This is to certify that the

dissertation entitled

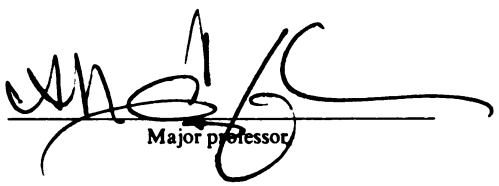
Toward a General Theory of Unicast-Based Multicast Communication

presented by

Barbara D. Birchler

has been accepted towards fulfillment
of the requirements for

Doctoral degree in Philosophy



Major professor

Date 8/21/97

TOWARD A GENERAL THEORY OF UNICAST-BASED MULTICAST COMMUNICATION

By

Barbara D. Birchler

A DISSERTATION

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

DOCTOR OF PHILOSOPHY

Department of Computer Science

1997

ABSTRACT

TOWARD A GENERAL THEORY OF UNICAST-BASED MULTICAST COMMUNICATION

By

Barbara D. Birchler

A common communication pattern that arises in many parallel and distributed computing applications is *multicast*, or one-to-many communication. In multicast, a single node in the system sends a message to multiple destinations in the system. Due to the complexity of implementing multicast in hardware, most machines offer only *unicast*, or one-to-one communication, in hardware. As a result, multicast must be implemented in software. A common software technique is Unicast-Based Multicast (UBM), in which multicast is accomplished by issuing multiple hardware unicast messages. In a UBM implementation, the number of nodes that can be informed can at most double in each time step, thus the lower bound on the time to complete multicast is $\lceil \log_2(d+1) \rceil$, where d is the number of multicast destinations. The goal of our research is to develop a general $\lceil \log_2(d+1) \rceil$ UBM implementation for realistic direct network systems.

Current direct network systems obey the *line-switching* model of communication and employ an *oblivious* routing scheme. In the line-switching model, the paths that are used to send messages simultaneously in the network must be edge-disjoint. The paths that are used to deliver messages are determined by the routing scheme of the network. In an oblivious routing scheme, each pair of

nodes in the network must use a specific prescribed path to deliver a message between them.

We introduce two realistic classes of oblivious routing schemes and develop line-switching UBM algorithms for arbitrary topologies that use these schemes. First, we address source limited inclusive routing schemes. We show that these routing schemes can be modeled by directed tree topologies. Within the context of directed trees, we show that a line-switching UBM algorithm can be closely approximated by a node-switching algorithm, in which paths used simultaneously must be node-disjoint as opposed to edge-disjoint. We then develop node-disjoint UBM algorithms for arbitrary directed tree topologies. The second class of routing schemes are shortest path oblivious routing schemes, in which the oblivious path between any pair of nodes must be a shortest path in the network. We develop a $\lceil \log_2(d+1) \rceil$ UBM implementation for arbitrary direct networks that employ a shortest path routing scheme. This result provides a theoretical basis that unifies the previous results on optimal multicast algorithms in specific direct network topologies. In addition, this result provides system designers with some simple, intuitive rules for creating routing schemes that guarantee multicast can be performed in minimum time.

To Edgar, who always says I can do it.

ACKNOWLEDGMENTS

This dissertation is the product of prayers, meaningful conversations, tears, and laughter, not only of my own, but of many others to whom I am forever in debt. The words that I write here cannot fully express my thanks, but I want you all to know that I truly appreciate the love, patience, and commitment that you have shown me these past three (or really five) years.

I want to thank the members of my committee. This, of course, would not have been possible without them. I thank Dr. Ni for giving insight into the "practical" and for his willingness to lend and ear and give advice whenever I asked. I will always aspire to give lectures as exciting as Dr. Palmer's "weird stuff" lectures. I pray that he never loses his excitement for graph theory. He appreciates mathematical beauty, and that is truly inspiring. I think Dr. Palmer appreciated my crazy graph cupcakes and attire more than anybody, and that really means a lot me even if it seems like a small thing. Of course most of the credit goes to my advisors. I thank Dr. Eric Torng for many insights and for lots of patience. From him I learned many valuable lessons about being a student, a researcher, and an advisor. Finally, if it were not for Dr. Esfahanian's confidence in my abilities, I would never have attempted to earn a PhD. He has been the best kind of mentor a student could hope for. He shows great excitement for his work (of course it's very easy to get excited about graph theory), he is a concerned teacher and advisor, and he has always shown me the utmost respect. Thank you.

I also owe many thanks to my family – my immediate family and my church family. I thank Christ Presbyterian Church in Pennsylvania for remembering me even though I live so far away, and I thank University Reformed Church for giving me a new home. A special thanks to the URC youth group, the Koinonia class, and the SIGAWOTS for their constant prayers and for helping me put everything into proper perspective. I thank my parents for their prayers and support, and for instilling in me that "protestant work ethic" that brought me here. I thank my brothers Rob (Four) and Pete (If-n-Six), the coolest numbers I have ever known, and my sister Deb for providing me with a get-away whenever I needed it and for her helpful insights about the many challenges life always seems to have to offer. I give a HUGE thanks to my nephew Ashwin and my niece Marissa who remind me of what is really important in life. They have never been anything but a joy to me.

Perhaps my most heartfelt thanks goes out to my friends who have been through it all along side me. I thank the "old" lunch group – Steve Turner, MP Jolly, Ron Sass and Edgar Kalns – who taught me how to be a grad student when I was totally clueless. I'm especially grateful to Ron – we have been through a LOT for a LONG time. I thank him for his willingness to listen and to share and to rescue me from Latex and all the other "computer stuff" that gives me (and never anyone else) problems. I also thank the "qualifier group" – Chuck Severance, Mark Brehob, Steve Wagner (icii), Natawut Nupairoj, Hugh Smith, and Jerry Gannod. We suffered together and we rejoiced together, but most importantly we LEARNED together!!

I thank my friend, icii. He is certainly the most innovative, creative, intelligent individual I know. We've been through thick and thin together. He was there when everyone else was and when nobody else was. To him I am forever grateful. Thanks for proof reading my papers (even the yucky notation and tedious proofs), for listening to me when I was a psych and letting me use your handkerchief, for working with me, for playing with me, for showing me your smiles and your frowns, and just for being an ic. "Friendship lasts forever."

I don't think there are words to express my gratitude to Edgar Kalns. He was my inspiration for this task. Edgar was a great example for me to follow. He always shows great confidence in my abilities, and he always knows how to motivate me and pick me up when I'm down. His love and support have been a constant source of strength and comfort to me. He also has a "plethora" of endearing qualities that are certainly too numerous to mention here. Edgars, es tev milu.

Finally, I have saved the best for last. My deepest thanks to Jerry. Thanks for your prayers, your advice, your jokes, your hugs, your patience, for reminding me that worry accomplishes nothing:

> "Who of you by worrying can add a single hour to his life? Since you cannot do this very little thing, why do you worry about the rest?" (Luke 12:25-26)

and most of all for your eternal commitment to see me though this and through every other challenge I will face. I love you, Jerry – not just that either – I'm crazy about you!!

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Chapter 1

Introduction

1.1 Motivation

The grand challenges in scientific computing, *e.g.*, pharmaceutical design, ocean modeling, and viscous fluid dynamics, can only be solved by computers that can provide Teraflops of computing power. Parallel computers are considered the most promising technology to solve such problems [30]. Because many (hundreds to thousands of processors) work together to solve problems, efficient communication among the nodes of a multicomputer is a vital component of system performance. One communication pattern that arises in parallel systems is *multicast*, or one-to-many communication. In multicast, one processor, called the *source*, sends a single message to multiple *destinations* in the system. *Unicast* and *Broadcast* are special cases of multicast. Unicast, also called one-to-one communication, involves a single source and a single destination. In broadcast, or one-to-all communication, the source sends a message to all of the other nodes in the network. Multicast communication is found in various applications such as parallel search algorithms [16], graph algorithms [34], and barrier synchronization [39]. Multicast is also used for cache-coherency protocols in distributed shared memory systems [36], and it has been defined in the MPI (Message

Passing Interface) standard [22] as an essential operation for implementing other collective communication operations such as *all-to-all gather* and *global reduction* in distributed memory machines. Because multicast communication is a component of many applications in parallel systems, efficient multicast implementation has been the subject of much study.

This dissertation addresses multicast communication paradigms for an important class of parallel systems, namely those that use a *direct network* interconnection structure [45]. Due to the complexity of hardware implementations of one-to-many communication, most existing direct network systems only support unicast communication in hardware. Thus, multicast must be provided in software. A common software technique is Unicast-Based Multicast (UBM), in which multicast is accomplished by issuing multiple hardware unicast messages. A UBM implementation consists of a number of time steps in which nodes that have previously received the multicast message send the message to uninformed nodes. Ideally, the UBM implementation should maximize the amount of parallel communication; that is, all the informed nodes should send messages during a single time step in order to reduce the total number of steps needed to inform all destinations. Thus, the lower bound on the time to complete multicast is $\lceil \lg(d+1) \rceil$ steps[1], where d is the number of multicast destinations. The objective of this research is to find a general $\lceil \lg(d+1) \rceil$ UBM implementation for realistic direct network systems.

1.2 Contributions

We study the problem of finding efficient *line-switching* UBM algorithms for *arbitrary* direct network systems that use an *oblivious* routing scheme. In the line-switching model of communication, all of the paths that are used to deliver messages simultaneously must be edge-disjoint. We focus on the line-switching communication model because it most closely fits popular direct network ar-

[1] We use the convention $\lg n = \log_2 n$.

chitectures; however, we will use other communication models as stepping stones toward efficient line-switching implementations. In addition, we address oblivious routing schemes, in which each pair of nodes in the network must use a specific prescribed path to deliver a message between them. The routing schemes in current systems, such as xy-routing in a mesh and e-cube routing in a hypercube, are examples of oblivious routing schemes. Finally, we address arbitrary topologies with two goals in mind. First, an algorithm for an arbitrary topology will be portable, *i.e.*, it can be used on any multicomputer topology. For example, a multicast algorithm developed for an arbitrary topology can be used on a Mesh, Hypercube, Torus, etc. The second reason we consider arbitrary topologies is that a general multicast algorithm gives practitioners insight about defining the best routing scheme for an arbitrary system. Routing schemes such as xy-routing and e-cube routing exist for particular multiprocessor systems; however, there is no general routing scheme for arbitrary topologies.

The main contributions of this dissertation are the introduction of two important classes of routing techniques for direct network systems, namely *Source Limited Inclusive Routing* and *Shortest Path Routing*, and the development of UBM algorithms for systems that employ these techniques.

1.2.1 Source Limited Inclusive Routing Schemes

Definitions and Properties

We define a realistic class of oblivious routing schemes called *source limited inclusive routing* schemes. This class of routing schemes makes two assumptions. First, it assumes that each node is only aware of its own oblivious routing paths. We call such a routing table a *source limited* routing table. Next, it assumes the system uses an *inclusive* routing scheme, a natural class of oblivious routing schemes used in most current multicomputers. We show that directed treès can be used to model source limited routing tables for an arbitrary topology that uses inclusive routing, and that a

directed tree model has a smaller storage requirement than a general routing table. Furthermore, as long as the maximum outdegree of the directed tree is not too large, this source limited routing table contains many (but not all) of the oblivious routing paths used by other nodes in the network.

Our first lower bound result shows that UBM in an arbitrary topology that supports oblivious routing is fundamentally different than UBM in an arbitrary topology that supports free routing. Specifically, we show that if the source limited routing table at a node is a complete, balanced, k-ary tree T with $|V(T)|$ nodes, a lower bound on the number of time steps necessary for performing a broadcast from this node is $\Omega(k \log_k |V(T)|)$, and we show this lower bound can be easily extended to apply to algorithms which have complete routing tables.

As a first step toward solving the general lower bound line-switching multicast problem, we show that the minimum length node-disjoint multicast schedule in any directed tree T is at most twice as long as the minimum length edge-disjoint multicast schedule for ditree T. We then show that we can reduce the problem of finding an optimal node-disjoint multicast schedule in a directed tree T to the problem of finding an optimal node-disjoint *broadcast* schedule in a related ditree T'.

Node-Disjoint UBM Algorithms for Ditrees

Because we can reduce multicast in a ditree to an equivalent broadcast problem, we focus on UBM algorithms for broadcast in directed tree topologies. We develop two polynomial time approximation algorithms, CA and SCA, for performing broadcast in directed trees under the node-switching model of communication. The biggest advantage of the CA algorithm is its simplicity. Its best case time complexity is $O(n \lg n)$ (which it achieves for broadcast in hypercubes, meshes and tori), and its worst case complexity is $O(n^2)$. The second approximation algorithm (SCA) has slightly higher complexity ($O(n^2)$ in the best case and $O(n^3)$ in the worst case), but it may also produce much shorter calling schedules than the CA algorithm. The broadcast schedules produced by SCA are

guaranteed to be no more than twice the length of an optimal length broadcast schedule.

We also describe a $O(n^3)$ algorithm that always produces optimal length node-disjoint broadcast schedules for directed tree topologies. This can easily be transformed to an edge-disjoint broadcast schedule with length at most twice that of an optimal edge-disjoint schedule. Because ditrees represent a class of routing schemes (source limited inclusive routing schemes), our algorithms can be applied to any topologies in that class (*e.g.*, hypercubes, meshes, and tori) as well as other arbitrary topologies.

1.2.2 Shortest Path Routing Schemes

The Generalized Path-Matching Problem

We also introduce a generalization of previously studied matching problems for undirected graphs. We define the Generalized Path-Matching (GP-matching) problem, which includes both the edge-matching problem and the path-matching problem. In a GP-matching, a subset X of vertices is matched using paths from a specified set P. We are interested in finding a perfect GP-matching, or one that matches all the vertices of X. Additionally, we focus on the edge-disjoint GP-matching problem (or GDP-matching problem). Sufficient conditions for the existence of a perfect GDP-matching are given. It is shown that if shortest paths are used to match vertices, a perfect GDP-matching always exists.

A UBM Algorithm for Shortest Path Routing Schemes

We show that the GDP-matching problem can be used to create an optimal UBM algorithm for an arbitrary direct network that uses a shortest path routing scheme. Because meshes, hypercubes and tori typically use shortest path routing schemes, this result implies that optimal UBM can be done in these direct network topologies. Thus, this result provides a theoretical basis that unifies all

previous results on optimal multicast algorithms in specific direct network topologies [21, 46, 47]. In addition, this result provides system designers with some simple, intuitive rules for creating routing schemes that guarantee multicast can be performed in minimum time, and it gives insight about updating routing schemes when adding on to a network incrementally.

1.3 Organization of the Dissertation

Chapter 2 describes the background on multicast communication in direct network systems. Chapter 3 discusses related work in multicast communication and matching problems in graphs. Chapter 4 describes source limited inclusive routing and several of its properties. Chapter 5 gives two approximation algorithms and one optimal algorithm for multicast in ditrees under the node-switching model of communication. Chapter 6 describes the generalized path-matching problem and its application to multicast in shortest path oblivious routing systems. Chapter 7 concludes the dissertation and discusses several areas for future investigation.

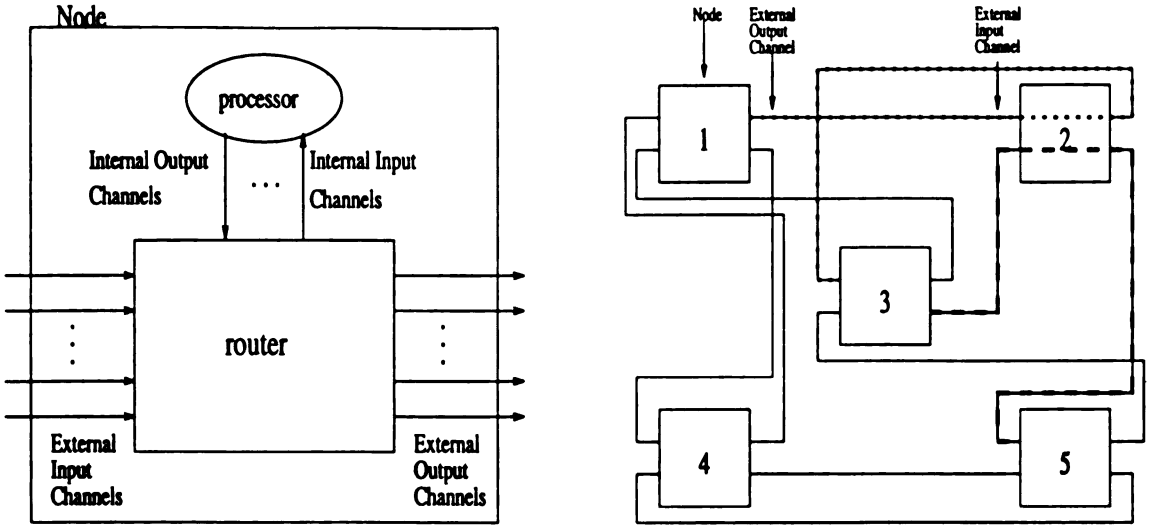
Chapter 2

Background

2.1 Direct Network Model

Efficient multicast communication is needed in parallel and distributed systems with widely varying architectures. The specific organization of the communication network used by the system restricts the manner in which multicast can be implemented. In this dissertation, we study multicast communication in networks that are based on a *direct network* interconnection structure.

Many direct network systems, for example the Intel Touchstone DELTA and the Intel Paragon [30] use the generic node architecture shown in Figure 2.1(a). Each node consists of a processor, local memory, and a separate router to handle communication-related tasks. Each router has several input and output channels. The router uses *internal* channels to communicate with its associated processor, and *external* channels are used for communication between nodes in the network. Normally, each input channel is paired with a corresponding output channel. Thus, the number of input and output channels is identical. If there is only one (input,output)-pair of internal channels, the system is called a *one-port architecture*. In a one-port architecture, a node can send only one message at a time and receive only one message at a time.



(a) Generic node architecture

(b) 5 node network

Figure 2.1: Direct network model

Because memory is not shared, all communication in a direct network system is done via message passing. Communication between nodes is either *direct* or *indirect*. Two nodes can communicate directly if an external output channel of one node is connected to an external input channel of the other node. For example, Figure 2.1(b) shows a possible interconnection of five nodes. In this network, node 1 can communicate directly with node 2. In contrast, sending a message from node 1 to node 3 requires indirect communication because other nodes must be used to relay the message.

There are two note worthy characteristic of systems that use the generic node design of Figure 2.1(a). First, the router is typically designed using a crossbar, which permits simultaneous transmission of several distinct messages provided each incoming message requires a different output channel. Second, the separation of the router from the local processor makes the router capable of relaying messages on its external channels without interrupting the local processor, allowing simultaneous computation and communication within each node [45]. To illustrate these concepts, consider the network in Figure 2.1(b). Suppose node 1 sends a message to node 3, using node 2 as an intermediate node (shown by the dotted line). Node 3 can simultaneously send a message to

node 5, also using node 2 as an intermediate node (shown by the dashed line), because it requires a different external channel of node 2's router. Furthermore, the computation taking place on node 2's processor is not interrupted because only node 2's router is being used to relay messages.

2.2 Switching Techniques

When a message is sent between two nodes in a network, the message travels along a routing path (which may be a direct communication link or a sequence of several communication links) in the network. The time needed to deliver the message depends on the characteristics of the underlying system. One factor that contributes to the message transmission time is the *network latency* of the system, or elapsed time between when the head of the message enters the network and the time when the tail of the message is received. Network latency is highly dependent on the type of switching mechanism that the system employs. Essentially two types of switching have been used: *store-and-forward* switching and *cut-through* switching.

Store-and-Forward Switching

Early direct networks used store-and-forward switching techniques. In a system that uses store-and-forward switching, an entire message is transferred from a node v to one of its neighbors w. If the message is not destined for w, *i.e.*, w is an intermediate node, the message is forwarded to one of w's neighbors when a buffer becomes available. The network latency for the store-and-forward switching technique is $(\frac{L}{b})d$, where L is the message length, b is the channel bandwidth, and d is the distance (number of links) the message must traverse. Because each message must be entirely received at a node before it can be forwarded, the amount of time needed to deliver the message is proportional to the path length d. For a more detailed discussion of routing algorithms based on store-and-forward switching see [48].

Cut-Through Switching

Cut-through switching was developed to reduce message transmission time. A cut-through scheme supports *non-local* communication primitives; that is, a node v can send a message to a non-neighboring node w in "unit" time. This is possible because messages are not stored at intermediate nodes unless the next required channel is unavailable. The message is sent along the path in a pipelined fashion. The network latency for cut-through switching is $(\frac{L_h}{b})d + \frac{L}{b}$, where L_h is the length of the header, b is the channel bandwidth, d is the length of the path traversed, and L is the total length of the message. If L_h is significantly smaller than L, then the dominant term is $\frac{L}{b}$. That is, network latency is relatively unaffected by d. In a multicast implementation, each node is sending essentially the same message (the message headers may be slightly different, but this will not significantly affect the message length.) Thus, we can assume that any (reasonably sized) multicast message can be delivered in unit time, where the "unit" is $\frac{L}{b}$, regardless of the length of the path traversed by the message [45].

Two common routing schemes that use the cut-through approach are *circuit routing* and *wormhole routing*. In circuit routing, before a node v can send a message to a node w, v establishes a circuit, or a path, from v to w. The message is then sent along this path in a pipelined fashion. Because the channels on the circuit are reserved, no buffers are needed at intermediate nodes.

In wormhole routing, a packet is divided into a number of *flits* (flow control digits). The header flit advances along the delivery path and the other flits follow in a pipelined fashion. If the header is blocked, then all trailing flits are also blocked. Consequently, small flit buffers are needed at intermediate nodes. Current commercial multicomputers such as the Cray T3D, NCUBE-2 [44], Intel Paragon [31], and IBM SP-2 [50] use wormhole routing. Ni and McKinley provide a thorough survey of wormhole routing techniques [45].

2.3 Direct Network Topologies

In a direct network system, the interconnection of the external input and output channels of the routers determine the topology of the system. Three popular network topologies that we will refer to often throughout this dissertation are: meshes, tori, and hypercubes.

2.3.1 Mesh

In an $n \times m$ 2-dimensional mesh, the nodes of the network are labeled by ordered pairs (v_i, v_j), where $0 \leq i \leq n-1$ and $0 \leq j \leq m-1$. Each node has a direct connection to at most four other nodes, namely (v_{i+1}, v_j), (v_{i-1}, v_j), (v_i, v_{j+1}), and (v_i, v_{j-1}) (if they exist). Figure 2.2 shows an example of a 2×3 mesh.

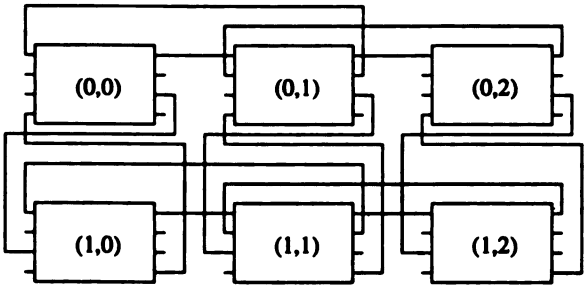


Figure 2.2: A 2×3 two-dimensional mesh

Higher dimension meshes are defined similarly. An n-dimensional mesh has nodes that are labeled by n-tuples $(v_1, \ldots, v_n)$. Each node has a direct connection to at most $2n$ other nodes that differ by one in each component of its n-tuple.

2.3.2 Torus

An n-dimensional torus is an n-dimensional mesh with *wrap around* edges. For example, in a two-dimensional torus each node (v_i, v_j) has exactly four direct connections to nodes $(v_{(i+1) \bmod 2}, v_j)$, $(v_{(i-1) \bmod 2}, v_j)$, $(v_i, v_{(j+1) \bmod 2})$, and $(v_i, v_{(j-1) \bmod 2})$. The edges that did not exist for the mesh

topology are the wrap around edges. Figure 2.3 shows an example of a 2×3 torus. The wrap around edges are shown in **bold** lines. Higher dimensional tori are defined similarly.

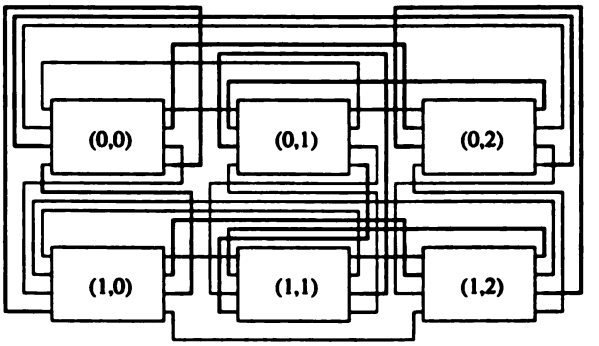


Figure 2.3: A 2×3 two-dimensional torus

2.3.3 Hypercube

An n-dimensional hypercube has 2^n nodes label by all possible n-bit binary strings. Each node has direct connections to n other nodes. Specifically, each node is connected to all nodes whose bit strings differ in exactly one bit. Figure 2.4 shows and example of a 3-dimensional hypercube topology.

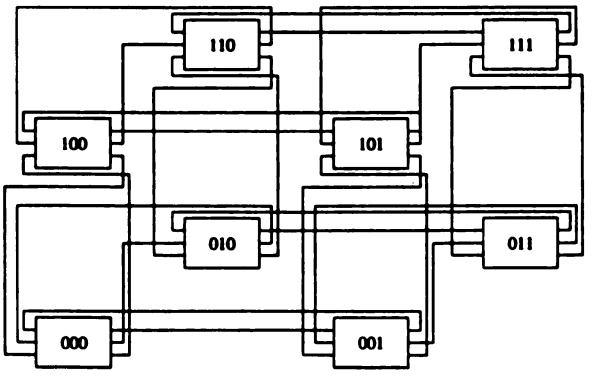


Figure 2.4: A three-dimensional hypercube

2.3.4 Symmetric Communication

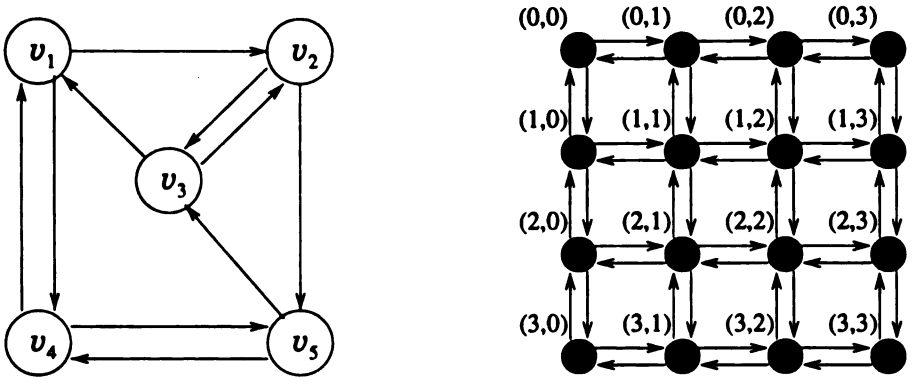
Meshes, Hypercubes, and Tori have the property that if node x has an external output channel connected to an external input channel of node y, then y also has an external output channel connected to an external input channel of node x. Topologies with this property are said to have *symmetric* (or "two-way") communication between nodes x and y.

In an arbitrary topology that uses the generic node architecture, communication may not be symmetric. For example, in the network shown in Figure 2.1(b) node 1 has a direct connection to node 2, but node 2 does not have a direct communication link to node 1. We will assume arbitrary topologies do not have symmetric communication unless otherwise stated.

