMC problem for n-cube graphs is NP-complete, we will reduce the known NP-complete problem of finding a Hamilton cycle for grid graphs to this problem.

First, given a grid graph $G(V,E)$ with $|V(G)| = k$, let's consider an n-cube graph $H(V,E)$ with $n = 4k$. For a node $q \in V(H)$, its address $b(q)=b_0(q)b_1(q)\cdots b_{n-1}(q)$ can be thought of as consisting of k blocks, each with length 4. Thus, we can express $b(q)=a_0(q)a_1(q)\cdots a_{k-1}(q)$, where $a_i(q) = b_{4i}(q)b_{4i+1}(q)b_{4i+2}(q)b_{4i+3}(q)$ for $0 \leq i < k$.

The next step is to select $K = \{u_0, u_1, \ldots, u_{k-1}\}$ where $K \subseteq V(H)$ such that $G(V,E)$ has a Hamilton cycle if and only if $H(V,E)$ has an OMC with length less than or equal to $6k$.

To begin with we apply the breadth-first search algorithm to $G(V,E)$ with an arbitrary node v_0 as a starting node. As a result, the node set $V(G)$ will be partitioned into w disjoint subsets $A_0, A_1, \ldots, A_{w-1}$, where $A_0 = \{v_0\}$ and $A_i = \{v : d_G(v_0, v) = i\}$ for $1 \leq i < w$. Next, we order the nodes of $V(G)$ as $v_0, v_1, \ldots, v_{k-1}$ in such a way that for every pair of nodes $v_i \in A_p, v_j \in A_h$, if $p < h$, then $i < j$.

The nodes in K can be selected by the following procedure.

1. Select $u_0 \in V(H)$, such that $a_0(u_0) = 1111$, $a_h(u_0) = 0000$ for $0 < h < k$.

2. For m=1 to $k-1$, select $u_m \in V(H)$, such that

   (a) Let $V_m = \{v_p : p < m \text{ and } (v_p, v_m) \in E(G)\}$. For each $v_p \in V_m$, let $U_{p,m} = \{v_q : p < q < m \text{ and } (v_p, v_q) \in E(G)\}$.

      i. if $|U_{p,m}| = 0$, $a_p(u_m) = 1000$

      ii. if $|U_{p,m}| = 1$, $a_p(u_m) = 0100$

      iii. if $|U_{p,m}| = 2$, $a_p(u_m) = 0010$

iv. if $|U_{p,m}| = 3$, $a_p(u_m) = 0001$

(b) If $|V_m| = 1$, then $a_m(u_m) = 1110$. If $|V_m| = 2$, then $a_m(u_m) = 1100$.

(c) For all other address blocks, i.e., v_h is not in $V_m \cup \{v_m\}$ for $0 \leq h < k$, we have $a_h(u_m) = 0000$.

By the ordering of the nodes in $V(G)$ and the definition of V_m, we have $|V_0| = 0$ and $1 \leq |V_m| \leq 2$ for $m > 0$. Since the degree of each node in a grid graph is at most 4, $|U_{p,m}| \leq 3$. Thus, K can be constructed by the above procedure in polynomial time. The following example should make the above procedure clear.

Example 4.1 *Consider the following grid graph G (see Fig. 4.2). After applying breadth-first search algorithm, the node set will be partitioned as $A_0 = \{v_0\}$, $A_1 = \{v_1, v_2\}$, $A_2 = \{v_3, v_4\}$, $A_3 = \{v_5, v_6\}$, and $A_4 = \{v_7\}$.*

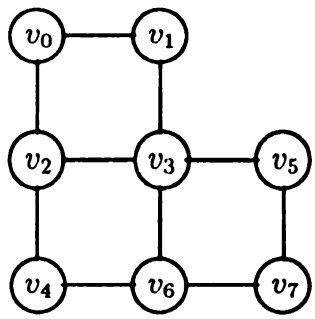


Figure 4.2. A simple grid graph with 8 nodes.

Consider $H(V, E)$, where $n = 4k = 4 \times 8 = 32$ and select $K = \{u_0, u_1, \ldots, u_7\}$ such that

$$a(u_0) = 1111, 0000, 0000, 0000, 0000, 0000, 0000, 0000$$

$$a(u_1) = 1000, 1110, 0000, 0000, 0000, 0000, 0000, 0000$$

$$a(u_2) = 0100, 0000, 1110, 0000, 0000, 0000, 0000, 0000$$

$$a(u_3) = 0000, 1000, 1000, 1100, 0000, 0000, 0000, 0000$$

$$a(u_4) = 0000, 0000, 0000, 1000, 1110, 0000, 0000, 0000$$

$$a(u_5) = 0000, 0000, 1000, 0000, 0000, 1110, 0000, 0000$$

$$a(u_6) = 0000, 0000, 0000, 0100, 0000, 1000, 1100, 0000$$

$$a(u_7) = 0000, 0000, 0000, 0000, 1000, 0000, 1000, 1100.$$

Note that in the above example every pair of nodes u_i and u_j in K, $0 \leq i, j \leq 7$, $d_H(u_i, u_j) = 6$ if (v_i, v_j) is in $E(G)$; otherwise, $d_H(u_i, u_j) = 8$. In what follows we will show that it is true for the general cases.

For each v_m in $V(G)$, let $U_m = V_m \cup \{v_m\}$. From the selection procedure, we have the following properties:

Property 1 : $\| a(u_m) \| = \sum_{v_q \in U_m} \| a_q(u_m) \| = 4$;

Property 2 : For any $v_p \in V_m$, $\| a_p(u_m) \| = 1$; and

Property 3 : For any $v_h \in V(G) - U_m$, $\| a_h(u_m) \| = 0$.

Since $d_H(u_i, u_j) = \| a(u_i) \oplus a(u_j) \| = \sum_{m=0}^{n-1} \| a_m(u_i) \oplus a_m(u_j) \|$, and $V(G) = \{v_0, v_1, \ldots, v_{n-1}\}$, then $d_H(u_i, u_j) = \sum_{v_m \in V(G)} \| a_m(u_i) \oplus a_m(u_j) \|$. By Property 3, $\sum_{v_h \in V(G)-U_i} \| a_h(u_i) \| = 0$ and $\sum_{v_h \in V(G)-U_j} \| a_h(u_j) \| = 0$, we have

$$\sum_{v_h \in V(G)-U_i \cup U_j} \| a_h(u_i) \oplus a_h(u_j) \| = 0.$$

Thus,

$$\begin{aligned} d_H(u_i, u_j) = & \sum_{v_m \in U_i \cup U_j} \| a_m(u_i) \oplus a_m(u_j) \| = \sum_{v_m \in U_i - U_i \cap U_j} \| a_m(u_i) \oplus a_m(u_j) \| + \\ & \sum_{v_m \in U_i \cap U_j} \| a_m(u_i) \oplus a_m(u_j) \| + \sum_{v_m \in U_j - U_i \cap U_j} \| a_m(u_i) \oplus a_m(u_j) \| . \end{aligned}$$

Again, by Property 3, we have

$$\sum_{v_m \in U_i - U_i \cap U_j} \| a_m(u_j) \| = \sum_{v_m \in U_j - U_i \cap U_j} \| a_m(u_i) \| = 0.$$

Finally, we have the following equation.

$$d_H(u_i, u_j) = \sum_{v_m \in U_i - U_i \cap U_j} \| a_m(u_i) \| + \sum_{v_m \in U_i \cap U_j} \| a_m(u_i) \oplus a_m(u_j) \| + \sum_{v_m \in U_j - U_i \cap U_j} \| a_m(u_j) \| . \quad (4.1)$$

Lemma 4.2 *For every pair of nodes u_i and u_j in K, $d_H(u_i, u_j) = 8$ if (v_i, v_j) is not in $E(G)$.*

Proof: By the definition of V_i and V_j, if $(v_i, v_j) \notin E(G)$, then $v_i \notin V_j$ and $v_j \notin V_i$. Since

$$U_i \cap U_j = (V_i \cup \{v_i\}) \cap (V_j \cup \{v_j\}) = (V_i \cap \{v_j\}) \cup (V_j \cap \{v_i\}) \cup \{v_i, v_j\} \cup (V_i \cap V_j),$$

we have $U_i \cap U_j = V_i \cap V_j$. Suppose $V_i \cap V_j = \emptyset$, Equation 4.1 becomes

$$d_H(u_i, u_j) = \sum_{v_m \in U_i} \| a_m(u_i) \| + \sum_{v_m \in U_j} \| a_m(u_j) \| .$$

That is, $d_H(u_i, u_j) = 4 + 4 = 8$ by Property 1. On the other hand, if $V_i \cap V_j \neq \emptyset$, then for each $v_p \in V_i \cap V_j$, we have $|U_{p,i}| < |U_{p,j}|$ since $i < j$ and $v_i \in U_{p,j}$. From Step 2(b) of the selection procedure, it is not difficult to check that

$$\| a_m(u_i) \oplus a_m(u_j) \| = \| a_m(u_i) \| + \| a_m(u_j) \|$$

Therefore, for $(v_i, v_j) \notin E(G)$, we have

$$\begin{aligned} d_H(u_i, u_j) &= \sum_{v_m \in U_i - U_i \cap U_j} \| a_m(u_i) \| + \\ &\quad \sum_{v_m \in U_i \cap U_j} \| a_m(u_i) \oplus a_m(u_j) \| + \sum_{v_m \in U_j - U_i \cap U_j} \| a_m(u_j) \| \\ &= \sum_{v_m \in U_i} \| a_m(u_i) \| + \sum_{v_m \in U_j} \| a_m(u_j) \| = 4 + 4 = 8. \end{aligned}$$

□

Lemma 4.3 *For every pair of nodes u_i and u_j in K, $d_H(u_i, u_j) = 6$ if (v_i, v_j) is in $E(G)$.*

Proof: By definition of V_i and V_j, if $(v_i, v_j) \in E(G)$, then $v_i \notin U_j$ and $v_j \in U_i$. We have $V_i \cap V_j = \emptyset$; otherwise, if there were a node $v_p \in V_i \cap V_j$, again by definition of V_i and V_j, there would have a triangular $\{(v_i, v_p), (v_j, v_p), (v_i, v_j)\}$ in G. However, this is impossible in a grid graph, G. Thus, $U_i \cap U_j = \{v_i\}$. Let's consider $\| a_m(u_i) \oplus a_i(u_j) \|$ in terms of the number of elements of V_i.

1. If $|V_i| = 0$, then $i = 0$ by the definition of V_i. From step 1 of the selection procedure, we have $a_0(u_0) = 1111$ and $\| a_0(u_j) \| = 1$ (Property 2). Thus,

$$\| a_i(u_i) \oplus a_i(u_j) \| = 3 = \| a_i(u_i) \| + \| a_i(u_j) \| - 2.$$

2. If $|V_i| = 1$, then $a_i(u_j) = 1110$ (step 2b). In this case, we must have $|U_{p,i}| \leq 2$ for a $v_p \in V_i$ and $p < i$, because we already have both $(v_p, v_i) \in E(G)$ and $(v_i, v_j) \in E(G)$, and the degree of each node in $V(G)$ is at most 4. Hence, $a_i(u_j)$ is either 1000, 0100, or 0010 by step 2 of the selection procedure. Thus, we have

$$\| a_i(u_i) \oplus a_i(u_j) \| = 2 = \| a_i(u_i) \| + \| a_i(u_j) \| - 2.$$

3. If $|V_i| = 2$, we have $|U_{p,i}| \leq 1$ for each $v_p \in V_i$ $(p < i)$ for the same reason as that of (2). Since $a_i(u_i) = 1100$ by step 2a of the selection procedure, we have

$$\| a_i(u_i) \oplus a_i(u_j) \| = 1 = \| a_i(u_i) \| + \| a_i(u_j) \| - 2.$$

Since $(v_i, v_j) \in E(G)$, we have

$$\begin{aligned} d_H(u_i, u_j) &= \sum_{v_m \in U_i - \{v_i\}} \| a_m(u_i) \| + \| a_i(u_i) \oplus a_i(u_j) \| + \sum_{v_m \in U_j - \{v_i\}} \| a_m(u_j) \| \\ &= \sum_{v_m \in U_i} \| a_m(u_i) \| - 2 + \sum_{v_m \in U_j} \| a_m(u_j) \| = 4 - 2 + 4 = 6. \end{aligned}$$

□

Now we can show that G has a Hamilton cycle if and only if H has an OMC for K with length less than or equal to $6k$. Suppose G has a Hamilton cycle $(v_{i_1}, v_{i_2}, \ldots, v_{i_k}, v_{i_1})$, then $(v_{i_j}, v_{i_{j+1}}) \in E(G)$ for $1 \leq i < k$ and $(v_{i_k}, v_{i_1}) \in E(G)$. By Lemma 4.3, for all the nodes u_{i_j} $(1 \leq j \leq k)$ of K, $d_H(u_{i_j}, u_{i_{j+1}}) = 6$ $(1 \leq j < k)$ and $d_H(u_{i_k}, u_{i_1}) = 6$. Hence, there exists a cycle $(u_{i_1}, \ldots, u_{i_2}, \ldots, u_{i_k}, \ldots, u_{i_1})$ of H, which contains all the nodes in K with length $6k$. That is, there must exist an OMC with length less than or equal to $6k$. Conversely, if H has an OMC for K with length less than or equal to $6k$, say $(u_{i_1}, \ldots, u_{i_2}, \ldots, u_{i_k}, \ldots, u_{i_1})$, then $d_H(u_{i_j}, u_{i_{j+1}}) \geq 6$ for $1 \leq j < k$ and $d_H(u_{i_k}, u_{i_1}) \geq 6$ by Lemma 4.2 and Lemma 4.3. Since the length of an OMC is less than or equal to $6k$, we must have $d_H(u_{i_j}, u_{i_{j+1}}) = 6$ $(1 \leq j < k)$ and $d_H(u_{i_k}, u_{i_1}) = 6$. Again, by Lemma 4.3, $(v_{i_j}, v_{i_{j+1}}) \in E(G)$ $(1 \leq i < k)$ and $(v_{i_k}, v_{i_1}) \in E(G)$. Therefore, the cycle $(v_{i_1}, v_{i_2}, \ldots, v_{i_k}, v_{i_1})$ is a Hamilton cycle of G. Thus, we have established the following theorem.

Theorem 4.5 *The OMC problem is NP-complete for n-cube graphs.*

Note that due to the regular structure of the n-cube graph, we don't need to construct H in the reduction. The input size of the reduced problem is max $\{n, k\}$, where k is the number of destinations. The reduction in the proof of Theorem 4.5 and solving the reduced problem are independent of the number of nodes in the n-cube graph.

By transforming the problem of finding a Hamilton path for grid graph G (see Lemma 1) into the problem of finding an OMP for an n-cube graph H in the same way as the one in the proof of Theorem 4.5, it can be shown that G has a Hamilton path if and only if H has an OMP with length equal to $6(k-1)$, where $k = |V(G)|$. So we have the following theorem.

Theorem 4.6 *The OMP problem for n-cube graphs is NP-complete.*

By using the above proof process, we can easily get the following theorem.

Theorem 4.7 *The OMS problem for n-cube graphs is NP-complete.*

Proof: We also reduce Hamilton path problem for grid graphs to this problem. Given a grid graph $G(V, E)$, we consider an n-cube graph $H(V, E)$ with $n = 4k$, where $k = |V(G)|$, which is the same as the one in the proof of Theorem 4.5.

By using the method in the proof of Theorem 4.5, it can be proved that for every pair of nodes u_i and u_j in K, $d_H(u_i, u_j) = 6$ if (v_i, v_j) is in $E(G)$, and $d_H(u_i, u_j) = 8$ if (v_i, v_j) is not in $E(G)$. By the same arguments as the proof of Theorem 4.3, we can show that G' has a Hamilton path if and only if H has an OMS for K with length less than or equal to $6(k-1)$. □

For the problem of finding a MST in n-cube graphs, it was proved in [53] that the MST problem is NP-complete for n-cube graphs.

Theorem 4.8 *The MST problem for n-cube graphs is NP-complete [53].*

4.3 Optimal Multicast in 3D Mesh Topology

We have studied the multicast issues for 2D mesh and hypercube topologies. These are the most popular topologies used in the current multicomputers. However, some other interconnection topologies are also used or will be adopted in the future multicomputers, for example, the cube-connected-cycle and general k-ary n-cube topologies. 3D mesh topology is also becoming popular. Because 2D graph is the subgraph of 3D graph, it is not difficult to obtain the following corollaries.

Corollary 4.1 *The OMC problem is NP-complete for 3D mesh graphs.*

Corollary 4.2 *The OMP problem is NP-complete for 3D mesh graphs.*

Corollary 4.3 *The MST problem is NP-complete for 3D mesh graphs.*

Corollary 4.4 *The OMS problem is NP-complete for 3D mesh graphs.*

It is still open if OMT problem is NP-complete or not for 2D mesh graphs. It is also open for 3D mesh graphs.

CHAPTER 5

BASIC HEURISTIC MULTICAST ROUTING ALGORITHMS

As shown in previous section, all the proposed multicast problems are NP-complete. Thus, several basic heuristic multicast algorithms for MP, MC, ST and MT problems are given in this chapter. The heuristic multicast algorithms for MS problem will be proposed in next chapter. The performance study for the routing algorithms will be presented in Chapter 7.

We present new heuristic distributed routing schemes for the multicast optimization problems. In our distributed routing algorithms, the source node must perform a preprocessing function (message preparation) to obtain some routing control information. This routing control information is carried in the message in conjunction with destination addresses in order to help forward nodes make efficient routing decisions. In this sense, our distributed routing algorithm may be considered as a hybrid algorithm.

For simplicity, for each node $v \in V(G)$, its address is also denoted by v. Edge and link (channel) will be used interchangeably. As mentioned before, the multicast set

K is $\{u_0, u_1, \ldots, u_k\}$, where u_0 is the source node and u_1 through u_k are destination nodes. Let $G(V, E)$ represent a 2D mesh topology or an n-cube topology. Without loss of generality, we assume that the width of at least one dimension of the 2D mesh is an even number.

5.1 Heuristic Routing Algorithms for MP and MC

The problem of finding an optimal MC (or MP) seems to be similar to *traveling salesman* problem [54]. In finding an MC (or MP) for multicast routing, however, the destination nodes cannot be visited in an arbitrary order since the host graph is mesh graph or n-cube graph. As a result, some well known heuristic algorithms for solving traveling salesman problem such as *nearest-neighbor* algorithm cannot be applied to this kind of problems. The proposed algorithms are heuristic in nature and make use of the following facts.

F1: There exists a Hamilton cycle (path) in an $N_1 \times N_2$ mesh graph when N_1 or N_2 is even.

F2: There exists a Hamilton cycle (path) in an n-cube graph for any integer n.

F3: Given a 2D mesh graph or an n-cube graph G, and a multicast set K of G, there exists an MC (or MP) in G.

Note that F3 is due to the fact that a Hamilton cycle (path) is an MC (MP) for any multicast set in G. Let C be a Hamilton cycle in the host graph G, where $C=(v_1, v_2, \ldots, v_m, v_1)$, $m = |V(G)|$, and $v_1 = u_0$. We define a mapping h such that $h(v_i) = i$ — the position of v_i in the cycle. For a given Hamilton cycle, C, which is known to all nodes, each node u can determine in advance $h(u)$ and the h values of its

neighboring nodes. In the description of the following routing algorithms, there are two parts. The first part (Fig. 5.1), message preparation, is performed by the source node, and the second part (Fig. 5.2), message routing, is performed by each forward node including the source node. In the message preparation part, a routing control field, D, which carries the destination addresses and some routing information, is prepended to the actual message. In the message routing part, the routing control field D may be modified. The basic idea in this part is to choose next node that is closest in the cycle to the next destination.

Algorithm: The sorted MP algorithm for Message Preparation

Input: Multicast set K and local address u_0

Output: A sorted destination node list D

Procedure:

1. For each u_i $(1 \leq i \leq k)$, if $h(u_i) < h(u_0)$, then set $f(u_i) = h(u_i) + m$, m=$|V(G)|$; otherwise, set $f(u_i) = h(u_i)$.
2. Sort the destination nodes $u_1, \ldots, u_k$ using the f value as the key. Put these sorted nodes in a list D in ascending order.

Figure 5.1. The sorted MP algorithm for message preparation

Theorem 5.1 *Let G represent a 2D mesh or an n-cube graph. Given a multicast set K, the edges selected by the sorted MP algorithm induce an MP for K in G.*

Proof: To prove all the edges selected by the algorithm induce an MP, we first need the following facts.

Fact 1: For any pair of nodes u and v in $V(G)$, if $f(u) = f(v)$, then $u = v$.

Algorithm: The sorted MP algorithm for Message Routing
Input: Local address w, sorted destination list $D = \{d_1, \ldots, d_\ell\}$
Output: A new destination list D' and a neighboring node w', or nothing.
Procedure:

1. If $w = d_1$, then $D' = D - \{d_1\}$ and a copy of the message is sent to the local node; otherwise, $D' = D$.
2. If $D' = \emptyset$, then return nothing.
3. Let d be the first node in D'. For each neighboring node v, set $f(v) = h(v) + m$ if $h(v) < h(u_0)$. Otherwise, set $f(v) = h(v)$. Select a neighboring node w' such that

$$f(w') = \max\{f(p) : f(p) \leq f(d) \text{ and } p \text{ is a neighboring node of } w\}$$

Figure 5.2. The sorted MP algorithm for message routing

Fact 2: If $f(w) < f(d)$, then $f(w) < f(w') \leq f(d)$, where w is the local node, d is the first node in the sorted list of the remaining destination nodes and w' is the next forward node selected by Step 3 (Fig. 5.2).

