This is to certify that the

dissertation entitled

MULTICAST COMMUNICATION: FROM PLATFORM-INDEPENDENT MODELING TO PLATFORM-DEPENDENT TUNING

presented by

Natawut Nupairoj

has been accepted towards fulfillment
of the requirements for

Ph.D. degree in Computer Science

Major professor

Date 5/11/98

MULTICAST COMMUNICATION: FROM PLATFORM-INDEPENDENT MODELING TO PLATFORM-DEPENDENT TUNING

By

Natawut Nupairoj

A DISSERTATION

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

DOCTOR OF PHILOSOPHY

Department of Computer Science

1998

ABSTRACT

MULTICAST COMMUNICATION: FROM PLATFORM-INDEPENDENT MODELING TO PLATFORM-DEPENDENT TUNING

By

Natawut Nupairoj

Multicast is an important system-level one-to-many collective communication service. Several collective communication services such as broadcast and scatter are a subset or a derivation of multicast. Due to the importance of multicast, many standard communication libraries have included a multicast service as their standard communication primitive. In addition, several parallel systems and high-speed switches provide special hardware support for multicast.

This thesis investigates several aspects of software-based multicast. To study how to efficiently implement a multicast service, the performance model for multicast is needed. Although multicast has been extensively studied, its performance metrics and benchmarking techniques have not been through-roughly examined. This research discusses the performance model of multicast which includes both metrics and how to measure them accurately.

A key issue in designing software multicast algorithms is to consider the trade-off be-

tween performance and portability. For most portable software multicast algorithms, they sacrifice their performance in order to achieve the portability. Thus, the performance of a portable algorithm is varied significantly from one platform to another. This thesis proposes the platform-independent multicast algorithm. The algorithm is portable, but still performs well on various platforms.

Without considering the underlying network architecture, the platform-independent multicast algorithm may not achieve truly optimal performance when implement in real networks. Some platform-dependent information such as the topology of the underlying network can be very useful to a multicast algorithm. This thesis addresses the platform-dependent tuning techniques to further optimize our platform-independent multicast algorithm.

Most multicast-related studies have been done in the ideal condition where other network traffic are not considered. However, in real networks, these network traffic may have significant impacts on multicast behavior. We investigate the performance of several multicast algorithms when other network traffic exist. We also study how to further improve our multicast algorithm when additional local network information is available.

To my grandmother and my parents

ACKNOWLEDGMENTS

The road toward my Ph.D. degree has been a very long one. There are so many people who have lent their hands to make this dissertation possible. First of all, I am deeply indebted to my advisor, Dr. Lionel M. Ni, for his tireless effort and boundless patience. He has taught me to be a critical-thinking person, to be an enthusiastic student, and most importantly, to understand the most valuable lesson in the true meaning of hard-working. His positive influence will have a profound impact on both my personal life and my future career.

I would like to thank other members of my dissertation committee: Dr. Philip K. McKinley, Dr. Matt W. Mutka, and Dr. Marianne Huebner, for their valuable comments and encouragement. I would like to thank Dr. Wenjian Qiao, Wei-Kuo Liao, and other ACS members who provided useful suggestions during my thesis work. I would also like to thank Dr. George C. Stockman, Lora Mae Higbee, Linda L. Moore, and friends from the CPS department for all their help.

Throughout the course of my study, there are both good time and bad time. The love and support from my family has given me the strength to go through those moments. I thank my grandmother, my parents, my sisters, and other family members for their constant

encouragement and support. I would like to share this accomplishment with them all.

A person cannot live alone in this world. I would like to thank all my friends who made my stay at Michigan State University very enjoyable and meaningful experience. In particular, I would like to thank my best friends, Prawit Sitirit, Sutthipong NaSongkhla, Piyarat Nanta, and Dr. Vanee Chonhenchob, for their friendship and their support. I would also like to thank: Wipada Wiwatana, Nudee Sermsrisuwan, Dr. Wimol Taoklam, Dr. Prabhas and Dr. Jaruloj Chongstitvatana, Dr. Rath Pichyakula, Prapassara Nilagupta, Annintita Vatcharasiritham, Sonatee Hongladaromp, Boonchoat Paosawatyanyong, John Hartzell, Paradorn Ramaboot, Pimsai Ramaboot, Eakachai Pavasiriporn, Uruchaya Sonchaeng, Chatpetch Saipetch, Patchrawat Uthaisombat, Poowanart Tianniam, Saharat Pongsree, Dawn Smith, Hiphop and Happy. The list can go on and on and still be far from being a complete one.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Chapter 1

Introduction

Scalable parallel computers (SPCs) have become the primary computing architecture to solve science and engineering problems, especially the so-called grand-challenge problems. Such systems are usually characterized by the distribution of memories among an ensemble of computing nodes. The nodes, each of which has its own processor, local memory, and other supporting devices, are typically interconnected by dedicated high-speed networks. Systems with hundreds or thousands of dedicated nodes connected by a dedicated network are usually called *massively parallel computers* (MPCs). These machines are among the most powerful SPCs currently available and have been reported to achieve the teraflops performance recently [1, 2]. The examples of MPCs are the TMC CM-5 [3], the NEC Cenju-3 [4], and the Cray-T3D [5]. Another promising computing platform to support the parallel processing is *network of workstations* (NOWs) [6]. Using the high-speed networks, such as ATM [7] and Myrinet [8], NOWs can deliver high performance with a fraction of MPCs cost. Even though the performance of NOWs are not comparable to the more powerful MPCs, NOWs are gaining more popularity because of its

excellent price-performance-ratio and its drastic performance improvement during the last few years.

1.1 Communication Services in SPCs

A common feature of those distributed-memory computing is to use explicitly or implicitly message passing to communicate and coordinate a group of participating processes. In shared-memory paradigm, users do not have to be concerned with message passing as systems implicitly use message passing mechanism to support shared-memory paradigm. Many parallel systems, however, support message-passing paradigm. In this case, users have to explicitly use communication services to communicate among processes. To support this paradigm, many communication libraries such as PVM [9], Chimp [10], and LAM [11], have been developed. Among them, PVM has received much attention and has been proven useful in programming many parallel applications. Although these communication libraries are widely used, the user interface for each library is quite different. To overcome this problem, MPI (*Message Passing Interface*) [12] has emerged as a standard communication library. The objective of MPI standardization effort is to enforce program portability and to achieve good performance across different distributed-memory computing platforms. While it is easy to support program portability, the program performance is still greatly dependent on the internal implementation of communication library, the underlying network protocols, and the network architecture.

Parallel programs usually facilitate two types of communication services: point-to-point and collective. A point-to-point communication, or known as *unicast*, involves two

communicating processes in the form of *send* and *receive* operations. This type of communication service is very fundamental to all communication subsystems, and its performance measurement techniques have been well-studied [13, 14, 15, 16]. Benchmark programs used to measure point-to-point communication performance include *ping* and *ping-pong*, where *ping* is mainly used to measure network throughput and *ping-pong* is mainly used to measure communication latency.

Collective communication which involves a group of processes is another frequently used communication pattern in parallel programs. This communication class is particularly useful to scientific applications which often require global data movement and global control in order to exchange data and synchronize the execution among processes. Providing such services can greatly simplify the programming effort and facilitate efficient implementation. For example, an iterative method can be used to approximate the solution for heat-transfered problem. The *global reduction* operation is useful to consider when the algorithm converges to the correct solution. Another example is *segmented scan* operation which can be used to perform prefix computation. Many of these collective communication services are defined in HPF [17] and MPI [12].

In general, collective operations are defined within a context called *process group* or *communication group* which specifies the domain or scope of participating processes in a communication operation. Collective operations can be classified into three subclasses: one-to-all, all-to-one, and all-to-all. The following describes each collective operation type and provides example operation based on MPI specification [12].

One-to-All

In one-to-all communication, one of the processes in the group becomes the sender sending messages to all other processes in the group (receiver). The sender can send same message to all receivers (*broadcast*) or distinct messages to each receiver (*scatter* or *personalized communication*. Figure 1.1(a) and 1.1(b) demonstrate the conceptual diagrams and the information flow diagrams of broadcast and scatter respectively. In each figure, each row of boxes represents data locations in one process. For example, in the broadcast, initially, P_0 contains data A_0. After the broadcast, all processes have it.

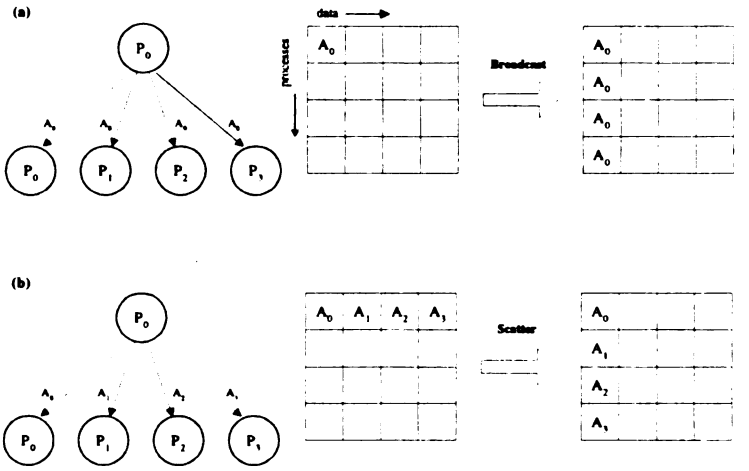


Figure 1.1: The one-to-all communication: (a) broadcast, (b) scatter.

All-to-One

All-to-one communication can be considered as an inverse of one-to-all operation. One process in the group is a receiver and all other processes are senders. An example of this communication is *gather* which allows the receiver to gather information from all other

processes. In addition, different messages from different senders may combine together to form a single message for the receiver. This operation is called *reduce*. The combining operator is usually commutative and associative such as addition, multiplication, maximum, etc. Figure 1.2(a) and 1.2(b) demonstrate the conceptual diagrams and the information flow diagrams of gather and reduce respectively.

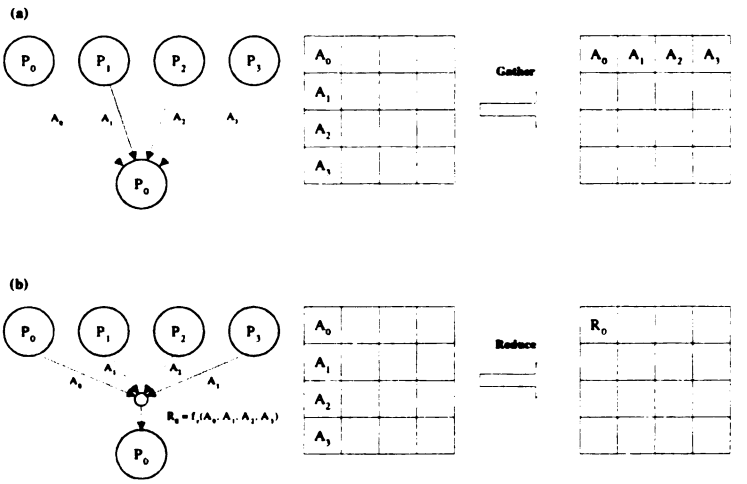


Figure 1.2: The all-to-one communication: (a) gather, (b) reduce.

All-to-All

In all-to-all communication, all processes in the group are both senders and receivers. Several operations are belong to this group such as *allgather* which is similar to gather but all processes in the group receive the result, *alltoall* which all processes perform scatter first and then gather results, a combined reduction and scatter operation (*reduce_scatter*), and *scan* operation which performs prefix computation across all members of the group. Figure 1.3(a) and 1.3(b) demonstrate the conceptual diagrams and the information flow diagrams of allgather and alltoall respectively.

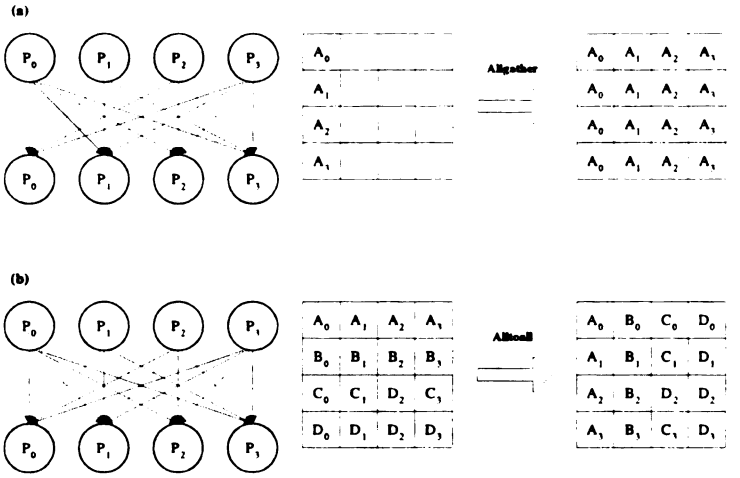


Figure 1.3: The all-to-all communication: (a) allgather, (b) alltoall.

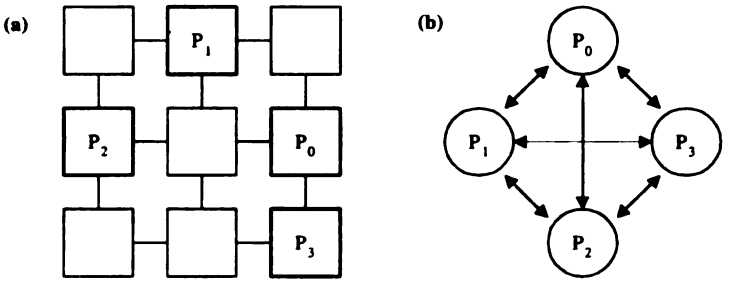


Figure 1.4: An example of a process group of 4 processes in 3 × 3 mesh (a) system's viewpoint, (b) process group's viewpoint.

It is important to distinguish between the *process view* and the *system view* of a collective operation. In a SPC, each individual application is usually allocated with a subset of processors in order to achieve the best performance and increase the system throughput. From the viewpoint of system or processors, a process group only involves a subset of processors. Consider an example of a process group of 4 processes in 3 × 3 mesh system as shown in Figure 1.4. From the process group's viewpoint, all processes in the group are fully connected. However, from the system's viewpoint, these processes are only allocated

in a subset of processors. When process 0 sends a broadcast message to all processes in the group, it performs multicast to a subset of processors. Thus, one-to-all operation in a process group is actually one-to-many operation from the system's point of view. For the purpose of system-level network performance evaluation, this study focuses on system-level multicast communication which is equivalent to an application-level broadcast within a process group. Furthermore, this study consider one process per processor. Therefore, the terms process, processor, and node will be used interchangeably.

1.2 Multicast Communication

Among those collective communication services, multicast communication is a frequently used communication pattern in many parallel applications. This operation can be invoked to deliver a message from a source (called *root*) to multiple destinations. Although users may call multiple sends to obtain similar functionality, providing such service can ease programming tasks and allow users to exploit hardware support and multicast algorithms fine-tuned specifically for the systems. In addition, multicast is a basic construct of several other collective communication services, including broadcast, scatter, and barrier synchronization [18]. Using efficient implementation of multicast service also improves the performance of the aforementioned services.

Multicast communication plays an important role not only in parallel applications, but also in other areas such as system protocols. For example, broadcast is used in write-invalidation for cache coherence protocol [19] and address resolution mechanism for TCP/IP [20]. Furthermore, new high-speed switches such as Myricom's Myrinet switch use broad-

cast to perform network topology mapping [8].

Recently, the multicast-related research has extended the usage of multicast service beyond scientific applications. Multimedia and wide area networks are among many applications which have received attention recently. Kompella *et al.* [21] recommend to use multicast routing to support high-bandwidth delay-sensitive applications such as video communication over computer networks. With development of IP multicast [22, 23, 24] and MBone [25], multimedia information can be delivered efficiently across the Internet as experimented in IEEE GLOBECOM'96 [26, 27]. It also creates several new research areas such as video-on-demand on the network [28], archive servers query [29], secure multicasting [30], and efficient world-wide-web information delivery [31].

1.2.1 Performance Evaluation

As multicast service has been used in a broad range of applications, comparing two multicast implementations can be a very difficult task. Some applications such as parallel programs require multicast implementations with low latency. In contrast, multimedia applications usually run on the system with limited resources such as wide area networks. To evaluate multicast implementations for this type of applications, we must also consider bandwidth consumption. Thus, there are two major criteria to evaluate multicast algorithms, *traffic cost* and *latency cost*.

Latency Cost

There are two popular approaches to evaluate the latency of a multicast algorithm. The first approach is to count the number of communication steps required to complete multicasting.

The communication step or *multicast step* is defined as a single send from the source to one destination. Consider an example of four-node multicast presented in Figure 1.5(a). In this example, node 0 is the root and three other nodes are the destinations. In the first step, node 0 sends a message to node 1. In the second step, node 0 sends a message to node 2 and node 1 sends a message to node 3. Note that since both node 0 and node 1 can distribute the message simultaneously, we consider both sends as one step. Thus, the cost of this multicast operation is 2 steps.

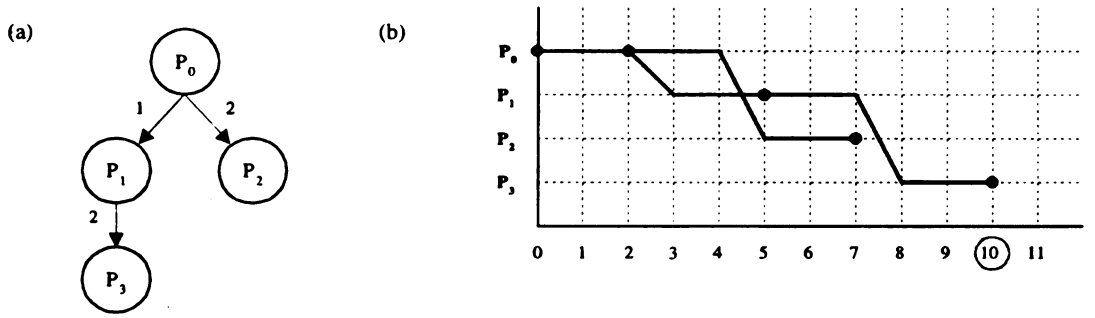


Figure 1.5: An example of how to evaluate multicast latency costs for a four-node binomial tree: (a) multicast steps, (b) multicast latency.

The other approach is to use the *multicast latency* to evaluate multicast implementations. The multicast latency is defined as the elapsed time between the root starts sending the message to the first node until the last node receives the message. Consider an example in Figure 1.5(b). Suppose a source has to wait for 2 time units after it sends a message to a destination before it can start sending to another destination. And let suppose it takes 5 time units to deliver a message. Obviously, node 0 starts sending to node 2 before node 1 finishes receiving which is quite contradicted to the evaluation based on multicast steps. Node 3 is the last node to finish receiving. Therefore, the multicast latency in this example

is 10 time units.

Traffic Cost

In the systems with limited resources such as wide area networks, bandwidth consumption of a multicast tree must be taken into consideration. Typically, each communication link in the network is associated with link bandwidth cost and link delay. Given a multicast tree in a network, its total bandwidth consumption is the sum of the bandwidth costs on the links in the multicast tree and its delay is the maximum delay from source to each destination. The least cost tree is called a *Steiner tree* and the problem of finding a Steiner tree is known to be NP-complete [32]. Thus, researchers use heuristic algorithms to construct low-cost multicast routes [33].

Optimizing the bandwidth consumption alone may not be sufficient to satisfy applications' requirements. For new applications such as multimedia, they are both delay and bandwidth sensitive. Thus, an efficient multicast tree must consume low bandwidth consumption and has bounded delay [33]. In addition, traffic concentration is also another metric needed to be considered since highly traffic concentration in some links due to multicast can create bottleneck to the system [34].

1.2.2 Implementations

One of the main focuses of multicast research is to find efficient implementations of multicast communication. Generally, we can categorize multicast implementations into two groups based on the hardware support requirements. The first group is multicast implementation with hardware support. In this group, there is a special hardware to support

efficient multicast communication. This support can be either a dedicated network or just additional support in data network. The second group is software-based implementation. In this group, a multicast service is implemented atop point-to-point reliable communication services. Both implementations have their pros and cons which are discussed next.

Hardware Implementations

Obviously, implementing multicast in hardware is much more efficient than software-based implementations As performance of multicast communication is quite critical to parallel applications, several MPC vendors have provided efficient multicast services in their hardware such as the TMC CM-5 [3] and the NEC Cenju-3 [4]. CM-5 uses the dedicated control network to perform multicast and other collective communication services. Unlike CM-5, the NEC Cenju-3 provides hardware-support for multicast in its MIN data network. However, these implementations have some serious limitations. For example, the deadlock due to multiple multicast is possible [35]. One possible solution used in CM-5 is to allow only one multicast at a time. However, the dedicated multicast network can be underutilized.