2.4 Graph Theoretic Formulation

When developing a multicast algorithm, it is convenient to have a mathematical model of the system. In particular, the topology of a direct network system can be modeled by a graph, and algorithms are developed using the graph model. In the system described in Section 2.1, each of the communication links from an external output channel of one node to an external input channel of another node represents a *one-way* communication link. Consequently, communication is not necessarily symmetric. Networks with this organization are best modeled by directed graphs, or *digraphs*. That is, a direct network topology is modeled by a digraph $G = (V, A)$, where V represents the nodes in the network and A represents the communication links. For example, the digraph in Figure 2.5(a) models the network pictured in Figure 2.1(b), and Figure 2.5(b) is a digraph that represents a 4×4 mesh topology.

Topologies that provide symmetric can be modeled by an undirected graph $G = (V, E)$, where V represents the nodes of the network and E represents two-way communication links between



(a) Network in Figure 2.1(b)

(b) 4 × 4 mesh

Figure 2.5: Digraph representations of direct networks

nodes. In particular, if edge $e = v_iv_j \in E(G)$ then v_i and v_j can send messages directly to each other along the edge e. We assume G is connected and simple, *i.e.*, there are no loops or parallel edges. The undirected graph model of a 4 × 4 mesh is seen in Figure 2.6. This undirected graph is

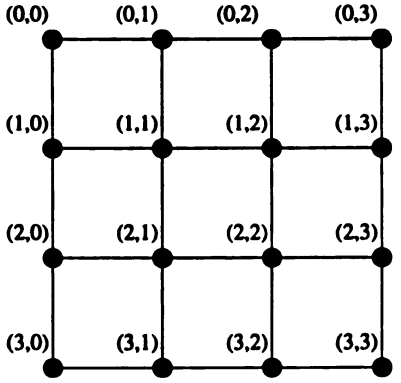
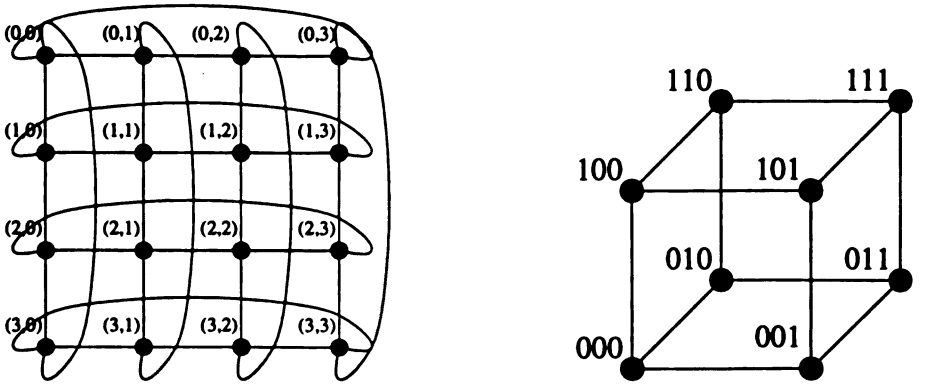


Figure 2.6: Graph model of a 4 × 4 Mesh

equivalent to the digraph model in Figure 2.5(b). For the sake of simplicity, we will use a general graph model to introduce main concepts and use directed graphs in special cases described later. The graph models for a 4 × 4 torus and a three-dimensional hypercube are shown in Figures 2.7(a) and 2.7(b). Notice that these graphs are much easier to decipher than Figures 2.3 and 2.4.

In a direct network, messages are sent between nodes using direct or indirect communication. The route that the message follows corresponds to a *path* in the graph model. A messages is sent



(a) 4 × 4 torus

(b) 3-dimensional hypercube

Figure 2.7: Graph models of direct networks

from source node v_i to destination node v_j along an (i,j)-path. A sequence $p = v_1 v_2 \ldots v_n$ is a path[1] if there is an edge between node v_i and node v_{i+1}, and if $v_i \neq v_j$ for all i, j. We call p an (i,j)-path if $v_1 = v_i$ and $v_n = v_j$; the path is denoted by $p(i,j)$. Direct communication implies that the path $p(i,j)$ used to send a message from node v_i to node v_j has length one, *i.e.*, $p(i,j)$ is precisely the edge $e = v_i v_j$. In indirect communication, an (i,j)-path uses intermediate nodes to forward a message to its destination, *i.e.*, $p(i,j) = v_i \ldots v_k \ldots v_j$, where v_k is an intermediate node.

To initiate communication between nodes in a direct network, a *message passing request* is made. A message-passing request is an ordered pair $M = (S, D)$, where $S \subseteq V(G)$ is the source set and $D \subseteq V(G)$ is the destination set. Message passing paradigms are classified according to the sizes of the source and destination sets. The multicast communication paradigm that we described above has the characteristics that $|S| = 1$ and $|D| \geq 1$. Two specific cases of multicast are *unicast* and *broadcast*. Unicast, or one-to-one, communication involves a single source and a single destination, *i.e.*, $|S| = |D| = 1$. In broadcast, also called one-to-all communication, $D = V(G) - S$. Thus, the multicast problem is formulated in graph theoretic terms as follows:

[1] A path is typically defined as an alternating sequence of vertices and edges. Since we consider only simple graphs, a path can be defined by a sequence of vertices.

Given a graph $G = (V, E)$ and a message passing request $M = (s, D)$ in G, find an efficient implementation for M in G.

2.5 Multicast Classification

Before a multicast algorithm can be developed for a specific system represented by a graph, the physical characteristics of the system must be defined, and any constraints that these characteristics impose must be incorporated into the graph model and multicast implementation.

Figure 2.8 shows a classification of different implementation methods for various constraints that have been defined for direct network systems. We describe each component of the classification in the following sections.

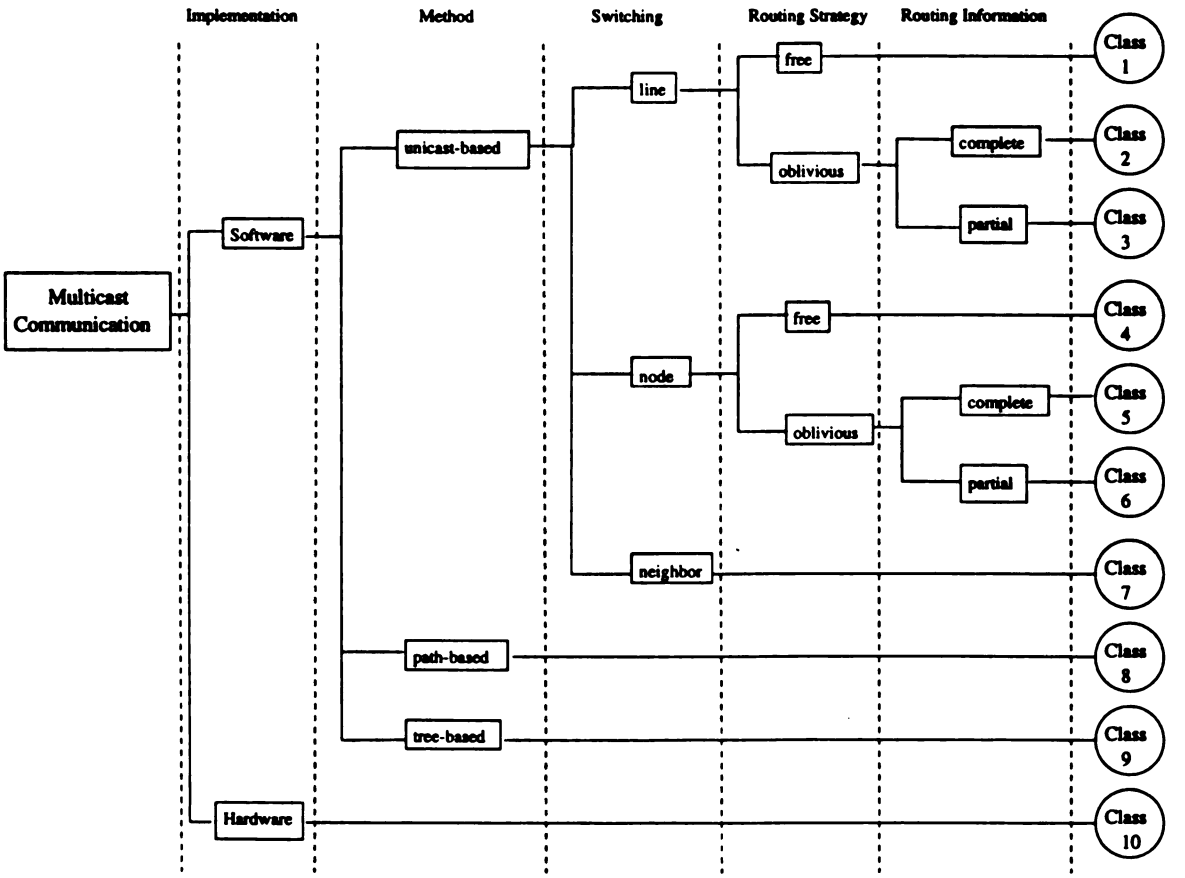


Figure 2.8: Multicast problem classfication

Our research addresses only 1-port architectures. However, the same classification applies to k-port architectures, in which each node has k pairs of internal channels. Let Class 1_k refer to a system with the characteristics of Class 1 in Figure 2.8 and a k-port architecture. We will assume all problems refer to a 1-port architecture unless otherwise stated. Thus, Class 1 (with no subscript) is understood to be Class 1_1.

2.5.1 Implementation

Like any operation, multicast communication can be implemented in either hardware or software. The NCUBE-2 offered a restricted multicast implementation in hardware; however, the hardware implementation was not deadlock free and was therefore disabled [40]. Due to the complexity of hardware implementations of multicast, most existing direct network systems only support unicast in hardware. Consequently, multicast must be offered in software. We address only software implementation of multicast communication in this dissertation.

2.5.2 Software Methods

Several software methods have been discussed in the literature: Unicast-Based Multicast (UBM), path-based multicast, and tree-based multicast. We desribe each of these in the following sections; however, our research addresses only UBM implementations.

Unicast-Based Multicast

Unicast-Based Multicast (UBM), in which multicast is accomplished by issuing multiple hardware unicast messages, has become a popular software technique for implementing multicast in direct network systems [21, 46, 47, 2, 4, 5]. Farley describes a *calling schedule* for implementing broadcast

communication [20, 21][2]. A calling schedule consists of several time steps of unit length during which one or more unicasts can be executed in parallel. Each unicast is characterized by three parameters: source, destination, and time. In order to clearly indicate the path used to perform a unicast, we represent a unicast as an ordered quadruple $(i, j, p(i,j), t)$ which is read as, "Node i sends a message to node j along path $p(i,j)$ during time step t." For a multicast request $M = (S, D)$, where $S = \{s\}$ and $D = \{d_1, d_2, \ldots, d_n\}$, we define the *informed set* of M at time step t with respect to a calling schedule C, denoted $I_t^C(M)$, to be the set of all nodes from $S \cup D$ that have received the message after time step t is executed under calling schedule C. Thus, $I_0^C(M) = \{s\}$ for any C and any M. The *length* of a calling schedule is the number of time steps in which calls are made.

To illustrate these concepts, consider multicast request $M = \{v_1, v_2, v_3, v_5\}$ in the graph of Figure 2.5(a). The calling schedule $C = \{(v_1, v_2, v_2v_3, 1), (v_1, v_5, v_1v_4v_5, 2), (v_2, v_3, v_2v_3, 2)\}$ implements M in G. One unicast call from v_1 to v_2 is made during the first step. During the second step, two unicast calls are made – one from v_1 to v_5 and another from v_2 to v_3. Thus, the informed sets are:

$$\begin{aligned} I_0^C &= \{v_1\} \\ I_1^C &= \{v_1, v_2\} \\ I_2^C &= \{v_1, v_2, v_3, v_5\}, \end{aligned}$$

and the length of C is two.

[2]Farley assumes a graph model. All concepts are easily extended to a digraph model by making edges directed.

Tree-Based and Path-Based Multicast

In both the tree-based and path-based approaches, the multicast destinations are specified in the message header. In the tree-based approach, a spanning tree rooted at the source node is found, and the multicast message is propagated along the tree. In particular, the message follows a common path until a branching point is encountered; the message is then replicated and sent along each branch, where the message header contains only the destinations of the branch to which it is sent.

In the path-based approach, the message is sent along a path (or multiple paths) that goes through each destination node. If the router discovers that the message is destined for its associated processor, it both copies the incoming message to the local processor and forwards the message along the path. Unlike the tree-based approach, no branching occurs in this approach.

Both of these methods are concerned with finding trees or paths that are deadlock free. There is no other measure of "efficiency" involved in these approaches. In addition, these approaches may required some additional hardware to support replication capabilities.

2.5.3 Switching

The edges of the graph (or arcs of the digraph) model described in Section 2.4 represent the physical lines of communication between the nodes in the direct network system. Various characteristics of the system, such as hardware and routing techniques, determine when the physical communication lines can and cannot be used. In a UBM calling schedule, messages may be sent simultaneously between multiple source-destination pairs. One constraint that a UBM algorithm must consider when scheduling simultaneous unicast calls is the communication model that applies to the direct network system. Farley defined three different communication models [21]. The distinguishing characteristic in the three variations is the requirements on the paths that can be used during simultaneous message transmissions. In the *line-switching* model, all the paths used simultaneously must be edge-

disjoint. In the *node-switching* model, all the paths used simultaneously must be node-disjoint. In the *neighbor-switching* model, all communication is direct, *i.e.*, each path used consists of a single edge.

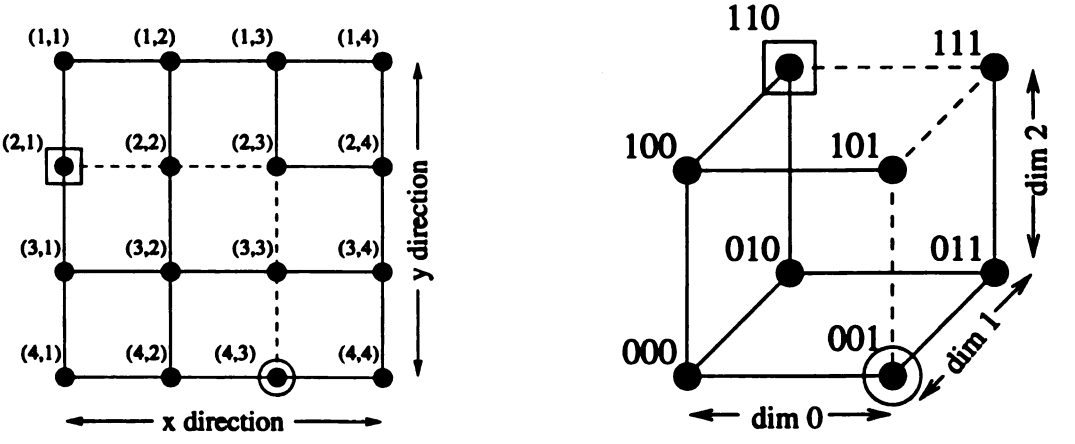
As the names suggest, these communication models depend on the type of switching used by the direct network. Store-and-forward switching is best described by the neighbor-switching model. In the neighbor-switching model, a message can only be sent to a neighboring node, and the time needed to send a message to a neighboring node is considered to be one unit. The line-switching model is applicable to current wormhole-routed, commercial multicomputers with generic nodes as depicted in Figure 2.1(a). Any call requires unit time because message transmission is relatively independent of path length. In addition, each router can relay several messages simultaneously provided the messages do not require the same external channels (*i.e.*, paths are edge-disjoint). Thus, two calls that use the same node can be made simultaneously, as long as the external channels used in each path are disjoint. Because most current commercial multicomputers use cut-through techniques, we will focus on the line-switching model of communication. Node-switching does not have an analogous switching technique; however, notice that the node-switching criterion also satisfies the line-switching criterion.

2.5.4 Routing Schemes

In the network in Figure 2.1(b), there may be multiple paths along which a message can be delivered between two nodes. For example, node 1 can send a message to node 3 using node 2 as an intermediate node, or node 1 could also use both 4 and 5 as intermediate nodes to deliver a message to node 3. The path along which the message is delivered depends on the *routing scheme* used. A routing scheme R of a direct network is a collection of all permissible paths along which a message can be delivered for each pair of nodes in the network. If a routing scheme R includes *every* (i, j)-path for

each v_i and v_j, the system is said to use *free* routing [46]. If, on the other hand, R includes *exactly one* (i, j)-path for every pair of nodes v_i and v_j, the system is said to use *oblivious* routing [35] or *restricted* routing [46]. UBM algorithms are highly dependent on the routing mechanism used by the underlying system.

In practice, parallel systems use oblivious routing rather than free routing. Current systems based on mesh and hypercube topologies use an oblivious routing scheme called dimension-ordered routing. For example, two dimensional mesh topologies typically use the xy-routing scheme. Suppose a message is being sent from source node (x_i, y_i) to destination node (x_j, y_j) in a 2D-mesh. The message is first routed in the x-direction (along row x_i) until it reaches column y_j. The message is then routed in the y direction (in column y_j) until it reaches its destination. To deliver a message from node $(2, 1)$ to node $(4, 3)$ in Figure 2.9(a), the path shown by dashed lines is used. The xy-routing scheme is oblivious because the specified path is unique and is used regardless of



(a) xy-routing in a 2D-mesh

(b) e-cube routing in a hypercube

Figure 2.9: Oblivious routing schemes.

network conditions. The Intel Paragon [31], a multiprocessor computer with a two dimensional mesh network, utilizes xy-routing. Note that a similar dimension-ordered routing can be defined for torus topologies.

Hypercube topologies typically use *e*-cube routing, in which the message also travels through the cube dimensions one at a time. Figure 2.9(b) shows the oblivious *e*-cube routing path used to deliver a message from node 110 to node 001 in a three dimensional hypercube. The NCUBE-2 [44], a multiprocessor computer with a hypercube network, utilizes *e*-cube routing.

The third type of routing combines both of the above models. In *partially-restricted* routing, part of the path is restricted and part of the path is free. This model was motivated by turnaround routing in multistage cube networks. In turnaround routing, any path can be used in the forward direction, but a specific path must be used in the backward direction [53]. We can define a similar type of routing in direct networks.

2.5.5 Routing Information

The final issue in the multicast classification of Figure 2.8 addresses the amount of routing information stored at each node. Ideally, each node has *complete* routing information. In other words, each node knows the path used to route a message between any pair of nodes in the network. If routing information is stored as a routing table, the storage requirements for complete information can be quite large. In order to reduce storage requirements, each node may contain only *partial* routing information. The least amount of information that a node can have is to store only the paths it uses to route messages to other nodes in the network. Thus, a node v_i knows how to route messages to all nodes in the network but does not know how node v_j routes a message to node v_k. We will refer to this as *source limited* routing information. Complete routing information and source limited routing information represent the two extremes of the amount of information that can be stored. There may be several levels between these extremes.

2.6 Legal UBM Calling Schedules

Finding a UBM algorithm for multicast request $M = (s, D)$ in a graph G is equivalent to finding a *legal* calling schedule for M. There are five requirements for a calling schedule C to be legal: that any path used to send a message must be contained in the routing scheme, that all the paths used to perform a unicast in any given time step t must be source and destination disjoint (*i.e.*, a node can only inform one destination in a time step, and a node can only be informed once in a time step), that only previously informed nodes can be the source of any unicast, that any destination node receives the message exactly once, and that all destinations in D are informed within a finite amount of time. In generalizing from broadcasting to multicasting, we add the requirement that only nodes in the source and destination set of multicast request M may be the source or destination of a unicast. We present below the formal requirements for a legal calling schedule C that implements a multicast request M under the line-switching model.[3] The requirements for legal calling schedules that implement node- and neighbor-switching are identical except for an appropriate modification to condition (vii).

(i) If $(i, j, p(i, j), t) \in C$ then $p(i, j) \in R$.

(ii) If $(i, j, p(i, j), t) \in C$ and $(m, n, p(m, n), t) \in C$ then $i \neq m$ and $j \neq n$.

(iii) For all $(i, j, p(i, j), t) \in C$, $i \in I_t^C(M)$.

(iv) For all $(i, j, p(i, j), t) \in C$, $j \notin I_t^C(M)$.

(v) There exists a time step t such that $I_t^C(M) = S \cup D$. We call t the *length* of C.

(vi) For all $(i, j, p(i, j), t) \in C$, $i, j \in S \cup D$.

(vii) If $(i, j, p(i, j), t) \in C$ and $(m, n, p(m, n), t) \in C$ are distinct unicasts, then $p(i, j)$ and $p(m, n)$ are edge-disjoint.

[3] Farley's requirements for a corresponding legal line-switching calling schedule include only conditions (iii) and (vii).

2.6.1 Inter-Class Relationships

We focus on UBM algorithms for direct network systems. However, there are seven distinct classes with the UBM framework. An obvious question that arises is: What is the relationship between the classes? Figure 2.10 shows the relationships between legal UBM schedules.

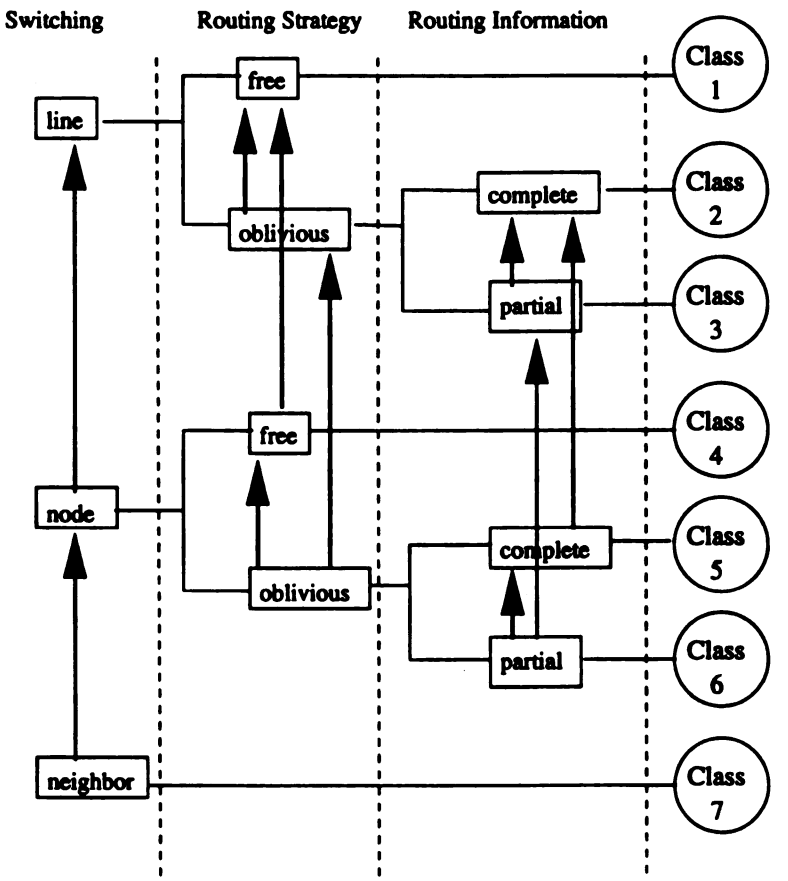


Figure 2.10: Legality relationships for UBM Calling Schedules

The paths used in any step of a legal node-switching schedule are node-disjoint. Thus, the paths are automatically edge-disjoint. This implies that any legal node-switching schedule is also a legal line-switching schedule. Similarly, because the neighbor-switching model only allows calls to be made along paths of length one, a legal neighbor-switching schedule is inherently node-disjoint. Thus, a legal neighbor-switching schedule is also a legal node-switching schedule (and therefore a legal line-switching schedule as well). With respect to the routing strategy, any legal schedule

for a system that uses oblivious routing is also a legal schedule for a system that employs free routing since the oblivious routing paths are a subset of the free routing paths. Similarly, any legal schedule that has access to only partial information is a legal schedule in a system that has complete information.

2.7 Performance Measures

Once we have determined the constraints of the system, there may be several legal calling schedules that satisfy a multicast request. Thus, we need to evaluate the performance of a UBM algorithm so that the most efficient calling schedule can be selected.

In general, efficient algorithms should use as little of the system's resources as possible. We consider two resources: time and traffic. Because communication overhead is so costly, it is desirable to minimize the amount of time (or number of steps) necessary to complete a multicast request. More formally, one objective is to find an algorithm that always produces legal calling schedules of minimum length. The lower bound on the number of steps for a legal UBM schedule is $\lceil \lg(|D|+1) \rceil$. A lower bound UBM implementation will require exactly $\lceil \lg(|D|+1) \rceil$; however, an optimal UBM schedule may require more steps if the lower bound is not achievable in the particular system in which the algorithm is used. Thus, there is an important distinction between optimal and lower bound UBM implementations. We define a *time efficient* UBM algorithm to be one that minimizes the number of steps needed to perform the multicast in the particular system (*i.e.*, an optimal implementation).

Another way to measure performance is by the number of communication links used. Using a small number of communication lines keeps the traffic low and allows better overall system performance. Thus, another objective is to minimize the sum of the length of the paths used in a legal calling schedule. We will call a UBM algorithm that minimizes the number of communication lines

that are used *traffic efficient*.

2.7.1 Optimality Implications

In Section 2.6.1, we showed the relationships between legal calling schedules for the different classes of UBM implementations. One may ask: Is there a similar relationship between optimal UBM implementations? The implications shown in Figure 2.10 are exactly the same with respect to lower bound implementations. Because no calling schedule can use fewer than $\lceil \lg(|D| + 1) \rceil$ steps, a lower bound implementation C that is legal for one class will clearly be a lower bound implementation for any other class in which C is a legal implementation.

Unfortunately, these implications do not hold for *optimal* implementations. For example, an optimal schedule for a system in Class 3 may require more than $\lceil \lg(|D| + 1) \rceil$ step. Because additional routing information is available for Class 2, an optimal schedule for a system in Class 2 may require only $\lceil \lg(|D| + 1) \rceil$ steps. A calling schedule C is legal for both classes, but not optimal for both classes. Very little is known about the optimality implications for the classes. This dissertation begins to answer this question.

Chapter 3

Related Work

Multicast, in particular broadcast, is one of the most widely studied communication paradigms in networks. In this chapter, we give a brief survey of the various multicast problems that have been addressed, focusing on unicast-based multicast implementations. Much of the early work in multicast communication considered the problem of broadcasting in multicomputer systems that assumed the neighbor-switching model. We describe this work in Section 3.1.1. As circuit routing techniques such as wormhole routing became popular, attention turned to multicast under the line-switching model. We discuss work in this area in Section 3.1.2.

In the UBM algorithms we describe, it is assumed that any message can be delivered in unit time. In recent years, several authors have used other models in attempt to more accurately model message transmission time on real networks. We describe several of these variations in Section 3.3.

In addition, parallel systems that are not based on direct networks are gaining popularity. For example, the IBM SP-2 is based on an indirect network, called a multistage interconnection network, as opposed to a direct network. We discuss multicast techniques that have been developed for these and other networks in Section 3.4.

Finally, because parallel machines and networks are often modeled by graphs, many problems

in parallel and distributed processing are closely related to problems in graph theory. In Chapter 6 we will show how a path-matching problem in graphs has direct application to multicast communication. Thus, in Section 3.5 we provide a brief discussion of previous matching problems in graphs.

3.1 Unicast-Based Multicast Communication in Direct Networks

3.1.1 Neighbor-Switching (Class 7)

In the neighbor-switching model, a node may only call one of its neighbors. For example, node $(1, 1)$ in Figure 3.1 can only make calls to nodes $(1, 2)$ and $(2, 1)$. In this model, we assume that the cost to deliver a message is uniform for all communication links, *i.e.*, a message can be delivered in unit time.