Fact 1 is obvious since each node has an unique f value. By the definition of f, there exists a neighboring node t of w such that $f(t)=f(w)+1$. That is, $f(w) < f(t) \leq f(d)$ since $f(w) < f(d)$. Note that $f(t) \leq f(w') \leq f(d)$ by the selection of w' in Step 3. Therefore, we conclude that Fact 2 is correct.

According to the above facts, given a local node w and a destination node d, where $f(w) < f(d)$, the next forward node w' can be determined uniquely by Step 3. Now we can prove the theorem by induction on k, the number of the destination nodes.

Basis ($k = 1$): we have $D = \{u_1\}$ and u_1 is the first (and the only) node in the remaining destination node list D'. The routing starts from u_0, $f(u_0) = 1$ and $f(u_0) < f(u_1)$. Suppose that $w_0 = u_0$ and, for $i > 0$, w_i is the forward node at

which source message arrives after traversing i edges from w_0, *i.e.*,, w_i is the forward node selected by w_{i-1} based on Step 3. It is easy to see that $f(u_0) \leq f(w_{i-1}) < f(w_i) \leq f(u_1)$ in terms of Fact 2. Since f is an integer mapping, there must exist an integer ℓ_1 such that $f(w_{\ell_1}) = f(u_1)$, and thus $w_{\ell_1} = u_1$ by Fact 1. Note that w_i is a neighboring node of w_{i-1}. Clearly, the edges selected by the algorithm induce an MP for $K = \{u_1\}$.

Induction: Assume the hypothesis is true for j $(k = j)$. For $k = j + 1$, without loss of generality, suppose that after sorting step in message preparation, $D = \{u_1, u_2, \ldots, u_j, u_{j+1}\}$, where $f(u_i) < f(u_{i+1})$ for $0 \leq i \leq j$. By the induction hypothesis, if $D = \{u_1, u_2, \ldots, u_j\}$, the edges selected by the algorithm induce an MP P, where $P = (w_0, w_1, \ldots, w_{\ell_j})$, $w_0 = u_0$, and $w_{\ell_j} = u_j$. For $D = \{u_1, u_2, \ldots, u_j, u_{j+1}\}$, after the source message has arrived at node u_j, we have $D' = \{u_{j+1}\}$. By using exactly the same arguments as those in the basis (by replacing u_j and u_{j+1} by u_0 and u_1, respectively), it can be shown that the algorithm will route the source message from node u_j, through a path P' to u_{j+1}, where

$$P' = (w_{\ell_j}, w_{\ell_j+1}, w_{\ell_j-2}, \ldots, w_{\ell_{j+1}-1}, w_{\ell_{j+1}}),$$

$w_{\ell_j} = u_j$ and $w_{\ell_{j+1}} = u_{j+1}$. Since $f(w_i) > f(u_j)$ for $\ell_j < i \leq \ell_{j+1}$ and $f(w_i) < f(u_j)$ for $0 \leq i < \ell_j$ by Fact 2, the edges selected by the algorithm induce a MP $P \cup P'$ for $\{u_1, u_2, \ldots, u_j, u_{j+1}\}$, where

$$P \cup P' = (w_0, w_1, \ldots, w_{\ell_j}, w_{\ell_j+1}, \ldots, w_{\ell_{j+1}-1}, w_{\ell_{j+1}}),$$

$w_0 = u_0$, and $w_{\ell_{j+1}} = u_{j+1}$. The proof of this theorem is completed by the principle of induction. □

Corollary 5.1 *The time complexity of the sorted MP algorithm for message preparation is $O(k \log k)$ where G represents 2D mesh or n-cube graphs, and k is the number of the destinations. The time complexity for message routing is $O(1)$ when G represents a 2D mesh topology and $O(n)$ when G is an n-cube topology.*

Proof: Note that the mapping h can be determined in advance for a given host graph and the starting address u_0. Thus, the computation of h value for each node takes $O(1)$ time. Step 1 of the message preparation part takes $O(k)$ time to set the f values for all of the destination nodes. Step 2 only sorts the k destination nodes. A well known sorting algorithm, such as quicksort, may be used with time complexity $O(klogk)$. In the message routing part, both Step 1 and Step 2 take $O(1)$ time when G is a 2D mesh or n-cube graph. Step 3 must examine all the neighboring nodes of w. Since each node in mesh graph G has at most 4 neighboring nodes, this step takes $O(1)$ time. On the other hand, for n-cube graphs, Step 3 can be done in $O(n)$ time since there are n neighboring nodes for each node in G. □

The routing algorithm for MC is similar to the MP algorithm. Suppose C is a Hamilton cycle in G, where $C = (v_1, v_2, \ldots, v_m, v_1)$ and $v_1 = u_0$. Again we define $h(v_i) = i$ for $i > 1$ and $h(v_1) = m + 1$. By adding u_0 (with $h(u_0) = m + 1$) to the destination list D, the MP algorithm can be used to find MC for K.

5.2 Heuristic Routing Algorithms for ST

For general host graph, Kou, Markowsky, and Berman presented an algorithm (KMB algorithm) for constructing Steiner tree [55]. The basic idea is to first construct an auxiliary complete graph G_I that consists of only the Steiner nodes and shows the distance in original graph G between any two of the Steiner nodes, then find a minimum spanning tree T_I of G_I, and replace each edge in T_I by its shortest path in

G and prune it to a Steiner tree. Wall modified the algorithm for use in a distributed environment [56].

Our Steiner tree algorithm is applied to 2D mesh or n-cube graph. As will be shown later, for any three nodes u, v and w in either graph, a node t can be located in a constant time such that t is the nearest node to w among the nodes in all the shortest paths between u and v. Therefore, in constructing a Steiner tree, we consider not only the Steiner nodes but also the nodes in shortest paths between Steiner nodes. It is not difficult to see that our algorithm is at least as good as KMB algorithm in the worst case.

The message preparation part (see Fig. 5.3) of the routing algorithm for ST will sort the destination nodes in some order. A forward node in the tree is called *bypass node* if it only forwards the message. If the message is replicated, the forward node is called a *replicate node*. The computational complexity of the routing algorithm executed at each bypass node is a constant in 2D mesh and n-cube topologies. Each replicate node will perform additional computation to determine which neighboring nodes are to receive a copy of the message. As to be explained later, the maximum number of replicate nodes is $k-1$, where k is the number of destinations. The basic idea of the message routing part (see Fig. 5.4) is to construct a Steiner tree that always adds a closest remaining destination node to it until all of the destination nodes are covered by the Steiner tree.

In Step 4(a) of Fig. 5.4, $d_e(u_i)$ can be computed as follows. Assume that $d_e(u_i) = d(u_i, v)$, $e = (s, t)$, and $v \in P_e$. v can be computed in the following way.

For a 2D mesh graph, Suppose that $x_1 = \min\{s_x, t_x\}$, $x_2 = \max\{s_x, t_x\}$, $y_1 = \min\{s_y, t_y\}$, and $y_2 = \max\{s_y, t_y\}$. Also, let $(u_i)_x = x_i$ and $(u_i)_y = y_i$. It is easy to see that

Algorithm: The greedy ST algorithm for Message Preparation

Input: Multicast set K

Output: A sorted multicast set node list $D = (u_0, u_1, \ldots, u_k)$.

Procedure:

1. Sort K in ascending order with $d(u_0, u_i)$, $1 \leq i \leq k$, as the key. Without loss of generality, suppose $d(u_0, u_1) \leq d(u_0, u_2) \leq \cdots \leq d(u_0, u_k)$ after sorting.

Figure 5.3. The greedy ST algorithm for message preparation

$$v_x = \begin{cases} x_1 & \text{if } x_i < x_1 \\ x_i & \text{if } x_1 \leq x_i \leq x_2 \\ x_2 & \text{if } x_i > x_2 \end{cases} \qquad v_y = \begin{cases} y_1 & \text{if } y_i < y_1 \\ y_i & \text{if } y_1 \leq y_i \leq y_2 \\ y_2 & \text{if } y_i > y_2 \end{cases}$$

For an n-cube graph, If $u_i = a_1 a_2 \ldots a_n$, $s = b_1 b_2 \ldots b_n$ and $t = c_1 c_2 \ldots c_n$. Suppose $v = d_1 d_2 \ldots d_n$, then we have, for $1 \leq j \leq n$

$$d_j = \begin{cases} a_j & \text{if } b_j \neq c_j \\ b_j & \text{if } b_j = c_j \end{cases}$$

Theorem 5.2 *Let G represent a 2D mesh graph or an n-cube graph. Given a multicast set K, the edges selected by the greedy ST algorithm induce a ST for K in G.*

Proof: The message routing part (Fig. 5.4) of the greedy ST algorithm in source node u_0 constructs a tree T which contains all the nodes in K. In Step 3 of Fig. 5.4, $V(T) = \{u, u_1\}$ and $E(T) = (u, u_1)$. Thus, T is a tree after Step 3. The i-th

Algorithm: The greedy ST Algorithm for Message Routing
Input: Local address w, destination list $D = (u, u_1, \ldots, u_\ell)$.
Output: Destination sublist(s): $D_1, D_2, \cdots$, where $D_i \cap D_j = \emptyset$ for $i \neq j$.
Procedure:

1. If $w \neq u$, set D_1=D and send the message to one of w's neighbor that is on a shortest path between w and u. Stop.
2. If $w = u$ and $D - \{u\} = \emptyset$, then the message is sent to the local node w. Stop.
3. Constuct a tree T with u as the root as follows. $E(T) \leftarrow \{(u, u_1)\}$.
4. For $i = 2, 3, \ldots, \ell$, do the following:
   (a) For each $e = (s, t) \in E(T)$
   - Define P_e as the set $\{v : v \in V(G),\ v$ is on a shortest path from s to $t\ \}$.
   - Let $d_e(u_i) = \min_{v \in P_e}\{d(u_i, v)\}$.

   (b) Find a v such that $d(u_i, v) = \min_{e \in E(T)}\{d_e(u_i)\}$, where v is in one of the shortest path(s) between s and t and $(s, t) \in E(T)$.
   (c) If $v \neq s$ and $v \neq t$, then $E(T) \leftarrow E(T) \cup \{(s, v), (v, t)\} - \{(s, t)\}$.
   (d) If $u_i \neq v$, then $E(T) \leftarrow E(T) \cup \{(v, u_i)\}$.
5. Suppose u has m sons in T, $r_1, r_2, \ldots, r_m$, set

   $D_i = \{v : v \in D$ and v is in the subtree of T with r_i as the root$\}$.

6. Send a copy of the message with D_i to one of w's neighbor which is in a shortest path between w and r_i for $1 \leq i \leq m$.

Figure 5.4. The greedy ST algorithm for message routing

iteration of Step 4 in Fig. 5.4 adds at least u_i to T while preserving cycle-free in T. It is not hard to see that each edge in $E(T)$ is replaced by a path in G whose edges are selected by the message routing part of the greedy ST algorithm. In Step 4 of Fig. 5.4, at most two nodes may be added to $V(T)$. Since there are $k-2$ iterations, T has at most $2(k-1)$ nodes. □

Corollary 5.2 *Consider the greedy ST algorithm with k destinations. The time complexity for the message preparation part for both 2D mesh and n-cube graphs is $O(klogk)$. The time complexity for the message routing part is $O(1)$ for the bypass nodes and $O(k^2)$ for the replicate nodes for both 2D mesh and n-cube graphs. The maximum number of replicate nodes is $k-1$.*

Proof: Since the distance between any two nodes can be computed in a constant time (by a simple hardware design) for both 2D mesh and n-cube graphs, the message preparation part in Fig. 5.3 takes $O(k \log k)$ time to sort the destination nodes.

In the message routing part, a bypass node simply executes Step 1 and a leaf (destination) node simply executes Step 2 in Fig. 5.4, which takes a constant time. For a replicate node, Step 3 in Fig. 5.4 takes a constant time. As mentioned above, $d_e(u_i)$ can be computed in $O(1)$ for both 2D mesh and n-cube graphs. Step 4(b) can be done in $O(k)$ time. Both Step 4(c) and Step 4(d) take $O(1)$ time. There are $k-1$ iterations of Step 4. Thus, the time complexity of the message routing part is $O(k^2)$.

In the worst case, if each replicate node makes only one addition copy of the message, it is trivial to see that the maximum number of replicate nodes is $k-1$. □

Note that the basic concept behind the greedy ST algorithm for message routing is to identify those closest replicate nodes to forward the message, where each replicate node, which may be a destination, is the root of a subtree. In order to obtain the

best replicate nodes, all destination nodes have to be considered. Thus, at the end of Step 4 (Fig. 5.4), a complete Steiner subtree rooted at the calling node is obtained. In other words, the source node is able to obtain the complete Steiner tree. Another approach to implement this algorithm is to pass the complete Steiner tree information in the message to all replicate nodes. In this case, the complexity required at each replicate node is reduced to $O(1)$ and $O(n)$ for 2D mesh and n-cube, respectively. However, each message will carry additional Steiner tree information. Thus, there is a tradeoff between message size and processing time. In both implementations, the amount of traffic generated is the same.

5.3 Heuristic Routing Algorithms for MT

Since the multicast routing algorithm for hypercube has been proposed in [19], we only present the routing algorithms of MT for 2D mesh.

In unicast communication in 2D mesh, message is generally delivered first in X-direction then Y-direction. The first multicast routing algorithm in Fig. 5.5, *X-first* multicast routing, is a natural extension of the one-to-one routing method.

It is easy to show the following theorem.

Theorem 5.3 *Given a multicast set K, the links selected by executing X-first multicast routing algorithm in each forward node in 2D mesh induce a multicast subgraph for K. The time complexity of the algorithm is of $O(k)$ for k input destination nodes.*

The X-first algorithm is simple and can be easily implemented in hardware. However, the selected path from the source to a destination node is only determined by the addresses of the source and the destination node. It is independent of the positions of the rest destination nodes. As a result, it often creates much unnecessary traffic.

X-first Multicast Algorithm

Input: Local address (x_0, y_0), destination list D, $D \neq \emptyset$.

Output: Destination node sublist(s): $D_{+X}, D_{-X}, D_{+Y}, D_{-Y}$, $D_i \subseteq D$ for $i \in \{+X, -X, +Y, -Y\}$ and $D_i \cap D_j = \emptyset$ for $i \neq j$.

Procedure:

1. set $D_i = \emptyset$ for i=+X, -X, +Y, -Y.
2. For each $(x, y) \in D$, do:
   Case (x, y) of
   $x > x_0$: $D_{+X} \leftarrow D_{+X} \cup \{(x, y\})\}$;
   $x < x_0$: $D_{-X} \leftarrow D_{-X} \cup \{(x, y)\}$;
   $x = x_0$ and $y > y_0$: $D_{+Y} \leftarrow D_{+Y} \cup \{(x, y)\}$;
   $x = x_0$ and $y < y_0$: $D_{-Y} \leftarrow D_{-Y} \cup \{(x, y)\}$;
   $x = x_0$ and $y = y_0$: send the message to local processor.
3. If $D_{+X} \neq \emptyset$, put D_{+X} to message head and send it to $(x_0 + 1, y_0)$;
   If $D_{-X} \neq \emptyset$, put D_{-X} to message head and send it to $(x_0 - 1, y_0)$;
   If $D_{+Y} \neq \emptyset$, put D_{+Y} to message head and send it to $(x_0, y_0 + 1)$;
   If $D_{-Y} \neq \emptyset$, put D_{-Y} to message head and send it to $(x_0, y_0 - 1)$;

Figure 5.5. X-first multicast algorithm

In the second multicast routing algorithm, we consider all the positions of the destination nodes in a multicast to select possible message passing path(s). The divided greedy multicast algorithm in Fig. 5.6 first divides the destination set into several subsets. The destination nodes in the same subset have the same one or two candidate neighboring nodes to which the message can be sent. If necessary, each subset can be further divided into two sublists, and the two sublists and the source message will be sent to the two candidate neighboring nodes respectively.

Theorem 5.4 *Given a multicast set K, the links selected by executing divided greedy multicast routing algorithm in each forward node in 2D mesh induce a multicast subgraph for K. The time complexity of the algorithm is of $O(k)$ for k input destination nodes.*

Proof: Starting from the source node, the algorithm is executed at each forward node. For an arbitrary destination node (x, y) which is in the input destination node list of a forward node, if it is in D_i of Step 2 for some i, $i \in \{+X, -X, +Y, -Y\}$, the message and (x, y) will be sent to a neighboring node which is in a shortest path from the current forward node to (x, y) in Step 6 . Otherwise, (x, y) is in P_i of Step 3 for some i, $0 \leq i \leq 3$. It will be in either S_{ix} or S_{iy} in Step 4.2. Finally, it will be put in either D_i or $D_{i+1 mod 4}$ in Step 5. By checking the definition of P_i and the routing decision made in Step 6, it is easy to see the message and (x, y) will be sent to a neighboring node which is in a shortest path from the current forward node to (x, y). Therefore, the message is forwarded from source node to each destination node along a shortest path. The links selected by executing the algorithm in each forward node induce a multicast subgraph for the given multicast set.

Step 1 and Step 2 need $O(k)$ and $O(1)$ time respectively. Step 3 to Step 6 each takes $O(k)$ time. Hence, the time complexity of the divided greedy multicast algorithm is $O(k)$, where k is the number of the input destination nodes. □

Divided Greedy Multicast Agorithm

Input: Local address (x_0, y_0), destination node list D, $D \neq \emptyset$.

Output: Destination sublist(s): $D_{+X}, D_{+Y}, D_{-X}, D_{-Y}$, $D_i \subseteq D$ for $i \in D$ and $D_i \cap D_j = \emptyset$ for $i \neq j$.

Procedure:

1. If $(x_0, y_0)=(x, y)$ for same (x, y), $(x, y) \in D$, send the message to local processor and $D \leftarrow D - \{(x, y)\}$;
2. If $D = \emptyset$, stop; else $S \leftarrow D$;
3. Set
   $D_{+X} \leftarrow \{(x, y) \mid (x, y) \in D, x > x_0 \text{ and } y = y_0\}$,
   $D_{+Y} \leftarrow \{(x, y) \mid (x, y) \in D, x = x_0 \text{ and } y > y_0\}$,
   $D_{-X} \leftarrow \{(x, y) \mid (x, y) \in D, x < x_0 \text{ and } y = y_0\}$,
   $D_{-Y} \leftarrow \{(x, y) \mid (x, y) \in D, x = x_0 \text{ and } y < y_0\}$;
4. Let
   $P_0 = \{(x, y) \mid (x, y) \in D, x > x_0 \text{ and } y > y_0\}$,
   $P_1 = \{(x, y) \mid (x, y) \in D, x < x_0 \text{ and } y > y_0\}$,
   $P_2 = \{(x, y) \mid (x, y) \in D, x < x_0 \text{ and } y < y_0\}$, and
   $P_3 = \{(x, y) \mid (x, y) \in D, x > x_0 \text{ and } y < y_0\}$;
   For i=0, 1, 2, 3 do:
   (a) $S_{ix} \leftarrow \emptyset, S_{iy} \leftarrow \emptyset$;
   $(x_{ix}, y_{ix}) \leftarrow (x_0, y_0), (x_{iy}, y_{iy}) \leftarrow (x_0, y_0)$;
   (b) For each destination (x, y) in P_i do:
   Let $H = \min\{\mid y - y_0 \mid, d_G((x, y), (x_{ix}, y_{ix}))\}$ and
   $V = \min\{\mid x - x_0 \mid, d_G((x, y), (x_{iy}, y_{iy}))\}$;
   If $H \leq V$, then $S_{ix} \leftarrow S_{ix} \cup \{(x, y)\}, (x_{ix}, y_{ix}) \leftarrow (x, y)$;
   else $S_{iy} \leftarrow S_{iy} \cup \{(x, y)\}, (x_{iy}, y_{iy}) \leftarrow (x, y)$;
5. Let S_{00}=S_{0x}, S_{01}=S_{3x}, and S_{10}=S_{0y}, S_{11}=S_{1y}, and S_{20}=S_{1x}, S_{21}=S_{2x}, and S_{30}=S_{2y}, S_{31}=S_{3y}; and $D_0 = D_{+X}$, $D_1 = D_{+Y}$ $D_2 = D_{-X}$ and $D_3 = D_{-Y}$;
   For i=0, 1, 2, 3 do:
   If $D_i \neq \emptyset$ or both $S_{i0} \neq \emptyset$ and $S_{i1} \neq \emptyset$ then $D_i \leftarrow D_i \cup S_{i0} \cup S_{i1}$;
   else $D_{(i-1) mod 4} \leftarrow D_{(i-1) mod 4} \cup S_{i0}$, $D_{(i+1) mod 4} \leftarrow D_{(i+1) mod 4} \cup S_{i1}$;
6. If $D_{+X} \neq \emptyset$, put D_{+X} to message head and sent it to $(x_0 + 1, y_0)$;
   If $D_{+Y} \neq \emptyset$, put D_{+Y} to message head and sent it to $(x_0, y_0 + 1)$;
   If $D_{-X} \neq \emptyset$, put D_{-X} to message head and sent it to $(x_0 - 1, y_0)$;
   If $D_{-Y} \neq \emptyset$, put D_{-Y} to message head and sent it to $(x_0, y_0 - 1)$.