Due to its attractive price-per-performance ratio, Network of Workstations (NOWs) have been popular as replacements of MPCs to run parallel applications. The basic principal of NOWs is to provide low-latency interconnection by using high-speed switches. Thus, new high-speed switches such as DEC's Autonet, DEC's Gigaswitch, Myrinet, and ATM provide efficient multicast services in their switches. However, multicast services provided in most implementations are unreliable. Several works have addressed the solution of reliable software multicast on top of unreliable hardware multicast [36, 37, 38].

Software Implementations

Although hardware multicast can provide excellent performance, many MPCs and NOWs do not have special hardware support for multicast services. Without the hardware support, these systems adopt software-based approaches to implement multicast services atop existing point-to-point communication services. This approach has been widely use in new communication libraries such as MPI and PVM. In this approach, processes form a tree structure which is called a *multicast tree* where the sender is the root node and destinations are internal nodes and leaves nodes of the tree. Based on the multicast tree structure, messages are forwarded from the root nodes using point-to-point communication. Multicast trees can be simple such as a sequential tree and a binomial tree and more sophisticated such as parameterized tree [39] and λ-tree [40] whose structures are adjusted based on the underlying network parameters.

When designing a software multicast algorithm, the key issue is to consider the trade-off between performance and portability. To optimize performance, researchers usually design multicast algorithms to utilize some features of the underlying network. One popular optimization technique is to design multicast algorithm to efficiently utilize communication channels based on the underlying network topology. Sullivan and Bashkow propose the well-known *spanning binomial tree* (SBT) algorithm [41] for the first-generation hypercube which used store-and-forward switching. Johnsson and Ho propose the *edge-disjoint spanning tree* (EDST) algorithm [42] which utilized communication channels more efficient than SBT. Barnett *et al.* [43] study broadcast problem in one-port wormhole-switched mesh networks and propose the *recursive-splitting broadcast* algorithm, the edge disjoint

fences (EDF) algorithm, and the *scatter-collect* algorithm. In addition, several topology-based optimization researches also focus on eliminating internal network contention of messages in multicast trees. In most real networks, network contention is likely to occur if concurrent message transmissions are not scheduled properly. In this case, the actual multicast latency can be longer than expected since the contention can prolong the multicast operation. McKinley *et al.* [44] address the contention-free multicast problem for network with dimension-ordered routing. The proposed algorithms, U-cube and U-mesh, are based on the recursive doubling technique used in SBT. Using dimension ordering of a source and destinations, multicast trees generated by U-cube and U-mesh algorithms are guaranteed to be contention-free on hypercube and mesh topologies respectively. Based on the same technique, Xu and Ni proposed U-min which is a contention-free multicast algorithm on bi-directional multistage interconnection networks (BMINs) with turn-around routing [45]. Their technique is based on lexicographical ordering to construct a contention-free multicast tree. Another popular optimization technique is to exploit the multi-port capability of the underlying network architecture. McKinley and Trefftz [46] propose the *Double Tree* (DT) algorithm for all-port wormhole-switched hypercubes. Ho and Kao [47] propose a heuristic algorithm which performed better than the DT algorithm. Robinson *et al.* [48] generalize the problem and studied multicast algorithms for all-port wormhole-switched hypercube. Tsai and McKinley [49] propose the *extended dominating node* (EDN) model to support multicast on multi-port wormhole-switched mesh network. McKinley *et al.* [50] provide an excellent survey of efficient multicast implementation on wormhole-switched MPCs.

Traditional models for message-passing systems assume a telephone-like communica-

tion medium between processors [51]. Under this assumption, the source always waits until the destination completes receiving the sending message before the source resumes its execution. In this case, multicast algorithms based on the recursive doubling technique requires the least amount of time to complete. However, several new communication models such as the Postal model [51] and the LogP model [52] indicate that by using techniques such as sending buffer and DMA, the source usually can resume its execution before the message is actually delivered. Thus, most multicast algorithms mentioned earlier which are based on the recursive doubling technique are not always optimal in practice. As a result, many recent studies have been focused on portable multicast algorithms. A portable multicast algorithm is usually based on a generic communication model which carries a sufficient number of critical system parameters to characterize the important features of different parallel architectures. The basic idea of the portable multicast algorithm is to generate a multicast tree based on the pre-determined parameters during the runtime. Since the parameters are measured from the real parallel systems, multicast algorithms based on this approach can generate a suitable multicast tree for the underlying system. Thus, the algorithms are both portable and efficient. Karp *et al.* [53] study optimal broadcast and other collective algorithms for the LogP model. Their algorithm is quite fast and simple and can guarantee the optimality when network parameters are fixed. Due to its simplicity, it is quite difficult to apply their algorithm to other complicated scenario such as when the other network traffics are taken into consideration. Bar-Noy and Kipnis propose multicast algorithms based on the postal model [51]. However, there are some limitations in the postal model such as the critical parameter λ is assumed to be an integer which is not practical. Bruck *et al.* [40] study several practical issues to implement broadcast and global com-

bine using the postal model. Although their works provide useful suggestions to overcome several limitations in the postal model, the proposed λ-tree broadcast algorithm is still too complicated to implement.

1.3 Motivation and Problem Definition

There are several open issues related to multicast service. First, to evaluate the performance of multicast communication accurately, we need to address two important issues. The first issue is to identify the most representative metrics. The chosen metrics must be able to characterize the actual overhead of the multicast service when it is invoked in applications. Moreover, the definition must be well-defined. The other important issue is to use correct measurement techniques. One common mistakes in performance evaluation is using incorrect measurement techniques. The results from incorrect techniques can be very misleading. Thus, it is necessary to verify that the techniques are actually measuring the desirable metrics.

Second, most of the current multicast implementations suffer several drawbacks. Generally, the design may not be scalable to large-scale systems. Many designs are platform-dependent and thus not portable to other platforms. For those platform-independent designs, the performance varies significantly from platform to another. To address these drawbacks, we focus on the efficient implementations of multicast communication based on an abstract parallel platform called the *parameterized* communication model which is the extension of the LogP model [52]. On one hand, our implementation is portable since it is based on the parameterized model which is an abstract model. On the other hand, our

implementation can perform well on a wide range of platforms since an optimal multicast tree is constructed based on a small set of easily measurable parameters which can be considered during the compilation phase. To further improve the performance of the multicast communication, the research is further extended to study the fine-tuning of the multicast algorithms by considering the architecture-dependent characteristics of the systems.

Third, most studies mentioned earlier have been done in the ideal condition where the effects of network traffic from other nodes are negligible. Although some studies measured results in the actual network, the results might not actually include the effect from the background network traffic. During the measurement, the processor-allocation schemes might guarantee contention-free or the measuring technique might focus only on the performance of multicast with no interference from other network traffics. For example, the effect of other network traffic can be eliminated by measuring performance of a multicast tree several times and choosing the minimum value. Thus, the performance of so-called optimal multicast algorithms when perform in the actual network may not be better or even worse than other conventional multicast algorithms such as binomial and sequential multicast trees. To address this problem, we study the effect of other network traffic on the multicast behavior.

1.4 Objectives and Thesis Outline

The primary objective of this research is to study the efficient software-based implementations of multicast communication for SPCs. As we intend to optimize our multicast algorithms for parallel programs, our main focus is to *reduce the overhead* of a multicast

service. In addition, our multicast algorithms must be *flexible* such that it is portable and also achieve good performance at the same time. Our algorithms must be *tunable* to exploit some characteristics of the underlying network. Finally, our algorithms must perform reasonably well when there are more than one application sharing the system.

SPCs usually have different configurations and features. To simplify our work, the platforms under investigation are assumed to have some common features. We assume that there is only one user process running on each processor and the workload on each processor follows the SPMD programming model. In addition, we assume that the underlying platform provides only reliable point-to-point communication services. We assume no special hardware support designed specifically for multicast. Although it is obvious that hardware support for multicast can greatly enhance multicast performance, these hardware support are not available in most platforms. We believe that software-based approach is quite flexible and scalable to large-scale systems.

In Chapter 2, we introduce a point-to-point communication model called the parameterized model which serves as the basic model throughout this study. Unlike other generic models, the parameterized model is designed specifically to model the one-to-many communication pattern and, hence, it contains only necessary parameters to model the one-to-many communication. Therefore, constructing an optimal multicast algorithm based on this model is practical since it is not too complicated We also state our assumptions, discuss how to measure the network parameters, and present benchmarking results from the IBM/SP1.

In Chapter 3, we discuss our multicast communication model. We explain our collective communication flow model which is the basic model for our multicast benchmark. We

then discuss the pitfalls of very-popular ping-based benchmarking. Then we present our approach and measurement results from the IBM/SP1.

In Chapter 4, we introduce techniques to find optimal multicast trees based on the parameterized model. We first explain the basic concept of constructing an optimal multicast tree which is based on the dynamic-programming approach. Although this approach can guarantee to find an optimal tree, its running time which is $O(k^2)$ (where k is the size of multicast tree) is not practical to use in a real implementation. With a simple observation, we can improve the running time to $O(k)$. This leads to our main algorithm, the *OPT-Tree* multicast algorithm. We present and compare experimental results of running three different multicast trees on the IBM/SP1.

In Chapter 5, we investigate architecture-dependent tuning of the OPT-Tree multicast algorithm. We propose two multicast algorithms, *OPT-Mesh* and *OPT-Min* which are optimized for wormhole-switched mesh networks and BMIN networks. Using flit-level simulators, we compare the performance of both algorithms and two other well-known network-dependent algorithms based on the binomial tree.

In Chapter 6, we use simulators to study the multicast behavior in various environments. The parameters include background traffic, topology, software overhead, and grain size. Based on the results, we introduce a new class of multicast algorithm called the *dynamic multicast* algorithm. This class of multicast tries to improve the performance by dynamically adjusting its structure based on the current network workload. In Chapter 7, we briefly review some related works and discuss the difference between our work and theirs. Finally, the concluding remarks as well as some possible future research directions are discussed in Chapter 8.

Chapter 2

Point-To-Point Communication Model

Point-to-point communication is an essential component in a message-passing system. This type of communication services usually involves two communicating processes in the form of *sending* and *receiving* primitives. As these primitives are available on most message-passing systems, they are typically facilitated as building blocks to construct more complicated communication services such as a software-based multicast service. Thus, the key for modeling software-based multicast service is to understand the cost of these point-to-point communication primitives.

Recent models of parallel architecture such as the postal model [51] and the LogP model [52] emphasize the importance of communication overhead. Still, these models are not quite suitable to model a multicast communication. The postal model uses λ as the only parameter to characterize the underlying network. λ is assumed to be constant for each system. This is not realistic since λ, in fact, depends on message size. The LogP model uses four parameters to identify a parallel system. This model works particular well for short messages but insufficient to model long messages. Later, the LogGP model [54]

has been proposed to address this limitation. Since this is a general-purpose model, it contains some parameters which is not necessary to model a multicast communication.

In this study, we introduce *the parameterized model* which is derived from the LogP model. This model is specifically aimed to be a basis for multicast communication modeling. It is simple and carries sufficient number of parameters to characterize the underlying parallel architecture. As it focuses on the communication at the application level, it includes all software communication overheads. Note that this model can also be used to model other one-to-many communication services such as scatter.

2.1 Assumptions

To simplify our model, we make a few assumptions regarding to the underlying parallel platform. As demonstrated in Figure 2.1, we assume that the platform being modeled consists of processors or nodes connecting to interconnection network. The underlying network topology is not specific. However, based on a node's viewpoint, the topology is assumed to be logically fully connected. In other words, the underlying topology is assumed to be similar to crossbar-switched. Thus, there is no contention when several nodes send messages to distinct destinations at the same time. [1]

We also assume that there is one application process running on each processor. This assumption eliminates the need to worry about context switching overhead when several user processes share the same processor. In fact, some communication libraries such as MPI-F [55] employ this strategy in order to implement the entire communication stack in

[1] We address the problem when this assumption is not true in Chapter 5.

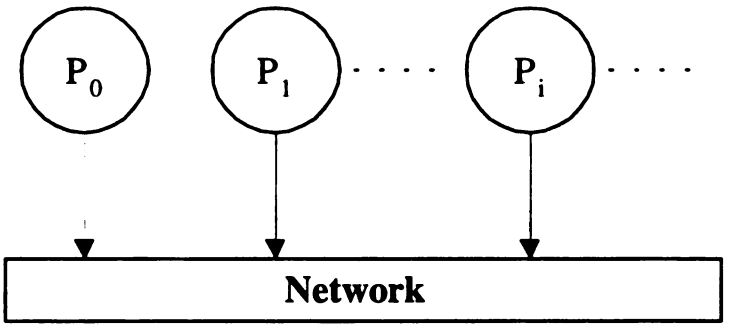


Figure 2.1: The abstract communication model.

user space and also avoid additional overhead due to security checking and packets multiplexing among different processes. Furthermore, we assume that the underlying communication subsystem provides only reliable blocking send and blocking receive point-to-point communication primitives. These two primitives are equivalent to MPI's standard mode send/receive functions (*MPI_Send* and *MPI_Recv*). According to MPI's specification, MPI_Send is considered complete when user's sending buffer can be reused and MPI_Recv is considered complete when the whole message has been moved to user's receiving buffer. Thus, when MPI_Send is returned, it does not guarantee that the destination has already received the message.

2.2 The Parameterized Model

The overhead of sending a message between two nodes involves three parameters including *sending latency* (t_{send}), *receiving latency* (t_{recv}), and *network latency* (t_{net}), as illustrated in Figure 2.2. t_{send} is the software latency in processing the sending message at the sender which includes the overhead of protocol processing, checksum computing, and, possibly,

memory copying. t_{recv}, similar to t_{send}, is the software overhead at the receiver. t_{net} is the time required to transmit the message across the network. In general, several significant factors contribute to t_{net}, including network bandwidth, underlying switching mechanism, and blocking time which is mainly due to network contention. In order to measure the network latency accurately, benchmarking must be conducted in a controlled environment such that the effect of network contention due to other unrelated messages can be avoided.

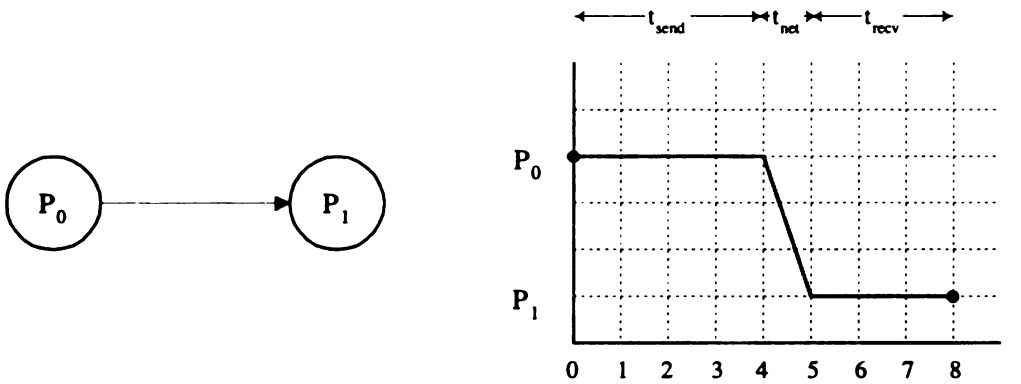


Figure 2.2: The basic point-to-point communication model.

Measuring the values of these three parameters are rather difficult since they are not application-level parameters. To get the accurate measurement, some special techniques, such as using hardware monitor [16] and software probe [15], are required. In addition, it is impractical to use these three parameters to model a multicast service. In multicast communication, there is only one sender (root node) and the other nodes in the group are receivers. Intuitively, after a receiver completely receives a multicast message, it can help the root node propagate the message to other receivers. To make the propagation efficient, all participants usually form a tree-like structure to dictate the ordering of the communication. Based on the multicast tree structure, each node uses the point-to-point

communication service to forward a message to each of its children in turn. We can make some observations from this implementation technique.

- A node may send to several nodes consecutively.
- A node, except the root node, receives from its parent first before sending to other nodes.
- A node, except the root node, has only one parent and receives a message from its parent only.

The first observation indicates that the model must include the latency between two consecutive sends. The second observation indicates that the model must include the end-to-end latency. And the last observation tells us that the model does not have to include the overhead due to the contention at a receiver. Based on these observations and the aforementioned assumptions, the basic model for multicast communication needs at most two parameters, *the holding latency* (t_{hold}) and *the end-to-end latency* (t_{end}). t_{hold} is the minimum time interval between two consecutive send or receive operations. t_{end} is the elapsed time between the time the sender starts sending a message and the time the receiver finishes receiving. Obviously:

$$t_{end} = t_{send} + t_{net} + t_{recv}$$

Figure 2.3 illustrates the relationship of the *application-level* parameters (t_{hold} and t_{end}) and *system-level* parameters (t_{send}, t_{net}, and t_{recv}). Note that the value of t_{hold} depends on how blocking send is implemented. Based on MPI's specification, a blocking send is returned when the user's sending buffer can be reused. Some implementations use tech-

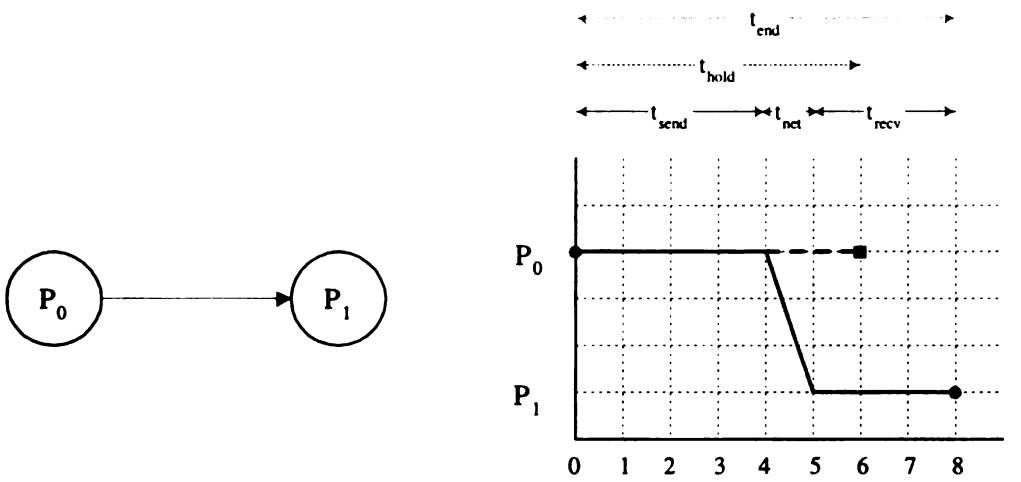


Figure 2.3: The point-to-point communication model.

niques, such as separate buffering, DMA, and dedicated communication processor, to hide some sending overhead from the sending processor. In this case, t_{hold} is less than t_{send}. In some implementations, however, a blocking send is returned when the receiver has received the message. This implementation still conforms to MPI's definition of standard mode sending operation (MPI_SEND) as it is guaranteed that, upon returning, user's sending buffer can be reused. In this case, t_{hold} can be greater than $t_{send} + t_{net}$ since t_{hold} may include the overhead of the acknowledgment from the receiver.

For a given network architecture, both application-level and system-level parameters usually depend on the size of sending/receiving message. Theoretically, when sending a message size m, all parameters can be decomposed into two components, startup latency and delay latency [44]. The startup latency is a fixed overhead of sending/receiving the first byte of the message. For example, it can be the cost of resolving destination's address, DMA setup, and buffer allocation. We use the delay latency to represent the per-byte cost

of sending/receiving the rest of the message. Thus:

$$
\begin{aligned}
t_{send}(m) &= t_{ss} + m.t_{sd} \\
t_{net}(m) &= t_{ns} + m.t_{nd} \\
t_{recv}(m) &= t_{rs} + m.t_{rd} \\
t_{hold}(m) &= t_{hs} + m.t_{hd} \\
t_{end}(m) &= t_{es} + m.t_{ed} \\
&= (t_{ss} + t_{ns} + t_{rs}) + m.(t_{sd} + t_{nd} + t_{rd})
\end{aligned}
$$

Note that t_{ns} is a fixed delay incurred at the sender when it prepares to transmit a message across the network. And t_{nd} is the cost of transmitting one byte over the network.

2.3 Benchmarking

Measuring the application-level parameters (t_{hold} and t_{end}) are quite easy and can be done at the application level. We use $Ping$ and $PingPong$ benchmarks to evaluate t_{hold} and t_{end} respectively. Unlike the application-level parameters, we cannot measure the system-level parameters accurately without special devices. However, their values can be estimated and will be used when we study more realistic model with simulation. The estimation technique will be discussed later.

2.3.1 Ping

The purpose of Ping benchmark is to measure (t_{hold}) by sending messages from one node to another. In this benchmark, the sender always tried to send as soon as it can and continuously sends messages to the receiver. In other words, when the current send is done, the next send is issued immediately. The communication model is shown in Figure 2.4.