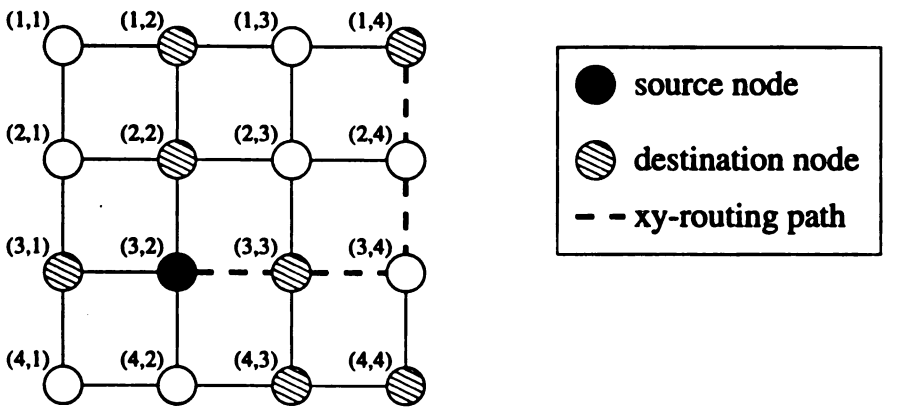


Figure 3.1: Multicast in node-switching model.

Before circuit routing became popular, most networks used store-and-forward routing. As mentioned in Section 2.5.3, networks that use store-and-forward routing are best modeled by the neighbor-switching assumption. Consequently, much of the early work on multicast in networks assumed the neighbor-switching communication model. Notice that the concept of free routing vs. oblivious routing is not applicable in the neighbor-switching model. There is only one "path" (of length one) between a node v and its neighbor w. Thus, node v is both free to use any available path

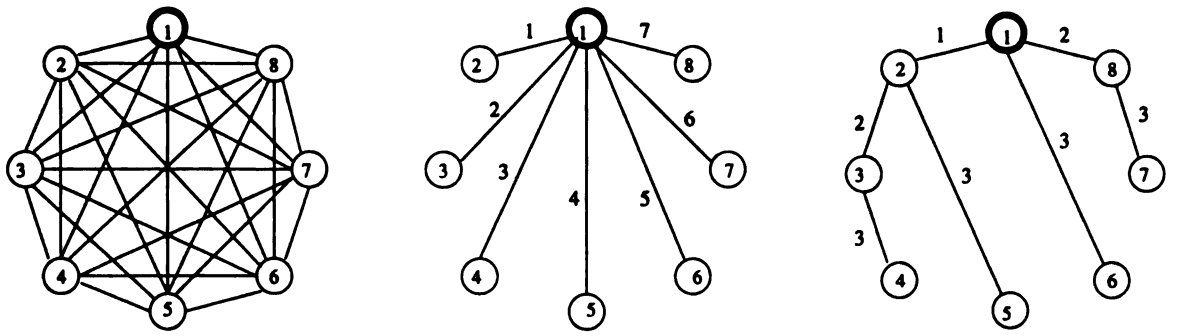
and restricted to a specific path when sending a message to w.

One drawback of the neighbor-switching model is that not all subsets of nodes may be reached using only local calls. For example, suppose a multicast request with $S = \{(3,2)\}$ and $D = \{(1,2),(1,4),(2,2),(3,1),(3,3),(4,3),(4,4)\}$ is made in the network in Figure 3.1. Under the neighbor-switching assumption, there is no way to deliver the message to node $(1,4)$ using only nodes in $S \cup D$. Thus, it is not surprising that most of the work that uses the neighbor-switching model concentrates on the broadcast problem, where $S \cup D = V(G)$. Other authors have allowed vertices that are not source or destination vertices to be involved so that multicast can be addressed. In this case, nodes that do not need the message are interrupted and required to forward the message.

Slater *et al.* show that the problem of determining the minimum amount of time required to broadcast from an arbitrary vertex of an arbitrary graph is NP-complete [49]. Because determining a minimum length broadcast schedule in an arbitrary graph is NP-complete, researchers have focused on other aspects of the problem. In the context of neighbor-switching, two classes of problems have been considered. The first problem is to find an efficient way to broadcast in a *given* graph. The second problem is to build a graph in which broadcast can always be done efficiently.

Finding Efficient UBM Broadcast Algorithms

In general, a broadcast request is made in a given graph, and a calling schedule (see Section 2.6) is used to implement the broadcast. The source node makes a call to one of its neighbors, and the other nodes may also make calls to neighbors after they have received the message. This continues until all nodes are informed. Typically, the edges that the messages travel along make up a *broadcast tree*. Figure 3.2 shows two broadcast trees for K_8, where the source node is node 1. The number on each edge shows the time step during which the edge is used to deliver a message. The broadcast tree in 3.2(b) requires seven steps while the tree in 3.2(c) requires only three. Many broadcast trees



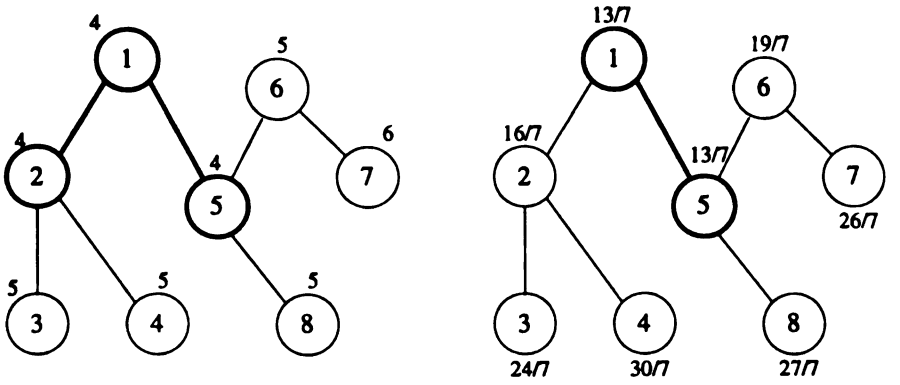
(a) Complete graph K_8 (b) 7-step broadcast tree (c) 3-step broadcast tree

Figure 3.2: Broadcast trees for K_8.

exist for a single graph. One objective that has been studied is to choose the "best" broadcast tree.

Because determining the minimum amount of time required to broadcast from an arbitrary vertex of an arbitrary graph is NP-complete, Slater *et al.* focus on broadcasting in trees. In particular, they develop an algorithm for finding the *broadcast center* of a tree. In general, a center of a graph is defined as follows: (1) each vertex is assigned a value according to some measure, *e.g.*, weight, distance, or time; (2) the center is the set of vertices that have minimum value. In [49], the measure being minimized is the number of time steps needed to broadcast from a vertex. That is, the broadcast center of a tree is the set of vertices in the tree from which broadcast can be completed in the least amount of time. Slater *et al.* show that the broadcast center of a tree is always a star with two or more vertices. For example, consider the tree in Figure 3.3(a). The number shown beside each node is the broadcast time from that node. Thus, the subgraph shown in bold lines is the broadcast center of the tree.

Koh and Tcha [43] use a different measure to find the broadcast center of a tree. They minimize the average time at which a vertex in the tree receives the message; this is called the *minisum* criterion, whereas the function used by Slater *et al.* is called the *minimax* criterion. The average time that each node receives the broadcast message is shown in Figure 3.3(b), and the center under



(a) Broadcast center under minimax (b) Broadcast center under minisum

Figure 3.3: Broadcast center of a tree.

the minisum criterion is again shown by bold lines. The center is not the same as that for the minimax criterion, however this is not always the case.

Bharath-Kumar and Jaffe [8] consider multicast in arbitrary networks, and they focus on traffic efficiency rather than on time. As mentioned earlier, not all subsets of nodes can be reached using local calls. Thus, some of the nodes used to deliver the message may *not* be destination nodes. Furthermore, the authors permit any subgraph to be used to deliver messages. Suppose a node must deliver a message to a non-neighboring node. A common technique is for the node to use local information to determine which link it will use to forward the message (we called this *source limited* information in Section 2.5.5), and therefore which node is the next to be responsible for delivering the message. Because the routing is done in this distributed manner, any subgraph may be used for the multicast. For example, Figure 3.4 shows a graph that is used for multicast that is not a tree. Node 1 is the originator of the message, and it is responsible for sending messages to nodes 6 and 7. Node 1 chooses to send the message destined for node 6 through intermediate node 2 and to send the message destined for 7 through intermediate node 5. Node 2 sends the message directly to node 6. Node 5 choose to send the message destined for 7 through node 6, thus creating a cycle in the multicast graph.

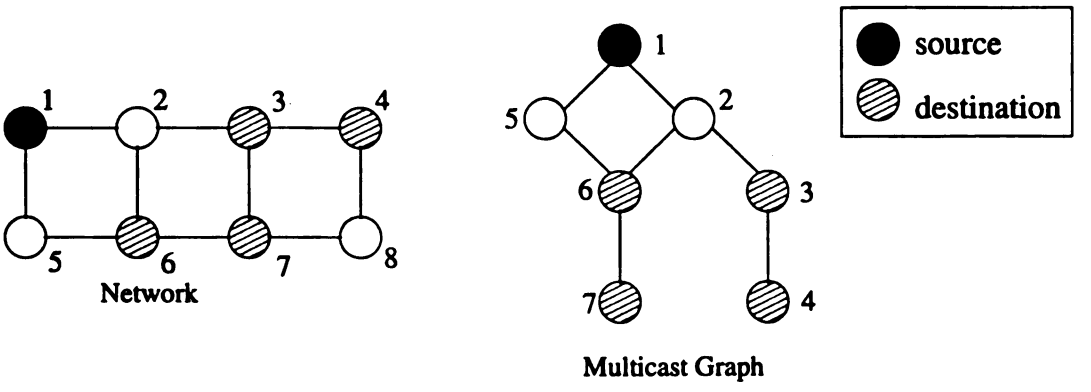


Figure 3.4: Multicast subgraph.

Within this context, Bharath-Kumar and Jaffe define two criteria used to measure the efficiency of the multicast implementation. Network Cost (NC) measures the number of communication links used to deliver the message and Destination Cost (DC) measures the average delay experienced by each destination. Their goal is to minimize NC because applications like file transfer do not rely heavily on message delay. The authors explain that, while optimizing DC is relatively easy, optimizing NC is an NP-complete problem. They discuss the feasibility of using DC optimal algorithms to approximate NC optimal solutions, and they devise several heuristics for minimizing NC.

Multi-port Architectures (Class 7_k)

Finding efficient neighbor-switching UBM algorithms has also been studied in systems with a multi-port architecture, *i.e.*, in systems where each node can send or receive more than one messages at a time. Choi and Esfahanian [14] study multicasting in such systems. They model the network as a graph and consider both time and traffic efficiency. In particular, they propose the Optimal Communication Tree (OCT) problem, in which a broadcast tree T that preserves the distances in G from each node to the source and that has the fewest number of nodes is to be found. In general, OCT is NP-complete. Choi and Esfahanian show that the OCT problem is NP-complete even for the n-cube and for graphs whose maximum degree is at most three. They also present some heuristics

for the OCT problem.

Efficient Broadcast Graphs

Other work under the neighbor-switching assumption has focused on characterizing the types of graphs in which broadcast can always be done efficiently. The broadcast time from a vertex v in a graph G is the minimum number of time steps needed to broadcast a message from v. The broadcast time of a graph G, $b(G)$, is the maximum broadcast time from any vertex in G. Because the number of informed nodes can at most double in each step, $b(K_n) = \lceil \lg n \rceil$. However, some edges can be removed from K_n, and the resulting graph still has broadcast time $\lceil \lg n \rceil$. A minimal broadcast graph on n vertices is a graph G with the properties that $b(G) = \lceil \lg n \rceil$, but any subgraph G' of G has $b(G') > \lceil \lg n \rceil$. Furthermore, $B(n)$ is the minimum number of edges in any minimal broadcast graph, and a minimum broadcast graph (mbg) on n vertices is a minimal broadcast graph with $B(n)$ edges. An mbg provides a network topology in which broadcast can be completed from every node in a minimum number of time steps.

Farley *et al.* determine $B(n)$ for $n \leq 15$ and for graphs with 2^k vertices [19]. $B(n)$ has been calculated for other specific values of n as well [42, 7], however it appears very difficult to calculate $B(n)$ in general. Thus, many authors have focussed on constructing *sparse* broadcast graphs. A sparse broadcast graph contains a small number of edges that is not necessarily the minimum number of edges of a broadcast graph [20, 13, 26]. Not all of the broadcast graphs found provide practical network configurations. In an attempt to make more realistic mbgs, construction of sparse broadcast graphs with bounded maximum degree has also been studied [37, 6]. All of the results mentioned above were for undirected broadcast graphs. Liestman and Peters found minimum broadcast *directed* graphs for some specific values of n [38].

Variations of this problem have also been considered. Gargano *et al.* study the mbg problem in

the presence of failures [25]. In particular, they want to find $B_k(n)$, the minimum number of edges in any graph on n vertices than can broadcast from any node in $\lceil \lg n \rceil$ time steps in the presence of up to k transmission failures.

3.1.2 Line-Switching (Classes 1-3)

In Section 2.5.3 we showed that current direct networks, in which each node has a processor and separate router, allow "long distance" calls to non-neighboring nodes to be made in unit time. Unlike the neighbor-switching model, several physical paths may exist between a source node s and destination node d. If the system allows any (s, d)-path to be used, it is said to employ a free routing scheme. If the system specifies a unique (s, d)-path for sending a message from s to d, the system is said to use oblivious [35] or restricted routing [46].

Free Routing (Class 1)

Farley first addressed *line-broadcasting* (broadcasting in a general graph under the line-switching assumption) [21]. He shows that under the line-switching assumption, broadcasting can be completed in minimum time ($\lceil \lg n \rceil$ time units) in any connected network of n nodes that employs free routing, regardless of message originator.

McKinley, *et al.* addressed the problem of performing multicast in wormhole-routed direct networks [46]. They showed that there exists a lower bound implementation for any multicast request $M = (s, D)$, *i.e.*, it requires exactly $\lceil \lg(|D| + 1) \rceil$ steps to deliver the message to all destinations in D, in systems that admit free unicast communication, regardless of topology. The general idea of their algorithm is as follows. First, construct a trail that includes the source node and all the destinations nodes. The message is delivered from the source to a destination at the "middle" of the trail, and the trail is then broken into two equal length subtrails each of which contains an

informed node. This is continued recursively until all destinations are informed, thus $\lceil \lg(|D|+1) \rceil$ steps are required. Neither Farley nor McKinley *et al.* consider traffic efficiency.

Kane and Peters [32] consider time and traffic efficient broadcasting in cycles. They construct algorithms that always use a minimum number of phases ($\lceil \lg n \rceil$, where n is the number of nodes in the network) and that also minimize the number of communication links used in each phase.

Oblivious Routing

With respect to time efficiency, problems in Class 1 have been solved, *i.e.*, optimal algorithms have been found for line-switching systems that allow free routing. Unfortunately, most commercial machines offer *only* oblivious unicast routing, particularly in wormhole routed networks where deadlock would result if free routing were used. Consequently, it is important to study efficient UBM algorithms for systems that employ oblivious routing.

Two popular topologies for multicomputers are the mesh and the hypercube. Both of these systems use an oblivious routing technique called dimension-ordered routing. Two commonly used dimension-ordered routing schemes are xy-routing and e-cube routing. In xy-routing, used in a two-dimensional mesh, a message moves first in the x direction and then in the y direction. For example, to deliver a message from node $(3,2)$ to node $(1,4)$ in Figure 3.1 the path shown by dashed lines is used. The hypercube uses e-cube routing, in which the message travels through the dimensions one at a time. McKinley *et al.* [46] use the calling schedule implementation described in Section 2.6 to develop lower-bound (on time) algorithms for doing multicast in hypercubes and meshes. Their algorithm was later extended to do multicast in torus networks [47].

Multi-port Architectures (Class 2_k)

Variations of this problem have also been studied. McKinley and Trefftz study broadcast in *all-port* wormhole-routed hypercubes [41]. All-port architectures have the same number of internal and external channels (see Figure 2.1(a)). They develop the double tree (DT) algorithm, and show that it completes broadcast in $\lceil \frac{n}{2} \rceil$ steps in an n-cube.

3.1.3 Node-Switching (Classes 4-6)

Because the node-switching model does not have a corresponding switching technique in direct networks it has not been studied as extensively as the other models. Farley addressed node-switching algorithms for Class 4, *i.e.*, for systems that allow free routing [21]. He shows that it may be impossible to find a minimum length, *i.e.*, $\lceil \lg n \rceil$, node-disjoint broadcast schedule in an arbitrary tree with n nodes. However, he also shows that for all n there does exist a tree with n nodes that allows minimum time node-disjoint broadcasting to be done.

3.2 Tree-Based and Path-Based Communicaton (Classes 8-9)

As discussed in Chapter 1, there are also multicast algorithms that are not based on making several unicast calls, *i.e.*, non-UBM algorithms. In tree-based and path-based multicast, the destinations are encoded in the header, and the router examines the header and copies or forwards the message. Because messages that traverse multiple communication links can hold several links at a time, deadlock can occur if the paths are not chosen carefully.

In [11], Byrd *et al.* describe three tree-based multicast protocols for deadlock-free multicast, however no specific algorithms are given. Lin and Ni show that in wormhole-routed multicomputers tree-based multicast algorithms can suffer from deadlock [40]. In particular, they show that deadlock can occur in the tree-based multicast algorithm implemented in the nCUBE-2. In [39], Lin *et al.* give

a tree-based algorithm for 2-D meshes that is deadlock-free if double channels are used; however, they explain that double channels incur extra cost and the performance of the tree-based algorithm in a 2-D mesh is inferior to other path-based approaches.

Lin *et al.* propose three deadlock free path-based multicast algorithms for any network that contains a Hamilton Path, and discuss the performance of these algorithms in 2-D meshes [39] and in hypercubes [40]. They divide the network into high-channel and low-channel subnetworks. The dual-path algorithm sends messages along two paths; one containing the destinations with labels less than the source, the other containing the destinations that are greater than the source. The multi-path algorithm is similar except that it divides the network into multiple subnetworks and uses multiple paths to deliver messages. The fixed-path algorithm is similar to the dual-path algorithm except that it uses longer paths that are the same regardless of the destination set. They use traffic to evaluate the two algorithms. In [39], they show the performance of the algorithms for an 8×8 mesh. They conclude that the dual-path algorithm is more efficient in that the network latency is relatively stable for any number of destinations.

The work by Lin *et al.* was extended by Tseng *et al.* [51]. The algorithms in [40, 39] are based on Hamilton paths in the multicomputer network. If faults occur in the network, they may destroy the Hamilton paths. Consequently, they propose a trip-based, rather than path-based, model for multicast communication that does not rely on the existence of a Hamilton path. The trip-based model is similar to the path-based approach. A trip that contains the source and destinations is found (nodes can appear multiple times in the trip). Forward and backward virtual graphs are constructed from the trip, and messages are sent along the forward and backward graph to the appropriate destinations. The path-based algorithms are a special case in which the trip is a Hamilton path. Tseng *et al.* discuss the performance of their algorithm in hypercubes, and show its adaptability in the presence of faults.

Boppana *et al.* [9] propose a deadlock-free multicast algorithm for wormhole routed mesh networks. They assume that each node has only one pair of channels associated with each source and destination for sending and receiving messages. They define *compatibility* between unicast and multicast routing algorithms. A unicast and multicast algorithm are compatible if there are no deadlocks between the messages routed by them. In the system Boppana *et al.* describe, the dual-path and multi-path algorithms described above are not compatible with *e*-cube routing. They propose the column-path routing algorithm which is compatible with *e*-cube routing and compare it to dual-path and multi-path. They show that its traffic efficiency is comparable to that of the dual-path algorithm.

3.3 Different Communicaton Models

The work we have discussed so far has assumed that a message is delivered in a single step, or in "unit" time. While this is a fairly good model, other frameworks have been developed to more closely model communication on actual machines. Such models include the Postal model, the LogP model, and the Parameterized model.

In the Postal model, a message transmission requires λ units. Thus, a message sent at time t will arrive at its destination at time $t+\lambda-1$. In [10], a method is given for constructing optimal multicast trees under this model. The authors assume a logical complete-graph topology, *i.e.*, they assume that communication is uniform between any pair of nodes in the network. The LogP model is a generalization of the Postal model. In the LogP model, L is the latency (time to send a message), o is the overhead (time that a processor is involved in transmitting or receiving and cannot perform other operations), and g is the gap (time between successive message transmissions). Karp *et al.* develop optimal broadcast implementations under this model [33]. Although the LogP model is more general than the Postal model, it also has shortcomings in modeling communication on real machines. Park,

et al. introduce the Parameterized model to more closely represent actual communication. In this model, message transmission is comprised of three parameters: the sending latency (t_{send}), the receiving latency (t_{recv}), and the network latency (t_{net}). Unfortunately, each of these parameters is rather difficult and costly to measure. Thus, the easily measured parameters t_{end}, or end-to-end latency, and t_{hold}, holding time, are introduced. As in [10], Park, *et al.* assume a logical complete-graph topology. They describe methods for constructing optimal multicast trees for one-port and multiport architectures under the Parameterized model.

In each of the above models, communication time is considered to be uniform regardless of the message size or the lenth of the path being traversed by the message. Fraigniaud and Peters [23] describe a method for measuring communication time that relies on message size and path length. In particular, they model the time to send a message of length ℓ over a path of length d as $\alpha + d\delta + \ell\tau$, where α represents startup time, δ represents the switching delay, and τ is the propagation time. They propose a circuit-switched algorithm for a torus that requires time $d\lg(n)\alpha + d\frac{n}{2}\delta + d\lg(n)\ell\tau$ under this model of communication.

3.4 Multicast in Systems Not Based on Direct Networks

Multicast has also been studied for various parallel and distributed systems that are not based on a direct network interconnection structure.

Gaber and Mansour study broadcast in radio networks [1]. In this problem, if two stations neighboring a node transmit simultaneously, their messages will collide, and neither of the messages will be received at the node. Thus, a node can receive a message only if exactly one of its neighbors transmits during a time slot. Gaber and Mansour develop a broadcast algorithm that requires $\Omega(D)$ time slots for sufficiently large D, where D is the diameter of the graph.

Huang and McKinley study multicast communication in ATM networks used for parallel com-

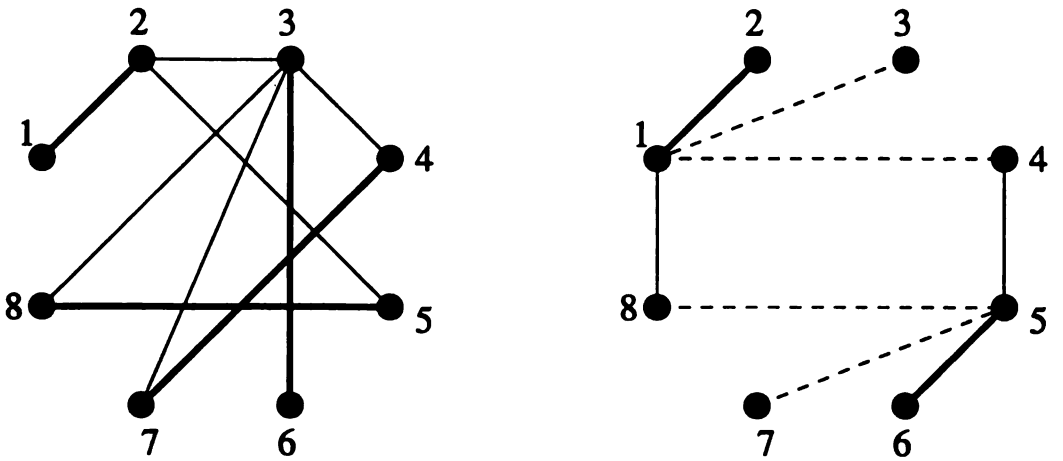
puting [29]. They show that multicast trees can provide a substantial benefit over a separate addressing technique, in which the source sends the message sequentially to each destination.

Multicast for systems based on indirect networks has also been addressed. Xu *et al.* describe a software technique for multicast in multistage cube networks that use turnaround routing [53]. In particular, they develop the $U\text{-}min$ algorithm for optimal multicast in such networks. The technique used is similar to the multicast algorithms developed for mesh and hypercube topologies in [46]. For a multicast request $M = (s, D)$, the destination addresses are put into a lexicographically-ordered chain. The destination in the "middle" of the chain is informed, and the chain is divided into two equal subchains. This process is repeated recursively for $\lceil \lg(|D| + 1) \rceil$ until all destinations are informed. Xu *et al.* implement their algorithm for an IBM SP-1 and show its superiority over other existing techniques such as Chameleon [27] and MPI-F [24] broadcast.

3.5 Matching Problems in Graphs

A *matching* in an undirected graph $G = (V, E)$ is a set $M \subseteq E(G)$ of pairwise independent edges (*i.e.,* no two edges share an endpoint). The endpoints of each edge $e \in M$ are said to be *matched.* We will refer to M as an *edge-matching* in G. For example, the **bold** edges in the graphs of Figure 3.5 are edge-matchings. Edge-matching has been used to model and solve many types of problems. For example, consider a graph in which vertices represent jobs and workers, and there is an edge between job j and worker w if worker w is able to perform job j. An edge-matching in such a graph represents jobs that can be done simultaneously by qualified workers. It is often desirable to find a *maximum* matching, where the largest possible number of vertices are matched. The matchings shown in Figure 3.5 are maximum matchings. In the example where jobs are matched to workers, a maximum matching represents the largest number of jobs that can be done simultaneously.

A maximum edge-matching in a graph may leave some vertices unmatched. If every vertex



(a) Perfect edge-matching

(b) No perfect edge-matching

Figure 3.5: Matchings in graphs.

in the graph can be matched, the matching is called a *perfect matching* (if there is an odd number of vertices then exactly one vertex is left unmatched.) The graph in Figure 3.5(a) has a perfect matching, but the graph in Figure 3.5(b) does not.

Wu and Manber introduce a generalization of an edge-matching called a *path-matching* [52]. In a path-matching, vertices of G are joined by simple paths rather than edges. Specifically, a path-matching M in G is a set of simple paths in which no two paths share the same end vertices. For path-matching, the definitions for *matched vertices* and *perfect matchings* are analogous to the definitions used in edge-matching, where an edge is replaced with a path. In Figure 3.5(b) the **bold edges** together with the paths shown by dashed lines represent a perfect path-matching in the graph. Various types of path-matchings can be defined by putting constraints on the paths contained in the matching M. For example, a *disjoint path-matching* (or DP-matching) is a path-matching in which paths in M are pairwise edge-disjoint. The path-matching in Figure 3.5(b) is a DP-matching.

Each of the matching problems described above has several commonly studied variations. The first variation is finding maximum (or perfect) matchings. As seen in Figure 3.5, perfect edge-matchings do not exist in all graphs, but several algorithms exist for finding maximum edge-

matchings in graphs [12]. Wu and Manber show that, unlike the case for edge-matchings, at least one perfect DP-matching exists in every connected graph, and they describe an algorithm for finding such a perfect DP-matching [52]. A second widely studied variation is further refinement of the maximum matching problem. In particular, several maximum matchings may exist in a graph, and the "best" of these matchings must be determined. This problem arises in weighted graphs, where a maximum matching that has minimum *cost* is desirable. There are two common ways to define the cost of a matching: as the sum of the weights of the edges in the matching or as the maximum weight edge in the matching (this is also called a *bottleneck* matching). Gabow and Tarjan provide a polynomial algorithm for finding a bottleneck edge-matching in a graph (*i.e.*, a maximum edge-matching whose maximum weight edge is as small as possible). Wu and Manber study both these measures of cost for path-matchings [52]. They define the *min-sum* and *min-max* variations of the DP-matching problem. In the min-sum problem, the objective is to minimize the sum of the weights of all edges used in the path-matching. In the min-max version, the objective is to minimize the maximal cost of a path in a matching, where the cost of a path is determined by the sum of the weights of its edges. Wu and Manber show that the min-max problem is NP-complete for general graphs. They also provide two polynomial algorithms for finding perfect min-max DP-matchings in trees. Although the bottleneck DP-matching problem is NP-complete, Datta and Sen show that it can be closely approximated. In particular, they give a 1-approximation algorithm for the bottleneck DP-matching problem [15].