Figure 5.6. Divided greedy algorithm

5.4 Illustrative Examples

Five examples are presented in this section in order to illustrate the operations of the proposed heuristic multicast algorithms.

Finding an MP in a 4×4 Mesh

Consider a 4×4 mesh, G. For simplicity, we use an integer to represent the address of each node as shown in Fig. 5.7. Obviously, $C = (0, 1, 2, 3, 7, 6, 5, 9, 10, 11, 15, 14, 13, 12, 8, 4, 0)$ is a Hamilton cycle in G. For each node x ($0 \leq x \leq 15$), Table 5.1 shows the corresponding $h(x)$. Consider the multicast set $K = \{9, 0, 1, 6, 12\}$, where $u_0 = 9$ is the source. Given $u_0 = 9$ as the source, the sorting key $f(x)$ for each node x is shown in Table 5.2.

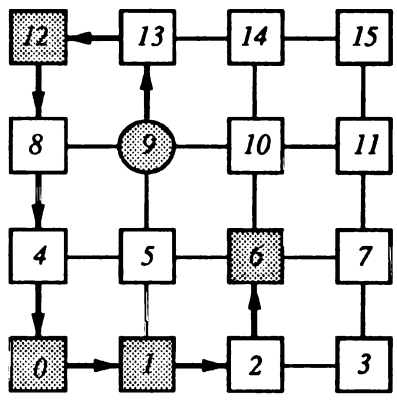


Figure 5.7. A 4×4 mesh with node 9 as the source.

Based on the sorted MP algorithm for message preparation shown in Fig. 5.1, we have a sorted destination list $D = \{12, 0, 1, 6\}$. Then, we apply the sorted MP algorithm for message routing (Fig. 5.2) for each node receiving the message. For

Table 5.1. A Hamilton cycle and the corresponding mapping h of a 4×4 mesh.

$h(x)$	x	$h(x)$	x	$h(x)$	x	$h(x)$	x
1	0	5	7	9	10	13	13
2	1	6	6	10	11	14	12
3	2	7	5	11	15	15	8
4	3	8	9	12	14	16	4

Table 5.2. The sorting key $f(x)$ and mapping $h(x)$ for each node x in a 4×4 mesh, where $u_0 = 9$.

x	$h(x)$	$f(x)$	x	$h(x)$	$f(x)$
0	1	17	8	15	15
1	2	18	9	8	8
2	3	19	10	9	9
3	4	20	11	10	10
4	16	16	12	14	14
5	7	23	13	13	13
6	6	22	14	12	12
7	5	21	15	11	11

node 9, its four neighboring nodes are 5, 8, 13, and 10. The corresponding f values are 23, 15, 13, and 9, respectively. Since $f(d = 12) = 14$, only node 13 and node 10 have f values less then 14 and $f(d = 13) > f(d = 10)$. Thus, node 13 is selected by node 9 to forward the message. By repeating the same procedure for each node receiving the message, we obtain the multicast path $(9, 13, 12, 8, 4, 0, 1, 2, 6)$ as shown in Fig. 5.7.

Finding an MP in a 4-cube

Consider a 4-cube shown in Fig. 5.8. One can easily obtain a Hamilton cycle shown in Table 5.3. For each node x, the corresponding $h(x)$ is also shown in Table 5.3. Suppose we have the multicast set $K = \{0011, 0100, 0111, 1100, 1010, 1111\}$. Given

$u_0 = 0011$ as the source, Table 5.4 lists the sorting key $f(x)$ for each node x.

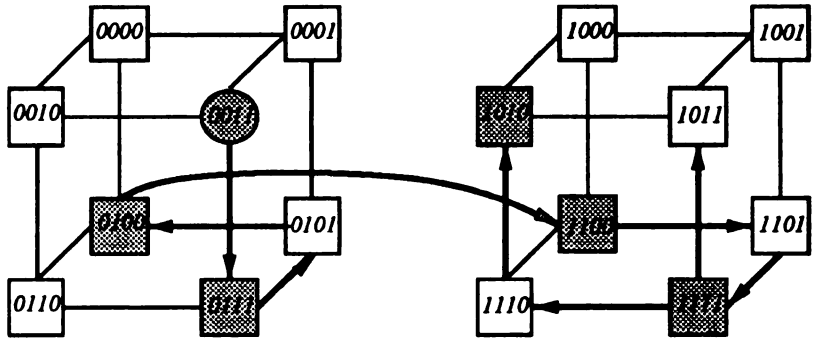


Figure 5.8. A 4-cube with node 9 as the source.

Table 5.3. A Hamilton cycle and the corresponding mapping h of a 4-cube.

$h(x)$	x	$h(x)$	x	$h(x)$	x	$h(x)$	x
1	0000	5	0110	9	1100	13	1010
2	0001	6	0111	10	1101	14	1011
3	0011	7	0101	11	1111	15	1001
4	0010	8	0100	12	1110	16	1000

By applying the sorted MP algorithm for message preparation, we have a sorted destination list $D = \{0111, 0100, 1100, 1111, 1010\}$. By applying the sorted MP algorithm for message routing (Fig. 5.2) to each node receiving the message, we obtain the following multicast path $(0011, 0111, 0101, 0100, 1100, 1101, 1111, 1110, 1010)$ as illustrated in Fig. 5.8.

Finding an ST in an 8×8 Mesh

Let G be an 8×8 mesh. Each node in G is indexed as $[i, j]$ for $0 \leq i, j \leq 7$.

Table 5.4. The sorting key $f(x)$ and mapping $h(x)$ for each node x in a 4-cube, where $u_0 = 9$.

x	$h(x)$	$f(x)$	x	$h(x)$	$f(x)$
0000	1	17	1000	16	16
0001	2	18	1001	15	15
0010	4	4	1010	13	13
0011	3	3	1011	14	14
0100	8	8	1100	9	9
0101	7	7	1101	10	10
0110	5	5	1110	12	12
0111	6	6	1111	11	11

Suppose node $(2,7)$, the source node, wishes to send a message to 5 destinations: $[0,5], [2,3], [4,1], [6,3]$ and $[7,4]$. By applying the greedy ST algorithm for message preparation (Fig. 5.3), we have $D = ([(2,7], [0,5], [2,3], [4,1], [6,3], [7,4])$. Note that if $d(u_0, u_i) = d(u_0, u_j)$, u_i and u_j can be placed in an arbitrary order. Let's consider node $[2,7]$, which is the source with the sorted input D, by applying the greedy ST algorithm for message routing (Fig. 5.4). Obviously, Step 1 and Step 2 are skipped. In Step 3, we have $E(T) = \{([2,7], [0,5])\}$. In the first iteration of Step 4, node $[2,5]$ is the nearest node to $[2,3]$ among the nodes in any shortest paths between $[2,7]$ and $[0,5]$. Thus, $[2,5]$ is selected in Step 4(b). After the first iteration of Step 4, $E(T)$ becomes $\{([2,7], [2,5]), ([2,5], [0,5]), ([2,5], [2,3])\}$. Nodes $[4,1]$, $[6,3]$, and $[7,4]$ are added to the Steiner tree in the second, third, and fourth iteration of Step 4, respectively. At the end of Step 4, we have $E(T) =$ { ([2,7],[2,5]), ([2,5],[0,5]), ([2,5],[2,3]), ([2,3],[4,3]), ([4,3],[4,1]), ([4,3],[6,3]), ([6,3],[7,4])}, which specifies the complete Steiner tree. Thus, in Step 5, we have output $D_1 = ([2,5], [0,5], [2,3], [4,1], [6,3], [7,4])$, which will be sent along with the message to the neighboring node $[2,6]$. Note that node $[2,6]$ is a bypass node which is the neighboring node on one of the shortest paths between $[2,7]$ and the next replicate node $[2,5]$.

Upon receiving the message, node [2,5] will execute the greedy ST algorithm for message routing. It will generate $D_1 = ([0,5])$ and $D_2 = ([2,3],[4,1],[6,3],[7,4])$. One copy of the message with D_1 will be sent to node [1,5], a bypass node, and another copy of the message with D_2 will be sent to the bypass node [2,4]. By repeating this procedure at all forward nodes, the complete Steiner tree routing pattern is shown in Fig. 5.9. Note that the selection of bypass nodes is based on the underlying shortest path routing algorithm.

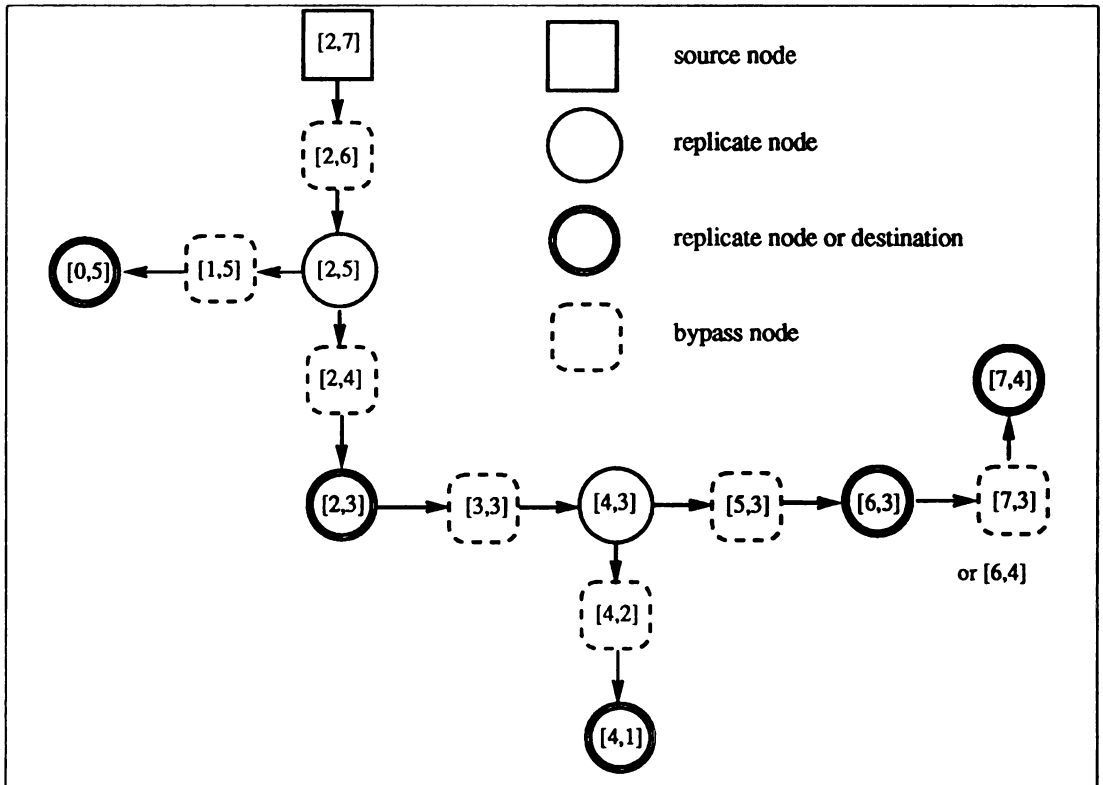


Figure 5.9. A complete ST routing pattern in an 8×8 mesh.

Finding an ST in a 6-cube

Let's consider a 6-cube with a multicast set K = { 000110, 010101, 001101, 000001,

101001, 110001 }. The sorted destination list is D = (000110, 010101, 000001, 001101, 101001, 110001). The source node 000110 is a replicate node. In this node, neither Step 1 nor Step 2 is executed, and we have $E(T) = \{(000110, 010101)\}$ in Step 3. Since node 000101 is the nearest node to node 000001 among the nodes in any shortest paths between 000110 and 010101, it is selected in Step 4(b). At the end of the first iteration of Step 4, $E(T)$ is {(000110,000101), (000101,010101), (000101,000001)}. In the second iteration, node 001101 is added to the Steiner tree, and node 101001 and node 110001 will be added to the Steiner tree in the third and fourth iteration, respectively. Finally, we have output D_1 = (000101, 010101, 000001, 001101, 101001, 110001). By repeating the greedy ST algorithm for message routing (Fig. 5.4) to each message receiving node, the resulting Steiner tree routing pattern is shown in Fig. 5.10.

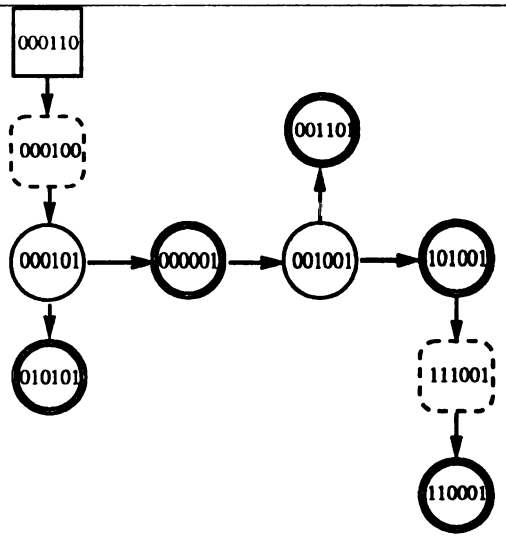


Figure 5.10. A complete ST routing pattern in a 6-cube.

Finding an MT in an 6 × 6 Mesh

The following example will be used to explain the X-first algorithm for MT. Con-

sider a 6 × 6 mesh in Fig. 5.11, Suppose source node (3, 2) wants to send message to (2,0), (3,0), (4,0), (1,1), (5,1), (0,2), (1,3), (2,5), (3,5) and (5,5). At the beginning, source node (3,2) runs X-first algorithm. After running the algorithm, we get

$D_{+X} = \{(4,0),(5,1),(5,5)\}$,

$D_{-X} = \{(2,5),(2,0),(1,3),(1,1),(0,2)\}$,

$D_{+Y} = \{(3,5)|\}$ and $D_{-Y} = \{(3,0)\}$,

which are to be added to message head, respectively, and sent to its neighboring nodes (4,2), (2,2), (3,3) and (3,1) respectively. Then the forward nodes (4,2), (2,2), (3,3) and (3,1) will run the algorithm and so on until all the destination nodes receive the message. The routing pattern is shown in Fig. 5.11, where each path from source to each destination node is always goes X-direction first then Y-direction. The total traffic for this example is 24.

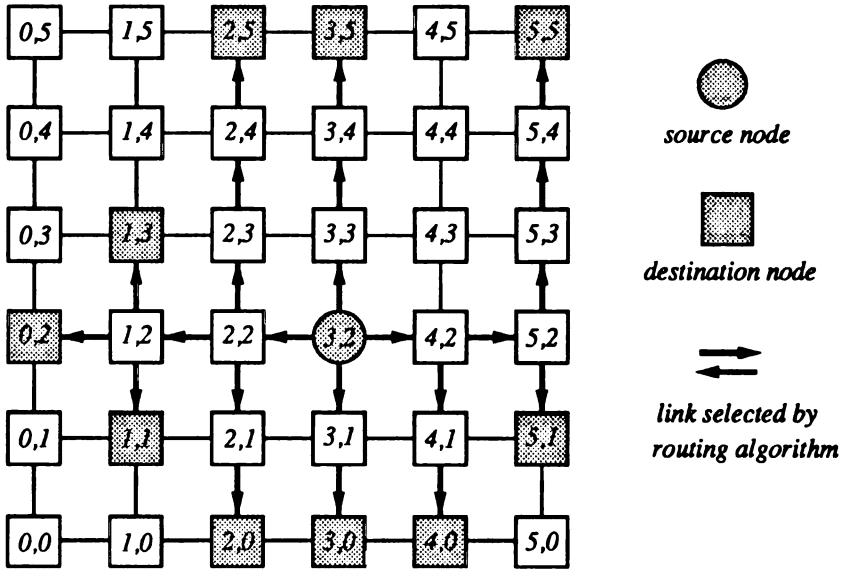


Figure 5.11. The routing pattern of X-first algorithm for the example.

The same example is used to clarify divided greedy algorithm which is much more complicated than X-first algorithm.

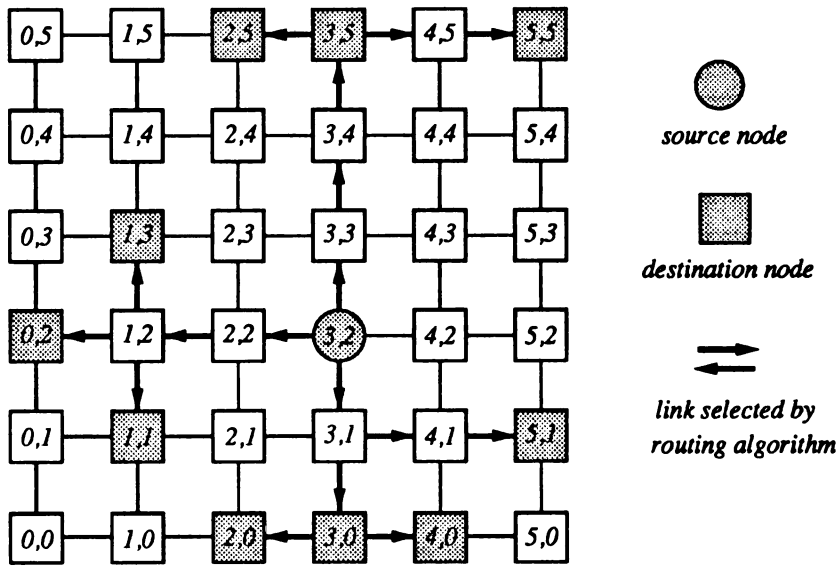


Figure 5.12. The routing pattern of divided greedy algorithm for the example.

Again, suppose source node (3,2) is going to send message to each destination node in destination list D, where D={(2,0), (3,0), (4,0), (1,1), (5,1), (0,2), (1,3), (2,5), (3,5), (5,5)}, as shown in Fig. 5.12. Initially, the only forward node, i.e. the source node (3,2) runs the divided greedy algorithm as follows. Step 1 and Step 2 will do nothing since no destination node here is equal to the local node (3,2) and the input destination list D is not empty. In Step 3, node (3,5) is put to D_{+Y} in order to sent to (3,3), node (0,2) is put to D_{-X}, and (3,0) to D_{-Y}, which will be sent to (2,2) and (1,3) respectively. By the definition of P_i, $0 \leq i \leq 3$, in Step 4, we have

$P_0 = \{(5,5)\}$, $P_1 = \{(1,3),(2,5)\}$,

$P_2 = \{(1,1),(2,0)\}$ and $P_3 = \{(4,0),(5,1)\}$.

After executing Step 4, P_i, $0 \leq i \leq 3$, is divided into two subsets S_{ix} and S_{iy}, $0 \leq i \leq 3$, where

$S_{0x} = \emptyset$, $S_{0y} = \{(5,5)\}$, $S_{1x} = \{(1,3)\}$, $S_{1y} = \{(2,5)\}$,

$S_{2x} = \{(1,1)\}$, $S_{2y} = \{(2,0)\}$ and

$S_{3x} = \{(5,1)\}$, $S_{3y} = \{(4,0)\}$.

Note that S_{0x} and S_{3x} are the candidate sets for D_{-X}, S_{0y} and S_{1y} for D_{+Y}, S_{1x} and S_{2x} for D_{-X}, and S_{2y} and S_{3y} for D_{-Y}. However, since S_{0x} is empty, in Step 5, its partner S_{3x} is not put to D_{+X}, and instead it will be merged with S_{3y} which in turn will be put to D_{-Y}. Therefore, by the end of Step 5, we get the output destination node sublists:

$D_{+Y} = \{(3,5),(2,5),(5,5)\}$, $D_{-X} = \{(0,2),(1,3),(1,1)\}$ and

$D_{-Y} = \{(3,0),(2,0),(4,0),(5,1)\}$.

D_{+Y}, D_{-X} and D_{-Y} will be sent, along with the source message, to (3,3), (2,2) and (3,1) respectively.

Next, the forward nodes are (3,3), (2,2) and (3,1). The algorithm is executed in each forward node with its input destination list in the same way as the one described above for node (3,2). Finally, the message will be sent to each destination node and the routing paths form a multicast subgraph for the source and destination nodes, which is shown in Fig. 5.12. Note that the destination node list for a non-source forward node can only be divided to at most two sublists of P_i, $0 \leq i \leq 3$, in Step 4. This is because it is shortest path routing. For example, for the forward node (3,3) with input destination node list $\{(3,5),(2,5),(5,5)\}$, the list can only be divided into two sets: P_0 and P_1 in Step 4 for further consideration. Thus, the execution of the algorithm in the non-source forward node will be much faster than that in source node.

CHAPTER 6

DEADLOCK-FREE MULTICAST WORMHOLE ROUTING

As described in Chapter 2, wormhole routing offers a best-case latency that is independent of the path length while requiring small amount of dedicated buffers. Wormhole routing has been a popular choice in new generation multicomputers. Since the blocked messages are not being buffered at each intermediate node in wormhole routing, its routing criteria and deadlock properties are quite different from those in the other switching technologies such as store-and-forward and virtual cut-through methods. This chapter focuses on the routing and deadlock issue in multicast communication in the most promising wormhole networks.