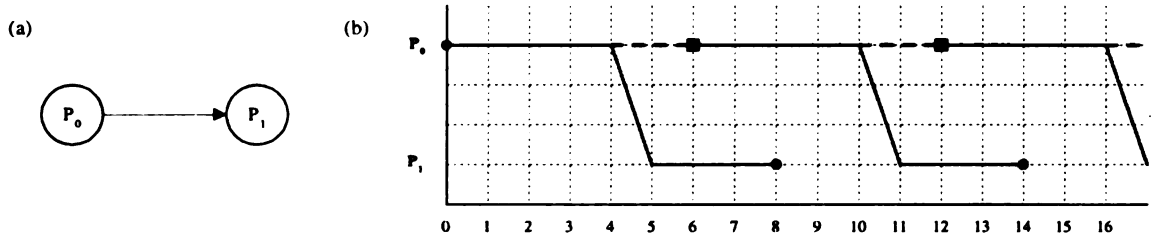


Figure 2.4: The Ping communication benchmark: (a) the model, (b) the timing diagram where $t_{send} = 4, t_{net} = 1, t_{recv} = 3$, and $t_{hold} = 6$.

The algorithm for Ping is rather straightforward. At the sender, each send is considered to be one iteration of the Ping communication. In practice, we may not be able to measure the elapsed time of a single iteration because the system clock resolution may not be fine enough. One solution is to measure the elapsed time of k iterations, and compute the average delay accordingly. The algorithm of the Ping benchmark is shown in Algorithm 2.3.1. Figure 2.4(b) shows the timing diagram of running a Ping benchmark on a hypothetical system, where $t_{send} = 4$, $t_{net} = 1$, $t_{recv} = 3$, and $t_{hold} = 6$.

Algorithm 2.3.1 Ping

Sender(P_0):
begin
  t_{start} := GetTime();
  for i := 1 **to** k **do**
    Send(message);
  endfor;
  t_{hold} = (GetTime() - t_{start})/k;
end.

Receiver(P_1):
begin
  for i := 1 **to** k **do**
    Recv(message);
  endfor;
end.

Note that the presence of other applications in the system may effect the results from the benchmarks. To eliminate this problem, we repeat the benchmarks several times and use the minimum values.

2.3.2 PingPong

The PingPong benchmark is aimed at measuring t_{end} by sending a message back and forth between two nodes. Unlike Ping, each node takes turn to become a sender and there is only one sender at a time.

Figure 2.5(a) shows the communication model of PingPong. Initially, P_1 sends a message to P_2 and waits for a returned message from P_2. When P_2 receives a message, it replies back to P_1. This is one iteration of PingPong communication. Similar to Ping, we measure the elapsed time of the k-iteration of PingPong communication and then find the average

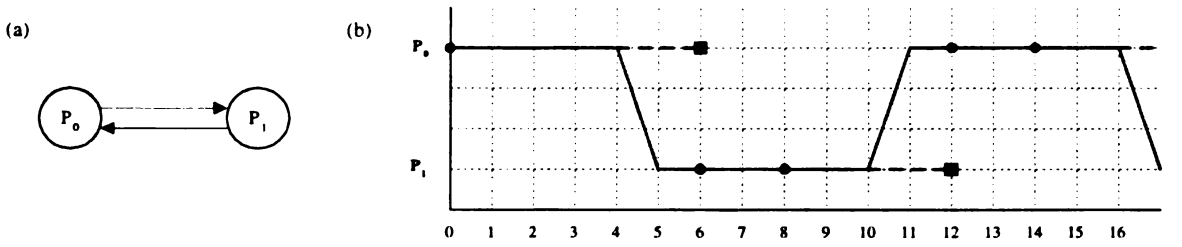


Figure 2.5: The PingPong communication benchmark: (a) model (b) timing where $t_{send} = 4, t_{net} = 1, t_{recv} = 3$, and $t_{hold} = 6$.

of the delay. The algorithm of the PingPong benchmark is shown in Algorithm 2.3.2. Figure 2.5(b) demonstrates the timing diagram of the PingPong benchmark. Obviously, the measured latency represents $2 \times t_{end}$.

Algorithm 2.3.2 PingPong

P_0:
begin
  t_{start} := GetTime();
  for i := 1 **to** k **do**
    Send(message);
    Recv(message);
  endfor;
  t_{end} = (GetTime() - t_{start})/(2 × k);
end.

P_1:
begin
  for i := 1 **to** k **do**
    Recv(message);
    Send(message);
  endfor;
end.

2.3.3 Estimating System-Level Parameters

As mentioned earlier, measuring the system-level parameters is quite difficult. However, since these parameters are used in our simulation studies, estimated values can be used. The technique presented here is quite similar to the technique used by Culler *et al.* in [56]. Although their techniques are designed for quantifying the LogP parameters on the Active Messages communication layer, we can apply their techniques to evaluate our system-level parameters.

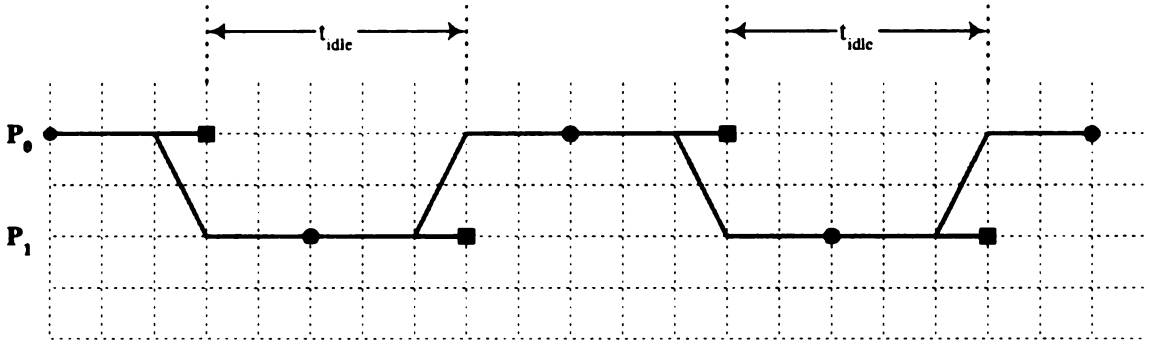


Figure 2.6: The idle periods in the PingPong benchmark.

To simplify the estimation process, we assume that the values of t_{hold} and t_{send} are the same. This is based on the fact that their values are quite close in most platforms. To estimate t_{net}, we consider the timing diagram of our PingPong benchmark in Figure 2.6. For each PingPong iteration, node P_0 idles after sending a message before receiving a reply from its peer (P_1). The idle period (t_{idle}) is $t_{net} + t_{end}$. Thus, if we can estimate t_{idle}, we can estimate t_{net} as t_{end} can be measured by using the PingPong benchmark. To estimate t_{idle}, we modify our PingPong benchmark by inserting a dummy loop of some harmless computations after P_0 sends a message as shown in Algorithm 2.3.3. Let t_{dummy} be the delay due to the dummy loop. Note that the relationship between the number of iterations

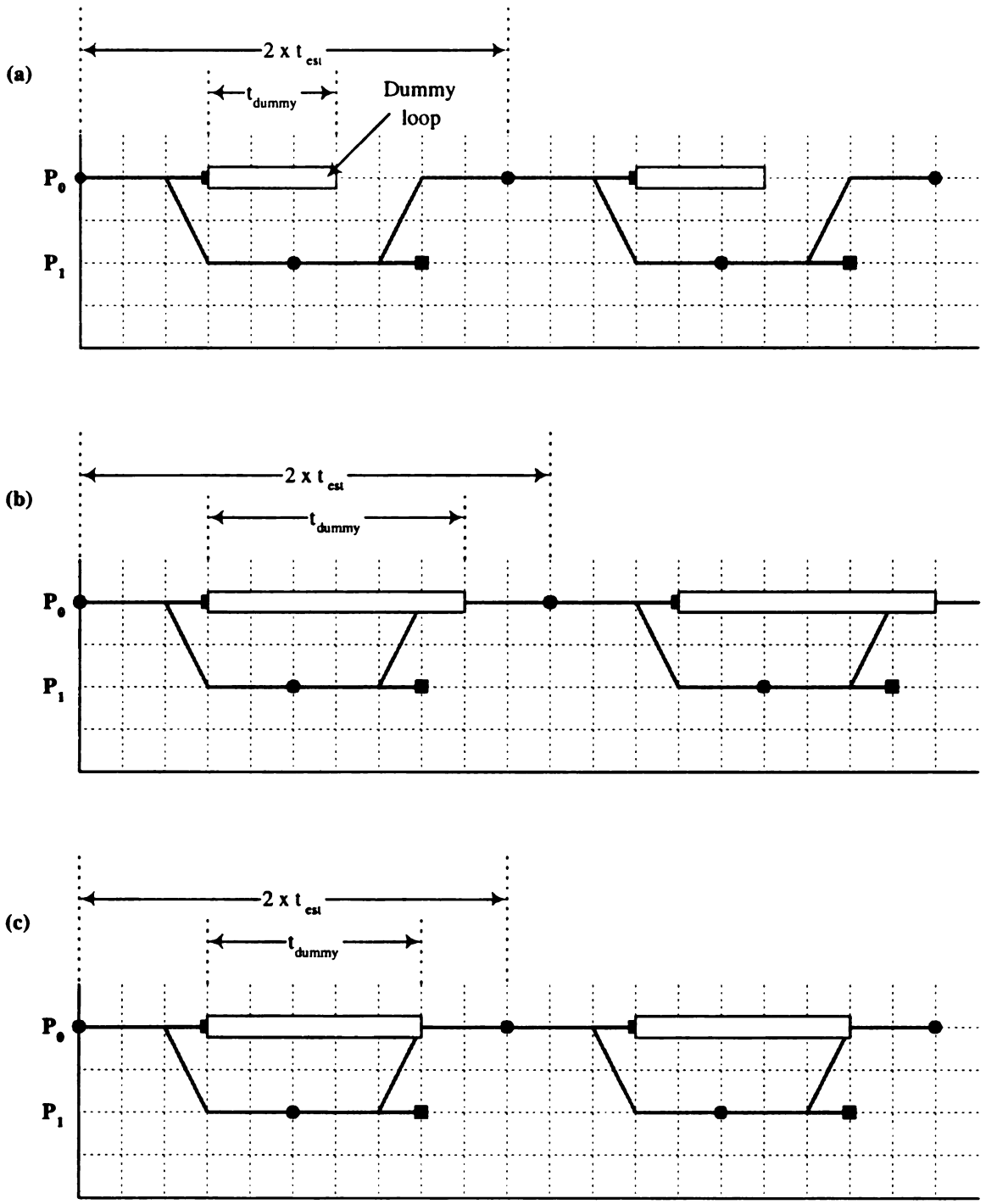


Figure 2.7: The effect of the dummy loop in the system-level parameters estimation algorithm: (a) $t_{dummy} < t_{idle}$, (b) $t_{dummy} > t_{idle}$, (c) $t_{dummy} = t_{idle}$.

in the dummy loop and t_{dummy} can be easily predetermined. As demonstrated in Figure 2.7, there are three possible cases: $t_{dummy} < t_{idle}$, $t_{dummy} > t_{idle}$, and $t_{dummy} = t_{idle}$. In the first case, the time measured (t_{est}) by this benchmark is still t_{end}. As we increase t_{dummy}, t_{est} remains unchanged since the delay due to the dummy loop is still shorter than the idle period. When $t_{dummy} > t_{idle}$ as shown in the second case, t_{est} increases. If we can find the maximum t_{dummy} before t_{est} increases as shown in the last cases, we can estimate t_{idle}. Thus, $t_{net} = t_{idle} - t_{end}$. Since we know t_{send}, t_{net}, and t_{end}:

$$t_{recv} = t_{end} - (t_{send} + t_{net})$$

Algorithm 2.3.3 System-Level Parameters Estimation

```
P0:
begin
   t_start := GetTime();
   for i := 1 to k do
         Send(message);
         Dummy loop(t_dummy);
         Recv(message);
   endfor;
   t_est = (GetTime() - t_start)/(2 × k);
end.

P1:
begin
   for i := 1 to k do
         Recv(message);
         Send(message);
   endfor;
end.
```

2.4 Experiments

We conducted the experiments on the 128-node IBM/SP at Argonne National Laboratory. Each node has an IBM/RS6000 processor (62.5 MHz) with 128 MB memory and 1 GB local disk. The peak performance is 125 MFlops per node. The communication library is MPI-F library version 1.41 [55]. Each data point in our results is the minimal value of 1000 measurements to reduce the overhead due to network contention from other programs. [2] However, if the sustained performance is desirable, the average values can be used instead.

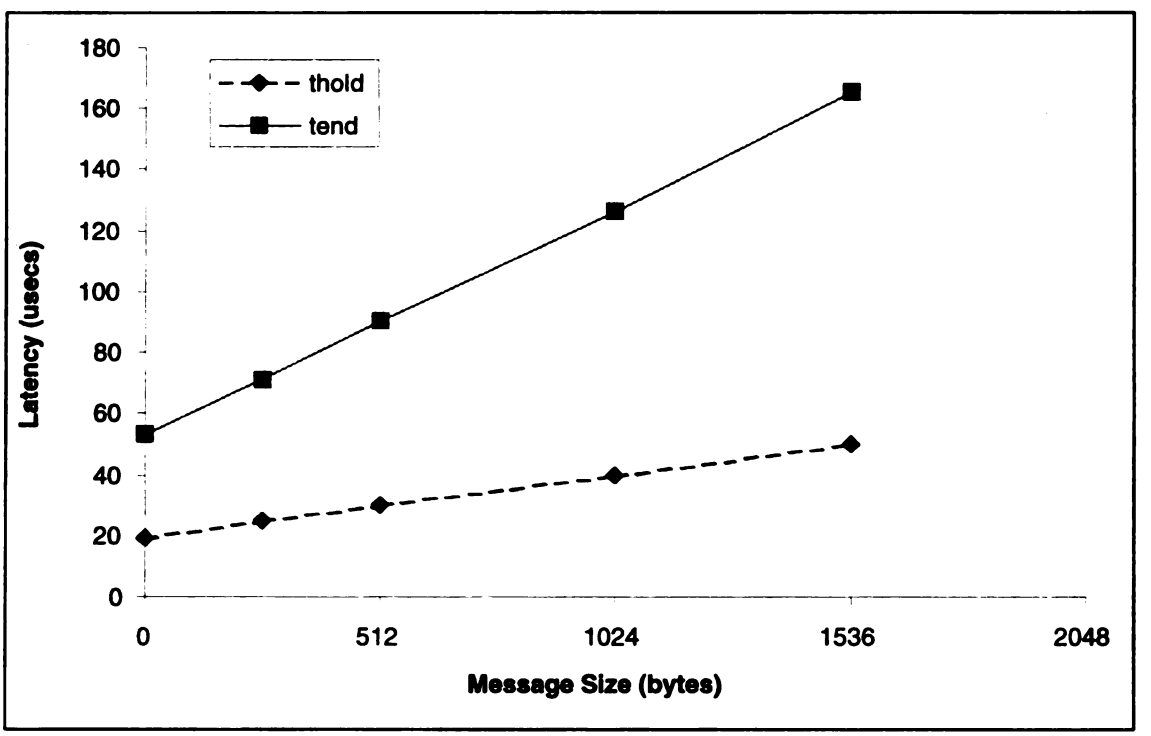


Figure 2.8: The results from Ping and PingPong benchmarks.

Figure 2.8 presents the results from Ping and PingPong benchmarks from our testbed system. The results indicate that for a message size m, t_{hold} and t_{end} of IBM/SP is $19.150+$

[2] It is difficult to reserve the whole machine for our measurement.

$0.02 * m$ and $53.295 + 0.07 * m$ respectively. As expected, these two parameters are dependent to the message size. This is mainly due to memory copying and checksum computing overheads.

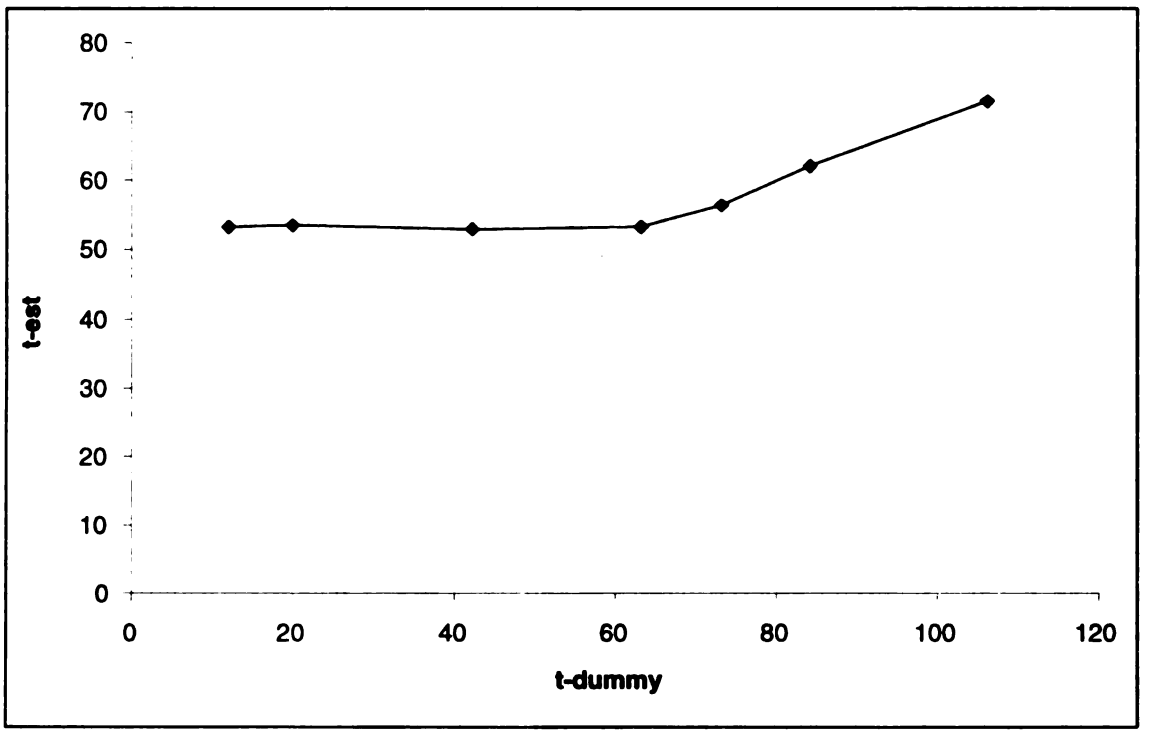


Figure 2.9: The relationship of t_{dummy} and t_{est}.

To measure the system-level parameters, we first determine t_{net} by estimating t_{idle}. As explained in Algorithm 2.3.3, we vary the value of t_{dummy} and observe where the value of t_{est} begins to rise. Figure 2.9 shows the results when testing with 1-byte messages. When t_{dummy} is small, t_{est} remains almost approximately 53.3 μsecs. Once we increase t_{dummy} to be more than 67.2 μsecs, t_{est} begins increasing proportionally. Thus, t_{net}(1-byte) is approximately $67.2 - 53.3 = 13.9$ μsecs. Note that this value is just an estimated value since we cannot precisely control the value of t_{dummy}. However, we project the value when the value of t_{est} increases. Based on this technique, t_{net} for IBM/SP is $13.900 + 0.02 * m$.

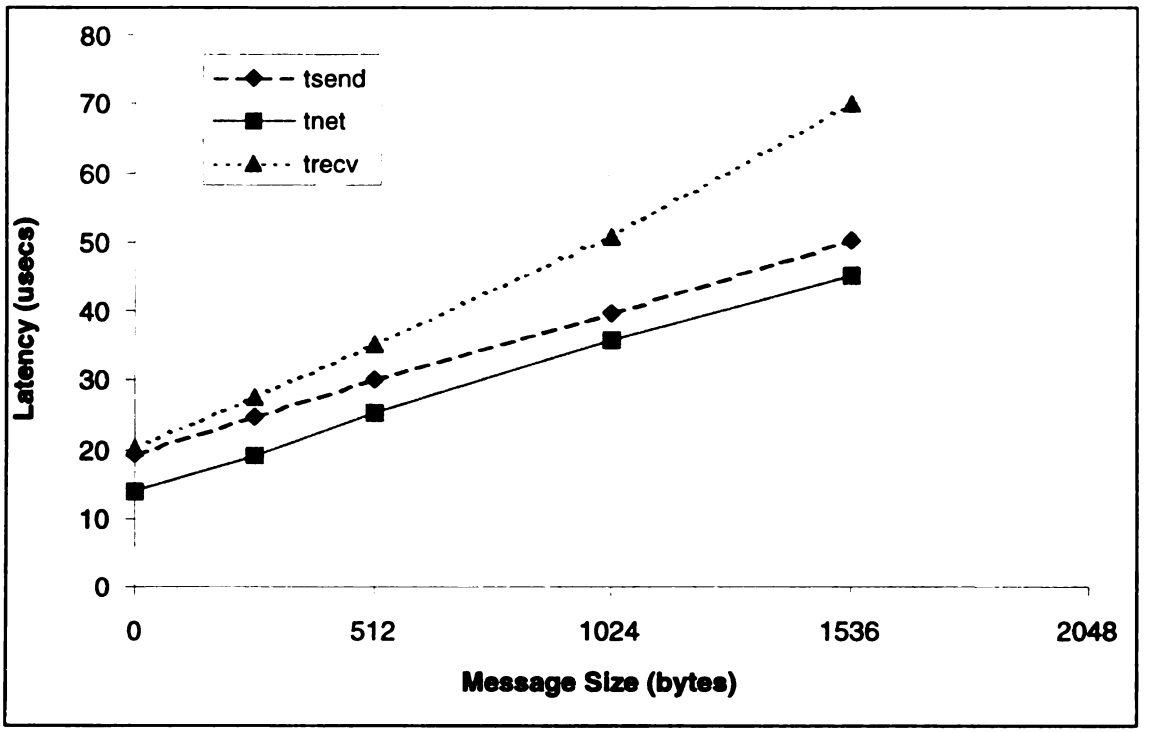


Figure 2.10: The results of estimating the system-level parameters.