3.6 Summary

Recall that in Chapter 1 we discussed various components of the multicast problem: network model, communication models, routing types and performance measures. The bulk of the work has been done using a graph model for the network. This necessarily implies that the network is symmetric,

i.e., if x can communicate directely with y then y can also communicate directly with x. Very little has been done in directed graphs, which model non-symmetric networks.

Both the neighbor-switching and line-switching communication models have been studied. Finding time efficient and traffic efficient multicast algorithms in systems that use neighbor-switching is NP-complete. However, neighbor-switching does not accurately model current routing techniques. Direct networks that use circuit routing techniques, such as wormhole routing, conform to the line-switching communication model. Because several paths may exist between a source and destination pair, many authors have considered free routing. Optimal multicast algorithms have been developed for arbitrary topologies that admit free routing. However, most commercial multicomputers use some type of oblivious routing. Optimal time UBM algorithms have been developed for hypercubes, meshes, and tori. However, none of the results for oblivious routing techniques are general, *i.e.*, they do not apply to any topology that uses an arbitrary oblivious routing scheme.

Finally, two metrics have been used to measure the performance of multicast algorithms: time and traffic. Very little of the previous work in multicast communication has addressed traffic efficiency. In particular, the problem of creating traffic efficient UBM algorithms has not been studied for arbitrary systems that admit free or oblivious routing.

Chapter 4

Source Limited Inclusive Routing

The first problem we address is finding *time* efficient line-switching UBM calling schedules in arbitrary topologies that use arbitrary oblivious routing schemes. In Section 4.1 we show that arbitrary oblivious routing schemes are difficult to model graphically and may exhibit poor performance. We discuss characteristics of oblivious routing schemes and describe a realistic class of oblivious routing schemes called *inclusive* routing schemes. In Section 4.1.3, we show that if we are given only *source limited* routing information that we can model these inclusive routing schemes as directed trees. In Section 4.2 we present several properties of multicast communication in directed tree topologies. In Section 4.2.3, we show that node-switching UBM algorithms provide a good approximation to line-switching algorithms in directed tree topologies, and in Section 4.2.5 we give a lower bound for broadcast in k-ary ditrees. The results presented in this chapter can be found in [2, 3, 4].

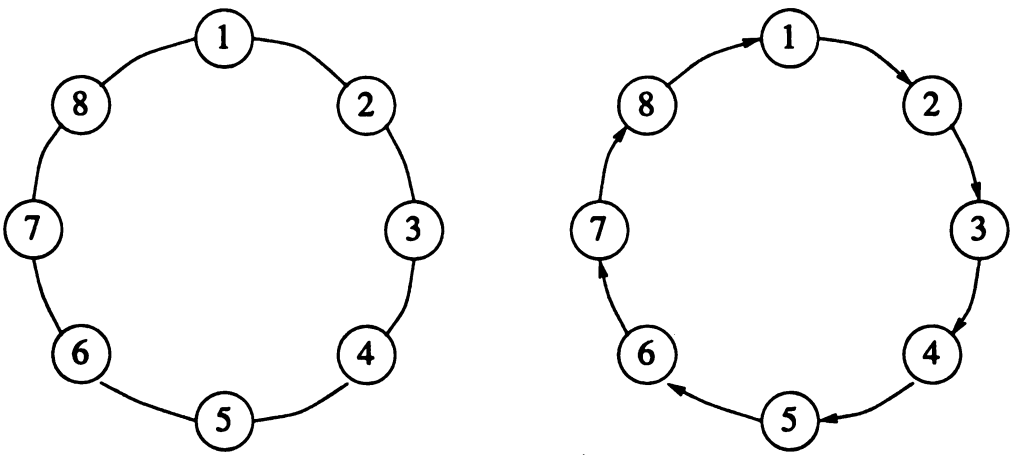
4.1 Oblivious Routing Models

In the previous work on multicast, authors modeled routing information as a graph. In order to develop correct multicast algorithms, the graph model should represent the routing information

accurately, *completely*, and *unambiguously*. Current graph models address free routing, however no model has been developed for oblivious routing.

4.1.1 Difficulties in Modeling General Oblivious Routing

Most of the previous work described in Chapter 3 was done using an undirected graph model. An undirected graph models a network in which communication is symmetric and free-routing is used. For example, consider the graph in Figure 4.1(a) that represents a bidirectional ring. The



(a) Bidirectional ring (b) Unidirectional ring

Figure 4.1: Graph and Digraph representations for rings

graph represents the routing information accurately, *i.e.*, there is no path in the graph that does not correspond to a physical connection in the network. The graph also includes complete routing information, in that there are no physical routes between nodes that are not represented in the graph. Finally, because free-routing is used, any path in the graph can be chosen to route a message. This is unambiguously shown in the graph.

Unfortunately, the graph model can not be used to represent networks that do not provide symmetric communication. Suppose that we have a unidirectional ring in which the unidirectional links are from node i to node $(i + 1) \bmod n$, where n is the number of nodes. The undirected graph in

Figure 4.1(a) does not accurately represent the routing information. For example, there is an edge from node 5 to node 4, but we can not deliver a message directly from 5 to 4 in the unidirectional ring. Thus, a different model must be used to capture the routing information. The directed graph in Figure 4.1(b) can be used to represent a unidirectional ring that supports free routing.

Both the undirected and directed graph models are used to represent systems that support free routing. If the system employs an oblivious routing technique, the situation changes. For example, suppose the non-symmetric direct network from Figure 2.1(b) uses the oblivious routing shown in Table 4.1. While the digraph model in Figure 2.5(a) is accurate and complete, it is not unambiguous.

Table 4.1: Oblivious Routing Table

Oblivious Routing Table					
source	destination	routing path	source	destination	routing path
v_1	v_2	v_1v_2	v_3	v_4	$v_3v_1v_4$
v_1	v_3	$v_1v_2v_5v_3$	v_3	v_5	$v_1v_4v_5$
v_1	v_4	v_1v_4	v_4	v_1	v_4v_1
v_1	v_5	$v_1v_2v_5$	v_4	v_2	$v_4v_5v_3v_2$
v_2	v_1	$v_2v_5v_3v_1$	v_4	v_3	$v_4v_5v_3$
v_2	v_3	$v_2v_5v_3$	v_4	v_5	v_4v_5
v_2	v_4	$v_2v_5v_4$	v_5	v_1	$v_5v_4v_1$
v_2	v_5	v_2v_5	v_5	v_2	$v_5v_3v_2$
v_3	v_1	v_3v_1	v_5	v_3	v_5v_3
v_3	v_2	v_3v_2	v_5	v_4	v_5v_4

There are three paths from node v_1 to node v_3 in the digraph ($v_1v_2v_3$, $v_1v_2v_5v_3$, and $v_1v_4v_5v_3$), but it is not clear that the path $v_1v_2v_5v_3$ must be used to send a message.

We have explored other graph models to solve the problem of ambiguity that exists in the digraph model. For example, one may use a multigraph[1], in which each edge is labeled with the (source,destination)-pair that uses it. Figure 4.2 shows the multigraph that corresponds to the oblivious routing in Table 4.1. Unfortunately, a multigraph model can be unmanageable for even small

[1] In a *multigraph* there can be multiple edges between a pair of vertices.

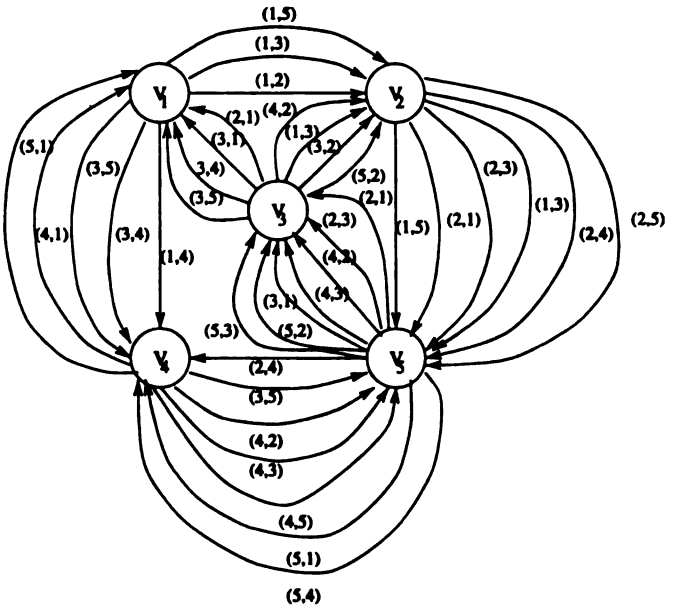


Figure 4.2: Multigraph model of oblivious routing

direct networks. Furthermore, multiple edges represent a single physical communication line. Thus, the notion of physically edge-disjoint paths is not present in the multigraph model.

We have discussed three graph models: undirected graphs, directed graphs, and multigraphs. None of these models is able to represent arbitrary oblivious routing schemes accurately, completely, and unambiguously. In addition to being difficult to model, arbitrary oblivious routing schemes may lead to extremely poor performance. For example, consider a ring network where the directed path used to transmit from any node v to any node w is the longer of the two paths in the ring from node v to node w. It is easy to see that performing a multicast in the ring with this oblivious routing scheme must take $\Omega(n)$ steps, where n is the number of destination nodes, as almost every pair of directed paths share an edge.

4.1.2 Inclusive Routing

In order to develop a realistic model for oblivious routing, we need to consider the defining characteristics of the routing and model the characteristics that best fit practical systems. One characteristic

to consider is the properties of the paths specified by the oblivious routing. In a general oblivious routing R, R contains exactly one (v_i, v_j) routing dipath for all i and j. A routing dipath may or may not provide additional information about other routing dipaths in the oblivious routing.

Suppose the oblivious routing path $p(i,j)$ uses intermediate node v_k, as shown in Figure 4.3. In an arbitrary oblivious routing scheme, the $p(k,j)$ routing path may or may not use any of the vertices or edges along $p(i,j)$. On the other hand, $p(k,j)$ may be precisely the subpath of $p(i,j)$

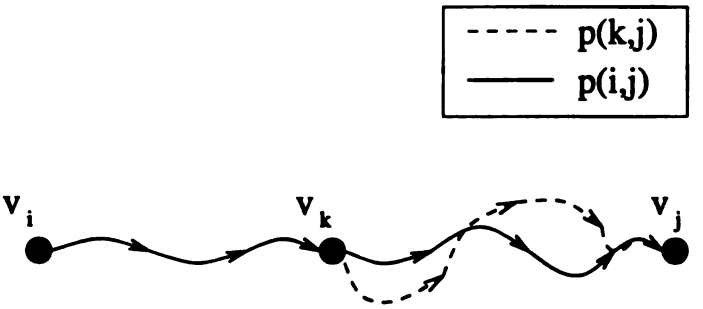


Figure 4.3: Oblivious routing paths

from v_k to v_j. Thus, one property of oblivious routing is whether or not a path gives any information about other oblivious routing paths. On one extreme, $p(i,j)$ gives no information about any other routing paths, *i.e.*, $p(i,j)$ is completely arbitrary. In the previous section we showed that arbitrary schemes are difficult to model and can exhibit poor performance.

Consequently, assuming that a dipath $p(i,j)$ provides no additional information about other routing paths is unrealistic. The other option is that $p(i,j)$ does give some information about other routing paths. We define *inclusive* routing schemes, a natural, realistic class of oblivious routing schemes. In an inclusive routing scheme, if the directed path used by node v to transmit to node w is $(v_0, v_1, \ldots, v_k)$ where $v_0 = v$ and $v_k = w$, then the directed path used by node v_i to transmit to node v_j for $0 \leq i < j \leq k$ is $(v_i, \ldots, v_j)$. Note that both xy-routing on a mesh and e-cube routing on a hypercube are specific examples of inclusive routing schemes.

4.1.3 Source Limited Routing Information

Another defining characteristic of a system that uses oblivious routing is the amount of routing information available to the router in each node. Nodes can store complete or partial routing information. Ideally, the router has complete routing information. However, in *massively parallel* systems, which have a large number of nodes, storing all routing information may be impractical. Thus, another option is that each router has only partial routing information. It is necessary for each node to contain the routing paths it uses to deliver messages to all other nodes in the system. Given this minimal amount of information, a node v_i knows how to route a message to an arbitrary node v_j, but v_i does not have information regarding the dipaths that v_j uses to send a message to it or any other arbitrary node. We will refer to routing tables of this form as *source limited* routing tables.

One clear disadvantage of storing only partial information is that helpful routing information is unavailable. For example, other informed nodes may be able to aid v_i in performing a multicast, but v_i cannot detect this using its limited routing information. However, source limited schemes can store enough information to give reasonably good performance. For example, Bharath-Kumar and Jaffe develop heuristics for NC routing, an NP-complete problem, using only source limited[2] information [8]. The heuristics performed well even though complete routing information was not used.

We restrict our attention to inclusive routing schemes because they seem to capture the class of oblivious routing schemes likely to be implemented in real networks. Furthermore, we concentrate on routing schemes that only have access to source limited routing information for three reasons: first, storing complete routing information may have unrealistic storage requirements; second, using only partial routing information has yielded good results in other multicast problems [8]; third, when used in conjunction with inclusive routing, there is a corresponding graph model that represents all

[2]Bharath-Kumar and Jaffe use the term local information.

the routing paths from the source to other nodes and some additional routing paths as well.

We use *directed trees* to model source limited inclusive routing. We use the definition of directed trees given in [18]. First, we define a *root* of a digraph. A digraph $G(V, E)$ is said to have a root r if $r \in V$, and there is a (r, v)-dipath for all $v \in V$. A digraph is called a *directed tree*, or ditree, if it has a root, and its underlying graph is a tree. We use a source's routing information to construct a directed tree that models the oblivious dipaths used in the inclusive routing scheme. Specifically, the ditree is rooted at the source node, and the dipaths in the directed tree correspond to the oblivious dipaths used by the source to send messages to each destination. For example, consider a direct network with eight nodes where the directed paths used by the source node r to communicate with the other seven nodes are shown in Table 4.2. We can represent all these dipaths using the directed tree in Figure 4.4. The problem of computing a minimum length multicast schedule in a network

Table 4.2: Source Limited Oblivious Routing Table

Routes from source node r	
destination	dipath
v_1	rv_1
v_2	rv_2
v_3	rv_3
v_4	rv_2v_4
v_5	rv_2v_5
v_6	rv_2v_6
v_7	$rv_2v_6v_7$

that employs an inclusive routing scheme given only source limited routing information is exactly the problem of computing a minimum length edge-disjoint multicast schedule in a directed tree T.

4.2 Properties of Multicast in Directed Trees

Because source limited inclusive routing schemes can be modeled as directed trees, we focus on the problem of performing multicast in directed tree topologies. We have not resolved the complexity of

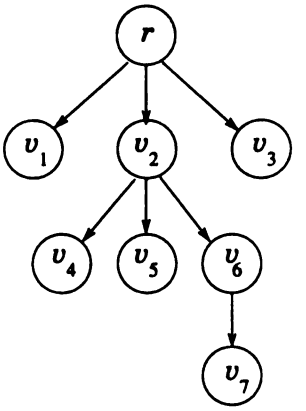


Figure 4.4: Ditree T with source node r

finding a minimum length *edge-disjoint* UBM schedule for ditrees, but in Section 4.2.3 we show that solving the related problem of computing a minimum length *node-disjoint* broadcast schedule in a directed tree T provides a good approximate solution to the edge-disjoint problem. In this section we present some properties that are useful for understanding and analyzing multicast communication in directed tree topologies.

4.2.1 Definitions

Definition 4.2.1 *A vertex v in a rooted ditree T with root r is at* **level** *i if and only if the length (in number of edges) of the dipath $p(r, v)$ in T is i.*

Definition 4.2.2 *The largest integer h for which there is a vertex at level h in a rooted ditree is called its* **height.** *Let $H(T)$ denote the height of ditree T.*

Definition 4.2.3 *Let T be a directed tree. If directed edge $vu \in E(T)$, then u is called a* **child** *of v, and v is called the* **parent** *of u. Let $children_T(v)$ denote the set $\{u | u$ is a child of v in $T\}$, and $parent_T(u)$ denote node u's parent in T. If there is a dipath from v to u in T, then u is called a* **descendant** *of v, and v is called an* **ancestor** *of u.*

Definition 4.2.4 *For any multicast algorithm A, let $\mathbf{A}(M, T)$ denote the multicast schedule produced by A for any multicast request $M = (r, D)$ in any directed tree T rooted at r, and let $|\mathbf{A}(M, T)|$ denote the length of the multicast schedule. When M is the broadcast problem, we only write $A(T)$ and $|A(T)|$. We say that A is a* **node-switching** *or* **line-switching** *algorithm if $A(M, T)$ is a legal node-switching or line-switching multicast schedule, respectively, for any multicast request M in any directed tree T.*

Definition 4.2.5 *Let* **OPT-ED**, **OPT-ND**, *and* **OPT-NB** *denote the set of multicast algorithms that always generate minimum length legal edge-disjoint, node-disjoint, and neighbor-switching multi-*

cast schedules, respectively. Let **ED**, **ND**, *and* **NB** *denote arbitrary elements of OPT-ED, OPT-ND, and OPT-NB, respectively.*

Definition 4.2.6 *Let $|T|$ denote the number of nodes in T.*

4.2.2 Decomposing Algorithms

An attractive feature of node-switching that makes analysis relatively easy is that each unicast decomposes the original multicast problem into two destination disjoint multicast subproblems. Consequently, node-switching algorithms with this property are called *decomposing* algorithms, formally defined below.

Definition 4.2.7 *Let T be a directed tree. For $v \in V(T)$, $\langle v \rangle$ denotes the subtree of T induced by v and all its descendants.*

Definition 4.2.8 *A node-switching algorithm A is a* **decomposing** *algorithm if it produces a broadcast schedule in which no node ever calls a descendant of a previously informed node. That is, the only informed node on any dipath used by any call in $A(T)$ for all ditrees T is the source of the call.*

To illustrate Definition 4.2.8, suppose we want to find a calling schedule C to implement multicast request $M = (r, D)$, where D is the destination set in directed tree T. Let the call $(r, v, 1)$ be the single unicast in the first step of C. After the call is made, we can decompose M into two subproblems: implementing multicast request $M' = (r, D - V(\langle v \rangle))$ in directed tree $T - \langle v \rangle$, and implementing multicast request $M'' = (v, D - V(T - \langle v \rangle))$ in ditree $\langle v \rangle$. The following result shows that we can restrict our attention to decomposing algorithms with no penalty.

Lemma 4.2.1 *There exists a decomposing algorithm A in OPT-ND.*

Proof: Let $A \in$ OPT-ND be a non-decomposing multicast algorithm for implementing multicast request M in ditree T, and let $(r, v, 1)$ be the first call in $A(M, T)$. We will construct a decomposing algorithm A' such that $A'(M, T)$ has the same length as $A(M, T)$ for any multicast request M in

ditree T. Note that all calls from nodes in $T - \langle v \rangle$ to nodes in $\langle v \rangle$ must pass through node v. Thus, only one such call can occur during any step t, and node v cannot be the source of a call during step t. We construct A' by replacing all calls of the form $(x, y, t) \in A(M, T)$, where $x \in V(T - \langle v \rangle)$ and $y \in V(\langle v \rangle)$ with the call (v, y, t). The same argument applies to the ditrees $\langle v \rangle$ and $T - \langle v \rangle$. Thus, we construct multicast schedule $A'(M, T)$ recursively as described above. The resulting multicast algorithm A' is a decomposing algorithm, and $A'(M, T)$ has the same length as $A(M, T)$. Thus, $A' \in$ OPT-ND. □

4.2.3 Relating Node-switching to Line-switching

We now address the relationship between OPT-ND and OPT-ED. Specifically, we show that a minimum length node-switching multicast schedule is at most twice as long as a minimum length line-switching schedule.

Lemma 4.2.2 *There exists a family of ditrees $\mathfrak{F}_T$ such that $|ND(T)| = (2 - \frac{2}{n})|ED(T)|$, where $T \in \mathfrak{F}_T$ and T has n nodes.*

Proof: Consider the multicast request $M = (v_1, \{v_2, v_3, \ldots, v_n\})$ (*i.e.*, broadcast) in the ditree T pictured in Figure 4.5. It is clear that under the node-switching model $|\mathrm{ND}(T)| = n - 1$. Under the line-switching model, v_1 can inform v_2, and then both v_1 and v_2 can make calls simultaneously until all destinations have been informed. Thus, $|\mathrm{ED}(T)| = 1 + \frac{n-2}{2} = \frac{n}{2}$. Comparing these gives us,

$$\frac{|\mathrm{ND}(T)|}{|\mathrm{ED}(T)|} = \frac{n-1}{\frac{n}{2}} = 2 - \frac{2}{n}.$$

□

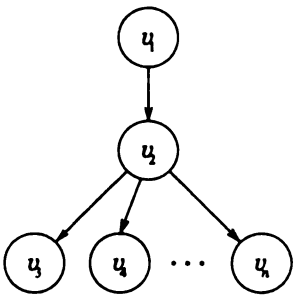


Figure 4.5: Node-switching vs. Line-switching

Lemma 4.2.3 *For all directed trees T and multicast requests M, $|ND(M,T)| \leq 2|ED(M,T)|$.*

Proof: We first state two facts about the directed paths used to perform unicasts in any time step of a legal line-switching calling schedule. First, for any uninformed node v, at most one unicast dipath will use that node. Since the tree is directed, a call through v must come through the directed edge $(parent_T(v), v)$. Only one call is permitted to use this edge. Second, for any informed node v, at most two unicast paths can use it. Again one call can use the edge $(parent_T(v), v)$. Since v is informed, it is the only other node that can use node v without using edge $(parent_T(v), v)$. Thus, if two calls use v, it must be the source of a message. Now consider all unicasts (s, d, t), *i.e.*, unicasts occurring in time step t, of a legal line-switching calling schedule. Let C_t be the set of all such unicasts. Because of the previous facts, the graph induced by the unicast dipaths in C_t is a *forest*, say F, of directed trees in which the maximum outdegree of any node is two. Furthermore, only leaves in F are destinations of unicast calls in step t. We will simulate the unicast calls of C_t using node-disjoint paths. For each leaf ℓ in F, make the nearest informed ancestor responsible for informing ℓ. Since any such informed node can be responsible for at most two leaves, we can simulate the unicasts of C_t in at most two steps using node-disjoint paths, and the lemma follows. □

From Lemma 4.2.3, we know that a proof that $|\mathrm{ND}(M,T)| \geq c$ immediately translates into a proof that $|\mathrm{ED}(M,T)| \geq \frac{c}{2}$. Similarly, if we find an algorithm A such that $|A(M,T)| \leq$

$c|\mathrm{ND}(M,T)|$ for any multicast request M on ditree T, then we know $|A(M,T)| \leq 2c|\mathrm{ED}(M,T)|$ for any multicast request M on ditree T. Therefore, we focus on lower bounding and approximating the minimum length of an optimal node-disjoint multicast schedule in the remainder of this paper, keeping in mind that we can directly use these results to approximate the minimum length of an edge-disjoint multicast schedule.

4.2.4 Reducing Multicast to Broadcast

Within the setting of node-switching algorithms, we now show that we can reduce any multicast request M in ditree T to an equivalent broadcast request in ditree $f(T)$ where f is defined below. Therefore, we are particularly interested in efficient implementations for broadcast requests in ditrees.

Definition 4.2.9 *For any ditree T and multicast request $M = (r, D)$, define ditree $f(T)$ as follows. First, let $f(T) = T$. Suppose there are i levels in T. Consider each node v in level i sequentially. If $v \notin D$, then let $children_{f(T)}(parent_T(v))$ be $children_{f(T)}(parent_T(v)) \cup children_T(v) - \{v\}$ and remove v from T. Continue this process for levels $i-1, \ldots, 1$.*

Figure 4.6 gives an example of the mapping f. The multicast request $M = (r, \{v_2, v_3, v_4, v_6, v_7, v_8, v_{10}\})$ in directed tree T is transformed to its corresponding broadcast request in ditree $f(T)$.

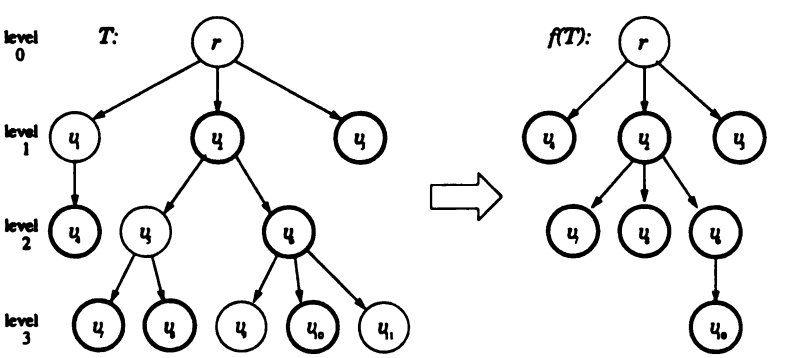


Figure 4.6: Mapping multicast to broadcast.

Lemma 4.2.4 *For any multicast request $M = (r, D)$ in any directed tree T, $|ND(f(T))| = |ND(M,T)|$.*

Proof: Let $p_T(x,y)$ denote the directed path $p(x,y)$ in a ditree T. It is clear from Definition 4.2.9 that for any node $x \in \{r\} \cup D$, $x \in p_{f(T)}(i,j)$ if and only if $x \in p_T(i,j)$; the mapping f simply eliminates nondestination nodes from $p_T(i,j)$ and leaves all destination nodes in the same order.

First, we show that if C is a legal node-switching calling schedule for implementing M in T, then C is also a legal node-switching calling schedule for implementing broadcast in $f(T)$. This implies that $|\text{ND}(M,T)| \geq |\text{ND}(f(T))|$.

Let C be a legal node-switching calling schedule for implementing M in T. Furthermore, let $(i,j,t) \in C(M,T)$ and $(k,l,t) \in C(M,T)$. Suppose that $p_{f(T)}(i,j)$ and $p_{f(T)}(k,l)$ intersect at node x. Then, $x \in \{r\} \cup D$. We know that since $x \in p_{f(T)}(i,j)$ and $x \in p_{f(T)}(k,l)$, x is also on both $p_T(i,j)$ and $p_T(k,l)$. Thus, $p_T(i,j)$ and $p_T(k,l)$ are not node-disjoint, contradicting the fact that C is a legal node-switching calling schedule. Thus, our assumption that $p_{f(T)}(i,j)$ and $p_{f(T)}(k,l)$ intersect is false, and C is a legal node-switching calling schedule for implementing broadcast in $f(T)$.

Next, we show that the converse is true. That is, if C is a legal node-switching calling schedule for implementing broadcast in $f(T)$, then C is also a legal node-switching calling schedule for implementing M in T. This implies that $|\text{ND}(f(T))| \geq |\text{ND}(M,T)|$.

Let C be a legal node-switching calling schedule for implementing broadcast in $f(T)$, and let (i,j,t) and (k,l,t) be two unicast calls that occur in $C(f(T))$. Suppose that $p_T(i,j)$ and $p_T(k,l)$ intersect. Let node x be the first vertex that $p_T(i,j)$ and $p_T(k,l)$ have in common. It must be the case that $x \notin \{r\} \cup D$, otherwise $p_{f(T)}(i,j)$ and $p_{f(T)}(k,l)$ intersect. Thus, there are two distinct directed edges vx (on $p_T(i,j)$) and ux (on $p_T(k,l)$). This results in a contradiction, because each node x in a ditree can have at most one parent. Thus, our assumption that $p_T(i,j)$ and $p_T(k,l)$

intersect is false, and C is a legal node-switching calling schedule for implementing M in T.