6.1 Deadlock Issue in Multicast Wormhole Routing

A practical multicast routing algorithm must be deadlock-free and should transmit the source message to each destination node using as little time and as few com-

munication channels as possible. For a given set of destination nodes, the multicast routing algorithm might deliver the message along a common path as far as possible, then branch in order to deliver the message to each destination node. Essentially, the message follows a tree-like route to the destinations. One possible approach to implementing such a multicast tree is to extend deadlock-free unicast routing algorithms to handle multicast traffic. For example, the E-cube algorithm has been proved to be deadlock-free for one-to-one communication in n-cubes [44].

The nCUBE-2 [6], which is the first hypercube multicomputer to use wormhole routing, uses this approach to support broadcast and a special form of multicast in which the destination nodes form a subcube. A tree is created in which each path from the source to a destination uses E-cube routing. In order to avoid buffering, when the tree branches and a node must send the message on multiple channels, all of the required channels must be available before transmission on any of them may take place. Hence, the branches of the tree proceed forward in a lock-step fashion. Blockage of any branch of the tree can prevent completion of the message transfer even to those destination nodes to which paths have been successfully created. Moreover, deadlock can occur using such a routing scheme. Figure 6.1 shows a deadlock configuration on a 3-cube. Suppose that nodes 000 and 001 simultaneously attempt to transmit broadcast messages $M0$ and $M1$, respectively. Figures 6.1a and 6.1b, respectively, depict the would-be trees for these broadcasts. Figure 6.2 shows a detailed diagram of six of the nodes involved in the broadcasts. Outgoing channel selectors are represented by solid rectangles, while input buffers are represented by open rectangles. The broadcast originating at node 000, $M0$, has acquired channels [000, 001], [000, 010], and [000, 100]. The header flit has been duplicated at node 001 and is waiting on channels [001, 011] and [001, 101]. The $M1$ broadcast has already acquired channels [001, 011], [001, 101] and [001, 000], but is waiting on channels [000, 001] and [000, 010]. The two broadcasts will block forever.

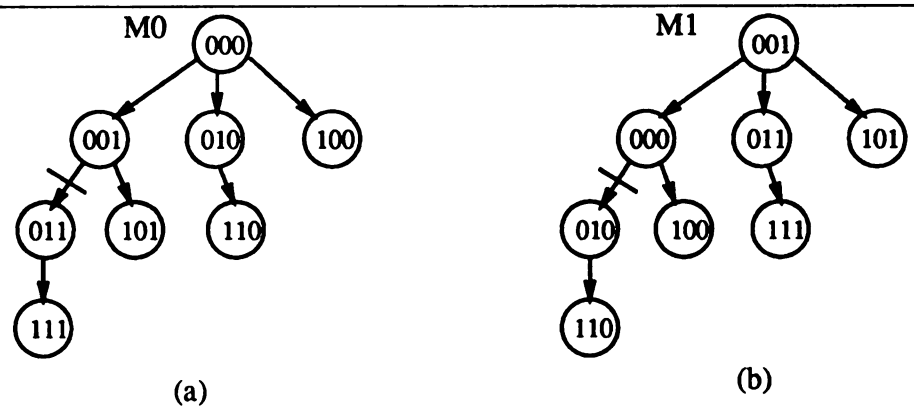


Figure 6.1. Deadlock in a 3-cube multicomputer.

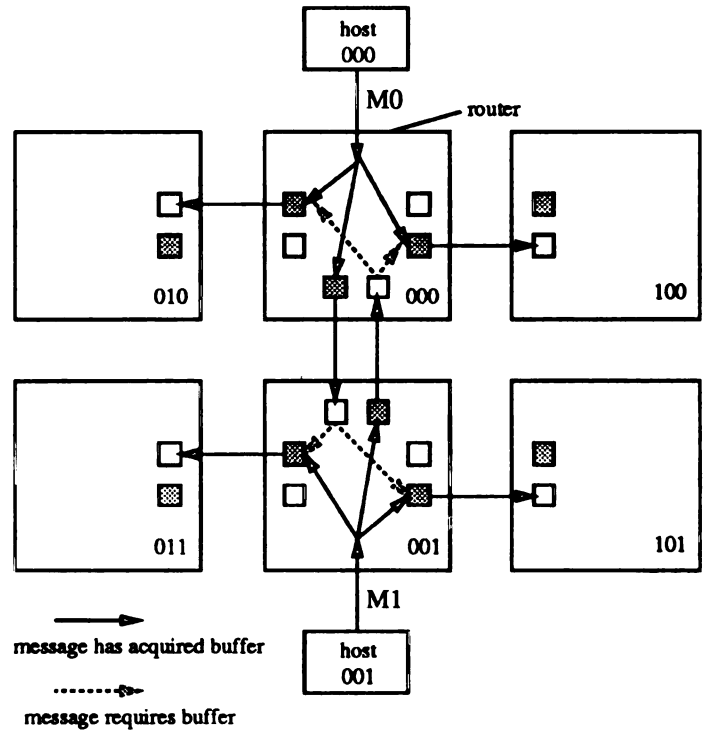


Figure 6.2. The detailed diagram of deadlock configuration.

In a similar manner, one may attempt to extend deadlock-free unicast routing on a 2D mesh to encompass multicast. One such unicast routing algorithm requires that messages be sent first in the X-direction and then in the Y-direction. It is straightforward to prove that this algorithm is deadlock-free. An extension of the X-first routing method to include multicast is shown in Figure 6.3, in which the message is delivered in to each destination in the manner described. As in the n-cube example, the progress of the tree requires that all branches be unblocked. For example, suppose the header flit in Figure 6.3 is blocked due to a busy channel [(4,2), (4,3)]. Unlike store-and-forward or virtual cut-through routing, node (4,2) cannot buffer the entire message. As a result of this phenomenon, the progress of messages in the entire routing tree must be stopped. All of the its flits stop forwarding and remain in contiguous channels of the network. In turn, other messages requiring segments of this tree are also blocked. Network congestion may be increased and degrade the performance of the multicomputer.

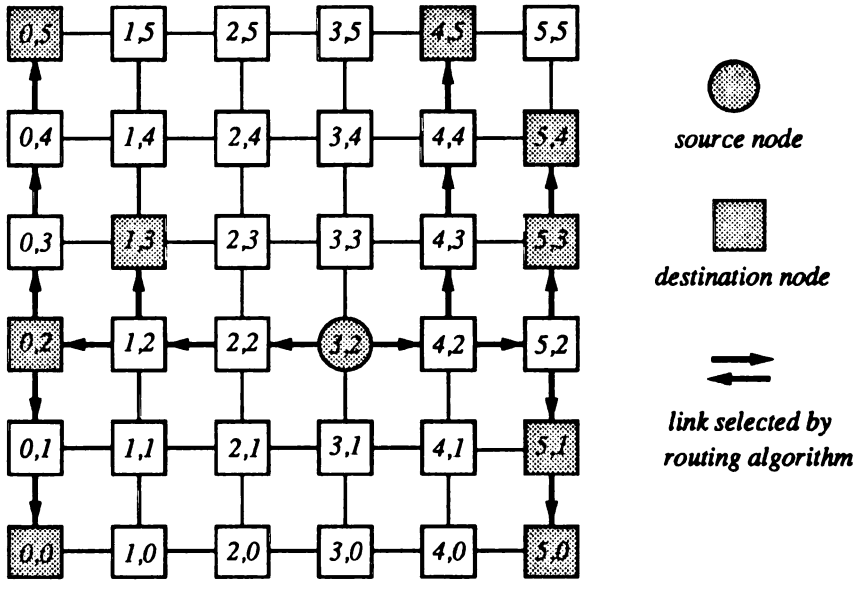
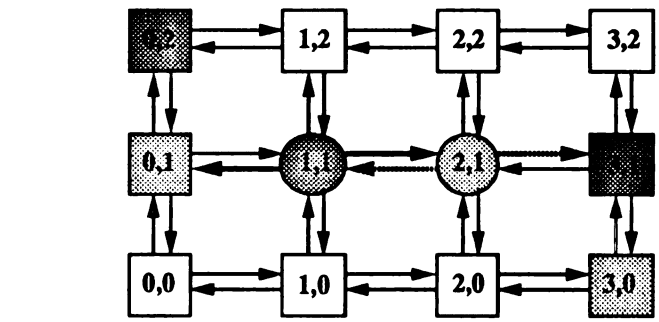


Figure 6.3. An X-first multicast routing pattern.

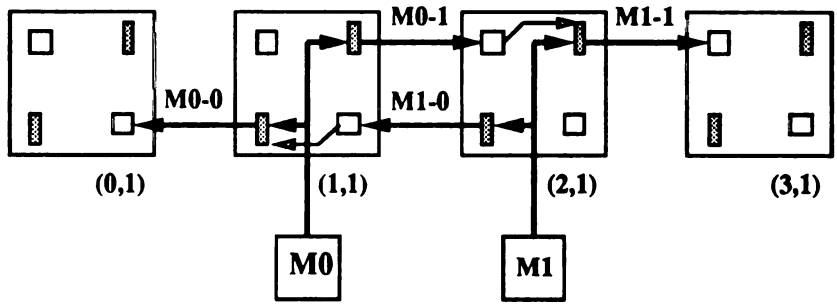
Moreover, this routing algorithm can lead to deadlock. Figure 6.4 shows a deadlock configuration in which two multicasts, $M0$ and $M1$, have the following communication patterns:

```
M0:
  source: (1,1)
  destinations: (0,2), (3,1)
  acquired: [(1,1),(0,1)],
            [(2,1),(3,1)];
  requiring: [(2,1),(3,1)];
```

```
M1:
  source: (2,1);
  destinations: (0,1), (3,0);
  acquired: [(2,1),(1,1)],
            [(1,1),(2,1)];
  requiring: [(1,1), (0,1)];
```



(a)



(b)

Figure 6.4. A deadlock situation in a 3×4 mesh.

In Figure 6.4a, $M0$ has acquired channel $[(1,1),(0,1)]$ and requires channel $[(2,1),(3,1)]$, while $M1$ has acquired channel $[(2,1),(3,1)]$ and requires channel $[(1,1),(0,1)]$. Figure 6.4b shows the details of the situation for the nodes (0,1), (1,1), (2,1), and (3,1). Because neither node (1,1) nor node (2, 1) buffers the blocked message, channels $[(1,1),(0,1)]$ and $[(1,1),(0,1)]$ cannot be released. Deadlock has occurred.

The goal of our research is to develop and evaluate deadlock-free multicast routing algorithms in order to support reliable and efficient parallel applications in multicomputers. In the next two sections, we present deadlock-free multicast wormhole routing schemes based on the concept of network partitioning for 2D mesh and hypercube respectively. The basic idea is to divide a multicast into several sub-multicasts, each routed in a different acyclic subnetwork. Because the subnetworks are disjoint and acyclic, no cyclic resource dependency can exist. Thus, the routing schemes are deadlock-free.

6.2 Deadlock-Free Multicast in 2D Mesh

We first present several deadlock-free multicast wormhole routing algorithms for 2D mesh multicomputers.

6.2.1 Tree-Like Deadlock-Free Multicast Routing Schemes

The first deadlock-free multicast wormhole method that we discuss uses a modification of X-first routing algorithm discussed earlier and shown to be susceptible to deadlock. In order to avoid cycles of channel dependencies, we first double each channel on the 2D mesh, then partition the network into four subnetworks, $N_{+X,+Y}$, $N_{-X,+Y}$, $N_{-X,+Y}$ and $N_{+X,-Y}$. Subnetwork $N_{+X,+Y}$ contains the unidirectional channels with addresses $[(i,j),(i+1,j)]$ and $[(i,j),(i,j+1)]$, subnetwork $N_{+X,-Y}$ contains channels with

addresses $[(i,j),(i+1,j)]$ and $[(i,j),(i,j-1)]$, and so on. Figure 6.5 shows the partitioning of a 3×4 mesh into the four subnetworks.

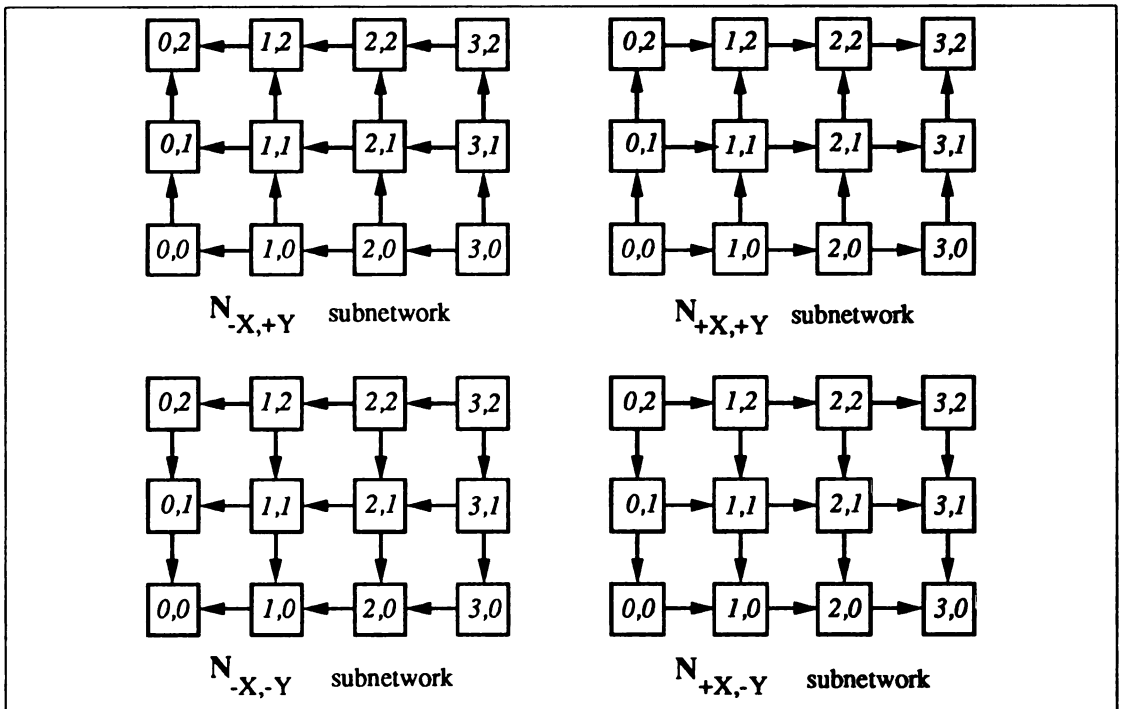


Figure 6.5. Network partitioning for 3×4 mesh.

For a given multicast, the destination node set D will be divided into at most four subsets, $D_{+X,+Y}$, $D_{-X,+Y}$, $D_{-X,+Y}$ and $D_{+X,-Y}$, according to the relative positions of the destination nodes and the source node u_0. Set $D_{+X,+Y}$ contains the destination nodes to the upper right of u_0, $D_{-X,+Y}$ contains the destinations to the upper left of u_0, and so on. Formally, we have

$$D_{+X,+Y} = \{ (x,y) | (x,y) \in D \text{ and } x > x_0, y \geq y_0 \},$$

$$D_{-X,+Y} = \{ (x,y) | (x,y) \in D \text{ and } x \geq x_0, y > y_0 \},$$

$$D_{+X,-Y} = \{ (x,y) | (x,y) \in D \text{ and } x < x_0, y \leq y_0 \}, \text{ and}$$

$D_{-X,-Y} = \{ (x,y) | (x,y) \in D \text{ and } x \geq x_0, y < y_0 \}$.

Specifically, the multicast will comprise into at most four submulticasts from u_0 to each of $D_{+X,+Y}$, $D_{-X,+Y}$, $D_{-X,+Y}$, and $D_{+X,-Y}$. The submulticast to $D_{+X,+Y}$ will be implemented in subnetwork $N_{+X,+Y}$ using X-first Y-next routing, $D_{-X,+Y}$ in subnetwork $N_{-X,+Y}$, and so on. The message routing algorithm is given in Fig. 6.6.

Algorithm: The double-channel X-first routing algorithm for subnetwork $N_{+X,+Y}$

Input: Destination set D', D'=$D_{+X,+Y}$, local address $v, v = (x,y)$;

Procedure:

1. If $x < Min\{x_i | (x_i, y_i) \in D'\}$, send message and D' to $(x+1, y)$. Stop.
2. If $(x,y) \in D'$, then $D' \leftarrow D' - \{(x,y)\}$ and a copy of the message is sent to the local node.
3. Let $D'_Y = \{(x_i, y_i) | x_i = x, (x_i, y_i) \in D'\}$. If $D'_Y \neq \emptyset$, send message and D'_Y to $(x, y+1)$; If $D' - D'_Y \neq \emptyset$, send message and $D' - D'_Y$ to $(x+1, y)$.

Figure 6.6. Double-channel X-first routing algorithm.

The following example is used to explain the algorithm. Consider the 6×6 mesh in Figure 6.3. At the source node, (3,2), the destination set

$D = \{(0,0), (0,2), (0,5), (1,3), (4,5), (5,0), (5,1), (5,3), (5,4)\}$,

is divided into

$D_{+X,+Y} = \{(4,5), (5,3), (5,4)\}$, $D_{-X,+Y} = \{(0,5), (1,3)\}$,

$D_{-X,-Y} = \{(0,0), (0,2)\}$, and $D_{+X,-Y} = \{(5,0), (5,1)\}$.

The message will be sent to the destination nodes in $D_{+X,+Y}$ through subnetwork $N_{+X,+Y}$, to the nodes in $D_{-X,+Y}$ using subnetwork $N_{-X,+Y}$, to the nodes in $D_{-X,-Y}$

using subnetwork $N_{-X,-Y}$, and to the nodes in D_{+-Y} using subnetwork $N_{+X,-Y}$, respectively. The routing pattern is shown in Figure 6.7.

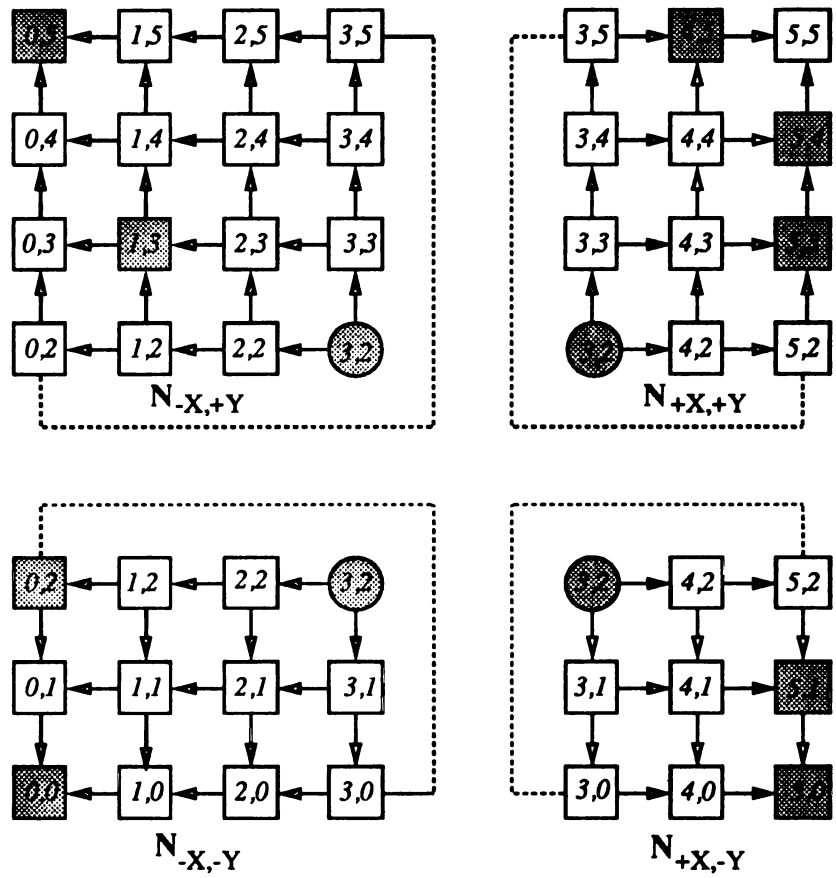


Figure 6.7. The routing pattern of the tree algorithm.

Assertion 1 *The double-channel X-first routing algorithm is deadlock-free.*

Proof: Because the subnetworks are channel-disjoint, we can show the assertion in each subnetwork separately. Without loss of generality, we only show it for subnetwork $N_{+X,+Y}$. First, we label the nodes such that node (0,0) has label 0, and and all nodes at distance i from node (0,0) have higher labels than all nodes at distance i-1 from (0,0). Figure 6.8(a) shows such a label assignment in the above

example. Next, we label all channels entering node with the same labels as the node as shown in Figure 6.8(b). Using X-first Y-next routing, a message entering a node on a channel labed with i always leaves on a channel labeled with a number greater than i. Therefore, no cycle dependency of the channels can exist and it is deadlock free. For the rest of the subnetworks, the proofs are similar to the above one. □

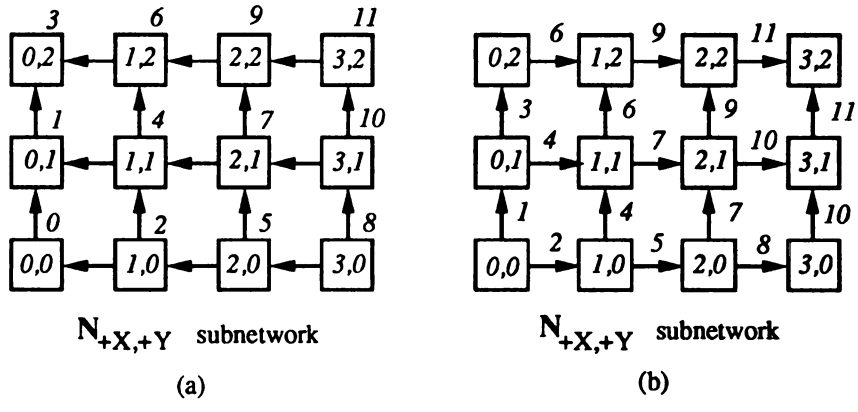


Figure 6.8. The label assignment for $N_{+X,+Y}$ for channel ordering.