Thus, we can calculate t_{recv} which is $20.245 + 0.03 * m$. The results for system-level parameters are shown in Figure 2.10.

2.5 Conclusion

Understanding the cost of point-to-point communication services is very crucial in modeling software-based multicast communication. In this chapter, we have discussed the model for point-to-point communication which will be used as a basis throughout this work. Unlike other models, the parameterized communication model are designed specifically for modeling multicast. The main goal of this model is to include only necessary and sufficient parameters to characterize the underlying network architecture. There are two types

of parameters: application-level parameters and system-level parameters. The application-level parameters are used in modeling multicast and system-level parameters are used in our simulation studies.

In addition, we have presented how to evaluate the network parameters. The application-level parameters are quite easy to measure and can be done at the user level. On the other hand, the system-level parameters are slightly more difficult to evaluate, especially when accuracy is desirable. However, in our work, it is sufficient to just estimate their values. We have presented the estimation techniques for the system-level parameters. To demonstrate our benchmarking techniques, we conducted experiments on the IBM/SP at Argonne National Lab and the results are summarized in Table 2.5.

Parameters	Latency
t_{hold}	$19.150 + 0.02 * m$
t_{end}	$53.295 + 0.07 * m$
t_{send}	$19.150 + 0.02 * m$
t_{net}	$13.900 + 0.02 * m$
t_{recv}	$20.245 + 0.03 * m$

Table 2.1: The communication parameters of the IBM/SP at Argonne National Lab.

Chapter 3

Multicast Communication Model

Multicast involves the communication of a group of nodes. One of the nodes, identified as the *root node*, is the sender. The other nodes in the group receive a message from the root node. Theoretically, all participants (including the root node) are assumed to complete a multicast operation at the same time. However, this is not always true. In this study, the definition of multicast is based on the MPI specification. A node in the group is considered to *start participating* a multicast communication service when it calls the multicast communication subroutine (*MPI_Bcast*). When the call is returned, that node has received the message and is considered to *finish participating* the multicast operation. The multicast operation is considered complete when all nodes have received the messages and returned from the subroutine. Based on this definition, all nodes may not finish participating at the same time. Moreover, some nodes might not have started participating yet. This makes multicast benchmarking even more difficult.

To evaluate the performance of multicast communication accurately, we need to address two important issues. The first issue is to identify the most representative metrics.

The chosen metrics must be able to characterize the actual overhead of the multicast service when it is invoked in applications. Moreover, the definition must be well-defined. The other important issue is to use correct measurement techniques. One common mistakes in performance evaluation is using incorrect measurement techniques. The results from incorrect techniques can be very misleading. Thus, it is necessary to verify that the techniques are actually measuring the desirable metrics.

To understand our performance evaluation techniques, it is important to understand the basic behavior and implementation techniques of multicast communication. As mentioned in Chapter 2, most software-based multicast implementations are based on the concept of *multicast trees* which is discussed in the next section.

3.1 Multicast Trees

Most of the multicast communication are implemented in software in which the nodes in the communication group form a tree-like structure to dictate the ordering of the communication. Based on the multicast tree structure (or *multicast tree*), each node uses the point-to-point communication service to forward a message to each of its children in turn. Each turn, when a node sends a message, is usually referred to as a *multicast step*.

Let consider examples shown in Figure 3.1. The first example is a *sequential tree* or known as *separate addressing*. In this approach, the root node (P_0) sends a separate message to each of the three nodes in turn. This approach was used to implement the multicast function, *Xmsend*, in the Symult 2010 [57]. Figure 3.1(b) is a well-known *binomial tree* based on the recursive-doubling technique [41]. In this approach, the number of nodes that

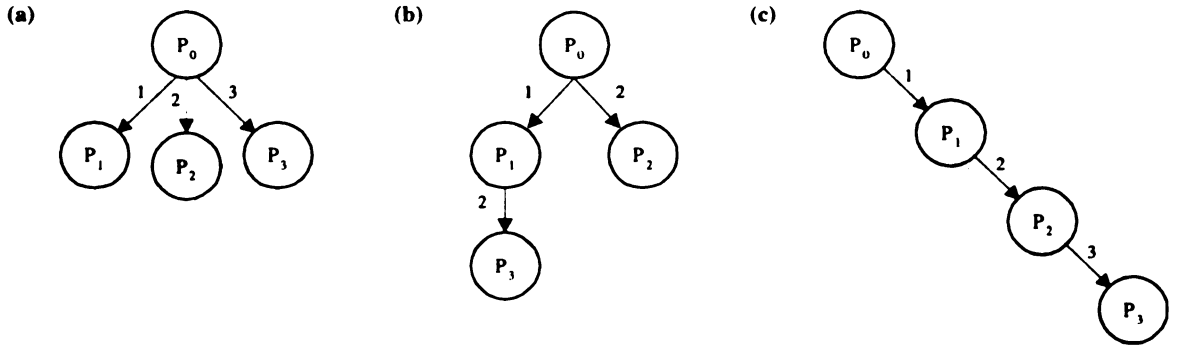


Figure 3.1: Three basic multicast tree structures: (a) the sequential tree, (b) the binomial tree, and (c) the chain tree

have already received the message increases by a factor of two after each multicast step. In Figure 3.1(b), the root node (P_0) sends a message to its first child (P_1) in the first step. In the second step, both P_0 and P_1 send messages to the other two nodes, P_2 and P_3. The last example is a *chain tree*. As shown in Figure 3.1(c), the root node (P_0) sends a message to its first child (P_1) in the first multicast step. In the second step, P_1 forwards the message to its successor, P_2, in the chain. The multicast operation is done when the message is completely forwarded to the last node in the chain, which is P_3 in this example.

3.2 Performance Metrics

The multicast step had been widely used as a metric for multicast communication for a long time because it is quite easy to evaluate. Theoretically, the performance of a multicast tree is dependent on the number of multicast steps required which is a function of the group size. For a group size of k, the sequential tree and the chain tree require $k - 1$ multicast steps, and the binomial tree requires $\lceil log_2 k \rceil$ multicast steps. According to the number of multicast steps required, the binomial tree is always the best.

In practice, the multicast step is not a good metric. This is because the multicast step can truly represent the performance of multicast when the communication is done in a lockstep fashion or a *rendezvous* strategy [58]. In other words, the sender waits until the receiver finishes receiving the message before it resumes execution. In this case, t_{hold} must equal to t_{end}. However, as discussed in Chapter 2, using communication buffer and intelligent network interface allows the sender to resume execution without having to wait for the message to be completely delivered. Thus, a tree which requires fewer multicast steps may actually perform worse than a tree which requires more steps. Consider the examples in Figure 3.1,. Suppose we assume $t_{send} = 2$, $t_{net} = 1$, $t_{recv} = 2$, and $t_{hold} = t_{send}$. By taking into account the sending and receiving latencies, the sequential tree shows the best performance. Therefore, in this study, the elapsed time between the multicast is issued until the last node receives the message is used as the performance metric for a multicast communication service. We refer this elapsed time as the *multicast latency* (t_{mcast}). In Figure 3.1, t_{mcast} of the sequential tree, the binomial tree, and the chain tree are 9, 10, and 15, respectively. Obviously, the sequential tree is the best.

One may argue that the metric should not consider the completion time of the last node because as soon as a node finishes participating, it can continue execution immediately. However, this is true only when the load can be dynamically balanced during the runtime. In typical data parallel programs, such as HPF, the workload is evenly-distributed among nodes using data distribution directives, such as Block and Cyclic [17]. Consider the case where the multicast is invoked, followed by some computation, and then a barrier synchronization, as illustrated in Figure 3.3. The nodes that finish earlier in the multicast tree (P_1 and P_2 in the example) have to wait until the last node ($P3$) reaches the barrier. Thus, t_{mcast}

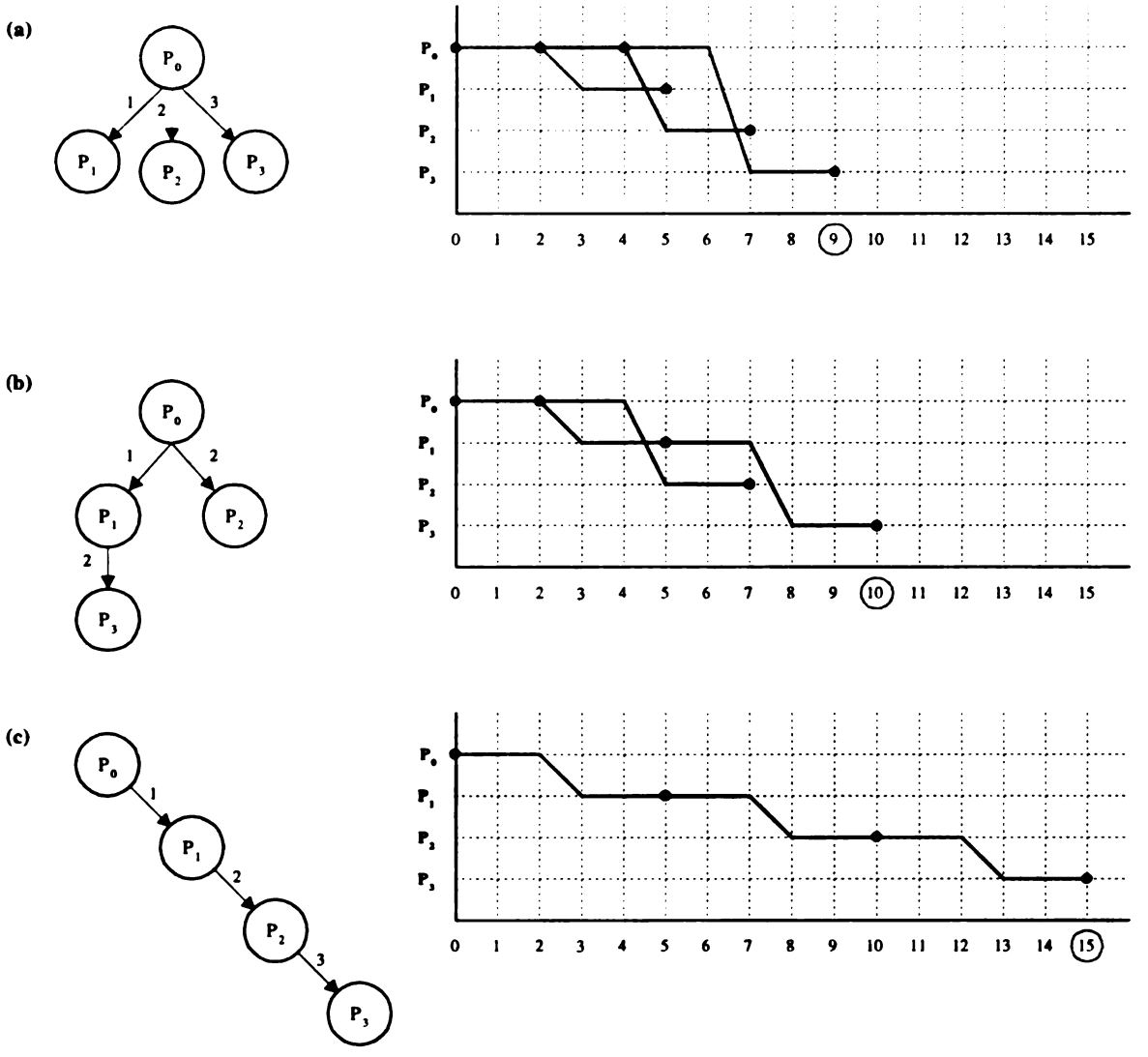


Figure 3.2: The timing diagrams of three basic multicast tree structures: (a) the sequential tree, (b) the binomial tree, and (c) the chain tree

is a fair metric to evaluate different multicast implementations.

3.3 Benchmarking Technique

In performance evaluation, using correct measurement techniques is as important as defining the most representative metrics. Unfortunately, without standard multicast benchmarks, one common mistake in multicast-related studies is using incorrect measurement tech-

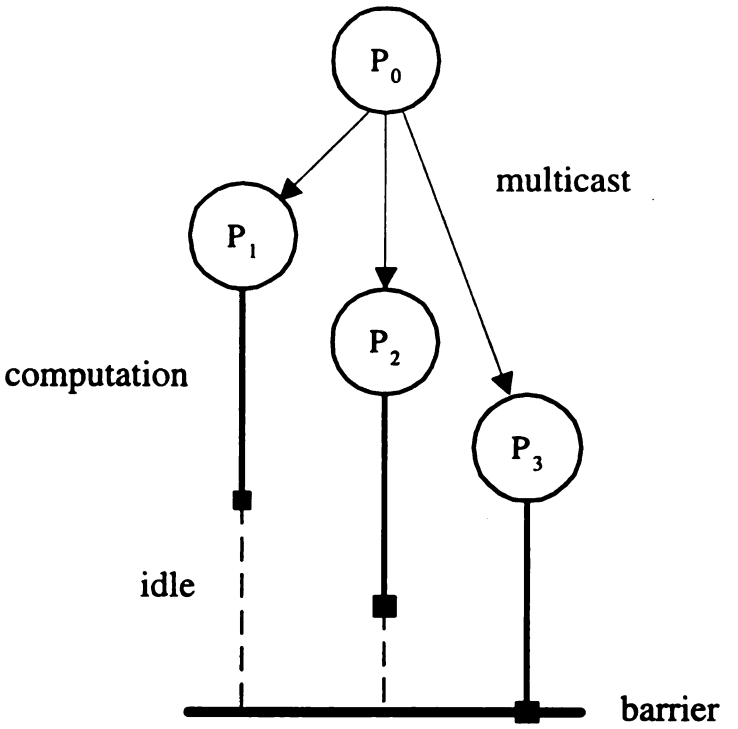


Figure 3.3: The importance of the multicast latency.

niques. The results from incorrect techniques can be very misleading. To explain this problem, we examine a ping-based benchmarking which has been used by some researchers. Algorithm 3.3.1 shows a ping-based multicast benchmarking code which measures the average latency at the root node (t_{root}) and the maximum latency among all participating nodes (t_{max}). The results can be a big misleading when comparing different multicast implementations since neither t_{root} nor t_{max} reflects the actual performance of a multicast tree. As shown in Figure 3.4, the results mislead us to conclude that the performance of the chain multicast tree is better than the binomial multicast tree in term of both latency and scalability, which is obviously wrong.

Algorithm 3.3.1 The ping-based multicast benchmark.

begin
  t_{start} := GetTime();
  for i := 1 **to** N **do**
    MPI_Bcast(message);
  endfor;
  t_{local} := (GetTime() - t_{start})/N;

  t_{root} := t_{local} from the root node;
  t_{max} := maximum reduce (t_{local});
end.

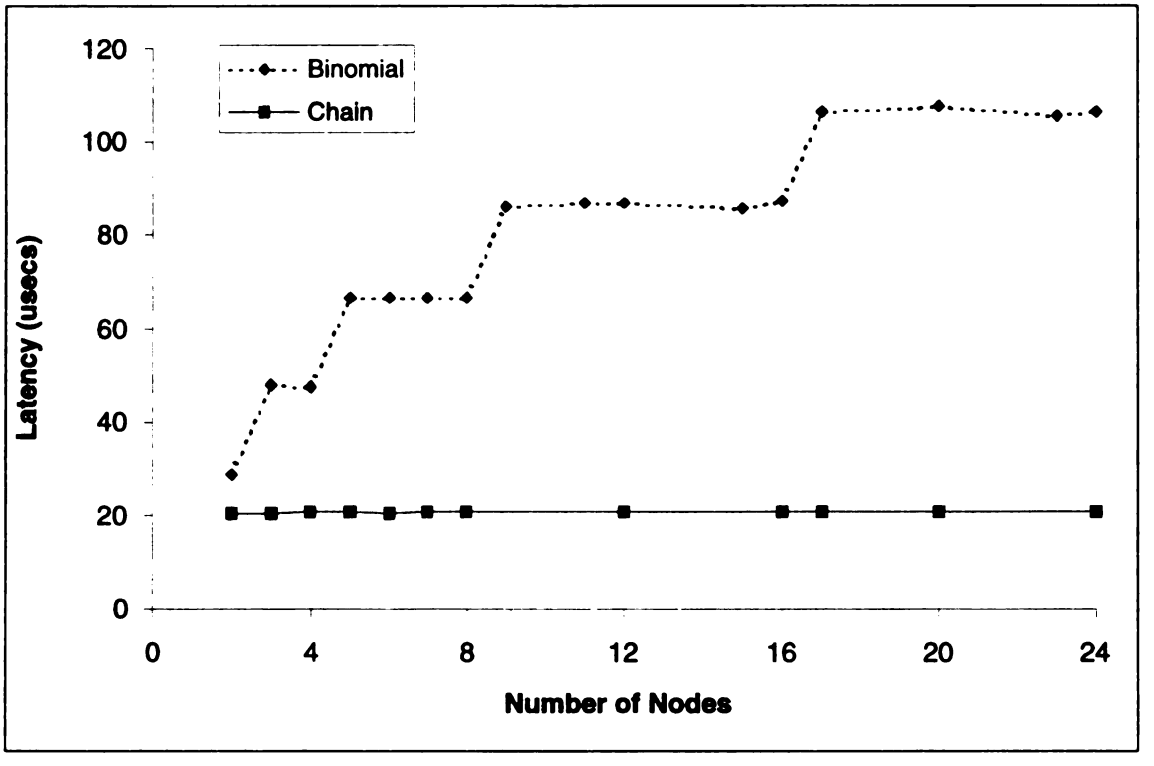


Figure 3.4: The ping-based measurement results of multicast (t_{root}) on the IBM/SP.

There are two major problems with the ping-based benchmark, pipelined effect and cross-iteration contention. If we roll out the loop in Algorithm 3.3.1, this algorithm measures the average latency of N consecutive multicast operations. Based on MPI definition,

the return from MPI_Bcast call does not guarantee that the whole operation is actually finished. Thus, some nodes may start the next call to MPI_Bcast routine even before all nodes finish the current MPI_Bcast operation. This creates a pipelined effect, and hence, the measured latency is less than the actual MPI_Bcast latency. The other problem is the contention between MPI_Bcast executed on different iterations. As mention earlier, some nodes may start the next call to MPI_Bcast routine while the current MPI_Bcast operation is not finished. The overlap of MPI_Bcast from different iterations may create network contention which can further skew the measurement results.

3.3.1 Collective Communication Flow Model

In this study, we use the collective communication flow model [59] to provide a framework for our multicast benchmark. We also provide the formal definition of multicast latency which is used as a metric. Although the definition of the multicast latency is very simple, it still contains some ambiguities. In SPMD environment, each node executes at its own speed. Some nodes may start participating a multicast operation late such that the multicast latency may be longer than expected. To eliminate these ambiguities, we redefine the multicast latency as follows:

Definition 1 *Let all nodes simultaneously call the multicast communication routine at time* t_0. *The* **multicast latency** (t_{mcast}) *is the elapsed time between* t_0 *and the earliest time when all nodes finish the call. Let* D *be the set of all destinations in the group and let the set of critical destinations* (D_c) *be a set of destinations that finish the call last.*

The main problem of this definition is that it is very difficult, if possible, to make all nodes call the multicast routine at the same time without using a special hardware. To make the multicast latency measurable, the measurement techniques are based on the *communication flow model.*

Definition 2 *A* **communication flow** $f(r, d)$ *is a chain of unicast operations propagating messages from the root node* r *to the destination* d.