Since $|\mathrm{ND}(M,T)| \geq |\mathrm{ND}(f(T))|$ and $|\mathrm{ND}(f(T))| \geq |\mathrm{ND}(M,T)|$, we can conclude that $|\mathrm{ND}(f(T))| = |\mathrm{ND}(M,T)|$. □

4.2.5 Lower Bound on Broadcast in k-ary Ditrees

Recall from Chapter 3 that for undirected graphs (and therefore undirected trees) there is always a line-switching broadcast schedule of length $\lceil \lg n \rceil$, where n is the number of nodes in the graph. This is not always the case in directed trees. In this section, we prove that performing broadcast in any full k-ary directed tree T^k of order n requires $\Omega(k \log_k n)$ time steps under the line-switching model. We prove this lower bound by showing that $|\mathrm{NB}(T^k)| = k \log_k n$. We then show that $|\mathrm{NB}(T)| \leq |\mathrm{ND}(T)| + H(T) - 1$ for all ditrees T. Since $H(T^k) = \log_k n$, it follows that $|\mathrm{ND}(T^k)| = \Omega(k \log_k n)$. Combining this result with Lemma 4.2.3, we show that $|\mathrm{ED}(T^k)| = \Omega(k \log_k n)$.

First we give a lower bound on the length of an optimal neighbor-switching algorithm that implements broadcast.

Lemma 4.2.5 *$|NB(T^k)| = k\lfloor \log_k(n) \rfloor$, where T^k is a full k-ary ditree of order n.*

Proof: Let T^k be a full k-ary ditree of order n. Since each node in the ditree must inform all of its children sequentially, the last node in level i receives the broadcast message at time $k \cdot i$. Since the last node to receive the message is at level $\lfloor \log_k n \rfloor$, it follows that the time to complete broadcast in T^k is $k\lfloor \log_k n \rfloor$. □

Now we show the relationship between |NB| and |ND|.

Lemma 4.2.6 *For all directed trees T, $|NB(T)| \leq |ND(T)| + H(T) - 1$.*

Proof: We prove this by induction on $H(T)$.

The base case when $H(T) = 1$ is obvious.

Now consider the inductive case. Assume that the lemma holds for all directed trees T' where $H(T') \leq k$. Now consider an arbitrary directed tree T with $H(T) = k + 1$. Let the root of T have m children. Name each of the m subtrees rooted at the children of T as T_i for $1 \leq i \leq m$ such that

$$|\mathrm{ND}(T_1)| \geq |\mathrm{ND}(T_2)| \geq \cdots \geq |\mathrm{ND}(T_m)|.$$

Note that in each step of ND, the root of T can only inform a node in one of the m subtrees. Thus, ND cannot begin informing nodes in $m - 1$ of the subtrees until the second step. It cannot begin informing nodes in $m - 2$ of the subtrees until the third step, and so on. This implies that

$$|\mathrm{ND}(T)| \geq \max\{|\mathrm{ND}(T_1)|, |\mathrm{ND}(T_2)| + 1, \ldots, |\mathrm{ND}(T_i)| + i - 1, \ldots, |\mathrm{ND}(T_m)| + m - 1\}. \quad (4.1)$$

We define a neighbor-switching algorithm for broadcast as follows. The root of T informs the root of T_1 in time step 1, the root of T_2 in time step 2, etc. In general, the root of T informs the root of T_i in time step i. An optimal neighbor-switching algorithm, NB, clearly does at least as well as the algorithm we have defined, so

$$|\mathrm{NB}(T)| \leq \max\{|\mathrm{NB}(T_1)| + 1, |\mathrm{NB}(T_2)| + 2, \ldots, |\mathrm{NB}(T_i)| + i, \ldots, |\mathrm{NB}(T_m)| + m\}. \quad (4.2)$$

Let i be the index that maximizes $|\mathrm{NB}(T_i)| + i$ for $1 \leq i \leq m$. From Equations 4.2 and 4.1, we know that $|\mathrm{NB}(T)| \leq |\mathrm{NB}(T_i)| + i$ and $|\mathrm{ND}(T)| \geq |\mathrm{ND}(T_i)| + i - 1$, respectively. In addition, we know that $H(T_i) \leq H(T) - 1$ for all $1 \leq i \leq m$. We apply our inductive hypothesis and these facts to complete our proof as follows.

$$\begin{aligned} |\mathrm{NB}(T)| &\leq |\mathrm{NB}(T_i)| + i \\ &\leq (|\mathrm{ND}(T_i)| + H(T_i) - 1) + i \\ &\leq |\mathrm{ND}(T)| + H(T_i) \\ &\leq |\mathrm{ND}(T)| + H(T) - 1 \end{aligned}$$

□

We use the previous results to derive a lower bound for the length of an optimal line-switching algorithm in a directed tree T.

Theorem 4.2.1 $|ED(T^k)| \geq \frac{1+(k-1)\log_k n}{2} = \Omega(k \log_k n)$ *for any full k-ary directed tree* T^k *of order* n.

Proof: From Lemma 4.2.3, we know that

$$|\mathrm{ED}(T^k)| \geq \frac{1}{2}|\mathrm{ND}(T^k)|.$$

Using Lemma 4.2.6, we have

$$|\mathrm{ED}(T^k)| \geq \frac{1}{2}(|\mathrm{NB}(T^k)| - H(T^k) + 1).$$

Substituting from Lemma 4.2.5, we get

$$|\mathrm{ED}(T^k)| \geq \frac{1}{2}(k \log_k n - H(T^k) + 1).$$

Finally, since the height of a full k-ary tree is $\log_k n$, it follows that

$$|\mathrm{ED}(T^k)| \geq \frac{1}{2}(k \log_k n - \log_k n + 1) = \frac{1 + (k-1)\log_k n}{2}.$$

□

We now extend the lower bound from Theorem 4.2.1 to a lower bound for edge-disjoint broadcasting in general digraphs.

Corollary 4.2.1 *There exists a digraph* G *with* n *nodes and with maximum indegree and outdegree of* k *such that* $|ED(G)| = \Omega(k \log_k n)$.

We construct a digraph G as follows. Let T_1 and T_2 be full k-ary trees of order $\frac{n}{2}$, rooted at r_1 and r_2, respectively. Orient all edges of T_1 away from r_1, and orient all edges of T_2 toward r_2. We call r_1 the *source root* and r_2 the *sink root*. For each node v in T_1 with outdegree zero, make a directed edge vw, where w is the corresponding node with indegree zero in T_2. Finally, add directed edge r_2r_1. Figure 4.7 shows an example of digraph G when k is three. Suppose that our routing

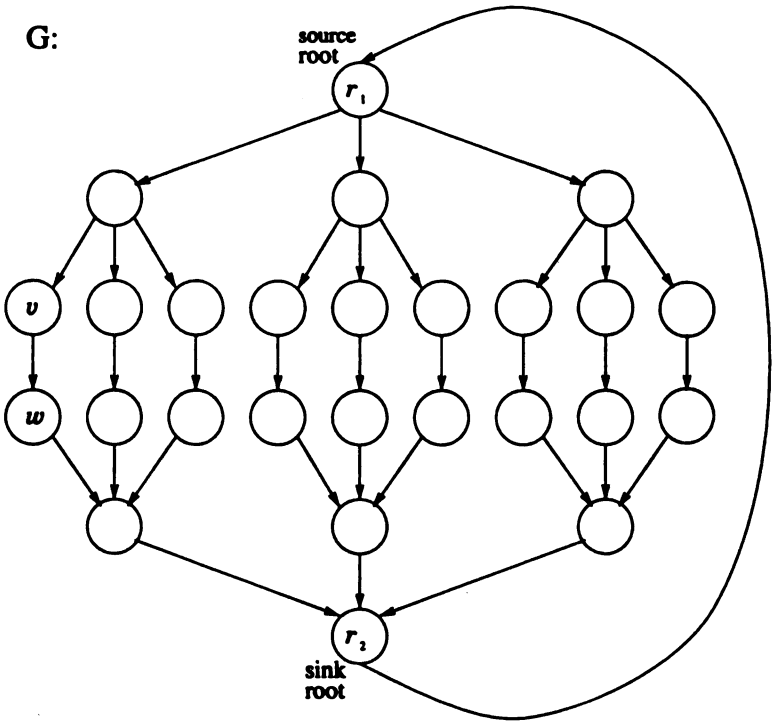


Figure 4.7: Broadcast requires $\Omega(k \log_k n)$.

scheme R includes all the dipaths in G. We then have a routing scheme in which each node has at least one dipath to any other node in the network. Thus, if we can lower bound the time to perform broadcast under the line-switching model in this network, we also lower bound the time to perform broadcast under the line-switching assumption in this network with some restricted routing scheme. The key observation is that at most *one* dipath in any time step can use edge r_2r_1. Thus, performing broadcast in this line-switched network from the source root, r_1, can be done no faster than twice as fast as performing broadcast in a full k-ary tree. The number of nodes in the network modeled by

G is twice the number of nodes in one of the full k-ary trees. Thus, the general lower bound result follows.

4.3 Summary

The line-switching model of communication represents networks that use circuit routing techniques such as wormhole routing. Although the multicast problem in such systems has been solved using a graph model, the graph model cannot accurately represent the oblivious routing information used in current multicomputer systems. We show that directed trees can be used to represent a realistic class of routing schemes called source limited inclusive routing schemes. We show that multicast requests in such systems can be transformed into equivalent broadcast requests with only a constant factor difference in the length of the calling schedule. We then address broadcast in directed trees, showing that $\mathrm{ED}(T) = \Omega(k \log_k n)$ for the complete, balanced k-ary directed tree T.

Chapter 5

Node-Switching Algorithms for Directed Trees

In Chapter 4, we described a realistic class of routing schemes, called source limited inclusive routing schemes, that can be modeled by directed trees. In addition, we showed that, in the context of node-switching UBM algorithms, a multicast request in a ditree T is equivalent to a broadcast request in a related ditree T'. Thus, we can limit our attention to performing broadcast in ditrees. In this chapter, we describe three node-switching UBM algorithms for broadcast in directed trees. In Sections 5.1 and 5.2 we describe two approximation algorithms for finding minimum length node-switching schedules for arbitrary ditrees. The Smart Centroid Algorithm, presented in Section 5.2, produces calling schedules that are at most twice the length of an optimal node-switching schedule. In Section 5.3, we present an optimal node-switching UBM algorithm for broadcast in directed trees.

5.1 Centroid Algorithm

In Section 2.5.4, we described the technique used for generating a minimum length line-switching calling schedule for a multicast request $M = (s, D)$ in a direct network that admits free routing. First, a trail containing the source and all destinations is constructed. Then, the trail is recursively broken into two trails of equal length during each time step. Breaking the problem in half at each step leads to $\lceil \lg(|D| + 1) \rceil$ steps to complete the multicast request. In this section, we present the Centroid Algorithm (CA) [3], which is built on a similar idea. At each step we attempt to break the directed tree representing the routing information into two *nearly equal* subtrees. We find an edge whose removal leaves two nearly equal sized ditrees. This edge will be adjacent to a *centroid* point of the underlying tree. We formally define the following terms that are used in the Centroid Algorithm.

5.1.1 Algorithm Description

Definition 5.1.1 *An* **end vertex** *is a vertex that has degree* 1.

Definition 5.1.2 *A* **branch** *at a vertex v of a tree T is a maximal subtree containing v as an end vertex.*

Definition 5.1.3 *The* **weight** *at a vertex v of T is the maximum number of edges in any branch at v.*

Definition 5.1.4 *A vertex c is a* **centroid point** *of a tree T if c has minimum weight, and the* **centroid** *of T consists of all such vertices.*

To create a node-disjoint broadcast schedule S for a ditree T, the assignment $S = CA(T, 1, \emptyset)$ must be made. That is, the initial parameter for CA are the entire ditree T rooted at r, time step 1, and an empty calling schedule. The CA algorithm recursively creates a node-disjoint broadcast schedule for T as follows:

Centroid Algorithm [CA] (T: ditree rooted at r; t: time step; S: calling schedule)

1. If $|V(T)| = 1$ return S
2. Find a centroid point c of T
3. Let $C_1, C_2, \ldots C_m$ be the components of $T - c$
4. Let $T_1 = C_i$, where $|C_i| \geq |C_j| \ \ \forall j \neq i$
5. If $r \in T_1$ then
   $S = S \cup \{(r, c, t)\}$
   else
   (a) Let c' be the root of T_1
   (b) $S = S \cup \{(r, c', t)\}$
6. Let $T_2 = T - T_1$
7. $S = \text{CA}(T_1, t+1, S) \cup \text{CA}(T_2, t+1, S)$

Suppose that T is a ditree with maximum outdegree k. CA is a simple recursive algorithm that works as follows. In each step, we find a centroid point c of the underlying tree of T. Removal of c divides T into m subtrees, where $0 \leq m \leq k+1$. We identify the largest one of these m subtrees, and call it T_1. If the root of T_1 is a child of c, then inform the root of T_1. If the root of T_1 equals the root of T, then inform c. Now divide the problem into the two subproblems T_1 and $T - T_1$, and continue.

Figure 5.1 shows the operation of CA on a broadcast request in an eight node ditree. The dark nodes are informed before the time step shown under each set of directed trees. First, the node to be

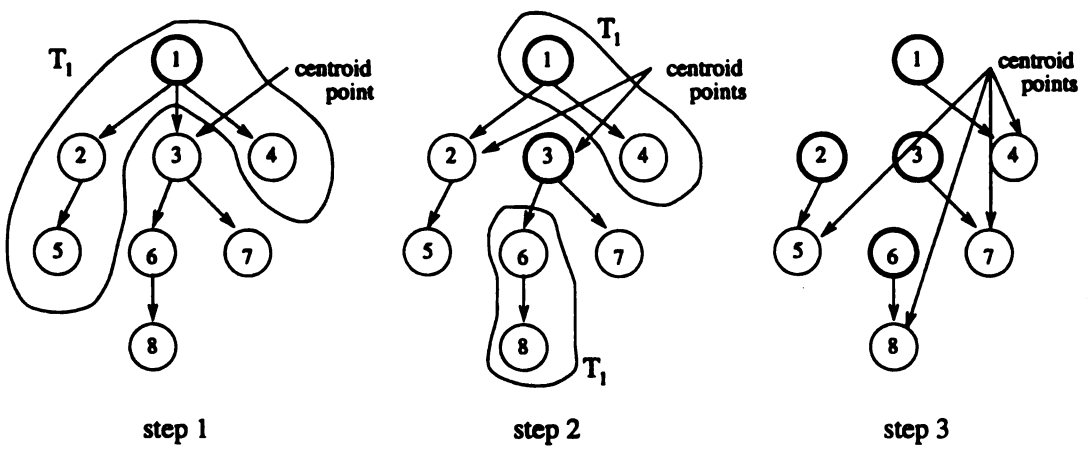


Figure 5.1: Centroid Algorithm at work.

called in the first step must be determined. Node 3 is chosen as a centroid point of T. Since, the root (node 1) is in T_1, the centroid point of T is informed in step 1, *i.e.*, the call (1,3,1) is made. The tree is then broken into two smaller trees, one rooted at node 1 and the other rooted at node 3, and the algorithm continues. For the ditree in Figure 5.1, CA produces an optimal calling node-disjoint schedule ($\lg(7+1) = 3$ time steps).

Theorem 5.1.1 $|CA(T)| = O(\log_{\frac{k+1}{k}} |T|)$ *for any directed tree T with maximum outdegree k.*

Proof: Let c be a centroid point of T's underlying tree. Removal of c leaves at most $k+1$ components. Let T' be the largest of the $k+1$ components. Because c is a centroid point, $|T'| \leq \frac{|T|}{2}$ [28, 17]. Because T' is the largest of the $k+1$ subtrees formed around c, $|T'| \geq \frac{|T|-1}{k+1}$. Thus, removal of T' from T gives, $|T - T'| \leq |T| - \frac{|T|-1}{k+1} = \frac{k}{k+1}|T| + \frac{1}{k+1}$. Thus, the maximum size of the problem decreases by a factor of at least $\frac{k}{k+1}$ in each time step, and the result follows. □

5.1.2 Time Complexity

Let $T_{CA}(n)$ be the worst case time complexity of the centroid algorithm on a ditree with n nodes. First, note that finding the centroid of a tree requires $O(n)$ time. Thus, we have the following recurrence relation for T_{CA}.

$$\begin{aligned} T_{CA}(n) &= \alpha_1 n + T_{CA}(kn) + T_{CA}(n - kn) + \alpha_2, \text{ where } 0 < k < 1 \\ T(1) &= \alpha_3 \end{aligned}$$

The CA algorithm is similar to the traditional QuickSort algorithm. In the worst case, the ditree T will be divided into two ditrees of size $n-1$ and 1 (*e.g.*, when T is a star). Thus, CA has worst case complexity of $O(n^2)$. However, when the ditree T has a uniform structure and can be divided

into nearly equal sized subtrees, the complexity is $O(n \lg n)$.

While the centroid algorithm does not produce optimal length broadcast schedules for all ditrees, its greatest advantages are its simplicity and speed for well-balanced ditrees. In the next section we show that the Centroid Algorithm produces optimal length broadcast schedules for some well known direct network topologies.

5.1.3 CA in Meshes and Hypercubes

Because the number of messages that are sent in each step of a multicast schedule for $M = (s, D)$ can at most double, the ideal number of time steps needed to complete a multicast is $\lceil \lg(|D| + 1) \rceil$. The centroid algorithm runs in $O(|D| \lg |D|)$ time and produces optimal length, *i.e.*, $\lceil \lg(|D| + 1) \rceil$, calling schedules for implementing broadcast in both mesh and hypercube topologies. Meshes use a specific type of inclusive routing called xy-routing. Under this scheme, a message is sent first in the x direction until it reaches the correct column. Then, the message is sent in the y direction until it reaches its destination. Because xy-routing is an inclusive routing scheme, we can represent source limited xy-routing information using a directed tree. Figure 5.2 shows the directed tree that

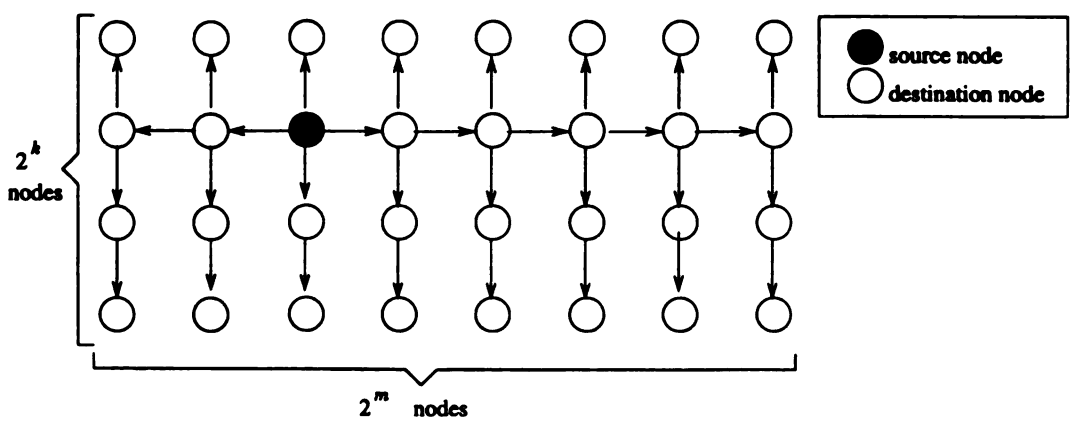


Figure 5.2: Directed tree representing $2^k \times 2^m$ mesh.

corresponds to the restricted routing of a 4 × 8 mesh when the node shown in black is the source of the broadcast. We show that the centroid algorithm produces an optimal line-switching calling

schedule for broadcasting in $2^k \times 2^m$ meshes.

Theorem 5.1.2 *CA produces optimal length node-disjoint broadcast schedules in* $2^k \times 2^m$ *meshes.*

Proof: Suppose we have a broadcast request from node $s = (row, col)$ in a $2^k \times 2^m$ mesh. We prove by induction on m that CA produces an optimal length broadcast schedule for a tree representing the restricted routing of a mesh.

The base case is $m = 0$, in which we have a $2^k \times 2^0$ mesh, *i.e.,* a path with 2^k nodes. Note that any centroid point of a path of length n has weight $\lceil \frac{n}{2} \rceil$. Thus, at each step CA will break the path into two paths with $\frac{2^k}{2}$ nodes. Consequently, CA will produce a broadcast schedule of length $\lg 2^k = k$, which is optimal.

Assume that CA produces optimal length broadcast schedules for all $2^k \times 2^i$ meshes, where $0 \leq i \leq m$. Now, we prove that CA produces an optimal calling schedule for a ditree representing the restricted routing for a $2^k \times 2^{m+1}$ mesh. Suppose that the source of the broadcast is at mesh coordinate (row, col). The centroid of the directed tree representing the mesh consists of the following two centroid points:

(1) $(row, \frac{2^{m+1}}{2})$

(2) $(row, \frac{2^{m+1}}{2} + 1)$

First, consider case (1). If $col > \frac{2^{m+1}}{2}$, then the centroid point $(row, \frac{2^{m+1}}{2})$ is called in the first step. If $col \leq \frac{2^{m+1}}{2}$, then $(row, \frac{2^{m+1}}{2} + 1)$, *i.e.,* the root of T_1, is called in the first step. Now consider case (2). If $col < \frac{2^{m+1}}{2} + 1$, then the centroid point $(row, \frac{2^{m+1}}{2} + 1)$ is called in the first step. Otherwise node $(row, \frac{2^{m+1}}{2})$ is called.

In either case, after the first call the original ditree is partitioned into two subtrees that represent meshes of size $2^k \times 2^m$. Thus, our inductive hypothesis holds for the two subtrees. Let $C(k, m)$ be the length of the broadcast schedule produced by CA for a $2^k \times 2^m$ mesh. Then,

$$\begin{aligned} C(k, m+1) &= C(k, m) + 1 \\ &= \lg(2^k \times 2^m) + 1 \\ &= \lg 2^k + \lg 2^m + \lg 2 \\ &= \lg 2^k + \lg 2^{m+1} \\ &= \lg(2^k \times 2^{m+1}) \end{aligned}$$

Thus, CA produces an optimal length edge-disjoint broadcast schedule for all $2^k \times 2^m$ meshes, for all $m \geq 0$.

□

Because the directed tree that represents the restricted routing in a $2^k \times 2^m$ torus is identical to a directed tree that represents a $2^k \times 2^m$ mesh, we have the following corollary to Theorem 5.1.2.

Corollary 5.1.1 *CA produces optimal length edge-disjoint broadcast schedules in $2^k \times 2^m$ tori.*

Hypercube topologies also use a specific kind of inclusive routing called *dimension-ordered* routing. In this scheme, the dimensions of the hypercube are ordered, and a message follows edges to its destination in increasing order of dimension. The restricted routing information in a hypercube with 2^k nodes can be represented by the directed *binomial tree* B_k, defined as follows.

Definition 5.1.5 *The binomial tree B_0 consists of a single node. The binomial tree B_k consists of two binomial trees B_{k-1} that are linked together such that the root of one is the leftmost child of the root of the other.*

Figure 5.3 shows the general form of a binomial tree. Figure 5.4 shows the directed tree that represents the restricted routing in a 16-node hypercube when broadcasting from node 0001. It is easy to verify that the ditree in Figure 5.4 is the directed binomial tree B_4. We show that CA creates an optimal calling schedule to implement a broadcast request from the root r in a directed binomial tree, *i.e.*, a hypercube.

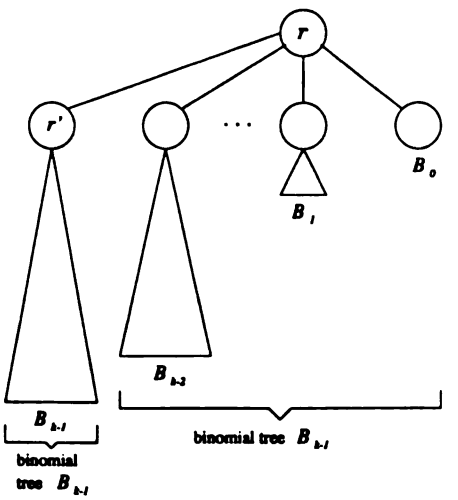


Figure 5.3: The binomial tree B_k.

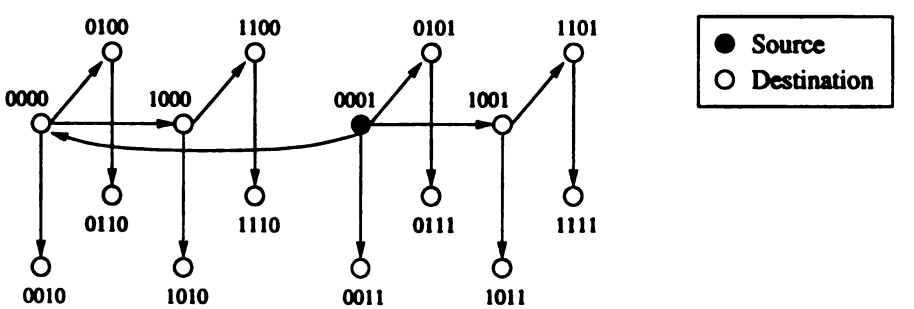


Figure 5.4: Ditree representation of a 16-node hypercube.

Theorem 5.1.3 *CA produces optimal length edge-disjoint broadcast schedules in k-dimensional hypercubes.*

Proof: We prove by induction on k that CA produces an optimal length broadcast schedule in the directed binomial tree B_k. The base case $k = 0$ is obvious.

Assume that when $k = n$, that CA produces an optimal length broadcast schedule in the directed binomial tree B_n, where $n \geq 0$. We must show that when $k = n + 1$, that CA produces an optimal length broadcast schedule in the directed binomial tree B_{n+1}. First, note that the centroid of B_{n+1} contains two nodes:

(1) the root of B_{n+1}

(2) the leftmost child of the root of B_{n_1}, *i.e.*, the root of a B_n subtree.