While the X-first multicast tree approach avoids deadlock, one of the major disadvantages is the need for double channels. It may be possible implement double channels with virtual channels [44], however, early analysis shows that the signaling for multicast communication may be quite complex. Another major disadvantage of tree-like routing is high block probability. As mentioned before, if one of branches of the routing tree is blocked, the progress of the entire tree has to be stopped.

Based on multicast star model proposed in Chapter 3, we next introduce multicast routing algorithms that are deadlock-free and which do not require additional physical or virtual channels. Deadlock situations arise in multicast trees when copies are

created and diverge at intermediate nodes. The algorithms in the next section avoid deadlock by enforcing the rule that a message, once in the network, may never be copied.

6.2.2 Path-Like Deadlock-Free Multicast Routing Schemes

We first present a network partitioning strategy based on Hamiltonian paths, which is fundamental to the deadlock-free routing schemes. A *Hamiltonian path* visits every node in a graph once and only once. It is not difficult to see that a 2D mesh has many Hamiltonian paths.

In our algorithms, each node i in a multicomputer is assigned a label, $\ell(i)$. In a network with N nodes, the assignment of the label to a node is based on the position of that node in a Hamiltonian path, where the first node in the path is labeled 0 and the last node in the path is labeled $N-1$. Each node is represented by its integer coordinate (x, y).

Figure 6.9(a) shows such a labeling in a 4×3 mesh. The labeling effectively divides the network into two subnetworks. The *high-channel subnetwork* contains all of the channels whose direction is from lower labeled nodes to higher labeled nodes, and the *low-channel network* contains all of the channels whose direction is from higher labeled nodes to lower labeled nodes.

First, let's consider the case of one-to-one communications. If the label of the destination node is greater than the label of the source node, the routing always takes place in the high-channel network; otherwise, it will take the low-channel network. Given a source and a destination node, it can be easily observed from Fig. 6.9 that there always exists a shortest path to deliver the message. Obviously, the performance of a routing scheme is dependent on the selection of a Hamilton path. Figure 6.10 shows the label assignment for a 4×3 mesh based on a different Hamilton path and the corresponding high-channel and low-channel networks. Under such a label

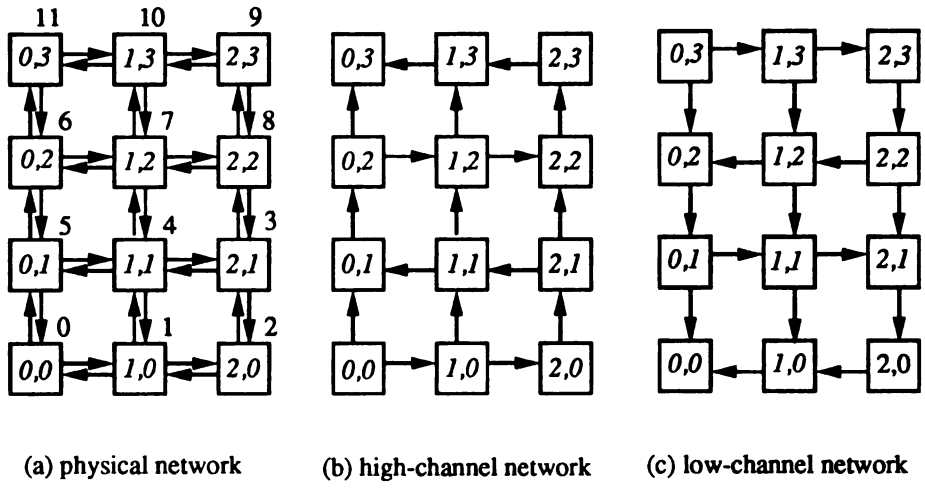


Figure 6.9. The labeling of a 4 × 3 mesh.

assignment, the routing paths between the nodes (1,0) (label 4) and (1,2) (label 8) take 4 channels in either direction (see Fig. 6.10(b-c)) rather than a shortest path which should take only 2 channels.

The label assignment function ℓ for a $m \times n$ mesh can be expressed as

$$\ell(x,y) = \begin{cases} y * n + x & \text{if } y \text{ is even} \\ y * n + n - x - 1 & \text{if } y \text{ is odd} \end{cases}$$

The first step in finding a deadlock-free multicast algorithm for the 2D mesh is to define a routing function that uses these two subnetworks in such a way as to avoid channel cycles. Let V be the node set of the 2D mesh. One routing function $R : V \times V \rightarrow V$, is defined as $R(u,v) = w$, such that w is the neighboring node of u, and

$\ell(w) = \max\{\ell(p) : \ell(p) \leq \ell(v)$ and p is a neighboring node of $u\}$, if $\ell(u) < \ell(v)$,

or

$\ell(w) = \min\{\ell(p) : \ell(p) \geq \ell(v)$ and p is a neighboring node of $u\}$, if $\ell(u) > \ell(v)$.

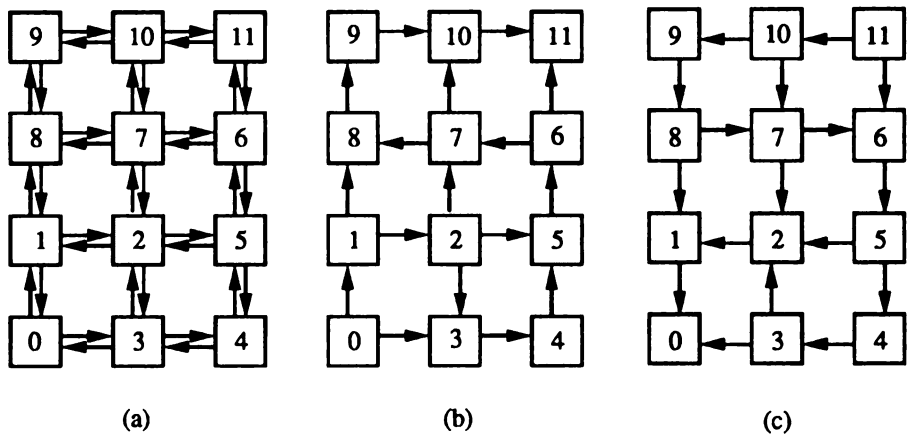


Figure 6.10. The other label assignment and channel partitioning for a 4×3 mesh.

For two arbitrary nodes r and t, a path from r to t can be selected by the routing function R as $(v_1, v_2, \ldots, v_k)$, where $v_1 = r, v_i = R(v_{i-1}, t) for 1 < i \leq k$, and $v_k = t$. The path is a partial order preserved for the assignment ℓ if $\ell(v_i) < \ell(v_{i+1})$ for $1 \leq i < k$ (that is, in the high-channel network), or $\ell(v_i) > \ell(v_{i+1})$ for $1 \leq i < k$ (that is, in the low-channel network). We have the following important lemmas.

Lemma 6.1 *For two arbitrary nodes u and v in a 2D mesh, the path selected by the routing function R is a shortest path from u to v, and it is a partial order preserved for the label assignment function ℓ.*

Proof: Since u and v are two arbitrary nodes in a 2D mesh, without loss of generality we can assume that $\ell(u) < \ell(v)$. We prove the lemma by induction on the distance $d(u, v)$.

Basis ($d(u, v)$=1): It is easy to check that $R(u, v) = v$ and $\ell(u) < \ell(v)$.

Induction: Assume the hypothesis is true for $d(u, v) = k$. We will prove that it is true for $d(u, v) = k + 1$. Suppose that $u = (x_u, y_u)$ and $v = (x_v, y_v)$, and $w = R(u, v)$.

Case $y_u = y_v$: By the definition of ℓ, $x_v > x_u$ if y_v is even, and $x_v < x_u$ if y_v is

odd. By the definition of routing function R, we have $w = (x_u + 1, y_u)$ for $x_v > x_u$, and $w = (x_u - 1, y_u)$ for $x_v < x_u$. Again, by the definition of ℓ, $\ell(u) < \ell(w) \leq \ell(v)$. Obviously, $d(w, v) = d(u, v) - 1 = k$ and $\ell(w) < \ell(v)$ since $w \neq v$.

Case $y_u < y_v$: By the definition of ℓ and R, when $y_v = y_u + 1$, if y_u is even and $x_v > x_u$ we have $w = (x_u + 1, y_u)$; if y_u is odd and $x_v < x_u$, then $w = (x_u - 1, y_u)$. Otherwise, $w = (x_u, y_u + 1)$. In all cases, we have $d(w, v) = d(u, v) - 1 = k$ and $\ell(u) < \ell(w) < \ell(v)$.

By the definition of ℓ for 2D mesh, it is impossible to have $y_u > y_v$, since $\ell(u) < \ell(v)$.

By the assumption of the induction, the path selected by routing function R for w and v, say $(w, u_1, \ldots, u_{k-1}, v)$, is a shortest path and a partial oreder preserved for ℓ with length k. Thus $(u, w, u_1, \ldots, u_{k-1}, v)$ is also such a path with length $k + 1$. The proof of this lemma is completed by the principle of induction. □

We are now ready to define three multicast routing heuristics that use the routing function R.

Dual-Path Multicast Routing

The first heuristic routing algorithm partitions the destination node set D into two subsets, D_H and D_L, where every node in D_H has a higher label than that of the source node u_0, and every node in D_L has lower label than that of u_0. Multicast messages from u_0 will be sent to the destination nodes in D_H using the high-channel network and to the destination nodes in D_L using the low-channel network.

The message preparation algorithm executed at the source node of the dual-path routing algorithm is given in Figure 6.11. The source node divides the destination node set into two subsets: D_H and D_L, which are then sorted in ascending order and descending order, respectively, with the label of each node used as its key for sorting.

The dual-path routing algorithm, shown in Figure 6.12, uses a distributed routing method in which the routing decision is made at each intermediate node. Thus the dual-path routing algorithm is executed at each intermediate node. Upon receiving the message, each node first determines whether its address is the first destination node in the message header. If so, the address is removed from the message header and the message is delivered to the host node. At this point, if the address field of the message header is not empty, the message is forwarded toward the first destination node in the message header using the routing function R.

Algorithm: Message preparation for the dual-path routing algorithm

Input: Destination set D, local address u_0, and node label assignment function ℓ;

Output: Two sorted lists of destination nodes: D_H and D_L placed in message header.

Procedure:

1. Divide D into two sets D_H and D_L such that D_H contains all the destination nodes with higher ℓ value than $\ell(u_0)$ and D_L the node with lower ℓ value than $\ell(u_0)$.
2. Sort the destination nodes in D_H and D_L using the ℓ value as the key, respectively. Put these sorted nodes in list D_H in ascending order and D_L in descending order, respectively.
3. Construct two messages, one containing D_H as part of the header and the other containing D_L as part of the header.

Figure 6.11. Message preparation for the dual-path routing algorithm.

Note that the label assignment function ℓ can be determined in advance for a given multicomputer topology. Thus, the label in Fig 6.11 and Fig. 6.12 for each node can be obtained in $O(1)$ time. Step 1 and Step 2 can be merged by using some well known

Algorithm: The dual-path routing algorithm
Input: A message with sorted destination list $D_M = \{d_1, \ldots, d_k\}$, a local address w and node label assignment ℓ;
Procedure:

1. If $w = d_1$, then $D'_M = D_M - \{d_1\}$ and a copy of the message is copied to the local node; otherwise, $D'_M = D_M$.
2. If $D'_M = \emptyset$, then terminate the message transmission.
3. Let d be the first node in D'_M, and let $w' = R(w, d)$.
4. The message is sent to node w' with address destination list D'_M in its header.

Figure 6.12. The dual-path routing algorithm.

sorting algorithm such as quick sort algorithm with time complexity $O(d \log d)$, where $d = |D|$. In the message routing part, both Step 1 and Step 2 take a constant time if we don't consider the time for message coping. Step 3 can be done in $O(1)$ time since the maximum outdegree for a node is 4 in 2D mesh. Thus we have established the following lemma and theorem.

Lemma 6.2 *The message preparation complexity for the dual-path routing algorithm is* $O(|D| \log |D|)$, *where* D *is the destination set.*

Theorem 6.1 *The dual-path routing algorithm in 2D mesh for a destination set* D *sends the source message to each destination node in* D *once and only once. The time complexity for each node is* $O(1)$.

Figure 6.13 shows an example of applying dual-path routing algorithm to a 6×6 mesh with (3,2) as the source node.

In order to show that the dual-path routing algorithm is deadlock-free, we need only prove that there can exist no cycle dependency among the channels, because

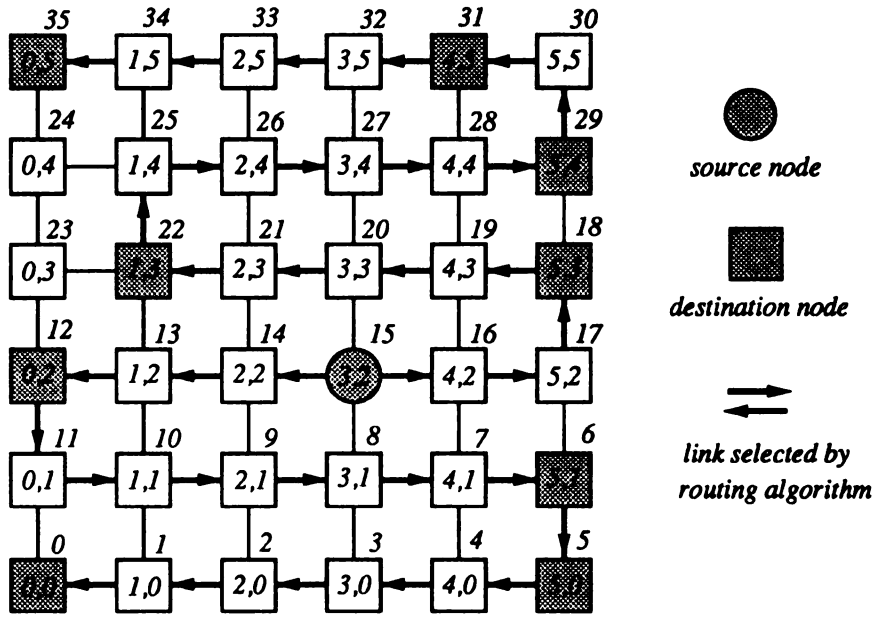


Figure 6.13. An example of dual-path routing in a 6 × 6 mesh.

the cycle dependency of the resource is a necessary condition for deadlock. By the definition of channel partitioning scheme, each of the high-channel and low-channel subnetworks comprises a separate sets of channels in the multicomputer. There are no cycles within each subnetwork, and hence no cyclic dependency can be created among the channels. Therefore, we have the following assertion.

Assertion 2 *The dual-path multicast routing algorithm for 2D mesh is deadlock-free.*

Multi-Path Multicast Routing

The performance of the dual-path routing algorithm is dependent on the location distribution of destination nodes. Consider the example shown in Figure 6.13 for a 6 × 6 mesh topology. The total number of channels used to deliver the message is 33 (18 in the high-channel network and 15 in the low-channel network). The maximum

distance from the source to a destination is 18 hops. In order to reduce the average length of multicast paths and the number of the channels used for a multicast, an alternative is to use a *multi*-path multicast routing algorithm, in which the restriction of having at most two paths is relaxed.

In a 2D mesh, most nodes have outgoing degree 4, so up to 4 paths can be used to deliver a message, depending on the location of the source node. The only difference between multi-path routing and dual-path routing concerns the message preparation algorithm at the source node. Figure 6.14 shows the message preparation of the multi-path routing algorithm, in which we further partition the destination sets D_H and D_L of the dual-path algorithm. The set, D_H is divided into two sets, one containing the nodes whose x-coordinates are greater than or equal to that of u_0 and the other containing the rest of the nodes in D_H. D_L is divided in a similar manner.

The rules by which ties are broken in partitioning the destination nodes depend on the location of the source node in the network and the particular labeling method used. For example, Figure 6.15a shows the partitioning of the destinations in the high-channel network when the source is the node labeled with 15. When the node labeled 8 is the source, the high-channel network is partitioned as shown in Figure 6.15b.

In the multi-path routing algorithm, Step 1 and Step 2 are exactly the same as those in dual-path algorithm. The two steps need $O(|D| \log |D|)$ time. In Step 3, we need to check at most two queues for each destination node in D_H, $|D_H| < D$, it takes $O(|D|)$ time. Step 4 is symmetric to Step 3, it also requires $O(|D|)$ time. The follow lemma is established.

Lemma 6.3 *The time complexity of message preparation of the multi-path routing algorithm is* $O(|D| \log |D|)$, *where* D *is the destination set.*

For the multicast example shown in Figure 6.13, the destination set is first divided into two sets D_H and D_L at source node (3,2), with D_H =

Algorithm: Message preparation for the multi-path routing in 2D mesh

Input: Destination set D, local address $u_0 = (x_0, y_0)$, and node assignment ℓ;

Output: Sorted destination node lists $D_{H1}, D_{H2}, D_{L1}, D_{L2}$ for 4 multicast paths.

Procedure:

1. Divide D into two sets D_H and D_L such that D_H contains all the destination nodes with higher ℓ value than $\ell(u_0)$ and D_L the nodes with lower ℓ value than $\ell(u_0)$.
2. Sort the destination nodes in D_H and D_L using the ℓ value as the key. Place these sorted nodes in list D_H in ascending order and D_L in descending order, respectively.
3. (Assume that $v_1 = (x_1, y_1)$ and $v_2 = (x_2, y_2)$ are the two neighboring nodes to u_0 with higher labels than that of u_0.)
   Divide D_H into two sets, D_{H1} and D_{H2} as follows:
   $D_{H1} = \{(x,y) | x \leq x_1 \text{ if } x_1 < x_2,\ x \geq x_1 \text{ if } x_1 > x_2\}$ and
   $D_{H2} = \{(x,y) | x \leq x_2 \text{ if } x_2 < x_1,\ x \geq x_1 \text{ if } x_2 > x_1\}$.
   Construct two messages, one containing D_{H1} as part of the header and sent the message to v_1, and the other containing D_{H1} as part of the header and sent the message to v_2.
4. Similarly, divide D_L into D_{L1} and D_{L2}.

Figure 6.14. Message preparation for the multi-path routing algorithm in 2D mesh.

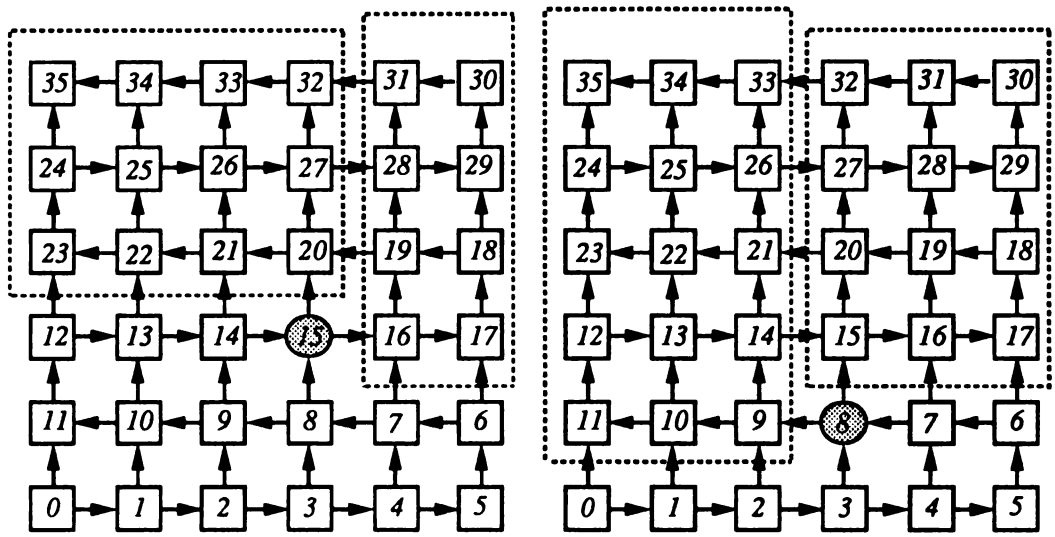


Figure 6.15. Multi-path destination address partitioning.

$\{(5,3),(1,3),(5,4),(4,5),(0,5)\}$ and $D_L = \{(0,2),(5,1),(5,0),(0,0)\}$. D_H is further divided into two subsets D_{H1} and D_{H2} at Step 3, with $D_{H1} = \{(5,3),(5,4),(4,5)\}$ and $D_{H2} = \{(1,3),(0,5)\}$. D_L is also divided into $D_{L1} = \{(5,1),(5,0)\}$ and $D_{L2} = \{(0,2),(0,0)\}$. The multicast will be performed using four multicast paths, as shown in Figure 6.16. Note that multi-path routing requires only 20 channels in the example, and the maximum distance from the source to destination is 6 hops. Hence, this example shows that multi-path routing can offer significant advantage over dual-path routing in terms of generated traffic and the maximum distance between the source and destination nodes.