Definition 3 *The* **flow latency** $t_f(r, d)$ *is the elapsed time between the root node* r *calls the multicast communication routine and the destination* d *returns from the call.*

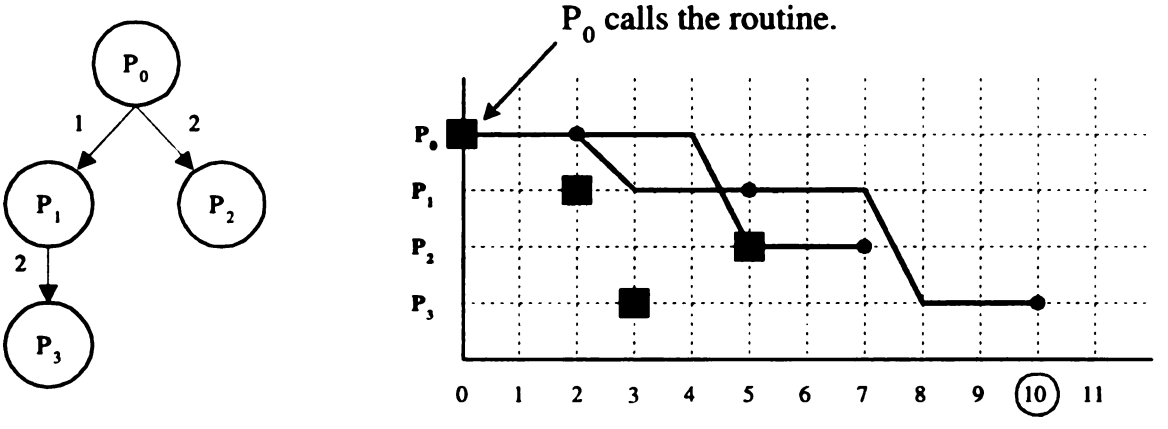


Figure 3.5: An example of the multicast communication service for 4 nodes. P_0, P_1, P_2, and P_3 call the multicast routine at time instance 0, 2, 5, and 3, respectively.

Figure 3.5 demonstrates an example of the multicast communication service for 4 nodes where node P_0 is the root node. In this example, there is only one critical destination. Thus, $D = P_1, P_2, P_3$ and $D_c = P_3$. In addition, there are 3 flows including $f(P_0, P_1)$, $f(P_0, P_2)$, and $f(P_0, P_3)$. The flow latencies are 7, 7, and 10, respectively. Based on the flow latency, we can derive the following lemmas.

Definition 4 . *Let r be the root node of the multicast operation. Let $t'_f(r,d)$ be the flow latency of $f(r,d)$ where all nodes simultaneously call the routine.*

Lemma 1 *Let r be the root node of the multicast operation.*

$$\begin{aligned} t_{mcast} &= \max_{d \in D}\{t'_f(r,d)\} \\ &= t'_f(r,d_c) \qquad \forall d_c \in D_c \end{aligned}$$

Lemma 2 *If the root node r is the last node to call the multicast communication routine, for all destination $d \in D$:*

$$t_f(r,d) = t'_f(r,d)$$

Proof: The proof for Lemma 1 is quite straight forward since it is derived from Definition 1 and Definition 4. For Lemma 2, we consider the fact that in multicast operation, the root node is the only source node and all other nodes are the destinations. All nodes, except the root node, have to receive a message before forwarding the message to other nodes. Thus, even though all nodes participate the multicast operation, the operation does not take place until the root node calls the multicast subroutine. Therefore, the flow latencies when the root node is the last one to call the multicast routine are equal to the flow latencies when all nodes call the routine at the same time.

Theorem 1 *If the root node r is the last one to call the multicast subroutine, for a critical destination $d_c \in D_c$ and for any destination $d \in D$:*

$$t_{mcast} = t_f(r,d_c)$$

$$t_f(r,d) \leq t_f(r,d_c)$$

Proof: See Lemma 1 and Lemma 2.

3.3.2 Multicast Benchmarking

Based on Theorem 1, there are three keys when measuring the multicast latency:

- Make the root node to be the last one to call the multicast routine.
- Measure the flow latencies of the root node and all destinations.
- Identify a critical destination. Its flow latency is the multicast latency.

It is quite simple to solve the first problem. Using a reduction operation before measurement, we can guarantee that the root node is always the last node to call the multicast operation. To solve the second problem, however, is more complicated. To measure a flow latency $t_f(r,d)$, we must know the time the root node r calls the multicast routine and the time a destination d returns from the call. Without a global clock, finding the elapsed time from two nodes is not straight forward. To overcome this difficulty, we use a technique similar to our PingPong benchmark. In our technique, the root node issues a multicast operation and wait for an ack message (1 byte) which is sent from node d *after node* d *finishes participating the multicast operation.* This is considered one iteration of our multicast benchmark. Then the root node computes the elapsed time of one iteration ($t_m(d)$). During the iteration, all other nodes only participate the multicast operation. To distinguish the destination d (whose flow latency is being measured) from other destinations, the destination d is referred to as the *responder*. By varying a responder, we can measure the flow

latency from r to each destination. The destinations whose flow latency are the longest are critical destinations.

To clearly explain the idea, let consider examples in Figure 3.6. There are four possible cases when the flow latencies are measured from different responders:

- the responder is not a critical destination and an ack arrives at the root node before the root node finishes participating.

- the responder is not a critical destination and an ack arrives at the root node after the root node finishes participating but before the multicast is actually ended.

- the responder is not a critical destination and an ack arrives at the root node after the multicast is ended.

- the responder is a critical destination.

For all but except the first case, we can find the flow latency $t_f(r, responder)$ by subtracting the latency of a 1-byte ack from $t_m(responder)$. Although we can not find the actual flow latency of the first case, we can ignore this case as this responder is not a critical destination. In the last case, the responder (P_4) has the longest flow latency. Thus, P_4 is the critical destination and its flow latency $t_f(P_0, P_4)$ is the multicast latency.

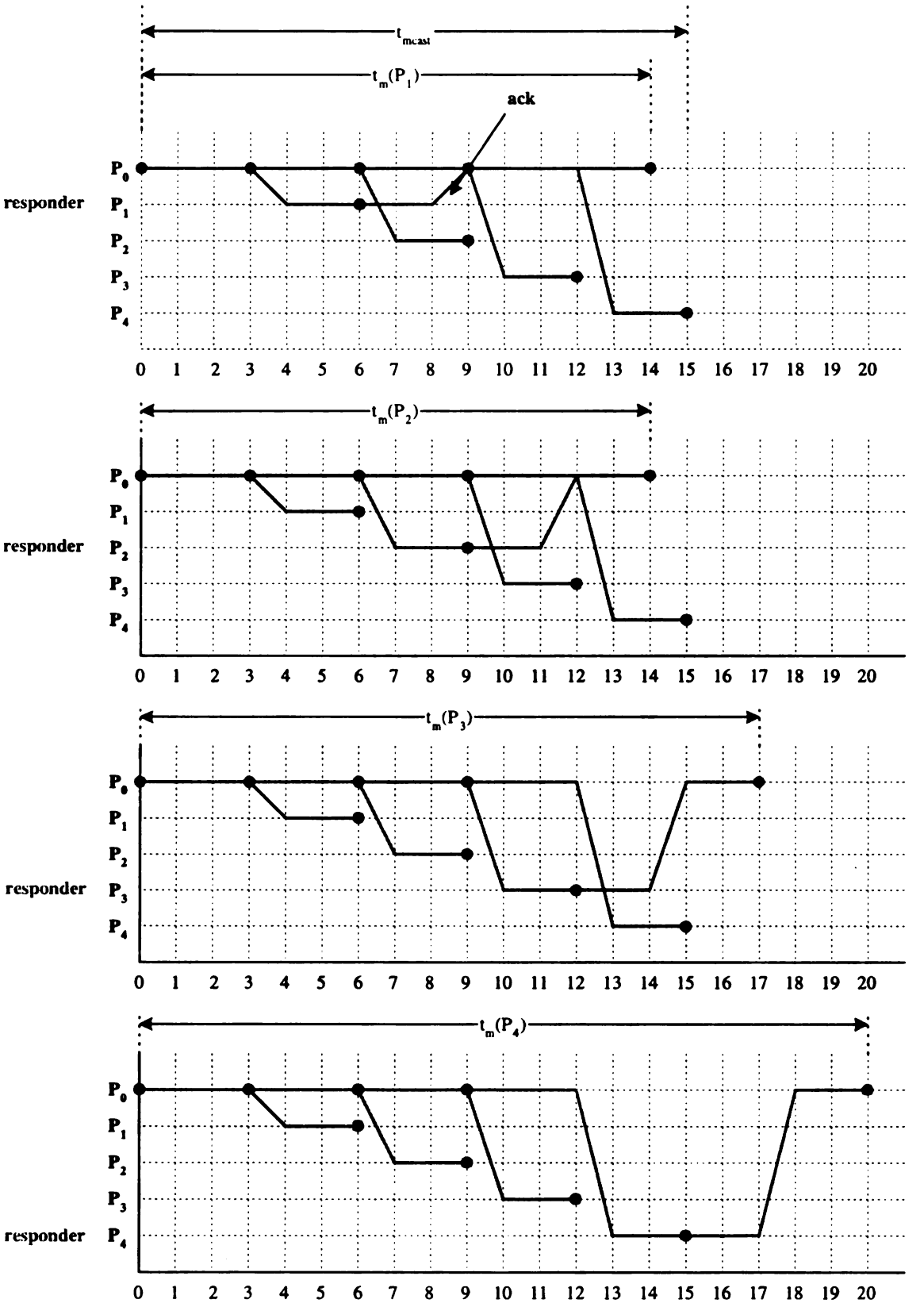


Figure 3.6: Four possible cases when measures the flow latency from the root node P_0 to P_i which is a responder.

Algorithm 3.3.2 **Measuring multicast communication.**

```
begin
   {Synchronize.}
   Reduce(root); {The root node finishes the last.}

   {Measurement.}
   if node_id == root then
         t_start := GetTime();
   endif;
   for i := 1 to N do
         Multicast_Routine();
         if node_id == root then
            Wait for an ack from responder;
         else if node_id == responder then
            {Dummy loop goes here.}
            Send a one_byte ack to the root;
         endif;
   endfor;
   if node_id == root then
         t_m(responder) := (GetTime() - t_start)/N;
         t_f(root, responder) := t_m(responder) - t_ack(one_byte);
   endif;
end.
```

Similar to Ping and PingPong benchmarks, it is quite difficult to measure a single iteration of multicast benchmark. This is because the system clock resolution is usually not fine enough. Thus, we measure the elapsed time of k iterations and compute the average delay. The algorithm to measure $t_f(root, responder)$ is shown in Algorithm 3.3.2. Since this algorithm measures a flow latency of a single responder, it has to run $k-1$ times if the group size is k. The value of responder changes from P_1 to P_{k-1} for each run, where P_0 is the root node.

One flaw of this algorithm is that $t_f(root, responder)$ computed in the algorithm may not be equal to the actual flow latency. This is because the root node may start the next iteration while some nodes do not finish the current iteration yet. Hence, it cannot be guaranteed that the root node is always the last node to call the routine all the time. In addition, the overlap of the multicast communication from different iterations may create network contention which will further skew the result.

To solve this problem, each responder has to wait t_{dummy} time before sending the ack message back. (The placement of the dummy loop in the code is shown in Algorithm 3.3.2.) Let the worst case measured latency in Algorithm 3.3.2 be t_{worst}. By choosing $t_{dummy} \geq t_{worst}$, it can guarantee that by the time the root node receives the ack message, all nodes have already started the next iteration. Thus, Theorem 1 holds. The responders that have the longest $t_m(responder)$ are the critical destinations. Therefore, the multicast latency can be measured by using one of the critical destinations as the responder and performing one more measurement without the dummy loop.

3.4 Experiments

We conducted the experiments on the 128-node IBM/SP at Argonne National Laboratory. Each node has an IBM/RS6000 processor (62.5 MHz) with 128 MB memory and 1 GB local disk. The peak performance is 125 MFlops per node.

Our benchmark is implemented using MPI-F library version 1.41 [55]. In order to fully utilize the high-performance switch, the library `euilib` with option `us` is used. Each data point in our results is the minimal value of 1000 measurements to reduce the

overhead due to network contention from other programs[1]. Note that the latency of the 1-byte acknowledgment message has been deducted from all data points.

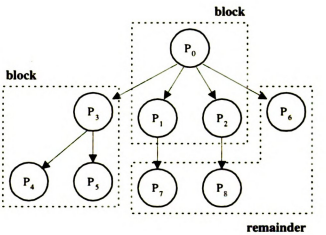
3.4.1 Results Interpretation

The fundamental concept of our benchmark is to select the last node by determining the time measured from different responders. To demonstrate how to interpret the results, we benchmark MPI_Bcast of MPI-F library, which is based on the *block-based binomial tree*. This tree is a combination of the binomial and sequential trees. The parameter `blocksize` determines the size of each block. First, some members of the process group are partitioned into fixed-size blocks, where the number of blocks must be a power of 2. Thus, some nodes may not belong to any block. Then, the root node multicasts a message to the first node in each block using the binomial tree. The first node in each block then sends the message to other members in the block using the sequential tree. Finally, the remaining nodes, say m of them, not belonging to any block are taken care by P_0 to P_{m-1}. The parameter `blocksize` determines the shape of the tree. If `blocksize` is one, the tree is a binomial tree. If `blocksize` equals to the group size, the tree is equivalent to a sequential tree.

Figure 3.7 shows the tree used in MPI-F for a group size of 9, where `blocksize` is 3. In this figure, there are two blocks: P_0, P_1, P_2 and P_3, P_4, P_5, where P_0 and P_3 are the first node of each block, and the two blocks form a two-node binomial tree. After P_0 sends the message to P_3, both processors send messages to other processors in their blocks. Then P_0,

[1] It is difficult to reserve the whole machine for our benchmarking.

P_1, and P_2 send messages to the remaining nodes which are P_6, P_7, and P_8. Figure 3.7 also shows the measured latency for each processor when the processor is the responder. From the timing diagram, P_8 is the last node. Thus, it is the critical destination. Its flow latency is the multicast latency.



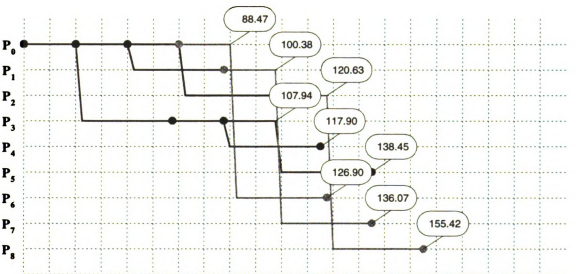


Figure 3.7: The flow latency of MPI_Bcast of MPI-F library on the IBM/SP with message size being 1 byte.

3.4.2 Comparison of Three Multicast Trees

We compare the performance of three different multicast trees with respect to two different message sizes. we consider the sequential tree, the binomial tree, and the block-based binomial (MPI-F) tree. Figure 3.8 illustrates the multicast latency (t_{mcast}) when the message size is 1 byte. When the size of tree is small, all trees exhibit the same performance. As the tree size become larger, the multicast latency of all trees increases. When the tree size is very large, the sequential tree performs the worst and both binomial and MPI-F tree perform considerably the same.

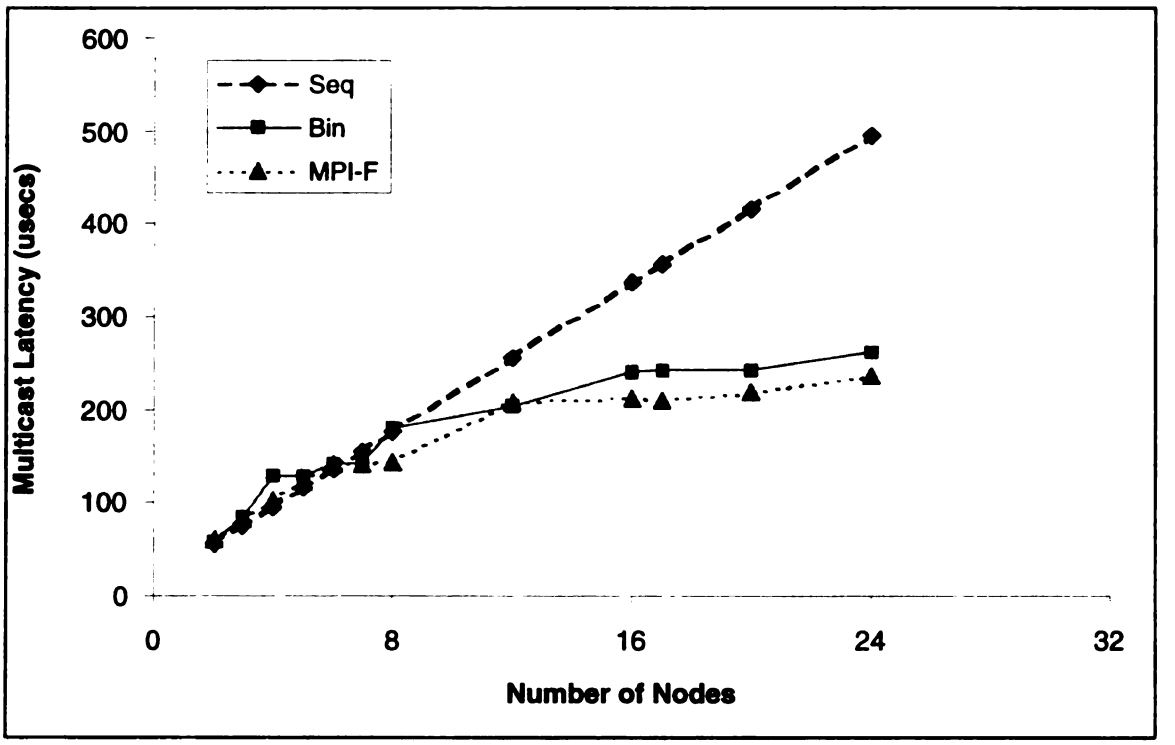


Figure 3.8: The multicast latency (t_{mcast}) of three multicast trees with message size being 1 byte.

When the message size is 1024 bytes, the results from three multicast trees are shown in Figure 3.9. With this message size, the difference of the performance of the binomial

tree and MPI-F tree is noticeable. This is due to the fact that the MPI-F tree is customized to perform well on the IBM/SP.

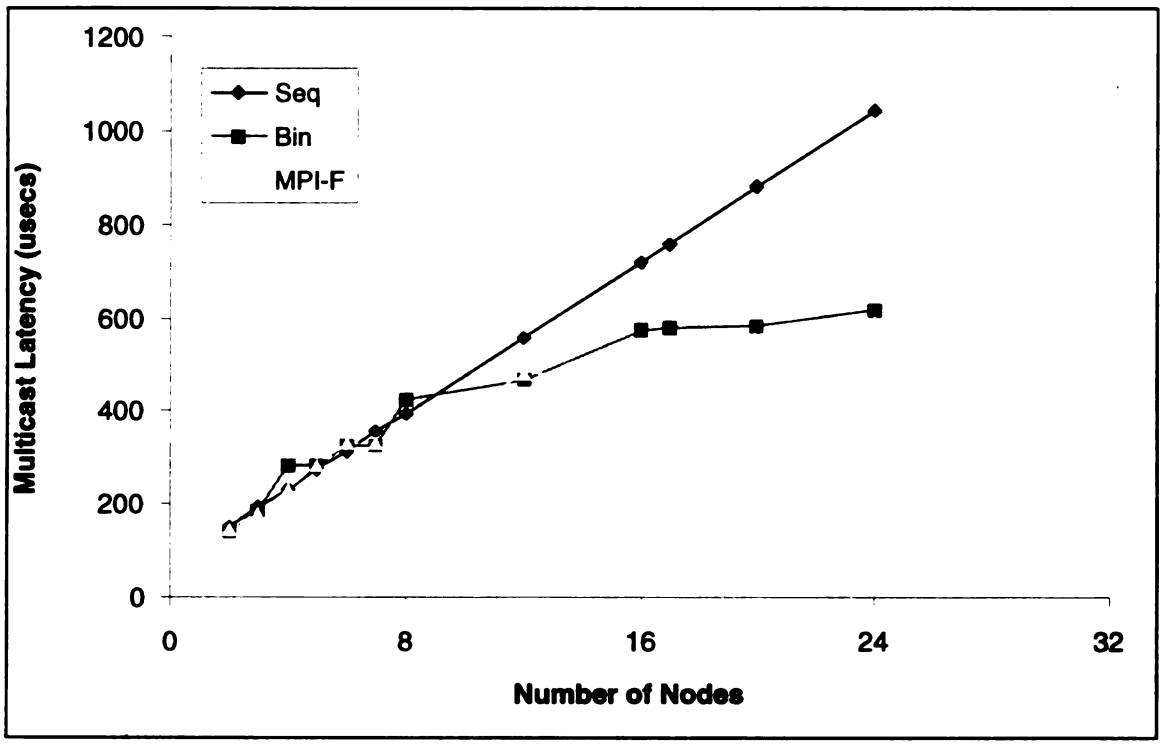


Figure 3.9: The multicast latency (t_{mcast}) of three multicast trees with message size being 1024 bytes.

3.4.3 Scatter Communication

The proposed benchmarking technique can be extended to other one-to-all collective communication services. The key idea of measuring the latency of this communication class is to identify the last node in the tree and measure the elapsed time. Figure 3.10 demonstrates the result of applying the proposed benchmarking technique to MPI_Scatter in MPI-F library. Since the latency increase of MPI_Scatter is linear, it is quite obvious that this function is based on a sequential tree approach, which was confirmed by checking the

source code [60]. Its performance is comparable to the sequential tree with slightly higher software latency.