For example, in Figure 5.3, nodes r and r' are in the centroid. In either case, the first unicast call produced by CA is $(r, r', p(r, r'), 1)$. Then, the tree is divided into two B_n subtrees, one rooted at r and the other rooted at r'. We know from our induction hypothesis that CA produces optimal calling schedules for the B_n subtrees. Let $C(n)$ be the length of the broadcast schedule for B_n produced by CA. Then,

$$\begin{aligned} C(n+1) &= C(n) + 1 \\ &= \lg(2^n) + 1 \\ &= (n) + 1 \\ &= \lg(2^{n+1}) \end{aligned}$$

Thus, CA produces an optimal length edge-disjoint broadcast schedule for B_k, for all $k \geq 0$. □

5.2 Smart Centroid Algorithm

While the Centroid Algorithm works well on many trees, particularly those which are relatively uniform in structure or those which have small maximum outdegree, it can perform quite poorly on others. The problem results from the fact that the algorithm always chooses to inform the subtree with the maximum number of nodes first. In some cases, this is a poor choice, as another subtree with slightly fewer nodes may require more time to perform a broadcast. To illustrate this concept, consider the tree T in Figure 5.5, which has nk nodes. The root of T is its centroid point, and it has

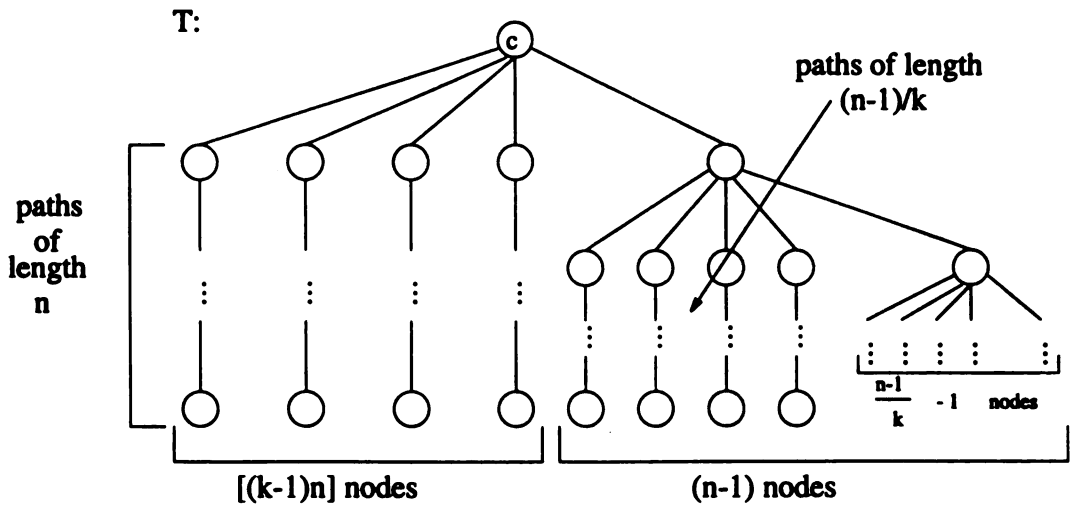


Figure 5.5: CA performs poorly.

k children. The first $k-1$ children are all roots of subtrees that are paths of length n. The kth child is the root of a tree with the same structure as T but has only $n-1$ nodes. The centroid algorithm will produce a calling schedule of length $\Omega(k \log_k |T|)$, whereas the optimal calling schedule will have length $O(k + \log_k |T|)$.

We present below a new algorithm which we call the Smart Centroid Algorithm (SCA) [3]. Similar to the CA algorithm, SCA is a recursive algorithm that creates a node-disjoint broadcast schedule S for a ditree T using the statement $S = SCA(T, 0, \emptyset)$. SCA is also based on finding a centroid of T, however SCA employs a better heuristic than the CA algorithm for determining

which child of the centroid point to inform first. Con sequently, we are able to guarantee that $|SCA(T)| \leq 4|\mathrm{ED}(T)|$ for all directed trees T.

5.2.1 Algorithm Description

We describe two algorithms: Smart Centroid Algorithm (SCA) and Smart Time (ST). ST returns the number of time steps necessary to complete broadcasting using SCA, *i.e.*, $|SCA(T)|$. SCA generates a calling schedule and uses ST to determine the order in which unicast calls are to be made. The Smart Centroid Algorithm works as follows. After identifying a centroid point c which has outdegree $m-1$, we decompose the problem into the subproblems $C_1, C_2, \ldots, C_m$ determined by the components of $T-c$. We compute $|SCA(C_i)|$ for each $i \leq m$. The root begins to inform these components in non-increasing order of calling schedule length. When the component containing the root of T is the next subtree to be informed, the root informs the centroid c. In subsequent steps, c is responsible for informing its remaining uninformed children, freeing the root to inform its own component.

Smart Time [ST] (T: ditree rooted at r)

1. If $|V(T)| = 1$ return 0
2. Find a centroid point c of T
3. Let $C_1, C_2, \ldots, C_m$ be the components of $T - c$
4. Let node c_i be the root of component C_i
5. For $i = 1$ to m do
   Let $max_i = ST$ on C_i, rooted at c_i
6. Reorder max_i, $1 \leq i \leq m$, such that $max_1 \geq max_2 \geq \cdots \geq max_m$
7. Return $\max_{1 \leq i \leq m}\{max_i + i\}$

Lemma 5.2.1 $|SCA(T)| \leq |ED(T)| + \lg |T|$ *for all directed trees* T.

Proof: We prove this by induction on $|T|$.

Smart Centroid Algorithm [SCA] (T: ditree rooted at r; t: time step; S: calling schedule)

1. If $|V(T)| = 1$ return S
2. Find a centroid point c of T
3. Let $C_1, C_2, \ldots, C_m$ be the components of $T - c$
4. Let node c_i be the root of component C_i
5. For $i = 1$ to m do
   Let $time_i = ST(C_i)$
6. Reorder the components such that $time_1 \geq time_2 \geq \cdots \geq time_m$
7. Let $i = 1$
8. While ($r \neq c_i$) do
9. Let $t = t + 1$
10. $S = S \cup \{(r, c_i, t)\}$
11. $S = SCA(C_i, t, S)$
12. Let $i = i + 1$
    End while
13. $S = S \cup \{(r, c, t + 1)\}$
14. $S = SCA(C_i, t + 1, S) \cup SCA(T - \cup_{j \leq i} C_j, t + 1, S)$

The base case when $|T| = 1$ is obvious.

Now consider the inductive case. Assume that the lemma holds for all directed trees T' where $|T'| \leq j$. We use the inductive hypothesis to show that the lemma holds for any directed tree T, where $j < |T| \leq 2j$. Consider an arbitrary directed tree T with $j < |T| \leq 2j$. Find a centroid point of this tree T. For the sake of simplicity, assume it is the root (the other case is handled in a nearly identical fashion). Let the root have m children. Break the tree into the m distinct subtrees rooted at the children of the root. Name these m subtrees so that $|SCA(T_1)| \geq |SCA(T_2)| \geq \cdots \geq |SCA(T_m)|$. Note $|T_i| \leq \frac{|T|}{2} \leq j$ for $1 \leq i \leq m$, so the induction hypothesis applies to each of these subtrees. Note also

$$|SCA(T)| = \max\{|SCA(T_1)| + 1, |SCA(T_2)| + 2, \cdots, |SCA(T_i)| + i, \cdots, |SCA(T_m)| + m\}.$$

Now let $\sigma(i)$ be a permutation on $\{1, \cdots, m\}$ such that $|\mathrm{ND}(T_{\sigma(1)})| \geq |\mathrm{ND}(T_{\sigma(2)})| \geq \cdots \geq |\mathrm{ND}(T_{\sigma(m)})|$. Note, in any time step of ND the root can only inform nodes in one of these m

subtrees. This implies that

$$|\mathrm{ND}(T)| \geq \max\{|\mathrm{ND}(T_{\sigma(1)})|, \ldots, |\mathrm{ND}(T_{\sigma(i)})| + i - 1, \ldots, |\mathrm{ND}(T_{\sigma(m)})| + m - 1\}.$$

To relate these two sets of values, we know from our induction hypothesis that $|\mathrm{ND}(T_1)| \geq |SCA(T_1)| - \lg j$, so $|\mathrm{ND}(T_{\sigma(1)})| \geq |SCA(T_1)| - \lg j$. Similarly, $|\mathrm{ND}(T_2)| \geq |SCA(T_2)| - \lg j$ and $|\mathrm{ND}(T_1)| \geq |SCA(T_1)| - \lg j \geq |SCA(T_2)| - \lg j$, so $|\mathrm{ND}(T_{\sigma(2)})| \geq |SCA(T_2)| - \lg j$. In general, $|\mathrm{ND}(T_{\sigma(i)})| \geq |SCA(T_i)| - \lg j$.

Now let $1 \leq i \leq m$ be the value such that $|SCA(T)| = |SCA(T_i)| + i$. Combining this with the above fact, we have

$$\begin{aligned} |\mathrm{ND}(T_{\sigma(i)})| &\geq |SCA(T_i)| - \lg j \\ |\mathrm{ND}(T_{\sigma(i)})| &\geq |SCA(T)| - i - \lg j \\ |\mathrm{ND}(T_{\sigma(i)})| + i + \lg j &\geq |SCA(T)| \end{aligned}$$

We know that $|\mathrm{ND}(T)| \geq |\mathrm{ND}(T_{\sigma(i)})| + i - 1$, thus

$$\begin{aligned} |SCA(T)| &\leq (|\mathrm{ND}(T_{\sigma(i)})| + i - 1) + 1 + \lg j \\ &\leq |\mathrm{ND}(T)| + 1 + \lg j \\ &\leq |\mathrm{ND}(T)| + \lg 2j \\ &\leq |\mathrm{ND}(T)| + \lg |T| \end{aligned}$$

and we are done.

□

Lemma 5.2.2 $|SCA(T)| \leq 2|ND(T)|$ *for all directed trees* T.

Proof: This follows from Lemma 5.2.1 and the fact that $|\mathrm{ND}(T)| \geq \lg |T|$. □

Theorem 5.2.1 $|SCA(T)| \leq 4|ED(T)|$ *for all directed trees* T.

Proof: This follows from Lemma 5.2.2 and Lemma 4.2.3. □

If the centroid point divides T into subtrees in which the largest tree requires the most time to perform broadcast, then the CA and SCA algorithms will produce the same broadcast schedule for T. For example, in the directed binomial tree B_k, there are two centroid points. In either case, B_k will be divided into two B_{k-1} directed trees. Thus, CA and SCA produce the same broadcast schedules for binomial trees (or equivalently for hypercube topologies). It is easy to see that CA and SCA also produce the same broadcast schedules for mesh topologies. This leads to the following corollaries of Theorem 5.1.2 and Theorem 5.1.3.

Corollary 5.2.1 *SCA produces optimal length edge-disjoint broadcast schedules in $2^k \times 2^m$ meshes (and tori).*

Corollary 5.2.2 *SCA produces optimal length edge-disjoint broadcast schedules in k-dimensional hypercubes.*

5.2.2 Time Complexity

Because SCA employs a better heuristic than CA we can guarantee that it will produce a broadcast schedule with length at most twice the optimal. However, this improvement over the Centroid Algorithm necessitates an increase in algorithm complexity. We show that SCA has worst case complexity of $O(n^3)$.

First, we must calculate the complexity of the Smart Time algorithm. Suppose removal of the centroid divides the tree into m components. Then we have the following recurrence relation for the time complexity of ST, denoted T_{ST}.

$$\begin{aligned} T_{ST}(n) &= \alpha_1 n + T_{ST}(a_1(n-1)) + \cdots + T_{ST}(a_m(n-1)) + c_2 m \lg m + \alpha_3 m + \alpha_4, \\ &\quad \text{where } \textstyle\sum_{i=1}^{m} a_1 = 1 \\ T_{ST}(1) &= \alpha_5 \end{aligned}$$

Similar to the CA algorithm, the worst case occurs when $m-1$ components are size 1, and one component is size $n-m$. Thus, the worst case complexity for ST is

$$T_{ST}(n) = \alpha_1 n + T(n-m) + \alpha_2 m \lg m + \alpha_3 m + \alpha_4.$$

Solving this recurrence gives

$$T_{ST}(n) = O(n^2).$$

We use this to calculate the worst case complexity, $T_{SCA}(n)$, of the Smart Centroid Algorithm. In step 5, ST is called m times. In the ideal case, each of the m components has size $\frac{n}{m}$, and the complexity of ST is $O(n)$. Since ST is called $O(n)$ times, step 5 has complexity $O(n^2)$. The worst case for the ST algorithm is when $m-1$ components have size 1 and the remaining component has size $n-1-m$. In this case the complexity of step 5 is $O(n^2)+(m-1)O(1)$. Since $m \leq n-1$, the complexity of step 5 is $O(n^2)$ in all cases. We have the following recurrence relation for the worst case complexity of SCA.

$$\begin{aligned} T_{SCA}(n) &= \alpha_1 n^2 + \alpha_2 n \lg n + \alpha_3 n + T_{SCA}(n-1) + \alpha_4 \\ T_{SCA}(1) &= \alpha_5 \end{aligned}$$

Solving this recurrence gives,

$$T_{SCA}(n) = O(n^3).$$

In the best case,

$$\begin{aligned} T_{SCA}(n) &= \alpha_1 n^2 + \alpha_2 n \lg n + \alpha_3 n + 2T_{SCA}(\frac{n}{2}) + \alpha_4 \\ &= O(n^2) \end{aligned}$$

While *SCA*'s complexity is not markedly worse than that of the CA algorithm, direct networks with uniform directed tree representations (such as meshes, hypercubes, and tori) can save time using the CA algorithm to calculate edge-disjoint broadcast schedules. For general topologies that use inclusive routing, *SCA* will provide a better approximation.

5.3 Optimal Decomposing Algorithm

Neither the CA algorithm nor the SCA algorithm is guaranteed to produce a minimum length broadcast schedule for every directed tree. In this section we present a $O(n^3)$ algorithm that always produces minimum length broadcast schedules [4].

5.3.1 Definitions

Recall from Lemma 4.2.1 that there is an optimal node-disjoint multicast algorithm that is a decomposing algorithm. It should be clear that of the decomposing algorithms, we can focus on those algorithms in which the root is not idle until all its children are informed. We now define some terms that we will use to both describe our optimal decomposing algorithm and to prove its optimality. We will illustrate these definitions using the following two minimum length broadcast schedules for the ditree of Figure 4.4. Note that these are the only minimum length broadcast schedules for T.

$$C^1(T) = \{(r, v_2, 1), (r, v_1, 2), (v_2, v_6, 2), (r, v_3, 3), (v_2, v_4, 3), (v_6, v_7, 3), (v_2, v_5, 4)\}$$

$$C^2(T) = \{(r, v_6, 1), (r, v_2, 2), (v_6, v_7, 2), (r, v_1, 3). (v_2, v_4, 3), (r, v_3, 4), (v_2, v_5, 4)\}$$

Definition 5.3.1 *Let $C(T)$ be a node-disjoint broadcast schedule for ditree T. We define the* **root schedule** *$C_r(T)$ of $C(T)$ to be $\{(v, w, t) \in C(T) \mid v = r\}$,* i.e., *the set of calls made by the root. For $1 \leq i \leq |C_r(T)|$, we define $v_i(C_r(T)) = w$ where $(r, w, i) \in C_r(T)$,* i.e., *the i^{th} node informed directly by the root. For $0 \leq i \leq |C_r(T)|$, we define $T_i = T - \bigcup_{j=1}^{i} (v_j(C_r(T)))$,* i.e., *the subtree of T that the root still has responsibility to inform after the ith time step of $C(T)$.*

$$C_r^1(T) = \{(r, v_2, 1), (r, v_1, 2), (r, v_3, 3)\}$$

$$C_r^2(T) = \{(r, v_6, 1), (r, v_2, 2), (r, v_1, 3), (r, v_3, 4)\}$$

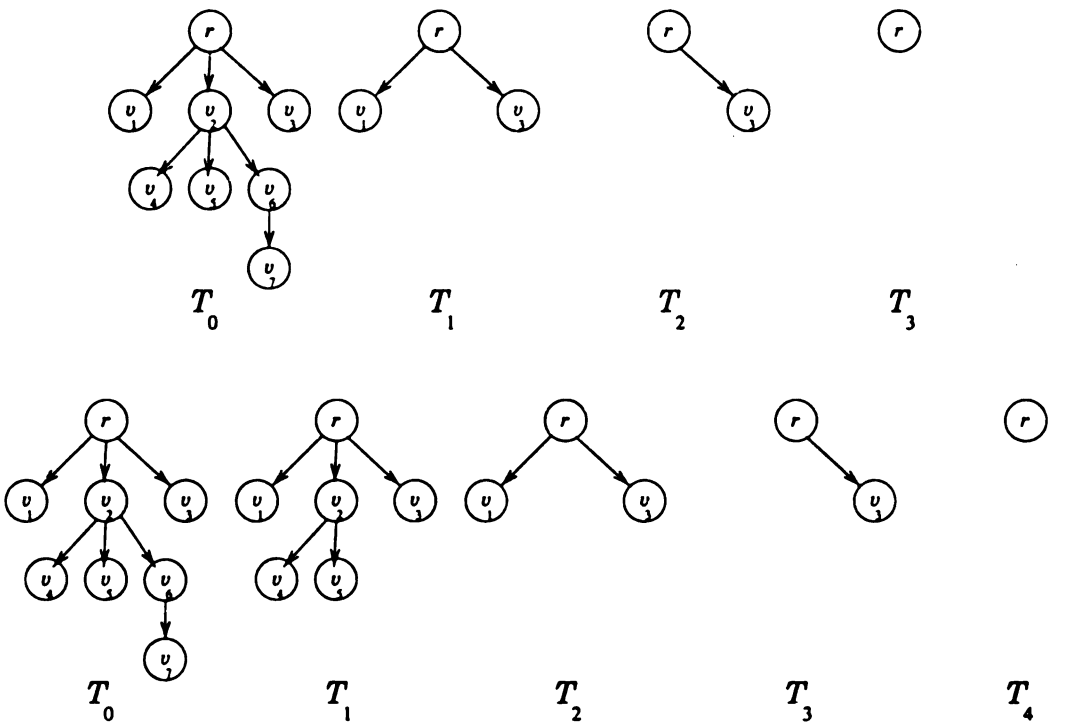


Figure 5.6: T_i's for broadcast schedules $C^1(T)$ and $C^2(T)$.

Definition 5.3.2 *Suppose $C(T_i)$ is a broadcast schedule for ditree T_i with root r_i such that $(r_i, v, 1) \in C(T_i)$. We define $C(T_{i+1})$, where $T_{i+1} = T_i - \langle v \rangle$, to be the broadcast schedule $C(T_i)$ modified as follows. All calls that involve nodes in $\langle v \rangle$ are deleted. All remaining calls have their time steps decremented by 1.*

Since $T_0 = T$,

$$C^1(T_1) = \{(r, v_1, 1), (r, v_3, 2)\}.$$

$$C^2(T_1) = \{(r, v_2, 1), (r, v_1, 2), (v_2, v_4, 2), (r, v_3, 3), (v_2, v_5, 3)\}$$

Definition 5.3.3 *Let $n = |C_r(T)|$. The* **labor vector** *of broadcast schedule $C(T)$ in ditree T, denoted $\vec{V}_C(T)$, is the vector $\vec{V}_C(T) = (|C(T_0)|, |C(T_1)|, \ldots, |C(T_n)|)$. Let $(\vec{V}_C(T))_i$ be the ith element of $\vec{V}_C(T)$,* i.e., *$(\vec{V}_C(T))_i = |C(T_{i-1})|$. Note that $(\vec{V}_C(T))_1 = |C(T_0)| = |C(T)|$ and that $(\vec{V}_C(T))_{n+1} = |C(T_n)| = 0$.*

$$\vec{V}_{C^1(T)} = (4,2,1,0)$$

$$\vec{V}_{C^2(T)} = (4,3,2,1,0)$$

Definition 5.3.4 *A broadcast schedule $C(T)$ is a* **least labor** *broadcast schedule if $\vec{V}_C(T)$ is lexicographically minimal over all broadcast schedules for T. The corresponding vector $\vec{V}_C(T)$ is called a least labor vector. In general, let $LLV(T)$ denote the least labor vector of T.*

Definition 5.3.5 *Let $MLS(T)$ be the set of all minimum length broadcast schedules for ditree T, and let $LLS(T)$ be the set of all least labor broadcast schedules for ditree T.*

It is not hard to verify that both $C^1(T)$ and $C^2(T)$ are in $MLS(T)$ and that $C^1(T) \in LLS(T)$ while $C^2(T) \notin LLS(T)$.

Definition 5.3.6 *We define the set of minimum first calls (MFC) of ditree $T = \langle r \rangle$ to be*

$$MFC(T) = \bigcup_{C \in MLS(T)} \{(r, v_i, 1) \in C\}.$$

Definition 5.3.7 *We define the set of least labor first calls (LLFC) of ditree $T = \langle r \rangle$ to be*

$$LLFC(T) = \bigcup_{C \in LLS(T)} \{(r, v_i, 1) \in C\}.$$

In our example,

$$MFC(T) = \{(r, v_2, 1), (r, v_6, 1)\}$$

$$LLFC(T) = \{(r, v_2, 1)\}$$

We now describe a decomposing algorithm ODA which is in OPT-ND. The algorithm ODA will work on a tree T in two phases, one bottom up and one top down. Let v be a node in T with p children d_i for $1 \leq i \leq p$. In the bottom up phase, ODA computes the root schedule $S_r(\langle v \rangle)$ and the least labor vector $\vec{V}_S(\langle v \rangle)$ of a least labor broadcast schedule $S(\langle v \rangle)$ for ditree $\langle v \rangle$ using the root schedules and the least labor vectors of least labor broadcast schedules $S(\langle d_i \rangle)$ previously computed for ditrees $\langle d_i \rangle$ for $1 \leq i \leq p$. In the top down phase, ODA pieces together the various

root schedules and least labor vectors to compute a least labor broadcast schedule $ODA(T)$ for the entire tree T.

5.3.2 The Optimal Substructure

The heart of the bottom up phase of ODA lies in determining which node of a directed tree $T = \langle r \rangle$ should be informed in the first time step. The following two lemmas are used to determine which node to call in the first time step.

Lemma 5.3.1 *Let T and T' be ditrees rooted at nodes r and r', respectively. Let each of r and r' have p children $d_1, d_2, \ldots, d_p$ and $d'_1, d'_2, \ldots, d'_p$, respectively, with the following properties:*

(i) The subtrees $\langle d_i \rangle$ and $\langle d'_i \rangle$ are isomorphic for all $i < p$

(ii) $LLV(\langle d_p \rangle) \leq LLV(\langle d'_p \rangle)$.

Then, $LLV(T) \leq LLV(T')$.

Proof: Let $C \in LLS(T')$. The basic structure of our proof is as follows. Let $v' \in LLFC(T')$. We show that there exists a schedule F for T with the property that $\vec{V}_F(T) \leq \vec{V}_C(T')$. In order to do this, we will choose a node $v \in V(T)$ such that

$$|F(T)| = \max\{|\mathrm{ND}(\langle v \rangle)|, |\mathrm{ND}(T - \langle v \rangle)|\} + 1 \leq \max\{|\mathrm{ND}(\langle v' \rangle)|, |\mathrm{ND}(T' - \langle v' \rangle)|\} + 1 = |C(T')|,$$

and

$$LLV(T - \langle v \rangle) \leq LLV(T' - \langle v' \rangle).$$

Since $C \in LLS(T')$, this proves that $LLV(T) \leq LLV(T')$. We prove the result by induction on $n = |C_{r'}(T')|$, the length of the least labor root schedule for T'.

The base case is when $n = 1$, *i.e.*, r' makes exactly one call. In this case, it is obvious $p = 1$ which means r' and r each have only one child. Furthermore, it is clear that $v' = d'_1$. We choose

$v = d_1$. In this case, from property (ii), it is easy to see that $|F(T)| = |\mathrm{ND}(\langle d_1 \rangle)| + 1 \leq |\mathrm{ND}(\langle d'_1 \rangle)| + 1 = |C(T')|$ and that $LLV(T - \langle d_1 \rangle) = LLV(T' - \langle d'_1 \rangle) = (0)$. Thus, $LLV(T) \leq LLV(T')$.

Our inductive hypothesis is that for all $n \leq k$, if properties (i) and (ii) hold, then $LLV(T) \leq LLV(T')$. We now prove the inductive step which is that if $|C_{r'}(T')| = k + 1$ and properties (i) and (ii) hold, then $LLV(T) \leq LLV(T')$. Note that $|C_{r'}(T' - \langle v' \rangle)| = |C_{r'}(T')| - 1 = k$, so the induction hypothesis will apply to ditrees $T' - \langle v' \rangle$ and $T - \langle v \rangle$ if conditions (i) and (ii) hold.

We consider several cases depending on v' and the characteristics of T' and T.

(1) Suppose $v' \in V(\langle d'_i \rangle)$, where $i < p$. We choose $v = v'$, which means $|\mathrm{NB}(\langle v \rangle)| = |\mathrm{ND}(\langle v' \rangle)|$. Therefore, showing $LLV(T - \langle v \rangle) \leq LLV(T' - \langle v' \rangle)$ also shows $|F(T)| \leq |C(T')|$. Because isomorphic subtrees are removed from T and T', properties (i) and (ii) still hold for $T' - \langle v' \rangle$ and $T - \langle v \rangle$. Therefore, by the induction hypothesis, $LLV(T - \langle v \rangle) \leq LLV(T' - \langle v' \rangle)$.

(2) Suppose $v' \in V(\langle d'_p \rangle)$. We consider two possibilities.

(i) Suppose $|\mathrm{ND}(\langle d_p \rangle)| = |\mathrm{ND}(\langle d'_p \rangle)|$. We again divide this into two cases.

(a) Suppose v' has the property $|\mathrm{ND}(\langle v' \rangle)| = |\mathrm{ND}(\langle d'_p \rangle)|$. Then choose $v = d_p$, *i.e.*, we make the call $(r, d_p, 1)$ in T. This gives us $|\mathrm{ND}(\langle v \rangle)| = |\mathrm{ND}(\langle d_p \rangle)| = |\mathrm{ND}(\langle d'_p \rangle)| = |\mathrm{ND}(\langle v' \rangle)|$. Thus, as in case (1), we merely need to show that $LLV(T - \langle v \rangle) \leq LLV(T' - \langle v' \rangle)$. Since $T - \langle v \rangle$ is a subtree of $T' - \langle v' \rangle$, $LLV(T_1) \leq LLV(T'_1)$.

(b) Suppose now that v' has the property that $|\mathrm{ND}(\langle v' \rangle)| < |\mathrm{ND}(\langle d'_p \rangle)|$. Since $|C(T')| \geq |\mathrm{ND}(\langle d'_p \rangle)|$, this implies $|C(T')| = \max\{|\mathrm{ND}(\langle d'_p \rangle)|, |\mathrm{ND}(T' - \langle v' \rangle)| + 1\}$. Choose $v \in LLFC(\langle d_p \rangle)$ which means $|\mathrm{ND}(\langle v \rangle)| < |\mathrm{ND}(\langle d_p \rangle)|$. Since

$|F(T)| \geq |\text{ND}(\langle d_p \rangle)|$, this implies $|F(T)| = \max\{|\text{ND}(\langle d_p \rangle)|, |\text{ND}(T - \langle v \rangle)| + 1\}$.

Since we have assumed that $|\text{ND}(\langle d_p \rangle)| = |\text{ND}(\langle d'_p \rangle)|$, we are assured that $|F(T)| \leq |C(T')|$ if $LLV(T - \langle v \rangle) \leq LLV(T' - \langle v' \rangle)$.

Since $v \in \langle d_p \rangle$ and $v' \in \langle d'_p \rangle$, condition (i) is still true of $T - \langle v \rangle$ and $T' - \langle v' \rangle$. Let $w \in LLFC(\langle d'_p \rangle)$, then $LLV(\langle d'_p \rangle - \langle w \rangle) \leq LLV(\langle d'_p \rangle - \langle v' \rangle)$. Because $v \in LLFC(\langle d_p \rangle)$ and condition (ii), $LLV(\langle d_p \rangle - \langle v \rangle) \leq LLV(\langle d'_p \rangle - \langle w \rangle)$. Therefore, $LLV(\langle d_p \rangle - \langle v \rangle) \leq LLV(\langle d'_p \rangle - \langle v' \rangle)$ and condition (ii) holds for $T - \langle v \rangle$ and $T' - \langle v' \rangle$. Therefore, by the induction hypothesis, $LLV(T - \langle v \rangle) \leq LLV(T' - \langle v' \rangle)$.

(ii) Suppose $|\text{ND}(\langle d_p \rangle)| < |\text{ND}(\langle d'_p \rangle)|$. We choose $v = d_p$, *i.e.*, we make the call $(r, d_p, 1)$ in T. Since $T - \langle v \rangle$ is a subtree of $T' - \langle v' \rangle$, $LLV(T - \langle v \rangle) \leq LLV(T' - \langle v' \rangle)$. We now must insure that $|F(T)| \leq |C(T')|$.