By the same argument as for Assertion 2, we have the following assertion.

Assertion 3 *The multi-path multicast routing algorithm for 2D mesh is deadlock-free.*

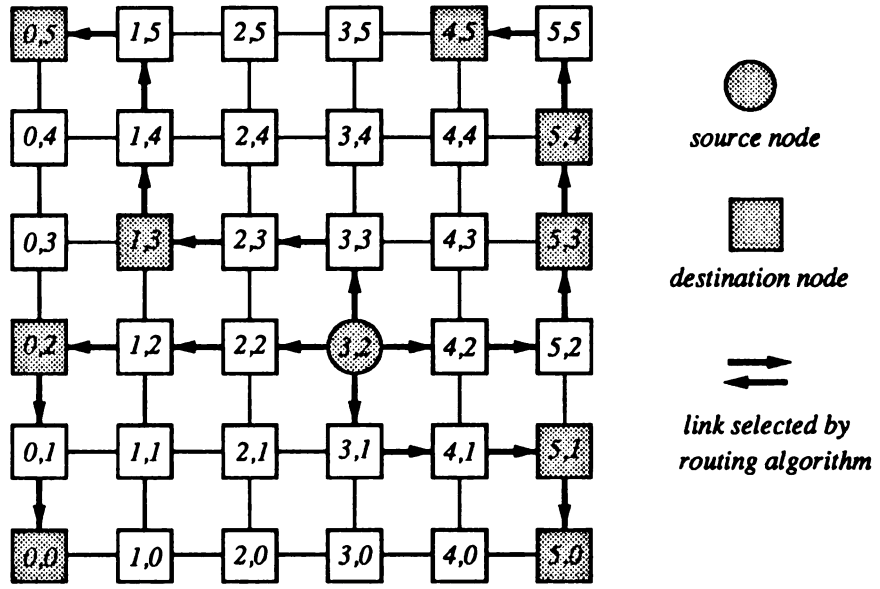


Figure 6.16. An example of multi-path routing in a 6 × 6 mesh.

Fixed-Path Multicast Routing

For purposes of comparison, we describe a third multicast algorithm called *fixed-path* routing. This routing scheme was suggested in [49]. Fixed-path routing is similar to dual path routing except that each path traverses all possible horizontal links before traversing a single vertical link. The upper path visits all nodes in increasing order until the last destination is reached. Similarly, the lower path visits all nodes in decreasing order until the last destination is reached. Figure 6.17 shows fixed path routing in a 6 × 6 mesh network for the source and destinations of the previous examples.

The total number of channels used to deliver the message is 35 (20 in the high-channel network and 15 in the low-channel network). The maximum distance from the source to a destination is 20 hops. Clearly, fixed-path routing is not as efficient as the other two approaches. However it is very simple to implement. We will discuss

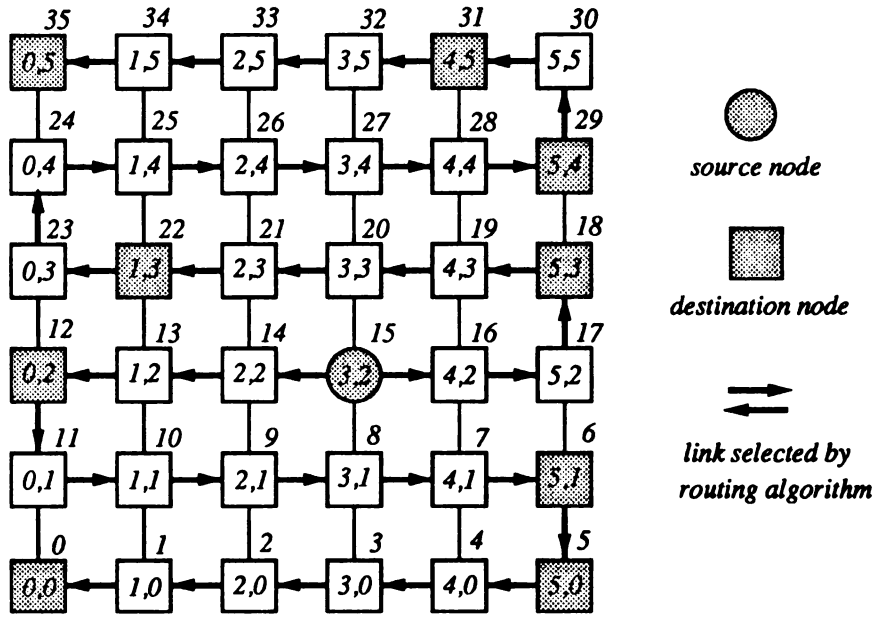


Figure 6.17. An example of fixed-path routing in a 6 × 6 mesh.

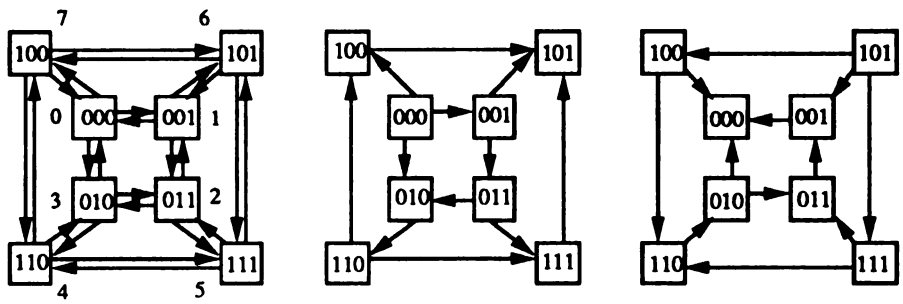
fixed-path routing further in the next chapter.

6.3 Deadlock-Free Multicast in Hypercube

The tree-like deadlock-free multicast routing algorithm using double channels is no longer suitable for hypercube topology since each node in n-cube has n neighbors. It is our conjecture that $O(n)$ channels between any two neighboring nodes are necessary to support tree-like deadlock-free multicast routing. As mentioned before, tree-like multicast routing pattern is not well suitable for wormhole networks. The remainder of this chapter will only consider the path-like routing schemes.

We also need to first partition network based on Hamiltonian paths. A hypercube has many Hamiltonian paths. Again, each node, say node i, in a multicomputer is assigned with a label, $\ell(i)$. The assignment of the label to a node is based on the order of that node in a Hamilton path, where the first node in the path is assigned

with label 0 and the last node in the path is assigned with label $m-1$ if there are m nodes in the network. Figure 6.18(a) shows a possible label assignment for 3-cube multicomputer. The corresponding high-channel and low-channel networks are also shown Fig. 6.18.



(a) physical network (b) high-channel network (c) low-channel network

Figure 6.18. The label assignment and the corresponding high-channel and low-channel networks for a 3-cube.

We also propose label assignment scheme hypercube topology and prove that the assignment scheme provides a shortest routing path for any given pair of source and destination nodes.

For an n-cube, the label assignment function ℓ for a node with address $d_{n-1}d_{n-2}\ldots d_0$ is

$$\ell(d_{n-1}d_{n-2}\ldots d_0) = \sum_{i=0}^{n-1} (c_i\overline{d_i}2^i + \overline{c_i}d_i2^i),$$

where $c_{n-1} = 0, c_{n-j} = d_{n-1} \oplus d_{n-2} \oplus \ldots \oplus d_{n-j+1}$ for $1 < j \leq n$. It is straightforward to check that for two arbitrary nodes u and v in n-cube, $\ell(u) \neq \ell(v)$ if $u \neq v$.

By defining the routing function as the same way as the one for 2D mesh, we have the following important lemma.

Lemma 6.4 *For two arbitrary nodes u and v in an n-cube, the path selected by the routing function R is a shortest path from u to v, and it is a partial order preserved for the label assignment function ℓ.*

Proof: Again, we assume that $\ell(u) < \ell(v)$, and prove by induction on the distance $d(u,v)$.

For $d(u,v)$=1, we have $R(u,v) = v$ and $\ell(u) < \ell(v)$. Assume that it is true for $k, (d(u,v) = k)$. Now, consider the case of $d(u,v) = k+1$. Suppose the j-th bit (from left to right) is the first bit that u and v differ.

1. Suppose that u is $a_{n-1}\ldots a_{n-j+1}0b_{n-j-1}\ldots b_0$ and v is $a_{n-1}\ldots a_{n-j+1}1c_{n-j-1}\ldots c_0$.

(a) Consider node $a_{n-1}\ldots a_{n-j+1}1b_{n-j-1}\ldots b_0$, if $\ell(a_{n-1}\ldots a_{n-j+1}1b_{n-j-1}\ldots b_0) < \ell(a_{n-1}\ldots a_{n-j+1}1c_{n-j-1}\ldots c_0)$,

then $R(u,v) = w$, where $w = a_{n-1}\ldots a_{n-j+1}1b_{n-j-1}\ldots b_0$ by the definition of R. We have $d(w,v) = d(u,v) - 1 = k$, and $\ell(u) < \ell(w) < \ell(v)$ by the definition of ℓ for n-cube. By the hypothesis of the induction, there is a shortest path selected by routing function R, say $(w, u_1, \ldots, u_{k-1}, v)$, which is a partial order preserved for ℓ with length k. Clearly, $(u, w, u_1, \ldots, u_{k-1}, v)$ is also such a path with length $k+1$.

(b) If $\ell(a_{n-1}\ldots a_{n-j+1}1b_{n-j-1}\ldots b_0) > \ell(a_{n-1}\ldots a_{n-j+1}1c_{n-j-1}\ldots c_0)$, assume that p is the first bit that $a_{n-1}\ldots a_{n-j+1}1b_{n-j-1}\ldots b_0$ and $a_{n-1}\ldots a_{n-j+1}1c_{n-j-1}\ldots c_0$ differ (from left to right), and $p > j$. Let $a_{n-1}\ldots a_{n-j+1}1b_{n-j-1}\ldots b_0$ be $a_{n-1}\ldots a_{n-j+1}1c_{n-j-1}\ldots c_{n-p+1}\overline{c_{n-p}}b_{n-p-1}\ldots b_0$. Consider node s,

$$s = a_{n-1}\ldots a_{n-j+1}0c_{n-j-1}\ldots c_{n-p+1}\overline{c_{n-p}}b_{n-p-1}\ldots b_0,$$

Since

$$\ell(a_{n-1}\ldots a_{n-j+1}1c_{n-j-1}\ldots c_{n-p+1}\overline{c_{n-p}}b_{n-p-1}\ldots b_0)$$
$$= \ell(a_{n-1}\ldots a_{n-j+1}1b_{n-j-1}\ldots b_0)$$

$> \ell(a_{n-1} \ldots a_{n-j+1} 1 c_{n-j-1} \ldots c_{n-p+1} \overline{c_{n-p}} b_{n-p-1} \ldots b_0)$, and

$$a_{n-1} \oplus \ldots \oplus a_{n-j+1} \oplus 0 \oplus c_{n-j-1} \oplus \ldots \oplus c_{n-p+1} \quad = \quad \overline{a_{n-1} \oplus \ldots \oplus a_{n-j+1} \oplus 1 \oplus c_{n-j-1} \oplus \ldots \oplus c_{n-p+1}},$$

by the definition of ℓ for n-cube, we have

$\ell(u) = \ell(a_{n-1} \ldots a_{n-j+1} 0 b_{n-j-1} \ldots b_0)$

$= \ell(a_{n-1} \ldots a_{n-j+1} 0 c_{n-j-1} \ldots c_{n-p+1} \overline{c_{n-p}} b_{n-p-1} \ldots b_0)$

$< \ell(a_{n-1} \ldots a_{n-j+1} 0 c_{n-j-1} \ldots c_{n-p+1} \overline{c_{n-p}} b_{n-p-1} \ldots b_0)$

$= \ell(s)$.

Note that $R(s, v) = v$, and $d(u, s) = d(u, v) - 1 = k$ and $\ell(u) < \ell(s) < \ell(v)$. Again, by the assumption of the induction, the path selected by R, say $(u, u_1, \ldots, u_{k-1}, s)$, is a shortest path and a partial order preserved for ℓ with length k. Hence, the path $(u, u_1, \ldots, u_{k-1}, s, v)$ selected by R is also such a path with length $k + 1$.

2. Suppose u is $a_{n-1} \ldots a_{n-j+1} 1 b_{n-j-1} \ldots b_0$ and v is $a_{n-1} \ldots a_{n-j+1} 0 c_{n-j-1} \ldots c_0$. It is symmetric to the proof of part 1. □

The labeling effectively divides the network into two subnetworks, the *high-channel subnetwork* and *low-channel network* similar to those for 2D mesh topology.

The basic idea of dual-path multicast routing algorithm is the same as the one for 2D mesh. Thus, algorithms in Fig. 6.11 and Fig. 6.12 can be used directly for hypercube topology.

By using the above label assignment, we have the following corollary.

Corollary 6.1 *The dual-path multicast routing algorithm for hypercube is deadlock-free.*

For example, consider a 4-cube with node 1100 as the source as shown in Fig. 6.19. The number outside a node in the figure is the assigned label of the node. Suppose

the destination nodes are 0100, 0011, 0111, 1000 and 1111. Clearly, D_L={0100, 0111, 0011} and D_H={1111, 1000}. For D_H, the source node will send the message first to node 1111, the first destination node in D_H through the high-channel network. Based on the label assignment scheme, node 1100 has three outgoing channels reaching nodes 1000, 1101, and 111, respectively. According to the routing function R, node 1101 will be selected to forward the message. By repeating the same procedure for each node receiving the message, we get the routing pattern shown in Fig. 6.19.

For n-cube, The time complexity of the message preparation is the same as the one for 2D mesh, i.e., $|D| \log |D|$. However, the routing algorithm in Fig. 6.11 needs to check at most $O(n)$ outgoing channels. We have the following lemma.

Theorem 6.2 *The dual-path routing algorithm of n-cube for a destination set D sends the source message to each destination node in D once and only once. The time complexity for each node is* $O(n)$.

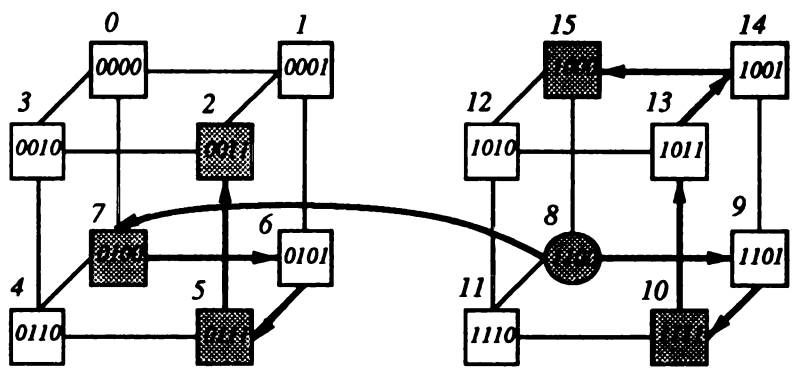


Figure 6.19. A 4-cube with node 1100 as the source for daul path routing.

In n-cube, each node has n neighboring nodes. Up to n multicast paths can be supported to deliver a message. Figure 6.20 depicts the message preparation of the

multi-path routing algorithm. The message routing part of the multi-path algorithm is the same as the one for dual-path routing.

Algorithm: Message preparation for the multi-path routing in hypercube

Input: Destination set D, local address u_0, and node assignment ℓ;

Output: Sorted destination node lists
$D_{H1}, D_{H2}, \ldots, D_{Hd}, D_{L1}, D_{L2}, \ldots, D_{Ld'}$
for at most $d + d'$ multicast paths.

Procedure:

1. Divide D into two sets D_H and D_L such that D_H contains all the destination nodes with higher ℓ value than $\ell(u_0)$ and D_L the nodes with lower ℓ value than $\ell(u_0)$.
2. Sort the destination nodes in D_H and D_L using the ℓ value as the key. Place these sorted nodes in list D_H in ascending order and D_L in descending order, respectively.
3. (Assume that $v_1, v_2, \ldots v_d$ are the d neighboring nodes to u_0 with higher labels than that of u_0, and $\ell(v_i) < \ell(v_{i+1}, 1 \leq i < d.)$
   Divide D_H into at most d sets, D_{Hi}, $1 \leq i \leq d$, as follows:
   $D_{Hi} = \{w | \ell(v_i) \leq \ell(w) < \ell(v_{i+1}\}$
   Construct at most d messages, each containing D_{Hi} as part of the header and sent the message to v_i for $1 \leq i \leq d$.
4. Similarly, divide D_L into $D_{L1}, D_{L2}, \ldots D_{Ld'}$.

Figure 6.20. Message preparation for the multi-path routing algorithm in hypercube.

Lemma 6.5 *The time complexity of message preparation of the multi-path routing algorithm for n-cube is* $\max\{O(|D| \log |D|), O(n|D|)\}$, *where D is the destination set.*

Proof: In the message preparation algorithm in Fig. 6.20, Step 1 and Step 2 are exactly the same as those in dual-path algorithm in Fig. 6.11. The two steps need

$O(|D|\log|D|)$ time. In Step 3, at most n queues for each destination node in D_H, $|D_H| < D$, need to be checked. It takes at most $O(n|D|)$ time. Step 4 is symmetric to Step 3, it also requires $O(n|D|)$ time. □

Figure 6.21 depicts an example of multi-path routing in a 4-cube multicomputers.

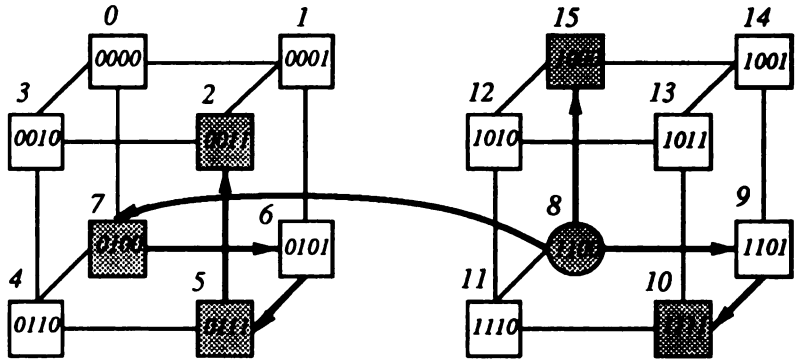


Figure 6.21. A 4-cube with node 1100 as the source for multi-path routing.

Corollary 6.2 *The multi-path multicast routing algorithm is deadlock-free for hypercube.*

Also, a fixed-path algorithm can be developed accordingly.

We present the performance study for these proposed algorithms in next chapter.

CHAPTER 7

PERFORMANCE STUDY OF THE ROUTING SCHEMES

In this chapter, we first present the simulation results of multicast routing algorithms for MC, MP and ST under static network condition. The average traffic of each algorithm is measured for different multicast sets. Then, we give the simulation results of the proposed deadlock-free multicast routing schemes. These results are measured not only by traffic but also by the network latency under dynamic network condition in which messages in network interact dynamically.

7.1 Performance Study under Static Network Traffic

We study the performance of the proposed heuristic multicast algorithms of MC, MP and ST for a 32×32 mesh and a 10-cube by simulation. A large number of randomly generated multicast sets with different number of destinations are tested to measure the traffic generated by the algorithms. The number of destination nodes, k, is chosen from 1 to 900. For a given k, a random number generator generates k integers within the range [0,1023], which represent k destination addresses. A generated integer i is

mapped to a 2D mesh node address (x, y), where $x = i \bmod 32$ and $y = \lfloor i/32 \rfloor$. In n-cube, the integer i is represented by the corresponding binary number. Each unit of traffic represents the transmission of one message over a link. The amount of traffic generated by the routing algorithm is averaged over 1000 runs for each k.

Figure 7.1 and Figure 7.2 show the amount of traffic generated by the MP algorithm in the 32×32 mesh and 10-cube topologies, respectively. Multicast can be performed by multiple one-to-one or broadcast communications. When using broadcast to implement multicast, the router sends the message to the local processor only when it is a destination node. The proposed algorithm always creates less traffic compared with multiple one-to-one and broadcast methods. For a 1-to-k multicast, it requires at least k units of the traffic [20]. The additional traffic is defined as the total amount of traffic minus k. For a broadcast with 1024 nodes, the traffic generated is always 1023. The additional traffic generated is $1023 - k$ for k destinations.

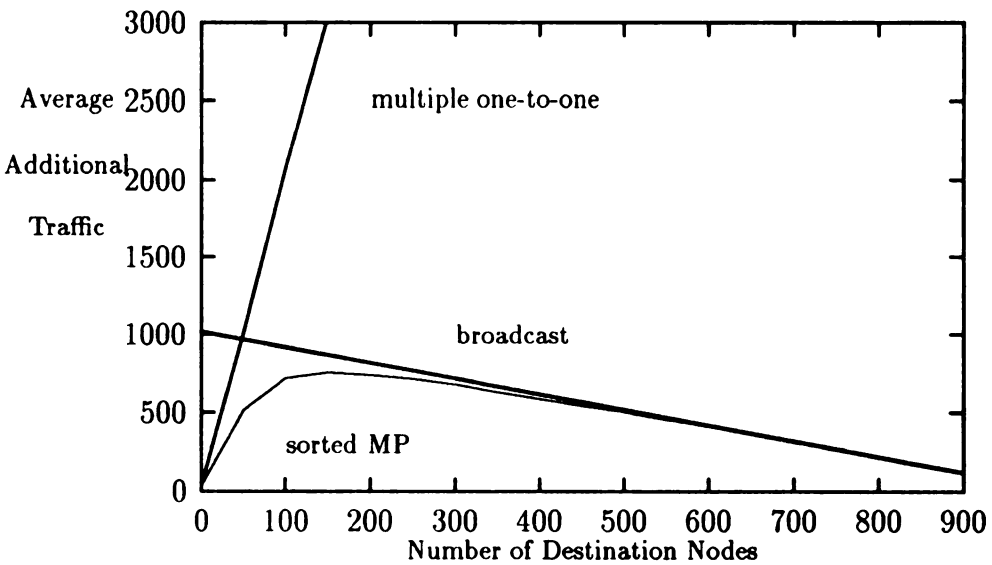


Figure 7.1. Performance of the sorted MP algorithm on a 32×32 mesh.