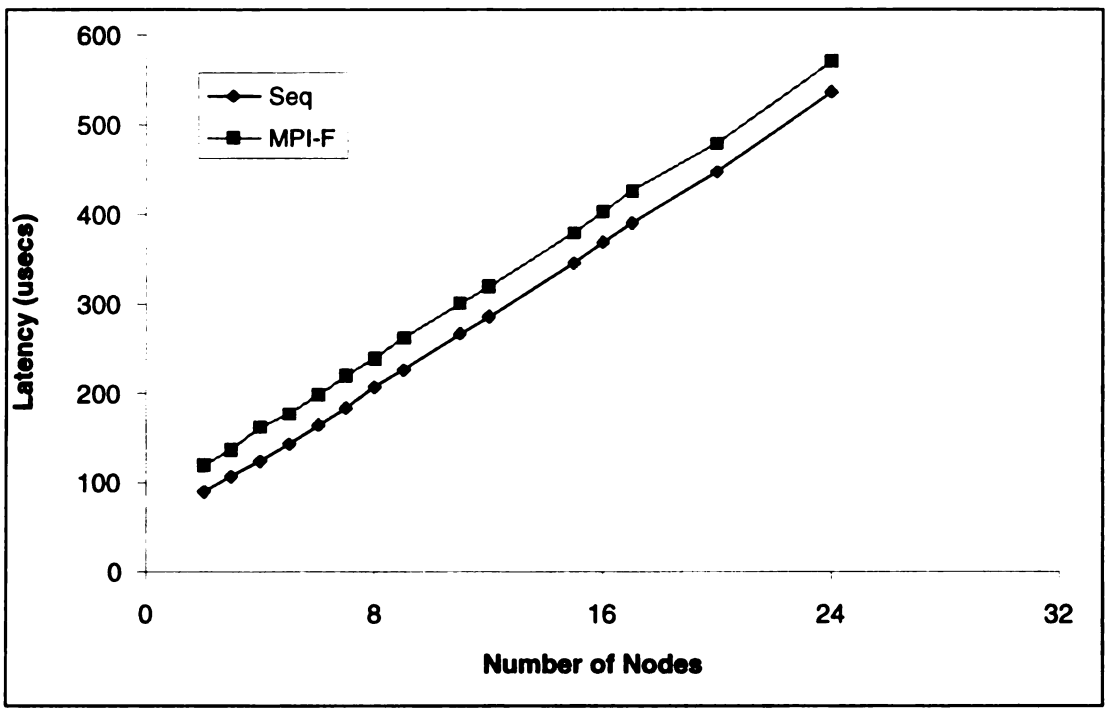


Figure 3.10: The communication latency of MPI_Scatter with message size being 1 byte.

3.5 Conclusion

In this chapter, we have discussed the model and benchmarking technique for multicast communication. Measuring the multicast latency is a challenging problem. We have shown that the popular ping-based benchmarking technique can be quite misleading. The proposed benchmarking technique can accurately measure the actual multicast latency without requiring a global clock and without having to know the detailed implementation of the multicast algorithm. To demonstrate and validate our technique, we conducted experiments

on the IBM/SP at Argonne National Laboratory and compared the empirical results with the results based on the multicast model. We also discussed the extension of our benchmarking technique to measure another one-to-all collective communication service, scatter. Note that this benchmark is used to evaluate multicast algorithms throughout this work.

Chapter 4

Platform-Independent Multicast

A key issue in designing a software multicast algorithm is to consider the trade-off between performance and portability. To achieve nearly optimal performance, some algorithms utilize the underlying network characteristics, for example EDN approach [49] and Scatter-Collect [43]. However, those algorithms are applicable to the intended architectures only and cannot be ported to other architectures. The other approach is to use standard communication services, send and receive, to propagate messages based on some standard multicast trees such as the sequential tree and the binomial tree. This *platform-independent* approach is simple and highly portable. Due to its simplicity and portability, the platform-independent multicast has been adopted by many implementations of standard communication libraries.

The main drawback of platform-independent multicast is that its scalability is usually poor and its performance typically varies significantly from one platform to another. Let consider Table 4 which demonstrates the performance of two multicast trees (8 nodes and 1-Kbyte) on two hypothetic systems. On $S1$, the sequential tree performs better than the

binomial tree. However, the performance of both trees are totally different on $S2$. The binomial tree obviously out-performs the sequential tree.

System	Sequential	Binomial
$S1$ ($t_{hold} = 20 + 0.02m$, $t_{end} = 55 + 0.07m$)	19631	21669
$S2$ ($t_{hold} = 25 + 0.03m$, $t_{end} = 40 + 0.04m$)	22718	12408

Table 4.1: The performance of the sequential tree and the binomial tree on two hypothetic systems. The multicast tree is 8 nodes with 1-Kbyte message.

To overcome this problem, we propose the *parameterized multicast algorithm* which is portable, but still performs well on various platforms. The main concept of our proposed algorithm is to characterize the underlying network using the parameterized communication model presented in Chapter 2 and then construct an optimal multicast tree based on two network parameters, t_{hold} and t_{end}. First, we measure network parameters, t_{hold} and t_{end}, during the compilation phase. When an application calls the multicast routine during the runtime, our multicast algorithm uses the predetermined network parameters to generate a multicast tree which yields the best performance. With this approach, our multicast algorithm is portable and does take into account network parameters. Consequently, the performance achievable, although may not be optimal, will be close to optimal.

4.1 Multicast Tree and Network Parameters

A previous example clearly shows that different multicast trees exhibit different performance. In fact, performance of a multicast tree, when run on different platforms, can be entirely different. Before we can construct an optimal multicast tree based on network parameters, we must first understand the relationship of multicast tree's shape and the un-

derlying network parameters.

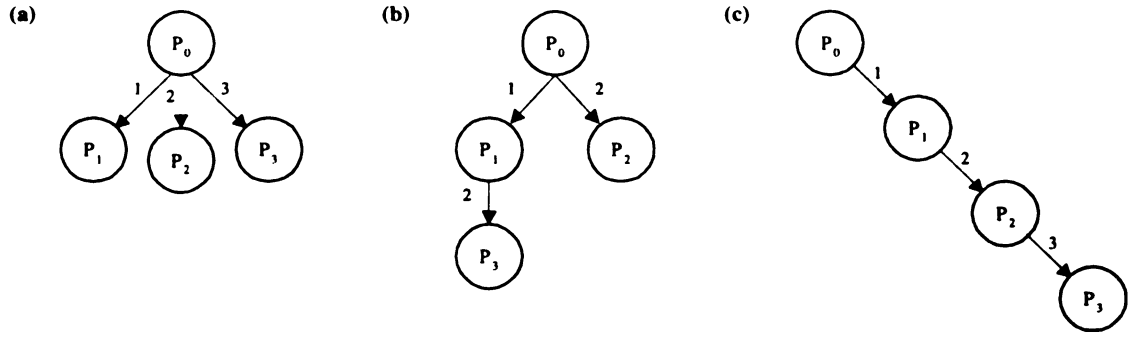


Figure 4.1: Three basic multicast tree structures: (a) the sequential tree, (b) the binomial tree, and (c) the chain tree

Let consider the aforementioned basic multicast trees, (a) the sequential tree, (b) the binomial tree, and (c) the chain tree. The structures of these trees are presented in Figure 4.1. The major differences among these trees are their depths and the out-degrees of the internal nodes. The sequential tree is not deep, but its out-degree of the root node is very high. On the contrary, the chain tree is very deep, but the out-degree at the root node and all internal nodes are one. Based on the concept of recursive-doubling, the binomial tree is deeper than the sequential tree but not as deep as the chain tree and its internal nodes have higher out-degree than the chain tree's but not as high as the sequential tree's.

We further study the effect of network parameters by examining the estimation of multicast trees' performance based on t_{hold} and t_{end}. Table 4.1 contains the estimated performance of three basic trees. Obviously, t_{hold} is very significant for the sequential tree. This is because the sequential tree is not deep and quite flat. Thus, in a system whose t_{hold} is much less than t_{end}, it is more efficient to allow the root node sending to all receivers. For the chain tree, t_{end} dominates since the tree is very deep. Thus, it is quite suitable for a system with t_{hold} greater than t_{end}. Since the binomial tree is quite complicated, it is very

Multicast Tree	Multicast Latency
sequential	$(k-2) \times t_{hold} + t_{end}$
chain	$(k-1) \times t_{end}$
binomial	case: $t_{hold} \leq t_{end}$ $\log_2 k \times t_{end}$ if $\log_2 k$ is an integer $max\{\lfloor \log_2 k \rfloor \times t_{end},$ $(\lfloor \log_2 k \rfloor - y + 1) \times t_{hold} + y \times t_{end}\}$ otherwise where $y = \lfloor \log_2 (k - 2^{\lfloor \log_2 k \rfloor}) \rfloor + 1$ case: $t_{hold} > t_{end}$ $\lfloor \log_2 (k-1) \rfloor \times t_{hold} + t_{end}$

Table 4.2: The performance of the sequential tree, the chain tree, and the binomial tree where k is number of nodes in the tree.

difficult to estimate its performance. However, its estimation and its structure suggest that this tree is the best in a system whose t_{hold} is approximately t_{end}. Figure 4.2 summarizes the relationship of t_{hold}, t_{end}, and the structure of an optimal multicast tree. Note that for a system with t_{hold} less than t_{end}, an optimal multicast tree is a combination of the sequential tree and the binomial tree which implies that the tree is *deeper* than the depth of the sequential tree and each internal node of the tree sends to more receivers (more *flat*) than the binomial tree. In a system with t_{hold} greater than t_{end}, an optimal multicast tree is a combination of the binomial tree and te chain tree.

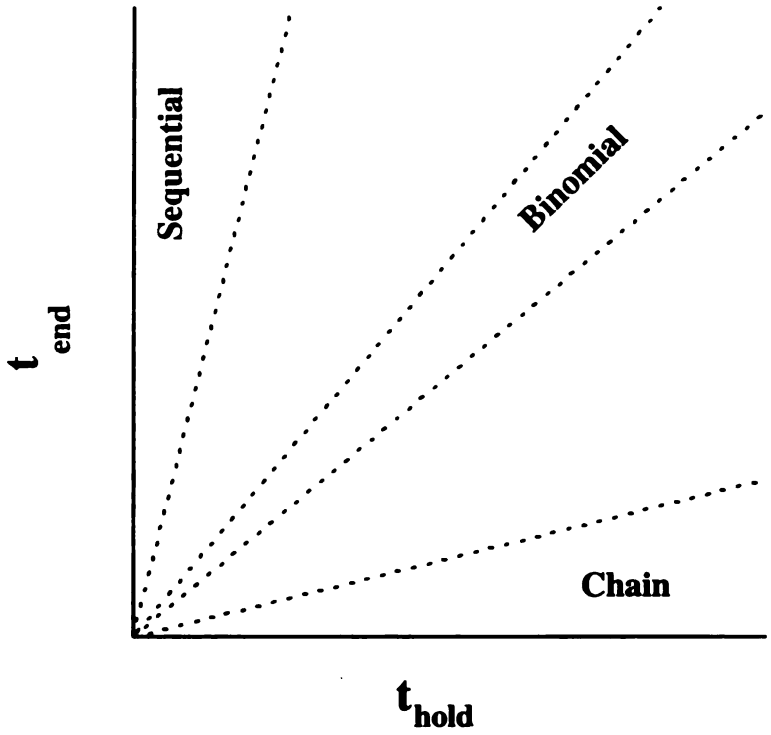


Figure 4.2: The relationship of t_{hold}, t_{end}, and the structure of an optimal multicast tree

4.2 Optimal Multicast Trees

Consider the construction of an optimal multicast tree with k nodes, $(P_0, P_1, \ldots, P_{k-1})$, where P_0 is the root node. In the first step, P_0 sends a message to P_j. Before P_j is receives the message completely, P_0 may be able to send the same message to other nodes in the tree depending on the values of t_{hold} and t_{end}. When P_j is ready to send messages, basically there are two multicast sub-trees: one is a j-node tree $(P_0, P_1, \ldots, P_{j-1})$ rooted at P_0 and the other is a $(k-j)$-node tree $(P_j, P_{j+1}, \ldots, P_{k-1})$ rooted at P_j as shown in Figure 4.3. The issue now becomes which node should belong to which subtree.

In order to construct an optimal multicast tree, two requirements must be satisfied. First, the node P_j must be chosen such that the generated multicast tree is optimal. Second, the

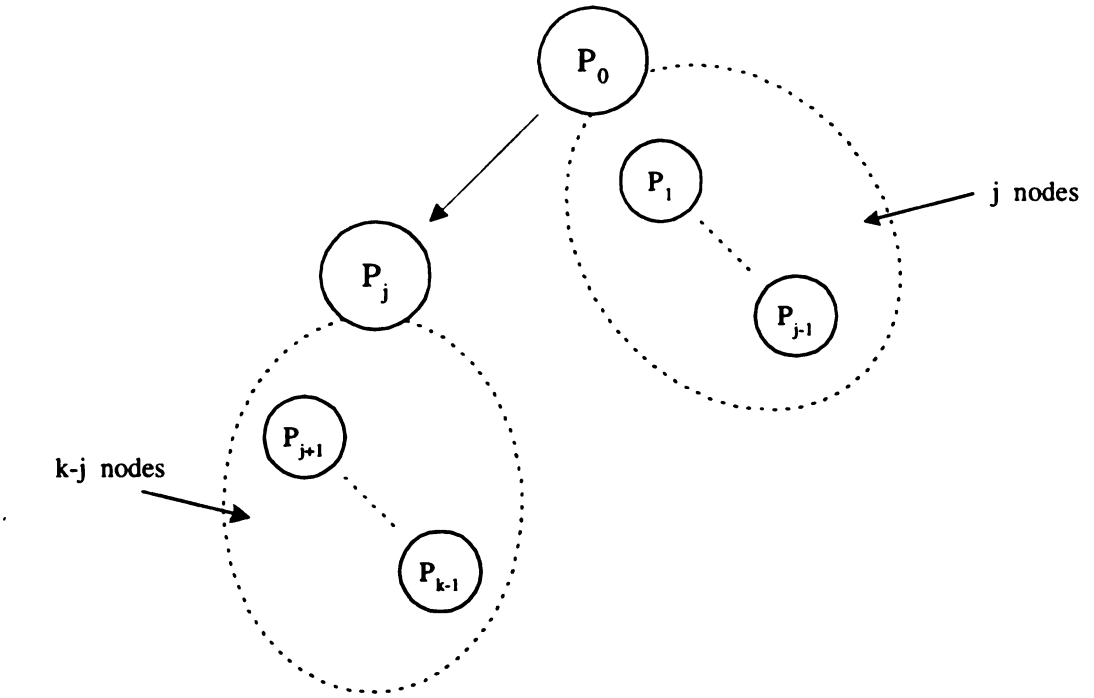


Figure 4.3: An optimal multicast tree with k nodes consists of two optimal multicast sub-trees.

two subtrees $(P_0, P_1, \ldots, P_{j-1})$ and $(P_j, P_{j+1}, \ldots, P_{k-1})$ must themselves be optimal. The same procedure is then recursively applied to each of the multicast sub-trees. Thus, this optimization problem exhibits two properties: optimal substructure and overlapping sub-problems. With the presence of these two properties, the dynamic programming technique is applicable to find an optimal solution [61].

Let $t[i]$ (for each i, $1 \leq i \leq k$) be the minimum latency required to multicast a message among i nodes $P_a, P_{a+1}, \ldots, P_{a+i-1}$ (for an a, $0 \leq a \leq k - i$) with node P_a as the root node. We then recursively define $t[i]$ as follows. If $i = 1$, there is only one node in the tree, thus $t[i] = 0$. When $i > 1$, P_a sends the message to some node P_{a+j} for $1 \leq j \leq i - 1$. After t_{hold} units, P_a can continue to transmit to nodes in its subtree of $(P_a, P_{a+1}, \ldots, P_{a+j-1})$, and after t_{end} time units, P_{a+j} has received the message and can

start the transmission to the nodes in its subtree of $(P_{a+j}, P_{a+j+1}, \ldots, P_{a+i-1})$. Therefore, the multicast latency among i nodes is in its subtree. This suggests the following recurrence for $t[i]$. Thus, the multicast latency is the maximum latency between the multicast latency of the subtree of $(P_a, P_{a+1}, \ldots, P_{a+j-1})$ plus t_{hold} and the multicast latency of the subtree of $(P_{a+j}, P_{a+j+1}, \ldots, P_{a+i-1})$ plus t_{end}. Note that t_{hold} is needed in the former because P_a must send a message to P_{a+j} before it sends messages to other nodes in its subtree, and t_{end} is needed in the latter because P_{a+j} has to receive the message from P_a before it can multicast the message to other nodes in its subtree. Therefore, we have

$$t[i] = \max(t[j] + t_{hold}, t[i-j] + t_{end})$$

To ensure the optimality, we must choose the node P_{a+j} such that the multicast latency is minimal. Thus, we have the following recurrence for $t[i]$.

$$t[i] = \begin{cases} 0 & \text{if } i = 1 \\ \min\limits_{1 \le j \le i-1} \{\max(t[j] + t_{hold}, \\ \qquad t[i-j] + t_{end})\} & \text{if } i > 1 \end{cases}$$

The optimal multicast latency of a k-node tree is $t[k]$ and this can be computed in $O(k^2)$ time by the dynamic programming. In particular, we compute values in the order $t[1], t[2], \cdots, t[k]$ and store them in an array. In order to compute $t[i]$ (for $1 < i \le k$), we consider each value of j from 1 to $i-1$, and determine the value of j for which $\max(t[j]+t_{hold},\ t[i-j]+t_{end})$ is minimized. Thus the total running time is $\sum_{i=1}^{k-1} i = O(k^2)$.

4.3 The $O(k)$ Algorithm

The overhead of $O(k^2)$ complexity of the dynamic-programming algorithm in the previous section is still too high to be used in the real world. However, it is possible to improve the running time to $O(k)$ by limiting the choices of j that need to be considered at each iteration. This is based on the observation about the nature of $t[i]$ which is described in the following lemma.

Lemma 3 *Let j_i denote the value of j for which the recursive definition of $t[i]$ achieves its minimum value (i.e., the best way to split a tree with i nodes is into subtrees of sizes j_i and $i - j_i$). Then, $j_2 = 1$ and, for $2 \leq i \leq k-1$, $j_{i+1} = j_i + 1$ or $j_{i+1} = j_i$.*

Proof: Without the loss of generality, for an optimal tree with i nodes, let the right subtree has j_i nodes and the left subtree has $j_i' = i - j_i$ nodes, as shown in Figure 4.4.

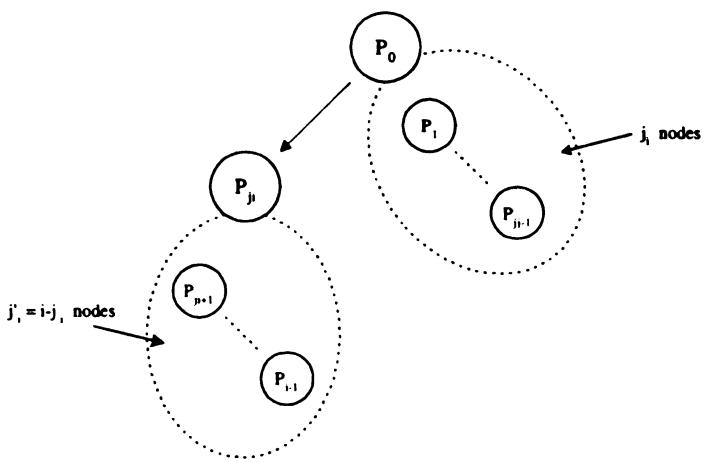


Figure 4.4: An optimal multicast tree with i nodes consists of two optimal multicast subtrees.