Since $LLV(T - \langle v \rangle) \leq LLV(T' - \langle v' \rangle)$, we know $|\text{ND}(T - \langle v \rangle)| \leq |\text{ND}(T' - \langle v' \rangle)|$. Therefore, we only need to show that $|\text{ND}(\langle v \rangle)| + 1 \leq |C(T')|$. By our assumption, $|\text{ND}(\langle v \rangle)| < |\text{ND}(\langle d'_p \rangle)|$. This means that $|\text{ND}(\langle v \rangle| + 1 \leq |\text{ND}(\langle d'_p \rangle)|$. Clearly, $|\text{ND}(\langle d'_p \rangle)| \leq |C(T')|$. Therefore, we conclude $|F(T)| \leq |C(T')|$ and the lemma is proven.

□

Lemma 5.3.2 *Consider any ditree T rooted at r. Let r have p children d_i, $1 \leq i \leq p$, such that $LLV(\langle d_1 \rangle) \geq \cdots \geq LLV(\langle d_i \rangle) \geq \cdots \geq LLV(\langle d_p \rangle)$, and let $(d_1, v, 1) \in LLFC(\langle d_1 \rangle)$, then*

(i) If $|ND(T)| = |ND(\langle d_1 \rangle)|$, then $(r, v, 1) \in LLFC(T)$.

(ii) If $|ND(T)| > |ND(\langle d_1 \rangle)|$, then $(r, d_1, 1) \in LLFC(T)$.

Proof: There are two cases to consider. First, let $|\text{ND}(T)| = |\text{ND}(\langle d_1 \rangle)|$. Clearly, we must call a node w in $\langle d_1 \rangle$ with the property that $|\text{ND}(\langle w \rangle)| \leq |\text{ND}(\langle d_1 \rangle)| - 1$; otherwise we cannot inform all

nodes in $\langle d_1 \rangle$ in $|\mathrm{ND}(\langle d_1 \rangle)|$ time steps. For all such w, $LLV(\langle d_1 \rangle - \langle v \rangle) \leq LLV(\langle d_1 \rangle - \langle w \rangle)$ by the definition of a LLFC. Thus, by Lemma 5.3.1 $LLV(T - \langle v \rangle) \leq LLV(T - \langle w \rangle)$, and $(r, v, 1) \in LLFC(T)$.

In the second case, $|\mathrm{ND}(T)| > |\mathrm{ND}(\langle d_1 \rangle)|$. Let C be a calling schedule that calls d_1 in the first step and then follows a least labor schedule for the remaining subtrees, and let F be a calling schedule that calls some $w \neq d_1$ in the first step and then follows a least labor calling schedule for the remaining subtrees. We will show that $\vec{V}_C(T) \leq \vec{V}_F(T)$, and consequently that $(r, d_1, 1) \in LLFC(T)$.

Let $T_1 = T - \langle d_1 \rangle$ and $T_1' = T - \langle w \rangle$. It follows that,

$$|C(T)| = \max\{|\mathrm{ND}(T_1)|, |\mathrm{ND}(\langle d_1 \rangle)|\} + 1 \quad \text{and} \quad |F(T)| = \max\{|\mathrm{ND}(T_1')|, |\mathrm{ND}(\langle w \rangle)|\} + 1.$$

Then,

$$(\vec{V}_C(T))_i = \begin{cases} |C(T)| & \text{if i=1} \\ (LLV(T_1))_{i-1} & \text{otherwise} \end{cases} \quad \text{and} \quad (\vec{V}_F(T))_i = \begin{cases} |F(T)| & \text{if i=1} \\ (LLV(T_1'))_{i-1} & \text{otherwise.} \end{cases}$$

Thus, we need to show $|C(T)| \leq |F(T)|$ and $LLV(T_1) \leq LLV(T_1')$.

We will first show that $|C(T)| = |\mathrm{ND}(T_1)| + 1$ and $|F(T)| = |\mathrm{ND}(T_1')| + 1$. If we can prove this, we need only show that $LLV(T_1) \leq LLV(T_1')$ to show that $\vec{V}_C(T) \leq \vec{V}_F(T)$. After calling d_1, we know that $|\mathrm{ND}(T_1)| \geq |\mathrm{ND}(T)| - 1$ or $|\mathrm{ND}(\langle d_1 \rangle)| \geq |\mathrm{ND}(T) - 1|$ or both. Because $|\mathrm{ND}(T)| > |\mathrm{ND}(\langle d_1 \rangle|$, we know that $|\mathrm{ND}(T_1)| \geq |\mathrm{ND}(T)| - 1 \geq |\mathrm{ND}(\langle d_1 \rangle)|$. Thus $|C(T)| = |\mathrm{ND}(T_1)| + 1$. Similar reasoning leads us to conclude $|\mathrm{ND}(T_1')| \geq |\mathrm{ND}(\langle w \rangle)|$ and $|F(T)| = |\mathrm{ND}(T_1')| + 1$.

We now show that $LLV(T_1) \leq LLV(T_1')$. There are three cases to consider:

(1) Let $w \in V(\langle d_1 \rangle)$. Clearly $LLV(T_1) \leq LLV(T_1')$.

(2) Let $w = d_i$, where $i \neq 1$. By Lemma 5.3.1 and our assumption that $LLV(\langle d_1 \rangle) \geq LLV(\langle d_i \rangle)$ for $i > 1$, $LLV(T_1) \leq LLV(T_1')$.

(3) Let $w \in V(\langle d_i \rangle)$, where $i \neq 1$. Then we know $LLV(T_1') \geq LLV(T - \langle d_i \rangle)$. Combining this with case (2) from above, we have $LLV(T_1) \leq LLV(T - \langle d_i \rangle) \leq LLV(T_1')$.

Since $LLV(T_1) \leq LLV(T_1')$ for all w, it follows that $(r, d_1, 1) \in LLFC(T)$. □

Lemma 5.3.2 describes the first call made by the root in a least labor broadcast schedule. The following decomposition property of least labor broadcast schedules will help us to apply Lemma 5.3.2 to the construction of complete least labor root schedules for any ditree T.

Lemma 5.3.3 *Let $L(T)$ be a least labor broadcast schedule for ditree T with $(r, v, 1) \in L(T)$. Then $L(T - \langle v \rangle)$ is a least labor broadcast schedule for $T - \langle v \rangle$.*

Proof: This follows in a straightforward manner from the definition of $L(T - \langle v \rangle)$ and the definition of least labor broadcast schedules. □

5.3.3 Computing Root Schedules and Labor Vectors

We now describe how we utilize Lemma 5.3.2 to compute the root schedule and the labor vector of a least labor broadcast schedule for any directed tree T with root node r. In all of the following algorithms we assume that r has p children $d_1, \ldots, d_p$ and that we are given the root schedules $S_r(\langle d_i \rangle)$ and labor vectors $\vec{V}_S(\langle d_i \rangle)$ of a least labor broadcast schedule $S(\langle d_i \rangle)$ for all d_i. Furthermore, the children of r are ordered such that $\vec{V}_S(\langle d_1 \rangle) \geq \cdots \geq \vec{V}_S(\langle d_i \rangle) \geq \cdots \geq \vec{V}_S(\langle d_p \rangle)$.

First, we define the procedure FIRSTCALL which decides which node v should be informed by r in the first time step. The correctness of FIRSTCALL, assuming the procedure TIME returns

the value of $|\mathrm{ND}(T)|$, follows immediately from Lemma 5.3.2.

FIRSTCALL (T)

1. Call TIME(T) which returns $|\mathrm{ND}(T)|$.
2. If $|\mathrm{ND}(T)| = |\mathrm{ND}(\langle d_1 \rangle)|$, then

   Set $v = v_1(S_r(\langle d_1 \rangle))$

   Else

   Set $v = d_1$.
3. Return v and $|\mathrm{ND}(T)|$.

Next, we define the procedure ROOTSCHED which repeatedly uses FIRSTCALL to create the root schedule $S_r(T)$ and the labor vector $\vec{V}_S(T)$ of a least labor broadcast schedule $S(T)$ for any directed tree T with root node r. The key to our ability to repeatedly apply FIRSTCALL stems from Lemma 5.3.3 which shows us how to quickly compute the root schedule and labor vector of a least labor broadcast schedule for $\langle d_1 \rangle - \langle v \rangle$ in steps (5a) and (5b) of ROOTSCHED.

ROOTSCHED (T)

1. Initialize $S_r(T) = \emptyset$ and $\vec{V}_S(T) = ()$ and $t = 1$.
2. Call FIRSTCALL(T) which returns v and L.
3. Update $S_r(T)$ and $\vec{V}_S(T)$ as follows
   (a) $S_r(T) = S_r(T) \cup (r, v, t)$
   (b) Append L to $\vec{V}_S(T)$
4. If $v = d_1$ then modify T as follows
   (a) Eliminate child d_1 and its associated root schedule and labor vector.
   (b) For $2 \leq i \leq p$, rename child d_i (and associated root schedule and labor vector) to be child d_{i-1}.
   (c) Decrement p by one.
   (d) If $p > 0$ increment t and Goto Step 2. Otherwise append 0 to $\vec{V}_S(T)$ and Exit.
5. Else modify T as follows
   (a) Replace $S_r(\langle d_1 \rangle)$ with $S_r(\langle d_1 \rangle - \langle v \rangle)$.
   (b) Replace $\vec{V}_S(\langle d_1 \rangle)$ with $\vec{V}_S(\langle d_1 \rangle - \langle v \rangle)$.
   (c) Rename children so that $\vec{V}_S(\langle d_1 \rangle) \geq \cdots \geq \vec{V}_S(\langle d_i \rangle) \geq \cdots \geq \vec{V}_S(\langle d_p \rangle)$.
   (d) Increment t and Goto Step 2.

5.3.4 Computing $|\mathrm{ND}(T)|$

We now describe procedure TIME and prove its correctness.

TIME(T)

1. Set $n = |\mathrm{ND}(\langle d_1 \rangle)|$
2. Call CHECK(n, T))
3. If CHECK returns "yes" then
   Return n.
   Else
   Increment n by 1 and goto 2

CHECK(n, T)

1. If $n = 0$ and r has children, return "no"

2. If $n \geq 0$ and r has no children, return "yes"

3. Else

   (a) If $|\text{OPT-ND}(\langle d_1 \rangle)| > n$, return "no"

   (b) If $|\text{OPT-ND}(\langle d_1 \rangle)| < n$, return(CHECK($n - 1, T - \langle d_1 \rangle$).

   (c) If $|\text{OPT-ND}(\langle d_1 \rangle)| = n$, let $v = v_1(S_r(\langle d_1 \rangle))$. Return(CHECK($n - 1, T - \langle v \rangle$)).

Lemma 5.3.4 *For all ditrees T and all $n \geq 0$, CHECK(n,T) returns "yes" if $|ND(T)| \leq n$, and CHECK(n,T) returns "no" if $|ND(T)| > n$,*

Proof: We will prove this lemma by induction on the variable n. Let $D_1 = \langle d_1 \rangle$.

The base case is when $n = 0$. Suppose $|\text{ND}(T)| = 0$. Clearly, this means there are no nodes to inform, *i.e.*, the root of T has no children. In this case, step 2 of CHECK applies, and "yes" is returned. Suppose $|\text{ND}(T)| > 0$. This means that r has children left to inform. In this case, step 1 of CHECK applies, and "no" is returned. Thus, for all ditrees T, CHECK(0,T) returns "yes" if $|\text{ND}(T)| \leq 0$, and CHECK(0,T) returns "no" if $|\text{ND}(T)| > 0$,

Now assume that for all ditrees T and for $n \leq k$, CHECK(n,T) returns "yes" if $|\text{ND}(T)| \leq n$, and CHECK(n,T) returns "no" if $|\text{ND}(T)| > n$. We now show that for all ditrees T, CHECK($k+1$,T) returns "yes" if $|\text{ND}(T)| \leq k+1$, and CHECK($k+1$,T) returns "no" if $|\text{ND}(T)| > k+1$.

Suppose $|\text{ND}(T)| > k+1$. There are three possibilities to consider:

(a) Suppose $|\text{ND}(D_1)| > k+1$. Then step 3(a) applies, and "no" is returned.

(b) Suppose $|\text{ND}(D_1)| < k+1$. Then step 3(b) applies, and CHECK($k+1$,T) is exactly CHECK(k, $T - D_1$). Since $|\text{ND}(D_1)| \leq k$ and $|\text{ND}(T)| > k+1$, we conclude

$|\text{ND}(T - \langle v \rangle)| \geq k+1$. By our induction hypothesis, CHECK($k, T - D_1$), and consequently CHECK($k+1, T$), returns "no."

(c) Suppose $|\text{ND}(D_1)| = k+1$. Then step 3(c) applies, and CHECK($k+1, T$) is exactly CHECK($k, T - \langle v \rangle$), where $v \in LLFC(D_1)$. Since $v \in LLFC(D_1)$ and $|\text{ND}(D_1)| = k+1$, we can conclude that $\text{ND}(\langle v \rangle)| \leq k$. Using the same argument from case (b), it follows that $|\text{ND}(T - \langle v \rangle)| \geq k+1$. By our induction hypothesis, CHECK($k, T - \langle v \rangle$) returns "no", and consequently CHECK($k+1$,T), returns "no."

Thus, we have shown that for all ditrees T, if $|\text{ND}(T)| > k+1$, CHECK($k+1, T$) returns "no."

Now suppose that $|\text{ND}(T)| \leq k+1$. Clearly $|\text{ND}(D_1)| \leq k+1$, so we only need to consider cases (b) and (c) from above.

(b) If $|\text{ND}(D_1)| < k+1$, then CHECK($k+1$,T) is exactly CHECK(k, $T - D_1$). From Lemma 5.3.2, we know that there exists a $v \in D_1$ such that $v \in LLFC(T)$. This, along with the fact that $|\text{ND}(T)| \leq k+1$, implies $|\text{ND}(T - \langle v \rangle)| \leq k$. Since $T - D_1$ is a subtree of $T - \langle v \rangle$, it follows that $|\text{ND}(T - D_1)| \leq k$. Thus, by our induction hypothesis, CHECK(k,$T - D_1$) returns "yes," and consequently CHECK($k+1$,T) also returns "yes."

(c) If $|\text{ND}(D_1)| = k+1$, then CHECK($k+1$,T) is exactly CHECK(k, $T - \langle v \rangle$) where $v \in LLFC(D_1)$. Because $|\text{ND}(D_1)| = k+1$ and $v \in LLFC(D_1)$ we know $|\text{ND}\langle v \rangle| \leq k$. Furthermore, we know from Lemma 5.3.2 that $v \in LLFC(T)$. Using this along with the fact that $|\text{ND}(T)| \leq k+1$, implies $|\text{ND}(T - \langle v \rangle)| \leq k$. Thus, by our induction hypothesis, we know CHECK($k, T - \langle v \rangle$), and consequently CHECK($k+1$,T), returns "yes."

Thus, we have shown that for all ditrees T, if $|\text{ND}(T)| \leq k+1$, CHECK($k+1, T$) returns "yes."

By the principle of mathematical induction, we have shown that for all ditrees T and all $n \geq 0$, CHECK(n, T) returns "yes" if $|\text{ND}(T)| \leq n$ and CHECK(n,T) returns "no" if $|\text{ND}(T)| > n$. □

The remainder of algorithm ODA can be implemented in a straightforward manner. CREATE and SCHED are the procedures that implement the bottom up and top down phases of algorithm ODA, respectively. CREATE simply calls ROOTSCHED in a bottom up fashion after creating empty root vectors and labor vectors equal to (0) for all nodes with outdegree 0. SCHED uses these least labor vectors and root schedules to create a complete minimum length node-disjoint broadcast schedule for the ditree T.

Optimal Decomposing Algorithm ODA

1. Call CREATE(T) to make least labor root schedules and least labor vectors for all $\langle v \rangle$ in T.
2. Call SCHED(T) using the entire output of CREATE(T) to create the broadcast schedule $ODA(T)$.

CREATE(T)

1. For all nodes v with outdegree 0, make $S_r(\langle v \rangle) = \emptyset$ and $\vec{V}_S(\langle v \rangle) = 0$.
2. Initialize $Completed = \{v \mid \text{outdegree}(v) = 0\}$
3. While $Completed \neq V(T)$ do the following
   (a) For all vertices v in $V(T) - Completed$
      (i) If all the children of v are in the set $Completed$ then call ROOTSCHED(T)
      (ii) Add v to the set $Completed$.

SCHED (T)

1. Initialize $D(T) = S_r(T)$ and $Completed = \{r\}$.
2. While $V(T) - Completed \neq \emptyset$
   (a) If all ancestors of v are in Completed
      (i) Call PROCESS($v, D(T)$)
      (ii) Add v to the set $Completed$

PROCESS $(v, D(T))$

1. $time = i$, s.t. $(w, v, i) \in D(T)$
2. $j = \min j$ s.t. $(w, v_j(D_r\langle v\rangle), k) \notin D(T)$ for all k, w.
3. For $t = j$ to $|D_r(\langle v\rangle)|$
   (a) $D(T) = D(T) \cup (w, v_t(D_r(\langle v\rangle)), time + t - j + 1)$, where $(w, v_t(D_r(\langle v\rangle)), t) \in D_r(\langle v\rangle)$.

Lemma 5.3.5 *Let T be a ditree rooted at r. CREATE(T) makes the root schedule and the labor vector of a least labor broadcast schedule for $\langle v\rangle$ for all $v \in V(T)$.*

Proof: This follows from Lemmas 5.3.2, 5.3.3 and 5.3.4. □

This leads us to our main result. The optimal algorithm ODA uses the least labor root schedules and vectors for all $\langle v\rangle$ in T to create an optimal node-disjoint broadcast schedule for T.

Theorem 5.3.1 *Algorithm ODA creates a minimum length node-disjoint broadcast schedule $ODA(T)$ for any directed tree T.*

Proof: This follows from Lemma 5.3.5 and the correctness of the procedure SCHED(T). □

5.3.5 Time Complexity

Table 5.1 gives the worst case time complexity for all the algorithms used by the Optimal Decomposing Algorithm (ODA).

We first calculate the time complexity of the CHECK algorithm. Let $T_{CHECK}(n)$ be the time complexity, where n is the number of nodes in ditree T. We have the following recurrence relation for the worst cast time complexity,

Table 5.1: Time Complexity for ODA

Worst Case Complexities	
Algorithm	Complexity
CHECK	$O(n)$
TIME	$O(n)$
FIRSTCALL	$O(n)$
ROOTSCHED	$O(n^2)$
CREATE	$O(n^3)$
SCHED	$O(n^2)$
PROCESS	$O(n)$
ODA	$O(n^3)$

$$\begin{aligned} T_{CHECK}(n) &= \alpha_1 + T_{CHECK}(n-1) \\ T_{CHECK}(1) &= \alpha_2 \end{aligned}$$

Solving this recurrence gives us

$$T_{CHECK}(n) = O(n).$$

Given the complexity of CHECK, it is easy to see that the TIME algorithm also has complexity $O(n)$. This implies that algorithm FIRSTCALL has time complexity $O(n)$ as well.

Now we can calculate the complexity of ROOTSCHED. The bulk of the work is done in step 2, in which algorithm FIRSTCALL is called. Steps 4(d) and 5(d) precede step 2. It is clear that the maximum number of times that 4(d) and 5(d) can occur is n. Thus, ROOTSCHED has complexity $O(n^2)$.

Given the complexity of ROOTSCHED, we are able to calculate the complexity of CREATE. CREATE can call ROOTSCHED at most n times, thus worst case complexity is $O(n^3)$.

The complexity of PROCESS is $O(n)$ since j in step 2 can be no larger than n. Because PROCESS is called at most n times by SCHED, the complexity of SCHED is $O(n^2)$. Thus, the

complexity of ODA is $O(n^3 + n^2) = O(n^3)$.

5.4 Summary

We describe two polynomial time approximation algorithms, CA and SCA, for performing broadcast in directed trees under the node-switching model of communication. The biggest advantage of the CA algorithm is its simplicity. Its best case time complexity is $O(n \log n)$ (which it achieves for broadcast in hypercubes, meshes and tori), and its worst case complexity is $O(n^2)$. While the SCA algorithm has slightly higher complexity ($O(n^2)$ in the best case and $O(n^3)$ in the worst case), it also may produce much shorter calling schedules. The broadcast schedules produced by SCA are guaranteed to be no more than twice the length of an optimal length broadcast schedule. Finally, we describe a $O(n^3)$ algorithm that always produces minimum length node-disjoint broadcast schedules for directed tree topologies. This can easily be transformed to an edge disjoint broadcast schedule with length at most twice that of an optimal edge disjoint schedule.

Chapter 6

Shortest Path Oblivious Routing

6.1 The Generalized Path-Matching Problem

In Chapter 3 we described *matching* problems that have been previously studied. Traditionally, a matching (or edge-matching) in a graph $G = (V, E)$ is defined as a set of pairwise independent edges. Wu and Manber introduced a generalization of an edge-matching called a *path-matching*. In a path-matching, vertices of G are joined by simple paths rather than edges [52]. Specifically, a path-matching M in a graph G is a set of simple paths in which no two paths share the same end vertices. Of particular interest is the problem of finding a perfect path-matching in which paths in M are pairwise edge-disjoint (a DP-matching).

In this Chapter, we introduce a *generalized path-matching*, which includes both edge-matchings and path-matchings. We show how a generalized path-matching can be used to create an optimal UBM algorithm for arbitrary topologies that employ a shortest path routing scheme. We will define a generalized path-matching for a *symmetric* digraph rather than an undirected graph. A directed graph $G = (V, A)$ is symmetric if for every arc $ij \in A(G)$, we have arc $ji \in A(G)$ as well. A generalized path-matching is defined as follows:

Definition 6.1.1 *Given a symmetric digraph graph $G = (V, A)$, a set P of directed paths in G, and a set $X \subseteq V(G)$ of vertices, we call $M_P \subseteq P$ a* **generalized path-matching (GP-Matching)** *on (G, P, X) if the origin and terminus of each directed path $p \in M_P$ are distinct vertices from X, and no two dipaths have a common origin or terminus.*

Note that every undirected graph has a corresponding symmetric digraph representation obtained by replacing each edge ab with two directed arcs ab and ba. Thus, a GP-matching can be thought of as being defined for an undirected graph, except the paths used to match vertices are directed. Directed paths are used to distinguish an ij-path from a ji-path. That is, if the prescribed set P of paths contains a directed ij-path, it does not necessarily contain a directed ji-path. Thus, we will assume an ij-path refers to a directed path with origin i and terminus j.

In the generalized case, a *perfect* GP-matching is one that matches all vertices from X. Both the edge-matching problem and the path-matching problem are specific cases of the GP-matching problem. In the edge-matching problem, $X = V(G)$ and P consists of all paths in G that are length one (*i.e.*, $P = A(G)$, the arcs in the symmetric digraph that corresponds to graph G). Thus, a GP-matching M_A for (G, V, A) is exactly an edge-matching in G. Similarly, if $P(G)$ denotes the set of all simple dipaths in $G = (V, A)$, then the GP-matching $M_{P(G)}$ for $(G, V, P(G))$ is exactly a path-matching in G.

In Section 3.5, we described two commonly studied variations of matching problems in graphs. The first variation is finding maximum (or perfect) matchings, and the second variation is finding minimum cost maximum matchings. We will focus on a slightly different variation; we wish to identify properties that guarantee a perfect GP-matching exists. We are particularly interested in perfect GP-matchings in which the directed paths are pairwise *edge-disjoint* (perfect GDP-matchings). Two directed paths $p(x_o, x_t)$ and $p(y_o, y_t)$ are edge-disjoint if they do not have any of their underlying edges in common. Unlike DP-matchings, perfect GDP-matchings do not always exist. In Section 6.2, we give sufficient conditions that guarantee an input instance (G, P, X) has a perfect

GDP-matching. In particular, we show that if P contains a shortest path between each pair of vertices in X, then there exists at least one perfect GDP-matching for (G, P, X). In addition, the proof provides a method for finding a perfect GDP-matching.

The GDP-matching problem has a direct application in *multicast* communication for direct network systems. In a line-switching UBM schedule, several messages may be sent concurrently along edge-disjoint paths in the network. In Section 6.3, we show how GDP-matchings can be used to maximize parallel communication and therefore to allow an optimal line-switching multicast algorithm to be constructed for direct network systems.

6.2 Sufficient Conditions for Perfect GDP-matching

Since every connected graph contains a perfect DP-matching, we know that when $X = V(G)$ and P contains all the paths in G, that there is a perfect GDP-matching for (G, P, X). It is not true in general that a perfect GDP-matching exists. For example, consider the graph C_8, pictured in Figure 6.1. Suppose $X = \{1, 2, 3, 8\}$. Let $p_L(i, j)$ be the longest ij-path, $p_c(i, j)$ be the "clockwise"

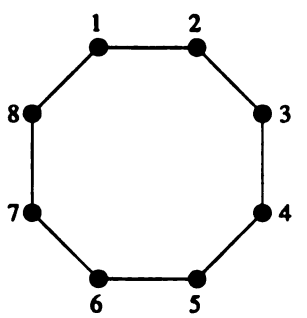


Figure 6.1: Cycle on 8 vertices.

ij-path, and $p_S(i, j)$ be a shortest ij-path in C_8. Furthermore, let

$$\begin{aligned} P_1 &= \{p_L(i,j)\ \forall i,j \in X\} \\ P_2 &= \{p_c(i,j)\ \forall i,j \in X\} \\ P_3 &= \{p_S(i,j)\ \forall i,j \in X\}. \end{aligned}$$

Clearly no perfect GDP-matching exists for (C_8, P_1, X) because every pair of paths must share a common underlying edge. If we use the paths in P_2, $M_{P_2} = \{p_c(1,2), p_c(3,8)\}$ is a perfect GDP-matching for (C_8, P_2, X). If we use P_3, the vertices matched in the previous example, using paths from P_2, cannot be matched by disjoint paths from P_3 (both $p_S(1,2)$ and $p_S(3,8)$ use the underlying edge between vertex 1 and vertex 2), but $M_{P_3} = \{p_S(1,8), p_S(2,3)\}$ is a perfect GDP-matching for (C_8, P_3, X).

Because perfect GDP-matchings do not always exist, a natural question that arises is: "under what conditions is there a perfect GDP-matching for instance (G, P, X)?" In this section, we give sufficient conditions for the existence of a perfect GDP-matching. In particular, we show that if X is any subset of vertices and P contains a shortest path for each pair of vertices in X, then there is a perfect GDP-matching for (G, P, X). The following Lemma is the key for finding edge-disjoint paths that match pairs of vertices in X.

Definition 6.2.1 *Let $|p(i,j)|$ denote the length of path $p(i,j)$.*

Lemma 6.2.1 *Let $G = (V, A)$ be a symmetric digraph, $X \subseteq V(G)$ a subset of vertices, and P be a set of directed paths that contains a shortest path for all pairs of vertices in X. Further, let $p(x,y)$ denote a shortest $xy-$path from set P. If there are two paths $p(x_1,y_1)$ and $p(x_2,y_2)$ that contain a common underlying edge, then either*

$$|p(x_1,x_2)| + |p(y_1,y_2)| < |p(x_1,y_1)| + |p(x_2,y_2)|$$

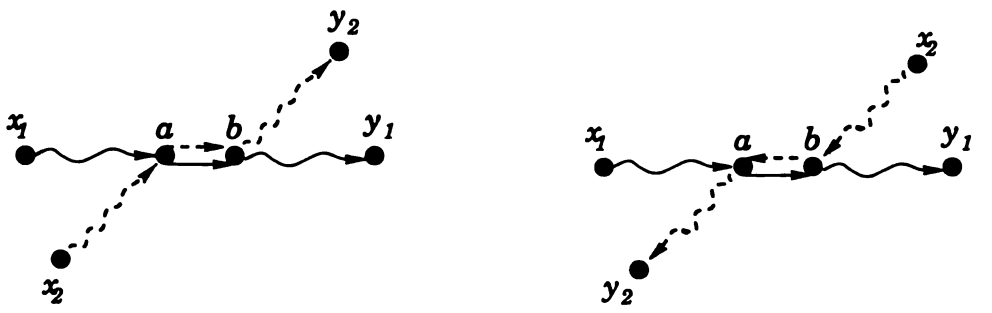
or

$$|p(x_1,y_2)| + |p(x_2,y_1)| < |p(x_1,y_1)| + |p(x_2,y_2)|.$$

Proof: Suppose that directed paths $p(x_1,y_1)$ and $p(x_2,y_2)$ are not edge-disjoint, and that they both contain the underlying edge ab. WLOG, assume that a occurs before b on $p(x_1,y_1)$. Then there are two cases to consider,

1. a occurs before b on $p(x_2,y_2)$ as pictured in Figure 6.2(a).