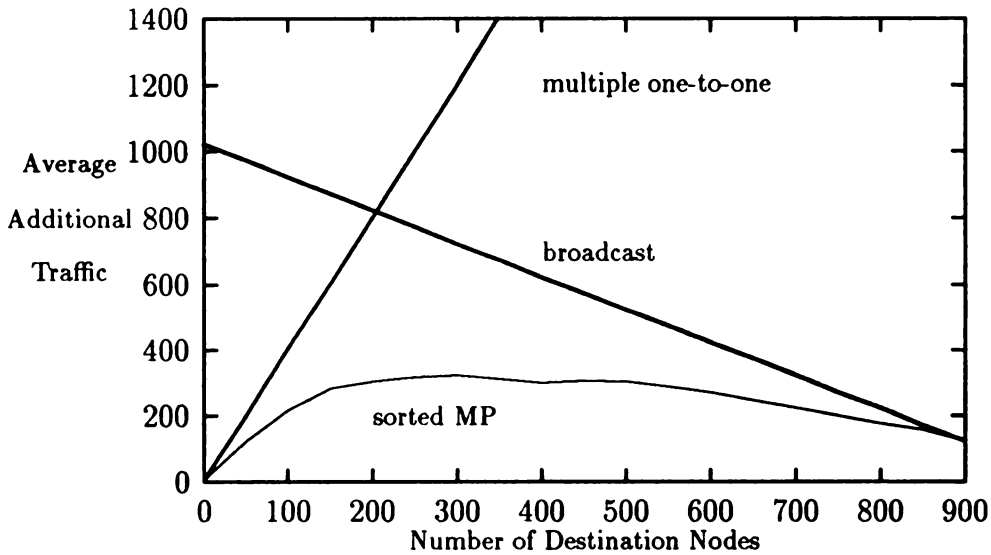


Figure 7.2. Performance of the sorted MP algorithm on a 10-cube.

Figure 7.3 and Figure 7.4 show the average additional amount of traffic obtained by simulating the greedy ST algorithm for 32×32 mesh and 10-cube, respectively. In Fig. 7.3, the performance of the greedy ST algorithm is compared with multiple one-to-one and broadcast communications. The performance of the greedy ST algorithm is much better compared with the other two approaches. For the n-cube graph, we compare our results with the LEN heuristic routing algorithm [20] which has demonstrated its superiority over multiple one-to-one and broadcast. The results of our routing algorithm show a significant improvement over the LEN algorithm in terms of the amount of traffic.

We also study the performance of MT algorithm by simulation on 16×16 mesh. Figure 7.5 plots the amount of traffic generated by each proposed MT algorithm for 16×16 mesh. It shows that the amount of traffic generated by X-first algorithm always creates much less amount of traffic compared with multiple one-to-one or broadcast

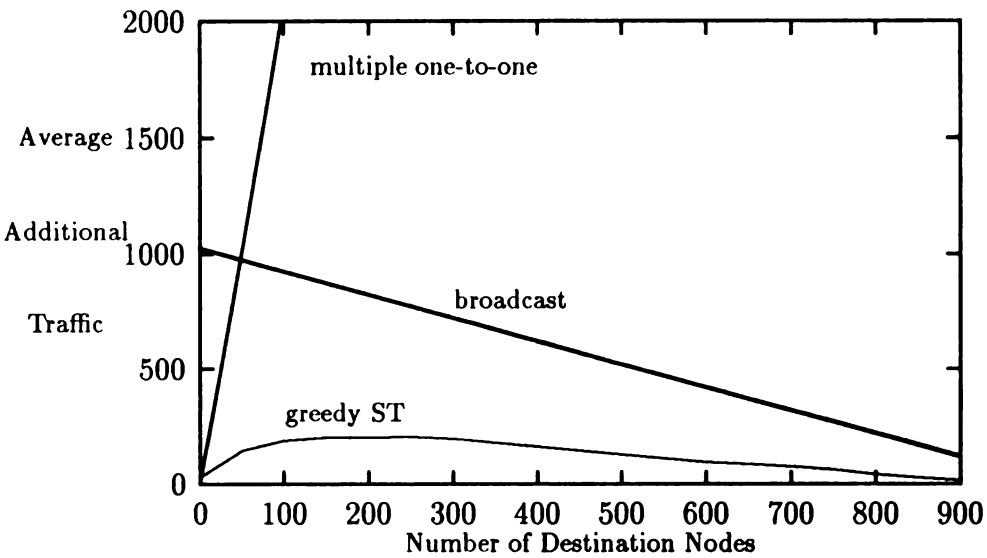


Figure 7.3. Performance of the greedy ST algorithm on a 32×32 mesh.

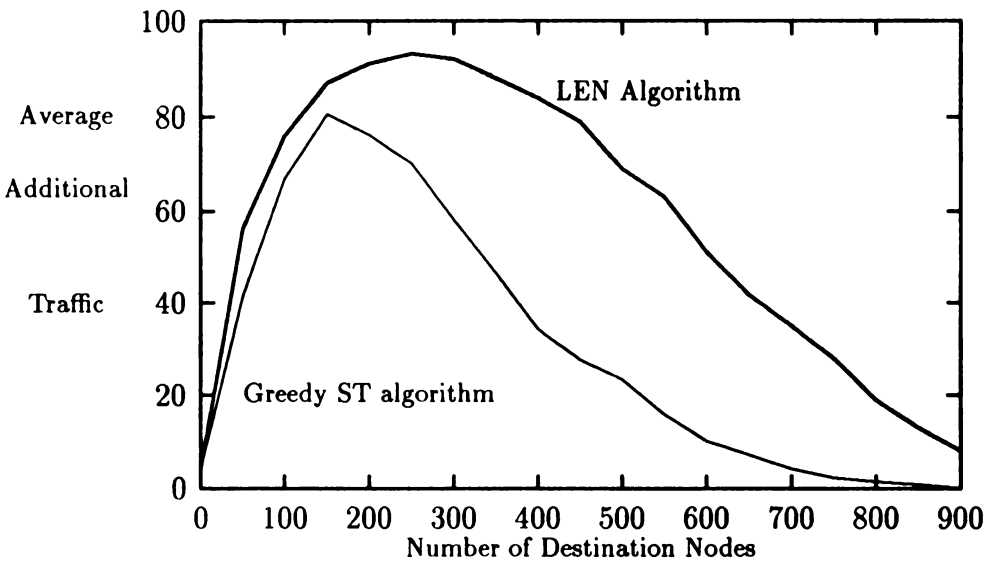


Figure 7.4. Performance of the greedy ST algorithm on a 10-cube.

methods. The traffic generated by divided greedy algorithm is always much less than that created by X-first algorithm.

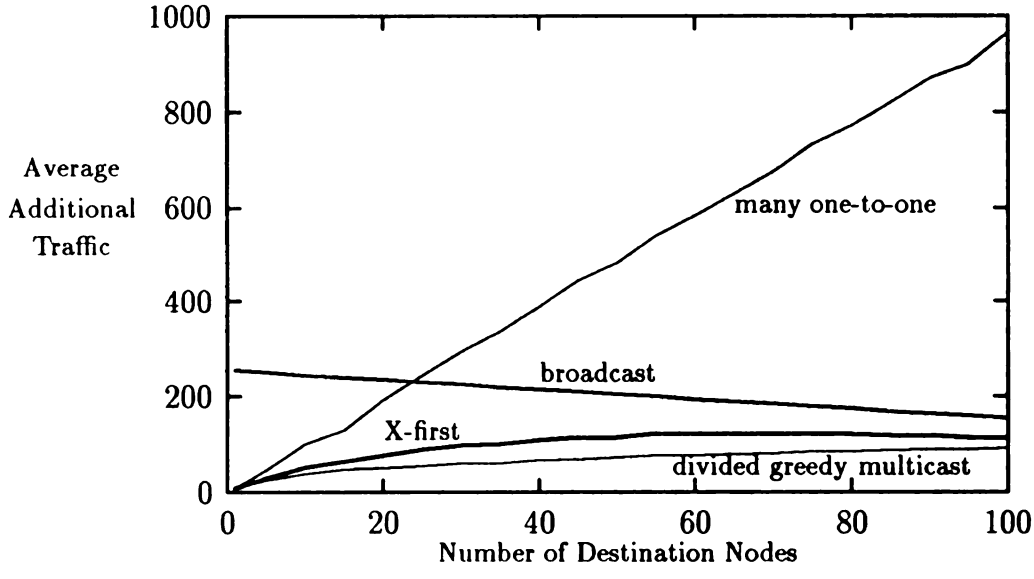


Figure 7.5. Performance of the X-first and divided greedy algorithms on a 16 × 16 mesh.

For deadlock-free multicast routing algorithms proposed in Chapter 6, Figure 7.6 and Figure 7.7 compare, for various numbers of destinations, the amount of "additional" traffic resulting from multi- and dual-path routing. This is a static measurement and does not depend on network traffic conditions, but gives an indication of the efficiency of the algorithm. The dynamic measurements for these routing algorithms will be given in next section.

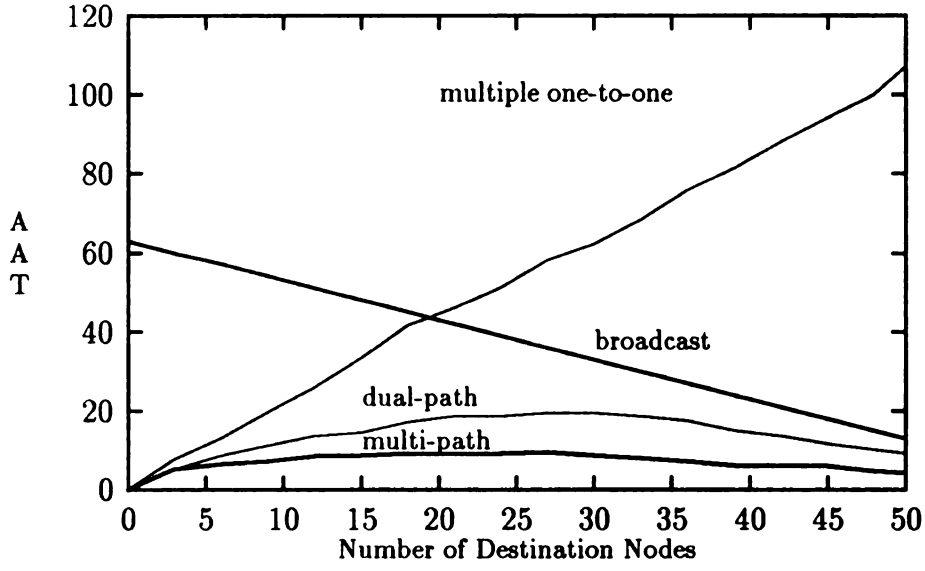


Figure 7.6. Performance of different multicast methods on a 6-cube.

7.2 Performance Study under Dynamic Network Traffic

The performance of a multicast routing algorithm not only is measured in terms of the delay and traffic resulting from single multicast message, but also depends on the interaction of the multicast message with other network traffic. In order to study these effects on the performance of the proposed multicast routing algorithms, we have written a simulation program to model the multicast communication in 8×8 mesh networks. In this section, we describe the program and results obtained by using it.

The simulation program used to model multicast communication in 2D mesh networks is written in C and uses an event-driven simulation package, CSIM [57]. CSIM allows multiple pseudo-processes to execute in a quasi-parallel fashion and provides a very convenient interface for writing modular simulation programs. Our simulation

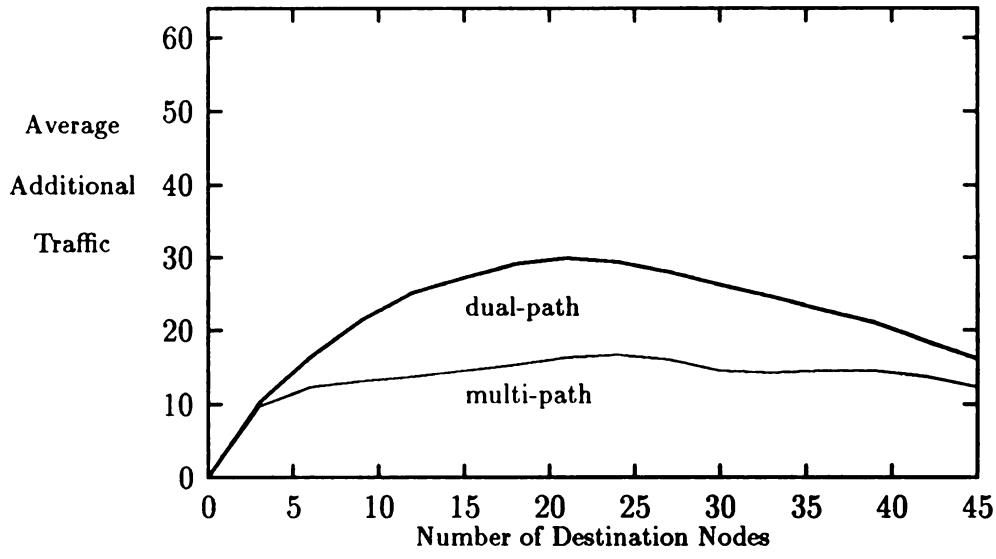


Figure 7.7. Performance of different number of destinations on 8 × 8 mesh.

program consists of several components, all of which run within the CSIM package. The main program activates 64 CSIM parallel processes, called *multicast generators*, one for each network node. Each multicast generator loops, creating *multicast* messages whose destinations are determined by a uniform random number generator. Each multicast messages is simulated with a pseudo-process that sends multicast messages to the destinations by creating *flit* pseudo-processes. Each flit pseudo-process models the transmission of a flit of the message. If there is a branch of the message at an intermediate node, the flit process will fork several flit processes, one for each new branch. A routing module for each routing algorithm is used by flit processes to determine the channels on which each message should be transmitted. Each channel has a single queue of message waiting for transmission. A statistics module gathers information concerning network traffic, time, for example, average network latency, using the method of *batch means* [58]. Although they are not shown in the figures, all simulations were executed until the confidence interval was smaller than 5 percent

of the mean, using 95 percent confidence intervals.

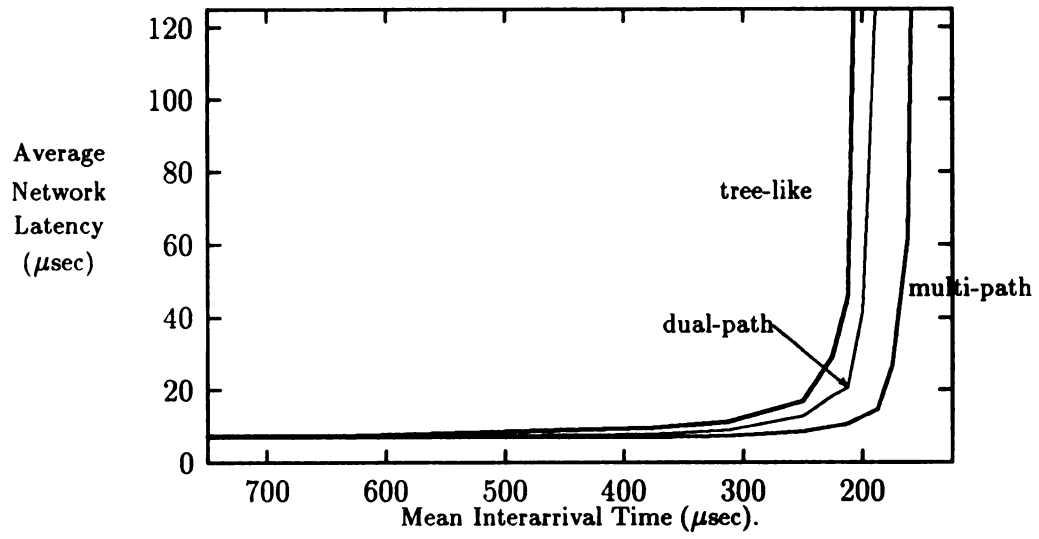


Figure 7.8. Performance under different loads on a double-channel mesh.

In order to compare the tree-like and path-like algorithms fairly, we first simulated each on a network that contained double channels. Figure 7.8 plots the average network latency for various network loads. The average number of destinations for a multicast is 10, and the message size is 128 bytes. The speed of each channel is 20 Mbytes/second. All three algorithms display good performance at low loads. As the load is increased, the path algorithms are not negatively affected as soon as the tree-like algorithm. This result is explained by the fact that in tree-like routing, when one branch is blocked, the entire tree is blocked. This type of dependency does not exist in path-like routing. Multi-path routing outperforms dual-path routing because, as we have shown earlier, paths tend to be shorter and less traffic is generated. Hence, the network will not saturate as quickly.

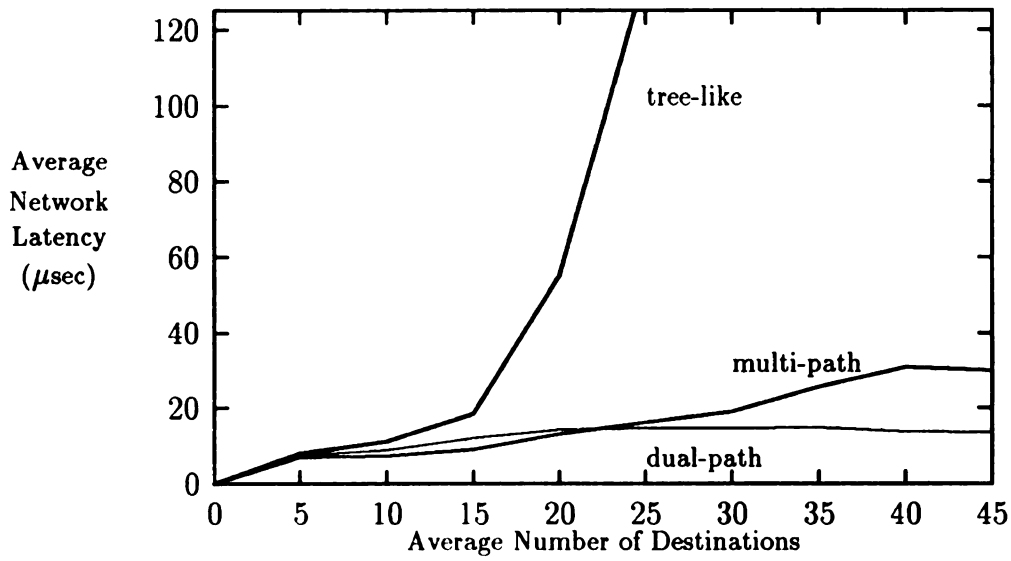


Figure 7.9. Performance of different number of destinations on a double-channel mesh.

The disadvantage of tree-like routing increases with the number of destinations. Figure 7.9 compares the three algorithms, again using double channels. The average number of destinations varies from 1 to 45. In this set of tests, every node generates a multicast messages with an average time between messages of 300 μsec. The other parameters are the same as for the previous figure. With larger sets of destinations, the dependencies among branches of the tree become more critical to performance and causes the delay to increase rapidly. The path algorithms still perform well, however. Notice that the dual-path algorithm outperforms the multipath algorithm for large destination sets. This result will be explained shortly. Our conclusion from Figures 7.8 and 7.9 is that tree-like routing is not particularly well suited for 2D mesh networks. First, it requires double channels in order to be deadlock-free. Second, its performance is worse than that for path-based schemes. The remainder of the simulation results concern only single link, path-based approaches.

As shown by example in Figures 6.13 and 6.16, multi-path routing usually requires fewer channels than dual-path routing. Because the destinations are divided into four sets rather than two, they are reached more efficiently from the source, which is approximately centrally located among the sets.

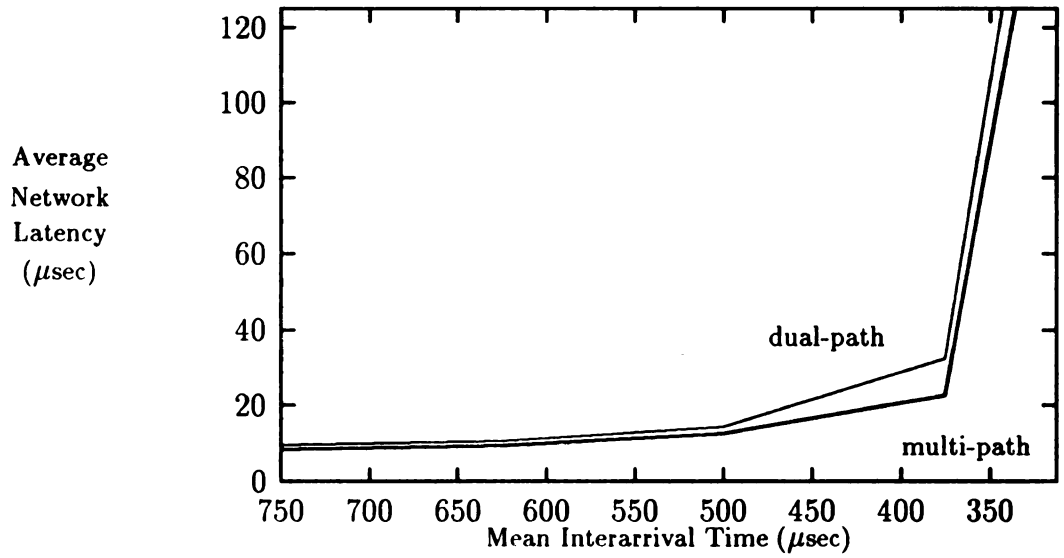


Figure 7.10. Performance under different loads.