Suppose the lemma does not hold. Let i_0 be such that for *any* optimal multicast tree with i_0 nodes, either:

1. $j_{i_0} \geq j_{i_0-1} + 2$

or 2. $j_{i_0} \leq j_{i_0-1} - 1$

Choose such an optimal multicast tree T^0 with i_0 nodes (T^0 has subtrees with j_{i_0} and j'_{i_0} nodes) with the minimum value of $A(T^0)$ where

$$A(T^0) = |j_{i_0} - j_{i_0-1}| + |j'_{i_0} - j'_{i_0-1}|$$

case 1: Suppose $j_{i_0} \geq j_{i_0-1} + 2$. This implies that

$$j'_{i_0} \leq j'_{i_0-1} - 1 \tag{4.1}$$

By deleting one node from the right subtree and adding one node to the left subtree, the multicast latency of the right subtree is

$$\begin{aligned} t[j_{i_0} - 1] + t_{hold} &\leq t[j_{i_0}] + t_{hold} \\ &\leq t[i_0], \qquad \text{from the definition of } t[i_0] \end{aligned}$$

and the multicast latency of the left subtree is

$$\begin{aligned} t[j'_{i_0} + 1] + t_{end} &\leq t[j'_{i_0-1}] + t_{end} \qquad \text{from Equation (4.1)} \\ &\leq t[i_0 - 1] \qquad \text{from the definition of } t[i_0 - 1] \\ &\leq t[i_0] \end{aligned}$$

The multicast latency of this new tree T^1 is still optimal (i.e., $t[i_0]$) and $\mathcal{A}(T^1) < \mathcal{A}(T^0)$. This is a contradiction to the choice of T^0.

cast 2: Suppose $j_{i_0} \leq j_{i_0-1} - 1$. This also implies that

$$j'_{i_0} \geq j'_{i_0-1} + 2 \tag{4.2}$$

By deleting one node from the left subtree and adding one node to the right subtree, the multicast latency of the left subtree is

$$\begin{aligned} t[j'_{i_0} - 1] + t_{end} &\leq t[j'_{i_0}] + t_{end} \\ &\leq t[i_0] \qquad \text{from the definition of } t[i_0]. \end{aligned}$$

and the multicast latency of the right subtree is

$$\begin{aligned} t[j_{i_0} + 1] + t_{hold} &\leq t[j_{i_0-1}] + t_{hold} \qquad \text{from Equation (4.2)} \\ &\leq t[i_0 - 1] \qquad \text{from the definition of } t[i_0 - 1] \\ &\leq t[i_0]. \end{aligned}$$

The multicast latency of this new tree T^2 is also optimal and $\mathcal{A}(T^2) < \mathcal{A}(T^0)$. It is a contradiction to the choice of T^0. This completes the proof of the lemma.

Based on Lemma 3, we can revise our dynamic programming algorithm to become an $O(k)$ algorithm as follows:

$$t[i] = \begin{cases} 0 & \text{if } i = 1 \\ t_{end} & \text{if } i = 2 \\ \min\{\ \max\{\ t[j_{i-1}] + t_{hold},\ t[i - j_{i-1}] + t_{end}\}, & \\ \qquad \max\{\ t[j_{i-1} + 1] + t_{hold}, & \\ \qquad\ t[i - 1 - j_{i-1}] + t_{end}\}\} & \text{if } i \geq 3 \end{cases}$$

where $j_2 = 1$.

Algorithm 4.3.1 OPT-TREE

Input: $rootid$: nodeid of the root node.
$nodeid$: nodeid of the current node.
k: number of nodes.
j_i: the size of the right subtree.

```
begin
  {Adjust node-id such that root-node has index = 0.}
  base := rootid;
  myid := nodeid − rootid;
  if myid < 0 then
      myid := myid + k;
  endif;

  while k > 1 do
      if myid < j_k then
          {I belong to the right subtree}
          if myid = 0 then
              {I am the root node}
              Send a message to node (base + j_k);
          endif;
          k := j_k;
      else
          {I belong to the left subtree}
          if myid = j_k then
              {I receive a message from the root node}
              Receive a message to node base;
          endif;
          {Node j_k becomes the root node of this subtree.}
          base := base + j_k;
          {Adjust node-id. and the size of the tree}
          myid := myid − j_k;
          k := k − j_k;
      endif;
  endwhile;
end.
```

Based on the revised algorithm, we develop the parameterized multicast algorithm (OPT-Tree) shown in Algorithm 4.3.1. In this algorithm, when we construct an optimal

multicast tree with k nodes, we first compute $t[i]$ where $1 \leq i \leq k$. Then, we recursively construct a tree based on the $t[i]$ table. Table 4.3 shows an example of calculating $t[k]$ for $k = 9$, $t_{hold} = 20$, and $t_{end} = 55$. An optimal multicast tree corresponding to Table 4.3 is shown in Figure 4.5. Note that if the node-id of the root node is not 0, we need to adjust the node-id for all node as shown in Algorithm 4.3.1. In summary, we obtain the following theorem.

Theorem 2 *The optimal multicast latency of a k-node tree can be computed in $O(k)$ time.*

Table 4.3: $k = 9$, $t_{hold} = 20$, and $t_{end} = 55$.

i	j_i	$i - j_i$	$t[j_i] + t_{hold}$	$t[i - j_i] + t_{end}$	$t[i]$
1	-	-	-	-	0
2	1	1	20	55	55
3	2	1	75	55	75
4	3	1	95	55	95
5	3	2	95	110	110
6	4	2	115	110	115
7	5	2	130	110	130
8	5	3	130	130	130
9	6	3	135	130	135

4.4 Experiments

We conducted the experiments on the 128-node IBM/SP at Argonne National Laboratory. Our implementation uses MPI-F library version 1.41 [55]. In order to fully utilize the high-performance switch, the library `euilib` with option `us` is used. Each data point in our results is the minimal value of 1,000 measurements to reduce the overhead due to network contention from other programs. We use the benchmarking technique presented in Chapter 3.

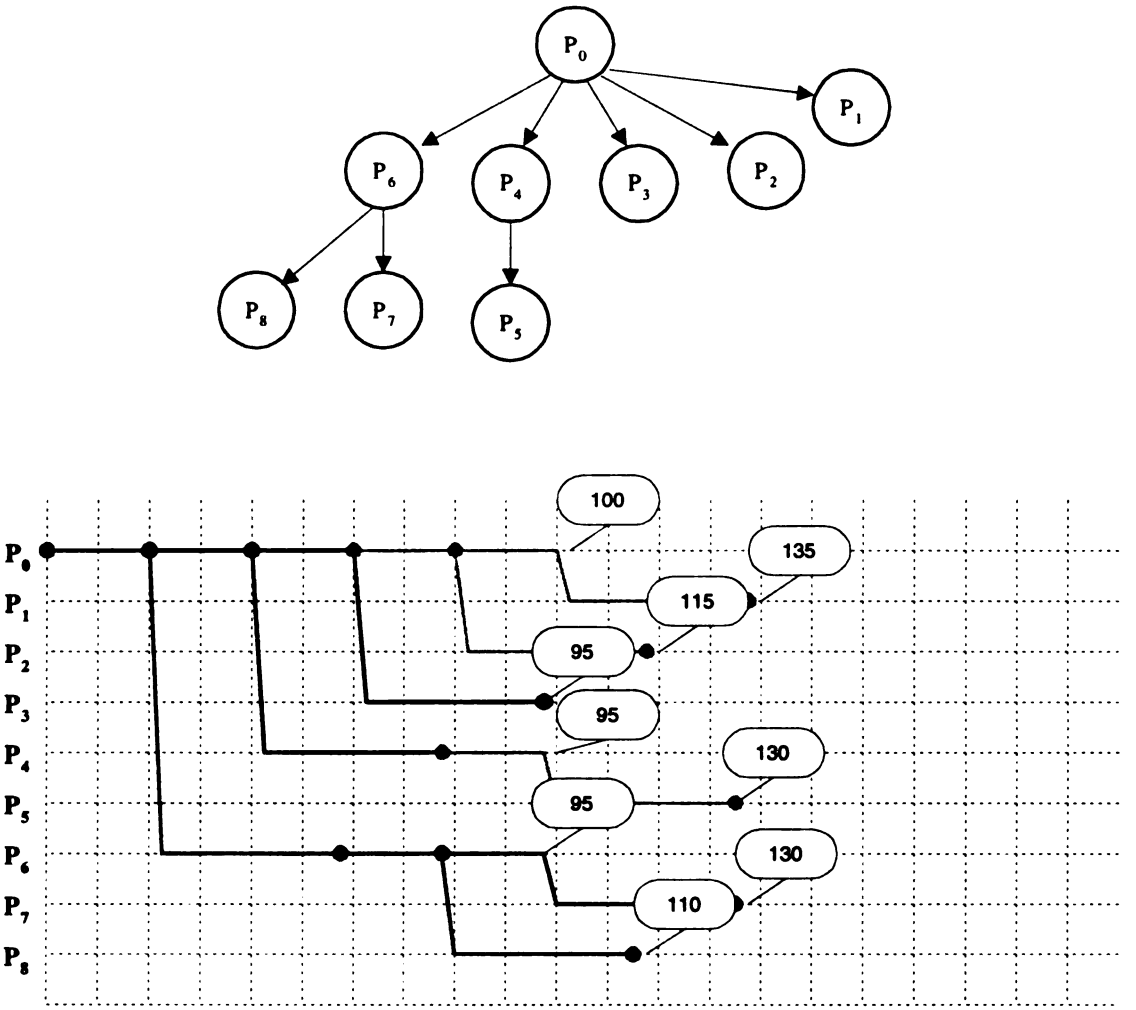


Figure 4.5: An optimal multicast tree with 9 nodes $t_{hold} = 20$, and $t_{end} = 55$.

In this Section, we discuss the results from our experiments by measuring the performance of three multicast tree: the popular binomial tree used by many researchers, the block-based binomial tree used in MPI-F, and the OPT-Tree tree, with respect to two different message sizes. The sequential tree and chain tree are not considered as they were known to perform poorly on the IBM/SP [62].

We compare the performance of three different multicast trees with respect to two different message sizes: 1 byte and 1K bytes. Figure 4.6 illustrates the multicast latency

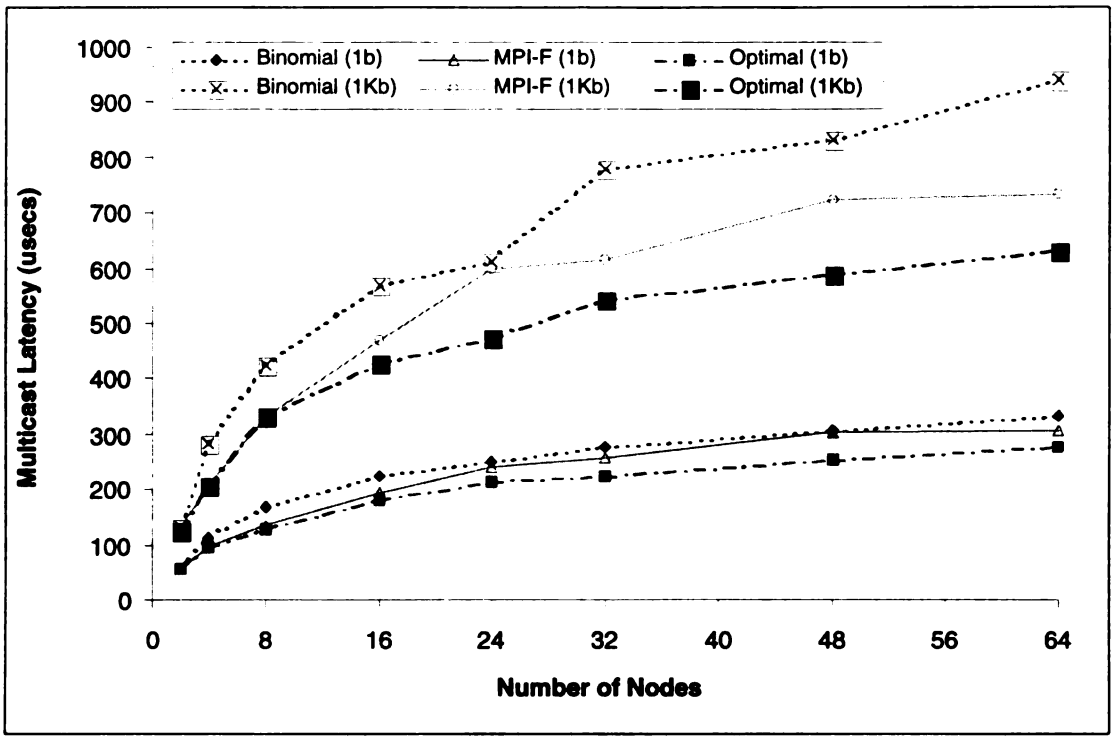


Figure 4.6: The multicast latency (t_{mcast}) of three multicast trees with message sizes being 1 byte and 1 Kbytes, respectively.

(t_{mcast}) of the three trees. For the case of the binomial tree, we were unable to measure the performance for group size greater than 24 due to the heavy use of the IBM SP at Argonne recently (we will complete the measurements in the final paper). However, it is clear that the performance of the binomial tree is the worst among the three trees. As the number of processors in a group increases, the multicast latency of all three trees increases. All three trees exhibit very close performance when the number of processors is small. In all cases, the OPT-Tree tree does provide the best performance. When the message is 1 byte, the MPI-F tree has slightly higher latency than the optimal tree. When the message size is 1024 bytes, the performance improvement of the OPT-Tree tree over the MPI-F tree is noticeable, which indicated that the proposed optimal multicast tree, even not considering

the underlying network topology, performs well on the IBM/SP.

4.5 Extension to the α-Port Architecture

Our previous discussion was based on the popular 1-port communication architecture. In some parallel machines, such as the TMC CM-5 and Intel/CMU iWARP, each processor has multiple communication ports. This section generalizes the construction of optimal multicast trees based on the α-port communication model.

As we defined earlier, t_{hold} is the time interval between two consecutive send operations through a single port. Let t_{int} denote the time interval that a processor can initiate another send operation from a different port. Obviously, we have $t_{int} < t_{hold}$; otherwise, multiple ports do not benefit. Given t_{int} and t_{hold}, the maximum number of ports, α_{max}, must satisfy the following inequality

$$(\alpha_{max} - 1)t_{int} < t_{hold} \leq \alpha_{max}t_{int}.$$

The proposed α-port optimal multicast algorithm can be applied to any value of $\alpha \leq \alpha_{max}$. Consider the construction of a multicast tree with k nodes $(P_0, P_1, \cdots, P_{k-1})$. The $O(k)$ time dynamic programming algorithm discussed in Section 4.3 for $\alpha = 1$ will be generalized to develop an $O(\alpha k)$ time algorithm for an arbitrary value of $\alpha \leq \alpha_{max}$.

Let $t[i]$ (for each i, $1 \leq i \leq k$) denote the minimum latency required to multicast a message among i nodes. We have $t[1] = 0$ and $t[2] = t_{end}$. For $i \geq 3$, $t[i]$ can be recursively defined as follows. Consider a multicast tree with i nodes. The source node P_0

sends the message to α nodes using α different ports. Here we assume that node P_0 sends the message to node P_{q_r} for $r = 1, 2, \cdots, \alpha$ in this order. That is, if node P_0 transmits to node P_{q_1} at time T, then node P_0 can transmit to node P_{q_r} $(2 \leq r \leq \alpha)$ after time $T + (r-1)t_{int}$. Thus, the multicast tree with i nodes has $\alpha + 1$ subtrees rooted at nodes $P_0, P_{q_1}, P_{q_2}, \cdots, P_{q_\alpha}$. Suppose the subtree rooted at node P_{q_r} has j_i^r nodes for $1 \leq r \leq \alpha$, and the subtree rooted at node P_0 has j_i nodes. (Throughout this section, the subtree with j_i^r (or j_i) nodes will denote the subtree rooted at node P_{q_r} (respectively, P_0) without any confusion.) This assumption implies that

$$\sum_{r=1}^{\alpha} j_i^r = i - j_i \tag{4.3}$$

Figure 4.7 shows the multicast tree with i nodes partitioned into $\alpha + 1$ subtrees.

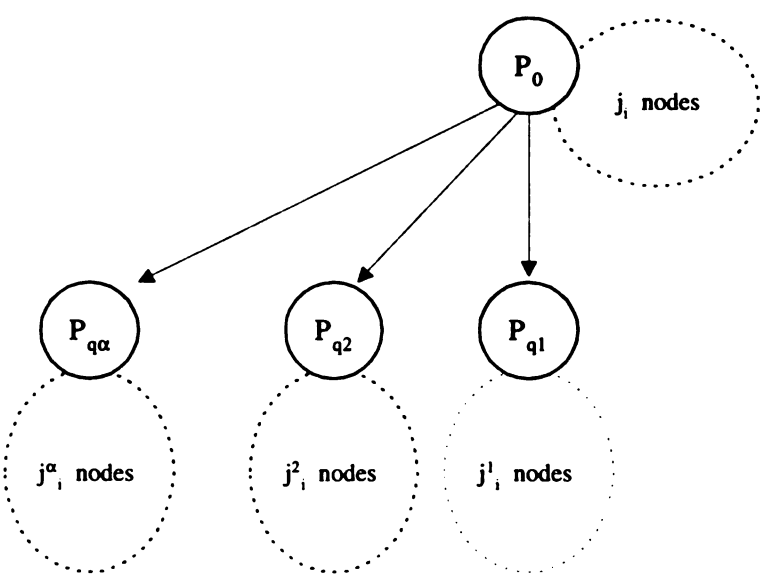


Figure 4.7: An optimal multicast tree using α -ports.

From the above discussion, we note the multicast latency of the subtree rooted at node

P_{q_r} $(1 \leq r \leq \alpha)$ is at least

$$t[j_i^r] + t_{end} + (r-1)t_{int} \tag{4.4}$$

and the multicast latency of the subtree rooted at node P_0 is at least

$$t[j_i] + t_{hold}. \tag{4.5}$$

To ensure the optimality, we must choose values for j_i^r for each r, $1 \leq r \leq \alpha$, such that the resulting multicast latency is minimal. Thus, we have the following recurrence for $t[i]$.

$$t[i] = \begin{cases} 0 & \text{if } i = 1 \\ \min\limits_{0 \leq j_i^1, \cdots j_i^\alpha \leq i-1} \{\max\{t[j_i] + t_{hold}, & \\ \max\limits_{1 \leq r \leq \alpha} \{t[j_i^r] + t_{end} + (r-1)t_{int}\}\}\} & \text{if } i > 1 \end{cases}$$

It is observed that all α ports may not be used when computing $t[i]$ for some values of i. For example, suppose the multicast tree with i nodes has $\alpha' + 1$ subtrees (i.e., the root of the tree has used only α' ports) for $\alpha' < \alpha$. In this case, j_i^r will denote 0 for $\alpha' < r \leq \alpha$. The optimal multicast latency $t[k]$ of a k-node tree is then computed in $O(\alpha k^{\alpha+1})$ time using the above recurrence. To verify this running time, we observe that there are at most k^α possible values of $j_i^1, \cdots, j_i^\alpha$, and for any given fixed values of $j_i^1, \cdots, j_i^\alpha$, $t[i]$ can be computed using α comparisons. As $1 \leq i \leq k$, it implies that $t[k]$ can be computed in $O(\alpha k^{\alpha+1})$ time.

Based on the following lemma, we improve the running time of our dynamic programming algorithm to $O(\alpha k)$ limiting the choices of $j_i^1, \cdots, j_i^\alpha$ at each iteration.

Lemma 4 *Let $j_i^1, \cdots, j_i^\alpha$ denote the values for which the recursive definition of $t[i]$ achieves its minimum value (i.e., the best way to split a tree on i nodes is into subtrees of sizes $j_i^1, \cdots, j_i^\alpha$ and $i - \sum_{r=1}^{\alpha} j_i^r$). Then, $j_2^1 = 1$ and $j_2^r = 0$ for $2 \leq r \leq \alpha$; and for $2 \leq i \leq k-1$, (i) $j_{i+1} = j_i$ or $j_{i+1} = j_i + 1$ and (ii) $j_{i+1}^r = j_i^r$ or $j_{i+1}^r = j_i^r + 1$ for $1 \leq r \leq \alpha$.*

Proof. The proof is similar to the proof of Lemma 3. Suppose the lemma does not hold for some value of α, say α_0, for $1 \leq \alpha \leq \alpha_{max}$. Let i_0 be such that for any optimal multicast tree with i_0 nodes, either $j_{i_0} \geq j_{i_0-1}+2$ or $j_{i_0}^{r_1} \geq j_{i_0-1}^{r_1}+2$ for some r_1, $1 \leq r_1 \leq \alpha_0$. Choose such an optimal multicast tree T^0 with i_0 nodes (T^0 has subtrees with j_{i_0}, $j_{i_0}^1, \cdots, j_{i_0}^{\alpha_0}$ nodes) with the minimum value of $\mathcal{A}(T^0)$, where

$$\mathcal{A}(T^0) = \mid j_{i_0} - j_{i_0-1} \mid + \sum_{r=1}^{\alpha_0} \{ \mid j_{i_0}^r - j_{i_0-1}^r \mid \}. \tag{4.6}$$

Suppose $j_{i_0} \geq j_{i_0-1} + 2$. Then, there exists r_0 such that

$$j_{i_0}^{r_0} \leq j_{i_0-1}^{r_0} - 1. \tag{4.7}$$

By deleting one node from the subtree with j_{i_0} nodes and adding one node to the subtree with $j_{i_0}^{r_0}$ nodes, the multicast latency of the decreased subtree is

$$\begin{aligned} & t[j_{i_0} - 1] + t_{hold} \\ & \leq t[j_{i_0}] + t_{hold} \\ & \leq t[i_0], \qquad \text{from the definition of } t[i_0] \end{aligned}$$

and the multicast latency of the increased subtree is

$$
\begin{aligned}
& t[j_{i_0}^{r_0} + 1] + t_{end} + (r_0 - 1)t_{int} \\
\leq\ & t[j_{i_0-1}^{r_0}] + t_{end} + (r_0 - 1)t_{int} \quad \text{from Equation (4.7)} \\
\leq\ & t[i_0 - 1] \qquad\qquad \text{from the definition of } t[i_0 - 1] \\
\leq\ & t[i_0]
\end{aligned}
$$

The multicast latency of this new tree T^1 is still optimal (i.e., $t[i_0]$) and $\mathcal{A}(T^1) < \mathcal{A}(T^0)$. This is a contradiction to the choice of T^0.