2. b occurs before a on $p(x_2, y_2)$ as pictured in Figure 6.2(b).



(a) Case 1 (b) Case 2

Figure 6.2: Two conflicting paths.

In the first case,

$$
\begin{aligned}
|p(x_1, y_1)| &= |p(x_1, a)| + |p(a, b)| + |p(b, y_1)| \\
|p(x_2, y_2)| &= |p(x_2, a)| + |p(a, b)| + |p(b, y_2)| \\
|p(x_1, y_1)| + |p(x_2, y_2)| &= |p(x_1, a)| + |p(a, b)| + |p(b, y_1)| \\
&\quad + |p(x_2, a)| + |p(a, b)| + |p(b, y_2)| \\
|p(x_1, y_1)| + |p(x_2, y_2)| &= (|p(x_1, a)| + |p(x_2, a)|) + (|p(b, y_1)| + |p(b, y_2)|) + 2 \\
|p(x_1, y_1)| + |p(x_2, y_2)| &\geq |p(x_1, x_2)| + |p(y_1, y_2)| + 2 \\
|p(x_1, y_1)| + |p(x_2, y_2)| &> |p(x_1, x_2)| + |p(y_1, y_2)|
\end{aligned}
$$

Similarly, in the second case,

$$|p(x_1, y_1)| + |p(x_2, y_2)| > |p(x_1, y_2)| + |p(x_2, y_1)|.$$

□

Lemma 6.2.1 leads naturally to a method for finding a perfect GDP-matching for (G, P, X).

Theorem 6.2.1 *Let $G = (V, A)$ be a symmetric digraph, and let $X \subseteq V(G)$ be a set of vertices from G. If P contains a shortest ij-path for all $i, j \in X$, then there is a perfect GDP-matching for (G, P, X).*

Proof: We describe a polynomial-time algorithm for finding a perfect GDP-matching for (G, P, X) when P contains a shortest ij-path for all $i, j \in X$.

Let $X = \{x_1, \ldots, x_n\}$, and let $p(x_i, x_j)$ be an arbitrary shortest ij-path from P. A perfect GDP-matching contains $k = \lfloor \frac{n}{2} \rfloor$ paths. We initialize a matching $M(0)$ to be a set of paths between k arbitrary pairs of vertices in X. WLOG, let $M(0) = \cup_{i=1}^{k} p_i$, where $p_i = p(x_i, x_{k+i})$.

Initially, $M(0)$ may not contain pairwise edge-disjoint paths. We describe a procedure that transforms $M(0)$ into a perfect GDP-matching after several iterations. Let $M(i)$ be the matching associated with iteration i in the procedure, and assume the path p_j has source s_j and terminus t_j. We transform $M(0)$ into a GDP-matching as follows:

1. Let $iteration = 0$
2. If there exist paths $p_i = (s_i, t_i)$ and $p_j = (s_i, t_j)$ from $M(iteration)$ that both contain the underlying edge ab, then increment $iteration$ and use Lemma 6.2.1 to create a new matching $M(iteration)$ as follows:

   If arc $ab \in p_i$ and arc $ab \in p_j$, then

   $$M(iteration) = M(iteration - 1) - p_i - p_j + p(s_i, s_j) + p(t_i, t_j)$$

   Else if arc $ab \in p_i$ and arc $ba \in p_j$, then

   $$M(iteration) = M(iteration - 1) - p_i - p_j + p(s_i, t_j) + p(s_j, t_i)$$

   Else there is no edge contention. STOP.
3. Repeat step 2.

If the procedure terminates with $iteration = i$, $M(i)$ is a perfect GDP-matching for (G, P, X) because all k paths in $M(i)$ are pairwise edge disjoint. Thus, we only need to show that the procedure terminates.

Define $size[M(i)] = \sum_{p \in M(i)} |p|$. Because each path p in $M(i)$ must contain at least one edge,

$$\forall i \ \ size[M(i)] \ \geq \ k. \tag{6.1}$$

By Lemma 6.2.1,

$$\forall i \ \ size[M(i+1)] \ < \ size[M(i)]. \tag{6.2}$$

Combining the facts in Equations 6.1 and 6.2 guarantees that the procedure must terminate. □

6.3 Multicast Algorithm using GDP-matching

The GDP-matching problem has a direct application to multicast communication. We present an optimal multicast algorithm that is based on Theorem 6.2.1. In a UBM implementation, the lower bound on the number of steps required to complete a multicast request $M = (s, D)$ is $\lceil \lg(|D| + 1) \rceil$. Thus, a multicast schedule with length $\lceil \lg(|D| + 1) \rceil$ is optimal. Ideally, we want to find an optimal UBM schedule C for implementing any multicast request M in any direct network topology represented by a graph G. Recall from Chapter 3 that optimal UBM algorithms have been developed for arbitrary network topologies that employ free routing, for mesh topologies that use xy-routing, for hypercube topologies that use e-cube routing, and for torus topologies that use dimension ordered routing. Unfortunately, multicast cannot always be performed in $\lceil \lg(|D| + 1) \rceil$

steps. A natural question to ask is, "Under what conditions can UBM be done in the optimal number of time steps, $\lceil \lg(|D| + 1) \rceil$?" In this section, we use Theorem 6.2.1 to show that any routing scheme that includes a shortest ij-path for all pairs (i, j) allows optimal line-switching UBM in any network topology that can be represented by a symmetric digraph (*i.e.*, a direct network that provides symmetric communication). First, some definitions are in order.

Definition 6.3.1 *Let $dis_G(i, j)$ be the length of a shortest ij-path in G.*

Definition 6.3.2 *A routing scheme R is called a* shortest path routing scheme *if for every pair of nodes v_i and v_j, R contains a routing path $p_R(i, j)$ of length $dis_G(i, j)$.*

Theorem 6.3.1 *There is an optimal line-switching UBM implementation for any shortest path routing scheme in any arbitrary symmetric graph topology.*

Proof: We give a constructive proof, *i.e.*, we describe a method to create a multicast schedule of length $\lceil \lg(|D| + 1) \rceil$ for any multicast request $M = (s, D)$ in any system represented by a symmetric digraph and that employs a shortest path routing scheme R. Let $p_R(s, d)$ be the routing path used by R to send a message from node s to node d.

In order to achieve multicast in $\lceil \lg(|D| + 1) \rceil$ time steps, the number of informed nodes must double in each time step. Thus, in the last step, half of the destination nodes are already informed and must make calls to the other half of the destinations. This is the underlying principle of the technique that we describe.

A multicast schedule is built from the bottom up. That is, the calls in time step $\lceil \lg(|D| + 1) \rceil$ are scheduled first, then the calls in step $\lceil \lg(|D| + 1) \rceil - 1$, and so on. We use Theorem 6.2.1 to create the optimal multicast schedule. The set X in Theorem 6.2.1 corresponds to the set of nodes involved in the multicast request (*i.e.*, $s \cup D$). The set of paths P is exactly the routing paths included in R. Because R contains a shortest path for each possible source-destination pair, Theorem 6.2.1 guarantees that a perfect GDP-matching can be found for $(G, R, s \cup D)$. Let the perfect GDP-matching

M_P be $\{p_R(s_1, d_1), \ldots, p_R(s_k, d_k)\}$, where $k = \lfloor \frac{|s \cup D|}{2} \rfloor$. The paths in M_P represent the calls that will be made in the last step of the multicast schedule. Let $S = \cup_{i=1}^{k} s_i$. Since the nodes in S are sources of unicast calls in step $\lceil \lg(|D| + 1) \rceil$, these nodes must be informed in $\lceil \lg(|D| + 1) \rceil - 1$ steps. Essentially, we have a new multicast request $M' = (s, D')$, where $D' = S - \{s\}$. We find a perfect GDP-matching for $s \cup D'$ as described above, and continue this process for a total of $\lceil \lg(|D|+1) \rceil$ steps, when the single call in the first step of the multicast algorithm is scheduled. □

Theorem 6.3.1 provides the theoretical basis for all the previous work on optimal multicast in direct network topologies. In [21, 46], optimal UBM algorithms are given for for any arbitrary network that uses free routing. In addition, optimal UBM algorithms for meshes that use xy-routing, hypercubes that use e-cube routing, and tori that use dimension ordered torus routing have been developed [46, 47]. Each of these is a corollary to Theorem 6.3.1.

Corollary 6.3.1 *There is an optimal line-switching UBM implementation for all $M = (s, D)$ in an arbitrary topology that employs free routing.*

Corollary 6.3.2 *There is an optimal line-switching UBM implementation for all $M = (s, D)$ in any mesh topology that employs xy-routing.*

Corollary 6.3.3 *There is an optimal line-switching UBM implementation for all $M = (s, D)$ in any hypercube topology that employs e-cube routing.*

Corollary 6.3.4 *There is an optimal line-switching UBM implementation for all $M = (s, D)$ in any torus topology that employs dimension-ordered torus routing.*

In addition, Theorem 6.3.1 gives guidelines for developing routing strategies. The theorem states that if shortest paths are used to route messages between all pairs of nodes, this is sufficient to guarantee an optimal multicast implementation. Current routing techniques use shortest paths. Using shortest paths can reduce overall message transmission latency and can reduce network traffic. In addition, shortest paths require less space to encode in routing table. In essence, Theorem 6.3.1

says that a simple, natural routing scheme in an arbitrary topology allows optimal multicast performance.

Finally, Theorem 6.3.1 has implications for updating routing schemes as new nodes are incrementally added on to the system. If new nodes are added to a system, routing tables must be updated to include the routes for new nodes. As long as all routing paths are shortest paths in the network, multicast can be done in optimal time. First, we must verify if existing routing paths are still shortest paths after the additional nodes are added. Then, any shortest path can be used for routes containing new nodes and to replace original routing paths that are no longer shortest paths. No particular structure or symmetry must be maintained by the routing paths, making it quite simple to update the routing scheme.

6.4 Summary

This chapter describes a generalization of previously studied matching problems for undirected graphs. The Generalized Path-Matching (GP-matching) problem is defined, which includes both the edge-matching problem and the path-matching problem. In a GP-matching, a subset X of vertices is matched using paths from a specified set P. We are interested in finding a perfect GP-matching, or one that matches all the vertices of X. Additionally, we focus on the edge-disjoint GP-matching problem (or GDP-matching problem). Sufficient conditions for the existence of a perfect GDP-matching are given. It is shown that if shortest paths are used to match vertices, a perfect GDP-matching always exists.

Sufficient conditions for optimal multicast communication in parallel systems based on direct networks are a direct consequence of the GDP-matching result. We showed that in a symmetric direct network system, a shortest path routing scheme allows optimal UBM communication in any arbitrary topology. Because meshes, hypercubes and tori typically use shortest path routing

schemes, this result shows that optimal UBM can be done in these direct network systems. Thus, this result provides a theoretical basis that unifies the optimal algorithms previously developed for these specific topologies [46, 47]. Since using shortest paths to route messages is sufficient to guarantee optimal time multicast, designing routing strategies for arbitrary topologies can be straight forward. Shortest paths should be used for routing, but routing paths do not need to be uniform or symmetric. Additionally, this result implies that adding on to a network incrementally should be relatively simple. Old routes may remain unchanged and new routes only need to satisfy the shortest path property. Finally, the proof provides a technique for constructing an optimal length multicast schedule for any multicast request in an arbitrary topology that uses shortest path routing.

Chapter 7

Conclusions and Future Work

The goal of our research is to provide a general UBM approach for efficient multicast communication in direct networks. This dissertation describes significant progress toward that end. We present two general models for realistic direct network systems and provide multicast algorithms for arbitrary topologies. We summarize these contributions in Section 7.1. Ideally, we want to develop time and traffic efficient UBM algorithms for arbitrary restricted routing systems in general topologies. In Section 7.2, we outline several open problems that merit future investigation.

7.1 Conclusions

7.1.1 Source Limited Inclusive Routing

Source limited inclusive routing schemes represent a realistic class of routing schemes for several reasons. First, all routing schemes that are implemented on current machines are inclusive routing schemes. Second, storing only source limited routing information allows for the smallest possible storage requirements for routing information. Furthermore, because all routing paths contain additional routing information within them, much of the system-wide routing information is available

even though nodes store only source limited information. Finally, source limited inclusive routing schemes have a natural graph theoretic model, namely directed trees, that makes them easy to work with.

Although the complexity of determining optimal line-switching UBM schedules for directed trees has not been determined, we show that node-switching UBM schedules provide a very good approximation for line-switching UBM schedules in ditrees. In addition, decomposing node-switching algorithms in a ditree have several advantages. First, multicast problems in ditrees under the node-switching assumption can be transformed to equivalent broadcast problems. Consequently we can focus on broadcast in directed tree topologies. Additionally, decomposing algorithms allow a ditree to be broken into two disjoint ditrees after a unicast call has been made. Because the resulting problems are disjoint, we know that the unicast paths used to deliver messages in different steps of the UBM schedule will not interfere with each other. Thus, the UBM algorithm does not need to be synchronous (*i.e.*, does not need a barrier at the end of each time step). Finally, the decomposing nature of node-switching algorithms makes analysis of these algorithms relatively simple.

7.1.2 Shortest Path Oblivious Routing

Although storing source limited information has very small storage requirements, helpful routing information may not be available, and therefore optimal algorithms cannot be determined. In the ideal case, all routing information is available to each node of the network. If complete information is stored, shortest path oblivious routing schemes are the most realistic schemes to use. Using shortest paths to deliver messages can reduce overall network traffic, and it also minimizes the storage for complete routing information.

We show that GDP-matchings can be used to create an optimal UBM algorithm for arbitrary direct network systems that employ a shortest path routing scheme. This result provides a theoretical

basis that unifies the previous results on optimal multicast algorithms in specific direct network topologies [21, 46, 47]. In addition, this result provides system designers with some simple, intuitive rules for creating routing schemes that guarantee multicast can be performed in minimum time.

7.2 Future work

7.2.1 Restricted Routing Model

In Chapter 4, we described the difficulties associated with finding a graph model for arbitrary restricted routing schemes. In addition, we present a ditree model for source limited inclusive routing schemes. The clear disadvantage of this model is that it does not use complete routing information to create multicast schedules. If complete routing information is available, we may be able to use this information to reduce the number of time steps needed to satisfy a multicast request. Other realistic assumptions or different modeling techniques may lead to a model that can represent arbitrary restricted routing information completely, accurately, and unambiguously.

7.2.2 Time Efficient Line-Switching UBM Algorithms

Our initial results have concentrated on time efficient multicast algorithms for source limited inclusive routing schemes and for shortest path oblivious routing schemes. This work stemmed from related work on time efficient multicast in systems that allow free routing. We present an optimal time line-switching UBM algorithm for shortest path schemes, and we show that we can approximate time efficient multicast for source limited inclusive routing schemes under the line-switching model; however, the existence of an optimal time line-switching algorithm for ditrees remains an open problem and deserves further investigation. In addition, we need to address the larger problem of optimal time line-switching UBM algorithms in other restricted routing schemes. Thus, we want to develop an optimal polynomial time UBM algorithm for arbitrary restricted routing schemes, or

prove that no such algorithm exists.

7.2.3 Traffic Efficient Line-Switching UBM Algorithms

The results in this dissertation focus on time efficient UBM algorithms. However, traffic efficiency is also a very important metric to consider. Ideally, a UBM algorithm will consider traffic efficient multicast in conjunction with time efficiency. One can start by addressing this issue in systems that allow free routing. The advantage of studying free routing first, is that free routed systems have a well defined graph model. Often, solving a problem in a simplified model gives insight about solving a harder problem. For example, the multicast algorithm in [46] was the foundation for the Centroid Algorithm presented in Chapter 5. In the same way, traffic efficient algorithms in free routing may provide insight about restricted routing systems.

As discussed in Chapter 3, systems that assume free routing have lower bound implementations for any multicast request. We will first consider the existing UBM algorithm proposed by McKinley *et al.* [46]. This algorithm is optimally *time* efficient; however, it does not consider *traffic* efficiency. Incorporating the notion of traffic efficiency into the existing algorithm can be explored. The algorithm relies on using an Euler trail to determine the unicast call that are made. The traffic efficiency is greatly influenced by the Euler trail that is chosen. Thus, it is important to characterize the Euler trail that produces the most traffic efficient multicast schedule.

Preliminary results not presented in this dissertation show that using the Euler trail method described by McKinley *et al.* is not optimally traffic efficient. In some networks, an optimal traffic efficient multicast schedule cannot be generated by any of the possible Euler trails. In order to produce optimal traffic efficient multicast schedules some other method must be used. Such methods merit further investigation.

7.2.4 Partially Restricted Routing

In addition to multicast in oblivious routing schemes, the issues of multicast in *partially* restricted routing schemes can be explored. We described one type of partially restricted routing in which part of the path is restricted and part of the path is free. This was inspired by turnaround routing in multistage cube networks. Other types of partial restricted may also be used. For example, we can restrict several of the nodes a path must travel through, but not the path itself. Very little is known about multicast performance for partially restricted routing schemes.

Bibliography

Bibliography

[1] Iris Bager and Yishay Mansour. Broadcast in radio networks. In *Proceedings 6th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 577–585, 1995.

[2] Barbara D. Birchler, Abdol-Hossein Esfahanian, and Eric Torng. Toward a general theory of unicast-based multicast communication. In Manfred Nagl, editor, *Lecture Notes in Computer Science*, volume 1017, pages 237–251. Springer-Verlag, 1995.

[3] Barbara D. Birchler, Abdol-Hossein Esfahanian, and Eric Torng. Toward a general theory of unicast-based multicast communication. Technical Report CPS-95-3, Michigan State University, March 1995.

[4] Barbara D. Birchler, Abdol-Hossein Esfahanian, and Eric Torng. Information dissemination in restricted routing networks. In *The International Symposium on Combinatorics and Applications*, pages 33–43, June 1996.

[5] Barbara D. Birchler, Abdol-Hossein Esfahanian, and Eric Torng. Sufficient conditions for optimal multicast communication. In *Proceedings of the 1997 International Conference on Parallel Processing*, pages 390–393, August 1997.

[6] Jean-Claude Bermond, Pavol Hell, Arthur Liestman, and Joseph Peters. Broadcasting in bounded degree graphs. *SIAM J. Disc. Math.*, 5(1):10–24, February 1992.

[7] Jean-Claude Bermond, Pavol Hell, Arthur Liestman, and Joseph Peters. Sparse broadcast graphs. *Discrete Applied Mathematics*, 36:97–130, 1992.

[8] Kadaba Bharath-Kumar and Jeffrey M. Jaffe. Routing to multiple destinations in computer networks. *IEEE Transactions on Communications*, 31(3):343–351, March 1983.

[9] Rajendra V. Boppana, Suresh Chalasani, and C.S. Raghavendra. On multicast wormhole routing in multicomputer networks. In *Proceedings of the 6th IEEE Symposium on Parallel and Distributed Processing*, pages 722–729, 1994.

[10] Jehoshua Bruck, Luc De Coster, Natalie Dewulf, Ching-Tien Ho, and Rudy Lauwereins. On the design and implementation of broadcast and global combine operations using the postal model. *IEEE Transactions on Parallel and Distributed Systems*, 7(3):256–265, March 1996.

[11] Gregory T. Byrd, Nakul P. Saraiya, and Bruce A. Delagi. Multicast communication in multiprocessor systems. In *Proceedings of the 1989 International Conference on Parallel Processing*, pages I–196–I–200, 1989.

[12] Gary Chartrand and Ortrud R. Oellermann. *Applied and Algorithmic Graph Theory*. International Series in Pure and Applied Mathematics. McGraw-Hill, Inc., New York, 1993.

[13] S.C. Chau and A.L. Liestman. Constructing minimal broadcast networks. *J. Combin. Inform. System. Sci.*, 10:110–122, 1985.

[14] Hyeong-Ah Choi and Abdol-Hossein Esfahanian. A message-routing strategy for multicomputer systems. *Networks*, 22:627–646, 1992.

[15] Alak K. Datta and Ranjan K. Sen. 1-approximation algorithm for bottleneck disjoint path matching. *Information Processing Letters*, 55:41–44, 1995.

[16] R. F. DeMara and D. I Moldovan. Performance indices for parallel marker-propagation. In *Proceedings of the 1991 International Conference on Parallel Processing*, pages 658–659, St. Charles, Illinois, August 1991.

[17] Narsingh Deo. *Graph Theory with Applications to Engineering and Computer Science*. Series in Automatic Computation. Prentice-Hall, Inc., Englewood Cliffs, N.J., 1974.

[18] Shimon Even. *Graph Algorithms*. Computer Science Press, Inc., Rockville, MD, 1979.

[19] A. Farley, S. Hedetniemi, S. Mitchell, and A. Proskurowski. Minimum broadcast graphs. *Discrete Mathematics*, 25:189–193, 1979.

[20] A. M. Farley. Minimal broadcast networks. *Networks*, 9:313–332, 1979.

[21] Arthur M. Farley. Minimum-time line broadcast networks. *Networks*, 10:59–70, 1980.

[22] Message Passing Interface Forum. Message passing interface forum. Technical Report CS-93-214, University of Tennessee, November 1993.

[23] Pierre Fraingiaud and Joseph Peters. Structured communication in torus networks. In *Hawaii Int. Conf. on Systems Science*, pages 584–593. IEEE, January 1995.

[24] H. Franke. MPI-F: An MPI implementation for IBM SP-1, February 1994. Available by anonymous ftp from info.mcs.anl.gov.

[25] Luisa Gargano, Arthur Liestman, Joseph Peters, and Dana Richards. Reliable broadcasting. *Discrete Applied Mathematics*, 53:135–148, 1994.

[26] M. Grigni and D. Peleg. Tight bounds on minimum broadcast networks. *SIAM J. Discrete Math.*, 4:207–222, 1991.

[27] W. Gropp and B. Smith. Users manual for the chameleon parallel programming tools. Technical Report ANL-93/23, Argonne National Laboratory, June 1993.

[28] Frank Harary. *Graph Theory*. Addison-Wesley Series In Mathematics. Addison-Wesley Publishing Company, Reading, Massachusetts, 1969.

[29] Chengchang Huang and Philip K. McKinley. Communication issues in parallel computing across ATM networks. *IEEE Parallel and Distributed Technology*, pages 73–86, Winter 1994.

[30] Kai Hwang. *Advanced Computer Architecture: Parallelism, Scalability, Programmability*. McGraw-Hill Series in Computer Science. McGraw-Hill, Inc., New York, 1993.

[31] Intel, Supercomputer Systems Division, Intel Corporation, Beaverton, OR 97006. *Paragon XP/S Product Overview*, 1991.

[32] Jave O. Kane and Joseph G. Peters. Line broadcasting in cycles. Technical Report CMPT TR 94-11, School of Computing Sciences, Simon Fraser University, Burnaby, B.C., Canada, December 1994.

[33] R. Karp, A Sahay, E. Santos, and K. Schauser. Optimal broadcast and summation in the LogP model. In *Fifth Symposium on Parallel Algorithms and Architectures*, June 1993.

[34] V. Kumar and V. Singh. Scalability of parallel algorithms for the all-pairs shortest path problem. Technical Report ACT-OODS-058-90, Rev. 1, MCC, January 1991.

[35] F. T. Leighton. *Introduction to Parallel Algorithms and Architectures*. Morgan Kaufmann Publishers, San Mateo, CA, 1992.

[36] Daniel Lenoski, James Laudon, Kourosh Gharachorloo, Anoop Gupta, and John Hennessy. The directory-based cache coherence protocol for the dash multi-processor. In *Proceedings of the 17th Annual International Symposium on Computer Architecture*, pages 148–159, May 1990.

[37] Arthur L. Liestman and Joseph G. Peters. Broadcast networks of bounded degree. *SIAM J. Disc. Math.*, 1(4):531–540, November 1988.

[38] Arthur L. Liestman and Joseph G. Peters. Minimum broadcast digraphs. *Discrete Applied Mathematics*, 37/38:401–419, 1992.

[39] Xiaola Lin, Philip K. McKinley, and Lionel M. Ni. Deadlock-free multicast wormhole routing in 2-d mesh multicomputers. *IEEE Transactions on Parallel and Distributed Systems*, 5(8):793–804, August 1994.

[40] Xiaola Lin and Lionel M. Ni. Deadlock-free multicast wormhole routing in multicomputer networks. In *The 18th Annual International Symposium on Computer Architecture*, pages 116–125, May 1991.

[41] Philip K. McKinley and Christian Trefftz. Efficient broadcast in all-port wormhole-routed hypercubes. In *Proceedings of 1993 International Conference on Parallel Processing*, volume 2, pages 288–291, St. Charles, Illinois, August 1993.

[42] S. Mitchell and S. Hedetniemi. A census of minimum broadcast graphs. *J. Combin. Inform. Systems Sci.*, 5:141–151, 1980.

[43] Jae moon Koh and Dong wan Tcha. Local broadcasting in a tree under minisum criterion. *Networks*, 23:71–79, 1993.

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

[45] L. M. Ni and P. K. McKinley. A survey of wormhole routing techniques in direct networks. *IEEE Computer*, 26:62–76, February 1993.

[46] P. K. McKinley, H. Xu, A-H. Esfahanian, and L. M. Ni. Unicast-based multicast communication in wormhole-routed networks. *IEEE Transactions on Parallel and Distributed Systems*, 5(12):1252–1265, December 1994.

[47] David F. Robinson, Philip K. McKinley, and Betty H.C. Cheng. Optimal multicast communication in wormhole-routed torus networks. In *Proceedings of the 1994 International Conference on Parallel Processing*, pages I–134–I–141, 1994.

[48] S. A. Felperin, et al. Routing techniques for massively parallel communications. *Proc. IEEE*, 79(4):488–503, April 1991.

[49] P. J. Slater, E. J. Cockayne, and S. T. Hedetniemi. Information dissemination in trees. *Society for Industrial and Applied Mathematics*, 10(4):692–701, November 1981.

[50] Craig B. Stunkel, Dennis G. Shea, Bulent Abali, Mark Atkins, Carl A. Bender, Don G. Grice, Peter H. Hochschild, Douglas J. Joseph, Ben J. Nathanson, Richard A. Swetz, Robert F. Stucke, Michael Tsao, and Philip R. Varker. Sp2 high-performance switch. *IBM Systems Journal*, 34(5):185–204, 1995.

[51] Yu-Chee Tseng, Dhabaleswar K. Panda, and Ten-Hwang Lai. A trip-based multicasting model in wormhole-routed networks with virtual channels. *IEEE Transactions on Parallel and Distributed Systems*, 7(2):138–150, February 1996.

[52] Sun Wu and Udi Manber. Path-matching problems. *Algorithmica*, 8(2):89–101, 1992.

[53] Hong Xu, Ya-Dong Gui, and Lionel M. Ni. Optimal software multicast in wormhole-routed multistage networks. In *Proceedings of the 1994 Supercomputing Conference*, pages 703–712, November 1994.