Figure 7.10 plots the average network latency time for various network loads for multi-path and dual-path routing. Only single channels are used, the average number of destinations for a multicast is 10, and the message size is 128 bytes. The speed of each channel is 20M bytes/second. Both algorithms display good performance at low loads. As the load is increased, multi-path routing offers slight improvement over dual-path routing. This is likely due to the fact that multipath routing introduces less traffic to the network.

One may conclude from the results given thus far that multipath routing is su-

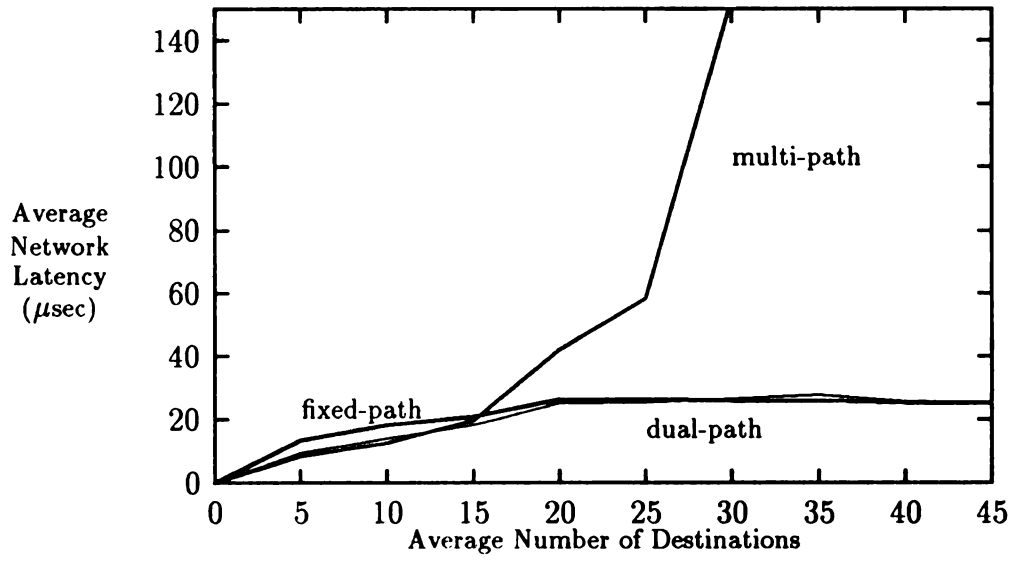


Figure 7.11. Performance of different number of destinations.

perior to dual-path routing. However, a major disadvantage of multipath routing is not revealed until both the load and number of destinations are relatively high. Figure 7.11 shows that under these conditions, dual-path routing performs much better than multi-path routing. The reason is somewhat subtle. When multi-path routing is used to reach a relatively large set of destinations, the source node will likely send on all of its outgoing channels. Until this multicast transmission is complete, any flit from another multicast or unicast message that routes through that source node will be blocked at that point. In essence, the source node becomes "hot spot." In fact, every node currently sending a multicast message is likely to be a hot spot. As the load increases, these hot spots will throttle system throughput and greatly increase message latency.

Hot spots are much less likely to occur in dual-path routing, and this fact accounts for its stable behavior under high loads with large destination sets. Although all of the outgoing channels at a node can be simultaneously busy, this can only result

from two or more messages routing through that node. Figure 7.11 also compares fixed-path routing to multi-path routing or dual-path routing. For a small number of destinations, fixed path routing traverses many unnecessary channels, creating more traffic and needlessly blocking more messages than multi- or dual-path routing. For a large enough number of destinations, however, dual- and fixed-path routing effectively become identical performance. Because fixed-path routing is much simpler than dual-path routing, it may be the best choice for messages with large destination sets.

CHAPTER 8

CONCLUSIONS

This chapter summarizes the major contributions of this dissertation research and outlines the direction of future research in this field.

8.1 Summary of Major Contributions

This dissertation research was motivated by the high demand for multicast communication in multicomputers.

We have studied various multicast routing evaluation criteria for multicomputers with different switching techniques. Based on a graph theoretical model, five optimization problems, namely the problems of finding OMP, OMC, MST, OMT, and OMS were identified.

This dissertation provided a theoretical foundation to the problems of OMP, OMC, MST, and OMS by showing that all these optimization problems are NP-complete for 2D mesh, n-cube, and 3D-mesh interconnection topologies. In the short time since its introduction in the early 1970's, NP-completeness has come to symbolize the abyss of inherent intractability that algorithm designers increasingly face as they seek to solve large and more complex problems. Indeed, discovering that a problem is NP-complete is usually just the beginning of work on that problem. The knowledge of

NP-completeness of a problem does provide invaluable information about how hard it would be. The work of proving the NP-complete problems was one of the main contributions of this dissertation research. Therefore, we could claim that it is extremely difficult to find optimal solutions for these problems for mesh or hypercube topologies in a reasonable amount of time. These results also justified our development of heuristic algorithms for these problems.

Designing efficient multicast protocols and routing algorithms naturally depends on the topology of the network, and also depends on the underlying switching mechanism used in the multicomputer. We have proposed new hybrid heuristic multicast routing algorithms of all the optimization problems for 2D mesh and hypercube multicomputers using various switching techniques. In general, they have much better performance than the current routing schemes for multicast, including multi-unicast, broadcast and some existing multicast algorithms.

In order to support reliable and efficient parallel computations in multicomputers, multicast routing schemes must be deadlock-free. We have focused on the deadlock issues of multicast communication using the most promising wormhole routing switching technique. We have shown that deadlock-free unicast routing algorithms, when extended to include multicast traffic, are no longer deadlock-free. A tree-based algorithm using X-first routing can be made to be deadlock-free if double channels are used in the network. It was observed that tree-like routing model is not suitable for multicast routing in wormhole networks. An alternative approach is to use a multicast star model. We presented three such algorithms based on star model that are deadlock-free without requiring additional channels. These algorithms were the first deadlock-free multicast algorithms using wormhole routing to be studied. These routing algorithms can be applied to any multicomputer networks that have Hamilton paths. The performance of the routing scheme is dependent on the selection of a good Hamilton path. For 2D mesh and hypercube topologies, we proposed methods

to select such a Hamilton path so that for any given two nodes, there exists a shortest routing path between them.

A simulation study has been conducted that compares the performance of these multicast algorithms under both static and dynamic network traffic conditions. Among the deadlock-free multicast routing algorithms, the dual-path routing algorithm offered the best overall performance over different traffic loads and destination set sizes. The major disadvantage of multi-path routing is that hot spots may occur under certain conditions, significantly degrading communication performance. A simpler fixed-path routing algorithm offers performance equal to the dual-path algorithm for large destination sets.

In summary, this dissertation research involved the study of routing criteria, modeling, investigation of optimal routing problems, routing algorithm development, deadlock issues, and performance evaluation of routing schemes.

8.2 Direction for Future Research

The following issues need to be addressed in future in order to better support multicast communication in multicomputers.

System Supported Multicast Service

System supported multicast service is able to offer a large number of applications improved performance and simplified programming. The dissertation research focused on multicast routing for hardware support. In order to provide multicast service for users, we must define a set of multicast primitive operations and develop the interface between application programs and system software, so that the underlying multicast facility can be easily used by the users. Also, a set of protocols must be developed to implement the multicast primitives using the functionality of the multicast facility.

In most existing multicomputers, multicast communication is not directly supported, the protocols have to map multicast primitives to many unicast messages.

Hardware Implementation – Multicast Router

In first generation multicomputers, communication functions were implemented by software. By hardware implementation, a dedicated hardware router is attached to each node. In order to fully utilize the resources of multicomputer, it is necessary to use such dedicated hardware routers so that the computation and communication can be overlapped, and the network latency be minimized. Some second generation multicomputers, such as Intel iPSC/2 uses dedicated hardware routers. In [29], a hardware router for unicast has been reported to support the virtual cut-through packet switching method. A hardware router for multicast communication is presented in [21].

In this dissertation, several deadlock-free multicast routing algorithms have been proposed to support multicast communication in wormhole networks. Hardware implementation for these routing algorithms is required to match the speed of the processors. If multicast is directly supported in hardware, the multicast protocols may be quite simple.

Adaptive Routing and Use of Virtual Channels

The routing algorithms proposed are deterministic. In order to increase the network throughput and to decrease the network latency, adaptive routing may be used. It can also support the fault tolerant routing. The main issue of adaptive routing is to avoid deadlock. Although some adaptive unicast routing schemes are proposed [36] [37], they are not directly applicable to the case of multicast communication.

Some adaptive unicast routing schemes use virtual networks [36]. A *virtual channel*

is a logic channel with its own message buffers, control, and data path. Several virtual channels may share a physical channel. The use of virtual channel increases the connectivity of the network and may easily support some adaptive routing schemes. Instead of partitioning the network into high-channel and low-channel networks as in the dissertation, the network may be partitioned into many sub-networks. The set of destination nodes then may be distributed to different sub-networks to support multiple multicast paths. The issue will be how many virtual channels are required and how the destination nodes should be partitioned. This is an interesting issue and deserves further study.

More Performance Study for Routing Schemes

We have conducted a simulation under both static and dynamic network traffic to study the performance of the proposed multicast routing schemes. The simulation was under the assumption that the distribution of the source node and destination nodes is uniform, and interval time to send a multicast message is uniformly random. Some benchmarks are necessary to run the simulation in order to get more convencing results. Another issue is to study the interaction between unicast and multicast traffic and how different multicast algorithms affect the performance of unicast wormhole routing.

BIBLIOGRAPHY

BIBLIOGRAPHY

[1] R. M. Karp and V. Ramachandran, "A survey of parallel algorithms for shared-memory machines," Tech. Rep. UCB/CSD 88/408, UCB/CSD, Mar. 1988.

[2] K. Hwang and F. A. Briggs, *Computer Architecture and Parallel Processing.* McGraw-Hill Book Company, 1984.

[3] W. C. Athas and C. L. Seitz, "Multicomputers: Message-passing concurrent computers," *IEEE Computer*, pp. 9–25, Aug. 1988.

[4] D. A. Reed and R. M. Fujimoto, *Multicomputer Networks: Message-Based Parallel Processing.* The MIT Press, 1987.

[5] J. L. Gustafson, S. Hawkinson, and K. Scott, "The architecture of a homogeneous vector supercomputer," in *Proceedings of the 1986 International Conference on Parallel Processing*, pp. 649–652, 1986.

[6] nCUBE Company, *NCUBE 6400 Processor Manual*, 1990.

[7] C.-T. Ho and S. L. Johnsson, "Distributed routing algorithms for broadcasting and personalized communication in hypercubes," in *Proceedings of the 1986 International Conference on Parallel Processing*, pp. 640–648, Aug. 1986.

[8] C. L. Seitz, J. Seizovic, and W.-K. Su, "The C programmer's abbreviated guide to multicomputer programming," Tech. Rep. Caltech-CS-TR-88-1, Department of Computer Science, California Institute of Technology, Jan. 1988.

[9] H.-W. Hsu and X. Lin, "Parallel algorithms for labeling image components," *Algorithms, International Symposium SIGAL'90,* Tokyo, Japan. *Lecture Notes in Computer Science 450,* Springer-Verlag, Berlin, New York, 1990.

[10] W.-J. Hsu, L. Wu, and X. Lin, "Optimal algorithms for labeling image components," in *Proceedings of 1990 International Conference on Parallel Processing,* August 1990.

[11] Z. Zeng, X. Lin, and T.-Y. Li, "An efficient parallel homotopy algorithm for unsymmetric eigenproblems with O(n) running time," in *Proceedings of Fifth SIAM Conference on Parallel Processing for Scientific Computing* (to be published in 1992), March 1991.

[12] M. Loper, "Distributed interactive simulation (DIS) requirement for multicast," Aug. 1991. Draft presented at Multipeer/Multicast Workshop in Orlando, Florida.

[13] R. F. DeMara and D. I. Moldovan, "Performance indices for parallel marker-propagation," in *Proceedings of the 1991 International Conference on Parallel Processing*, vol. I, (St. Charles, IL), pp. 658–659, Aug. 1991.

[14] S. G. Akl, *The Design and Analysis of Parallel Algorithms.* Englewood Cliffs, New Jersey: Prentice-Hall, 1989.

[15] V. Kumar and V. Singh, "Scalability of parallel algorithms for the all-pairs shortest-path problem," *Journal of Parallel and Distributed Computing*, pp. 124–138, Oct. 1991.

[16] X.-H. Sun, L. M. Ni, F. A. Salam, and S. Guo, "Compute-exchange computation for solving power flow problems: The model and application," in *Proc. of the Fourth SIAM Conference on Parallel Processing for Scientific Computing* (J. Dongarra, P. Messina, D. C. Sorensen, and R. G. Voigt, eds.), pp. 198–203, Dec. 1989.

[17] H. Xu, P. K. McKinley, and L. M. Ni, "Efficient implementation of barrier synchronization in wormhole-routed hypercube multicomputers," Tech. Rep. MSU-CPS-ACS-47, Department of Computer Science, Michigan State University, East Lansing, MI, October 1991.

[18] C. Moler and D. Scott, "Communication utilities for the iPSC," iPSC Technical Report No. 2, Intel Scientific Computers, 1986.

[19] Y. Lan, A. H. Esfahanian, and L. M. Ni, "Distributed multi-destination routing in hypercube multiprocessors," in *Proceedings of the Third Conference on Hypercube Computers and Concurrent Applications*, pp. 631–639, Jan. 1988.

[20] Y. Lan, A. H. Esfahanian, and L. M. Ni, "Multicast in hypercube multiprocessors," *Journal of Parallel and Distributed Computing*, pp. 30–41, Jan. 1990.

[21] Y. Lan, L. M. Ni, and A. H. Esfahanian, "A VLSI router design for hypercube multiprocessors," *Integration: The VLSI Journal*, vol. 7, pp. 103–125, 1989.

[22] Y. Saad and M. H. Schultz, "Topological properties of hypercubes," *IEEE Transactions on Computers*, vol. C-37, pp. 867–872, July 1988.

[23] R. Arlauskas, "iPSC/2 System: A second generation hypercube," in *Proceedings of the Third Conference on Hypercube Computers and Concurrent Applications*, (Pasadena, CA), pp. 38–42, Association for Computing Machinery, Jan. 1988.

[24] J. P. Hayes, T. Mudge, Q. F. Stout, S. Colley, and J. Palmer, "A microprocessor-based hypercube supercomputer," *IEEE Micro*, vol. 6, pp. 6–17, Oct. 1986.

[25] W. J. Dally, "Performance analysis of k-ary n-cube interconnection networks," *IEEE Transactions on Computers*, vol. C-39, pp. 775-785, 1990.

[26] C. L. Seitz, W. C. Athas, C. M. Flaig, A. J. Martin, J. Seizovic, C. S. Steele, and W.-K. Su, "The architecture and programming of the Ametek Series 2010 multicomputer," in *Proceedings of the Third Conference on Hypercube Concurrent Computers and Applications, Volume I*, (Pasadena, CA), pp. 33–36, Association for Computing Machinery, Jan. 1988.

[27] L. M. Ni, "Communication issues in multicomputers," in *Proceedings of the Fifth Workshop on Parallel processing*, (Taiwan), Dec. 1990.

[28] P. Kermani and L. Kleinrock, "Virtual cut-through: A new computer communication switching technique," *Computer Networks*, vol. 3, no. 4, pp. 267–286, 1979.

[29] W. J. Dally and C. L. Seitz, "The torus routing chip," *Journal of Distributed Computing*, vol. 1, no. 3, pp. 187–196, 1986.

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

[31] L. M. Ni abd P. K. Mckinley, "A Survey of Routing Techniques in Wormhole Networks," Tech. Rep. CPSACS 46, Michigan State University, Oct. 1991.

[32] D. C. Grunwald and D. A. Reed, "Networks for parallel processors: Measurements and prognostications," in *Proceedings of the Third Conference on Hypercube Concurrent Computers and Applications, Volume I*, (Pasadena, CA), pp. 610–619, Association for Computing Machinery, Jan. 1988.

[33] X. Lin and L. M. Ni, "Multicast communication in multicomputers networks," in *Proceedings of the 1990 International Conference on Parallel Processing*, pp. III–114–III–118, Aug. 1990.

[34] H. Sullivan and T. R. Brashkow, "A large scale homogeneous machine," in *Proceedings of the 4th Annu. Symp. Comput. Architecture*, pp. 105–124, 1977.

[35] C. Lang, "The Extension of object-oriented languages to a homogeneous concurrent architecture," Tech. Rep. 5014:TR:82, California Institute of Technology, Department of Computer Science, 1982.

[36] D. H. Linder and J. C. Harden, "An adaptive and fault tolerant wormhole rputing strategy for k-ary n-cubes" *IEEE Transactions on Computers*, pp. 2–12, Jan. 1991.

[37] C. Glass and L. M. Ni, "Adaptive wormhole routing in multicomputer networks," Tech. Rep. CPSACS 35, Michigan State University, Oct. 1991.

[38] K. D. Gunther, "Prevention of deadlocks in packet-switched data transport systems," *IEEE Transactions on Communications*, pp. 512–524, Apr. 1981.

[39] P. M. Merlin and P. J. Schweitzer, "Deadlock avoidance in store-and-forward networks - I: Store-and-forward deadlock," *IEEE Transaction on Communication*, pp. 345–354, 1980.

[40] S. Toueg and J. D. Ulman, "Deadlock-free packet switching networks," in *Proc. of the 11th ACM symposium on Theory Computing*, pp. 89–98, 1979.

[41] S. Toueg, "Deadlock- and livelock-free packet switching networks," in *Proc. of the 12th ACM symposium on Theory Computing*, pp. 94–99, 1980.

[42] D. Gelernter, "A DAG-based algorithm for preventing of store-and-forward deadlock in packet networks," *IEEE Transactions on Computers*, pp. 709–715, Oct. 1981.

[43] S. F. Nugent, "The iPSC-2 direct-connect communications technology," in *Proceedings of the Third Conference on Hypercube Computers and Concurrent Applications*, pp. 51–60, Jan. 1988.

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

[45] F. Harary, *Graph Theory.* Readings, Massachusetts: Addson-Wesley, 1972.

[46] G. T. Byrd, N. P. Saraiya, and B. A. Delagi, "Multicast communication in multiprocessor systems," in *Proceedings of the 1989 International Conference on Parallel Processing*, pp. I-196 - I-200, 1989.

[47] M. R. Garey and D. S. Johnson, *Computer and Intractability, A Guide to the Theory of NP-completeness.* Freeman, 1979.

[48] X. Lin and L. M. Ni, "Deadlock-free multicast wormhole routing in multicomputer networks," in *Proceedings of 18th Annual International Symposium on Computer Architecture*, pp. 115-125, May 1991.

[49] X. Lin, P. K. Mckinley, and L. M. Ni, "Performance study of multicast wormhole routing in 2D-mesh multicomputers," in *Proceedings of the 1991 International Conference on Parallel Processing*, pp. I-345 - I-343, 1991.

[50] H. A. Choi and A. H. Esfahanian, "On complexity of a message-routing strategy for multicomputer systems," in *Proc. of the 16th International Workshop on Graph-Theoretic Concepts in Computer Science*, (West Berlin, West Germany), June 1990.

[51] A. Itai, C. H. Papadimitriou, and J. L. Szwarcfiter, "Hamilton path in grid graphs," *SIAM J. Computing*, vol. 11, pp. 676 - 686, Nov. 1982.

[52] M. R. Garey and D. S. Johnson, "The rectilinear Steiner tree problem is NP-complete," *SIAM J. Appl. Math 32*, pp. 826–834, 1977.

[53] R. L. Graham and L. R. Foulds, "Unlikelihood that minimal phylogenies for realistic biological study can be constructed in reasonable computational time," *Mathematical Biosciences*, vol. 60, pp. 133–142, 1982.

[54] D. S. Johnson and C. H. Papadimitrion, "Computational complexity," in *The Traveling Salesman Problem* (E. L. Lawler, J. K. Lenstra, A. H. G. R. Kan, and D. B. Shmoys, eds.), John Willery & Sons, 1984.

[55] G. M. L. Kou and L. Berman, "A fast algorithm for steiner trees," Tech. Rep. Research Report RC 7390, IBM, Nov. 1978.

[56] D. W. Wall, *Mechanisms for Broadcast and Selective Broadcast.* PhD thesis, Stanford University, June 1980.

[57] H. D. Schwetman, "CSIM: A C-based, process-oriented simulation language," tech. rep., 1985.

[58] A. M. Law and W. D. Kelton, *Simulation Modeling and Analysis.* New York: McGraw-Hill, 1982.