Next, assume that $j_{i_0}^{r_1} \geq j_{i_0-1}^{r_1} + 2$ for some r_1, $1 \leq r_1 \leq \alpha_0$. By deleting one node from the subtree with $j_{i_0}^{r_1}$ nodes, the multicast latency of the resulting subtree is

$$
\begin{aligned}
& t[j_{i_0}^{r_1} - 1] + t_{end} + (r_1 - 1)t_{int} \\
\leq\ & t[j_{i_0}^{r_1}] + t_{end} + (r_1 - 1)t_{int} \\
\leq\ & t[i_0] \qquad\qquad \text{from the definition of } t[i_0].
\end{aligned}
$$

We also note that as $j_{i_0}^{r_1} \geq j_{i_0-1}^{r_1} + 2$, either

$$j_{i_0} \leq j_{i_0-1} - 1 \tag{4.8}$$

or for some r_0, $1 \leq r_0 \leq \alpha_0$,

$$j_{i_0}^{r_0} \leq j_{i_0-1}^{r_0} - 1. \tag{4.9}$$

When Equation (4.8) holds, by adding the node deleted from the subtree with $j_{i_0}^{r_1}$ nodes to

the subtree with j_{i_0} nodes, the multicast latency of the increased subtree is

$$
\begin{aligned}
& t[j_{i_0}+1]+t_{hold} \\
& \le t[j_{i_0-1}]+t_{hold} && \text{from Equation (4.8)} \\
& \le t[i_0-1] && \text{from the definition of } t[i_0-1] \\
& \le t[i_0].
\end{aligned}
$$

Hence, the resulting multicast tree, say T^2, is still optimal.

When Equation (4.9) holds, by adding the node deleted from the subtree with $j_{i_0}^{r_1}$ nodes to the subtree with $j_{i_0}^{r_0}$ nodes, the multicast latency of the increased subtree is

$$
\begin{aligned}
& t[j_{i_0}^{r_0}+1]+t_{end}+(r_0-1)t_{int} \\
& \le t[j_{i_0-1}^{r_0}]+t_{end}+(r_0-1)t_{int} && \text{from Equation (4.9)} \\
& \le t[i_0-1] && \text{from the definition of } t[i_0-1] \\
& \le t[i_0].
\end{aligned}
$$

Again, the resulting multicast tree, say T^3, is optimal. Finally, we note that $\mathcal{A}(T^2) < \mathcal{A}(T^0)$ for the former case and $\mathcal{A}(T^3) < \mathcal{A}(T^0)$ for the latter case. Either case is a contradiction to the choice of T^0. This completes the proof of the lemma. ∎

Note that Lemma 4 implies $t[i] = \min\{A,\ B\}$ for $i > 1$, where

$$A = \max\{t[i-1],\ t[j_{i-1}+1]+t_{hold}\} \tag{4.10}$$

and

$$B = \min_{1 \le r \le \alpha} \{\max\{t[i-1],\ t[j_{i-1}^r + 1] + t_{end} + (r-1)t_{int}\}\}. \tag{4.11}$$

As Equation (4.11) implies

$$B = \max\{t[i-1], \min_{1 \le r \le \alpha} \{t[j_{i-1}^r + 1] + t_{end} + (r-1)t_{int}\}\},$$

this together with the fact that $t[i] = \min\{A,\ B\}$ gives

$$t[i] = \max\{t[i-1],\ \min\{t[j_{i-1} + t_{hold}, \min_{1 \le r \le \alpha} \{t[j_{i-1}^r + 1]$$

$$+t_{end} + (r-1)t_{int}\}\}\}. \tag{4.12}$$

We now revise our dynamic programming algorithm based on Equation (4.12) which runs in $O(\alpha k)$ time in the following.

$$t[i] = \begin{cases} 0 & \text{if } i = 1 \\ t_{end} & \text{if } i = 2 \\ \max\{t[i-1],\ \min\{t[j_{i-1} + 1] + t_{hold}, & \\ \quad \min_{1 \le r \le \alpha}\{t[j_{i-1}^r + 1] + t_{end} & \\ \quad +(r-1)t_{int}\}\}\} & \text{if } i \ge 3 \end{cases}$$

where $j_2^1 = 1$ and $j_2^r = 0$ for $2 \le r \le \alpha$.

Theorem 3 *The optimal multicast latency of a k-node tree can be computed in $O(\alpha k)$ time when each node has α communication ports.*

Table 4.4: The optimal multicast tree of a 3-port communication architecture with $k = 12$, $t_{int} = 10$, $t_{hold} = 22$, and $t_{end} = 55$. Columns C_1 - C_4 denote $t[j_{i-1} + 1] + t_{hold}$, $t[j^1_{i-1} + 1] + t_{end}$, $t[j^2_{i-1} + 1] + t_{end} + t_{int}$, and $t[j^3_{i-1} + 1] + t_{end} + 2t_{int}$, respectively.

i	j_i	j_i^1	j_i^2	j_i^3	C_1	C_2	C_3	C_4	$t[i]$
1	-	-	-	-	-	-	-	-	0
2	1	1	0	0	-	-	-	-	55
3	1	1	1	0	77	110	65	75	65
4	1	1	1	1	77	110	120	75	75
5	2	1	1	1	77	110	120	130	77
6	3	1	1	1	87	110	120	130	87
7	4	1	1	1	97	110	120	130	97
8	5	1	1	1	99	110	120	130	99
9	6	1	1	1	109	110	120	130	109
10	6	2	1	1	119	110	120	130	110
11	7	2	1	1	119	120	120	130	119
12	7	3	1	1	121	120	120	130	120

Table 4 shows an example for $k = 12$, $t_{int} = 10$, $t_{hold} = 22$, $t_{end} = 55$, and $\alpha = 3$.

4.6 Conclusion

Designing a portable algorithm to achieve good performance on different parallel platforms is highly demanded. Based on the proposed parameterized communication model, efficient methods to construct optimal multicast trees are proposed for both 1-port and α-port communication architectures. Here the term "*optimal*" applies on the basis of the parameterized communication model, not on any specific parallel machine. The proposed communication model is more suitable for those machines supporting cut-through switching and having a rich interconnection topology (i.e., the network is able to support many simultaneous transmissions without much contention). For such parallel machine, we claim that the proposed architecture-independent multicast algorithm is near-optimal as it is obviously that the per-

formance of an algorithm can be improved by considering some machine-specific features, such as the network topology and routing algorithm, which cannot be captured as a generic system parameter.

The proposed parameterized communication model is useful in the design of portable communication libraries. Many techniques used in this paper can be extended to implement some other collective communication services, such as scatter. When a system changes or upgrades its critical component, such as new processors, new host interface, or new communication protocols, the system has to run some benchmark programs to obtain the new measure of system parameters. The corresponding communication library has to be recompiled to take effect of those new system parameters.

Chapter 5

Platform-Dependent Tuning Multicast

In Chapter 4, we proposed a multicast algorithm called *OPT-tree* which is truly portable and still provides good performance. We proved that the OPT-tree algorithm generates an optimal multicast tree. The optimality is based on the assumption that the underlying networks has no communication contention. In other words, the two network parameters, holding latency and end-to-end latency, of a point-to-point communication between any two nodes must remain constant, for a given message size, regardless of the location of the nodes. This assumption is justifiable if the network is logically fully connected. For systems using wormhole-switching and carrying small messages, the proposed multicast algorithm does provide near optimal performance.

To achieve a truly optimal performance, it is necessary to consider the architecture-dependent characteristics of a system, such as network topology and switching mechanism. In most real networks, network contention is likely to occur if concurrent message transmissions are not scheduled properly. In this case, the actual multicast latency of a multicast tree can be longer than expected. Several works have been done in order to construct

contention-free multicast trees for various network topologies, for example the U-mesh algorithm [44] for mesh network and the U-min algorithm [45] for bi-directional multistage interconnection networks (BMINs). However, most algorithms are based on the binomial tree which is efficient only on the networks with some restricted network parameters.

In this chapter, we study the architecture-dependent tuning of the parameterized multicast algorithm such that it can achieve truly optimal performance. We examine how to order nodes in the multicast tree to avoid the contention based on the underlying network topology. We consider wormhole-switched mesh networks and BMINs which are popular in the market, such as the Intel Paragon and the IBM/SP series.

5.1 The Problems

Given t_{hold} and t_{end}, the OPT-tree algorithm has been proved to construct the optimal architecture-independent multicast trees. This optimality bases on the assumption that t_{hold} and t_{end} remain constant for a given message size regardless of the ordering of the nodes in the multicast tree. This assumption is true on some specific networks such as fully-connected network. However, this may not be the case for all other networks, especially for the network that employs wormhole-switching as the switching mechanism. Although the communication latency in wormhole-switching network is distance-insensitive, one drawback of the wormhole-switching network is that the contention, when occurs, can prolong t_{end}. As the software-based multicast involves several point-to-point communications, the contention is possible which could increase the multicast latency of the tree, and hence, prevent the tree from becoming optimal. The following example is used to illustrate issues

and difficulties involved in implementing optimal multicast communication in wormhole-switching network.

Figure 5.1 shows an optimal multicast tree of 7 nodes when implemented in the 16-port BMIN network with t_{hold} = 20 and t_{end} = 55. Suppose a multicast message is sent from source 0011 to six destinations 0001,0110,0111,1010,1100,1101. Let us consider the contention in term of the *multicast step* defined to be a sending step in a multicast tree. At step 2, the channel collision is possible since the message from node 0011 to node 0111 and from node 0001 to node 0110 use a common channel. When the channel collision occurs between two messages transmitted simultaneously within the same step in a multicast tree is known as *stepwise contention* [44]. If we consider the latency of the multicast tree, we found that the stepwise contention at step 2 does occurs. Both messages use the same channel at the same time as the sender node 0011 and node 0001 start sending at time 20 and 55 respectively. The other type of the contention is based on the fact that the send may not start at the same time since sending in a multicast tree is not synchronous. This type of contention is called *depth contention*. In the example in Figure 5.1, the depth contention occurs between the message from node 0111 to node 1100 and from node 0011 to node 1101.

The best solution for avoiding the contention problem is to prevent a common channel used by two different senders at any time. This can be done by ordering the node so that the contention can be avoided. Let consider an example in Figure 5.2. Suppose the multicast tree in Figure 5.2(a) is optimal. However, the contention occurred when node 0 sends to node 2 and node 5 sends to node 6 can prevent the multicast tree to achieve its truly optimal performance. By reordering nodes in the multicast tree based on the underlying

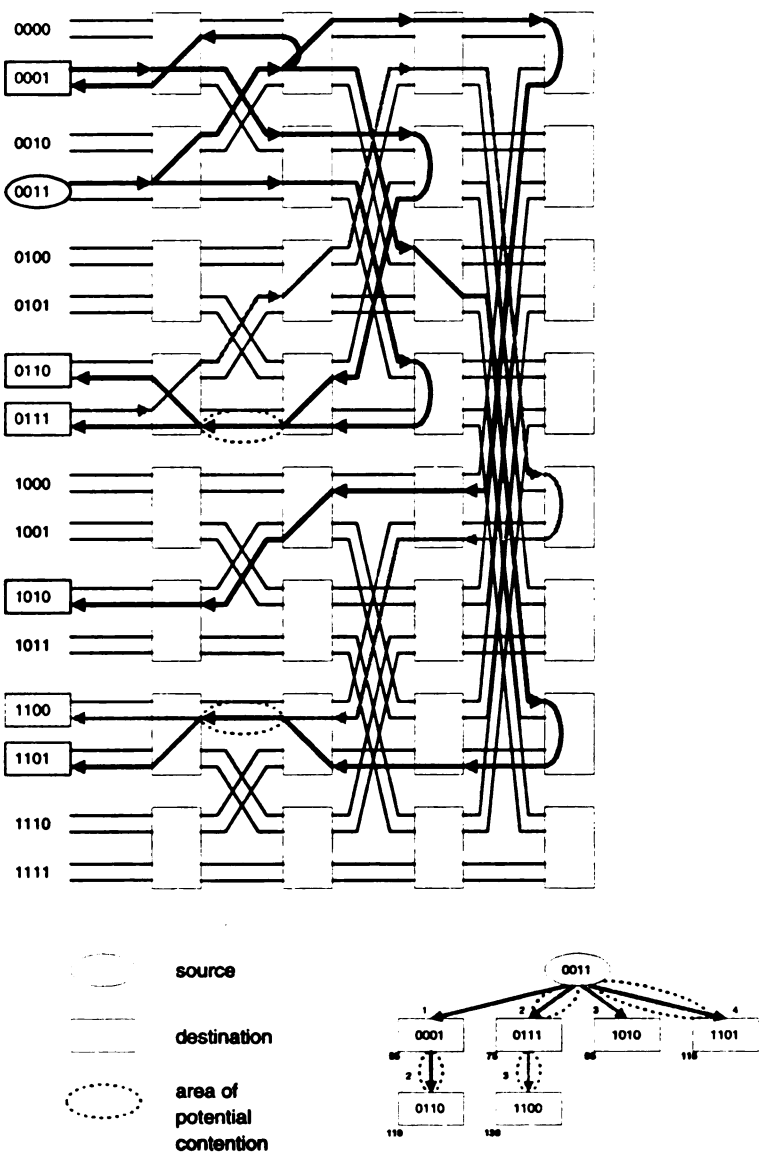


Figure 5.1: An optimal multicast tree which channel contention are possible.

network characteristics, the contention can be eliminated as shown in Figure 5.2(b). Thus, we can reorder multicast nodes in Figure 5.1 such that the multicast tree is contention-free. Figure 5.3 shows the same tree as in Figure 5.1 with a new ordering scheme. Obviously, no channel is shared by more than one message at any time. Thus, no contention occurs.

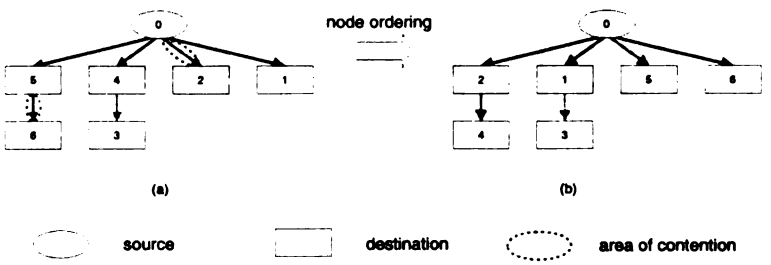


Figure 5.2: (a) Contention is possible. (b) Contention-free multicast.

5.2 Optimal Multicast in Mesh Networks

Constructing contention-free multicast trees on wormhole-routing mesh network was studied in [44]. Based on the dimension-ordered chain and recursive-doubling technique, U-mesh algorithm can construct an efficient contention-free multicast tree for a mesh network. Observe that U-mesh trees are binomial trees, and binomial trees are optimal only if they are implemented on networks with $t_{hold} = t_{end}$. In this section, we propose a new algorithm called OPT-Mesh which is the minimum time implementation of the parameterized multicast tree on a mesh network. Based on the dimension-ordered chain and parameterized multicast tree, OPT-Mesh tree can construct a contention-free optimal multicast tree that matches network parameters. We base our proof on some notations and results discussed in [44].

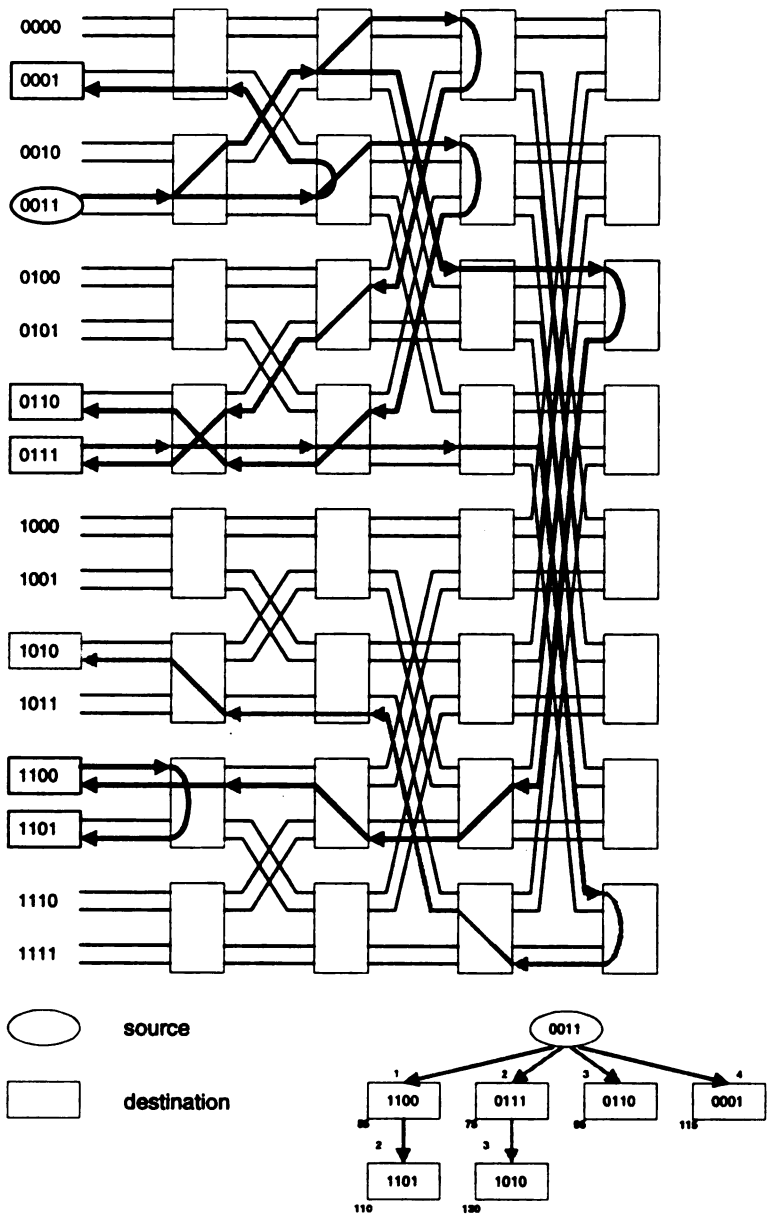


Figure 5.3: A contention-free ordering scheme of the multicast tree.

The address of a node x in a finite n-dimensional mesh with radix m is represented by $\sigma_{n-1}(x)\sigma_{n-2}(x)\cdots\sigma_0(x)$, where $\sigma_i(x) \in \{0, 1, \cdots, m-1\}$. The path from node a to node b resulting from dimension-ordered routing is denoted by $\rho(a, b) = (a; x_1, x_2, \cdots, x_k; b)$, where the x_i's are the sequence of intermediate routers constituting the routing path. The binary relation *dimension order*, denoted by $<_d$, is defined between two nodes a and b as follows: $a <_d b$ if and only if either $a = b$ or there exists a j such that $\sigma_j(a) < \sigma_j(b)$ and $\sigma_i(a) = \sigma_i(b)$ for all i, $j + 1 \leq i \leq n - 1$. A sequence of nodes $\{x_1, x_2, \cdots, x_m\}$ is a *dimension-ordered chain* if and only if all the elements are distinct and the sequence is dimension-ordered, that is, if $x_i <_d x_j$ for all i and j, $1 \leq i < j \leq m$.

Theorem 4 *[44] If $u <_d v <_d x <_d y$, then (i) $\rho(u, v)$ and $\rho(x, y)$ are arc-disjoint, (ii) $\rho(y, x)$ and $\rho(v, u)$ are arc-disjoint, and (iii) $\rho(v, u)$ and $\rho(x, y)$ are arc-disjoint.*

Let j_i for $1 \leq i \leq k$ be the output value computed by the OPT-tree algorithm shown in Algorithm 4.3.1. (The parameterized multicast tree with i nodes has two subtrees, where the size of the subtree containing the root is j_i and the size of the other subtree is $i - j_i$.) The OPT-mesh algorithm is given in Algorithm 5.2.1. The source and destination addresses are sorted into a dimension ordered chain denoted Φ. The source node successively divides Φ of size i into two parts of sizes j_i and $i - j_i$. If the source node is in the lower part, then it sends a copy of the message to the lowest node (with respect to $<_d$) in the upper part. This node will be responsible for delivering the message to the other nodes in the upper part, using the same OPT-mesh algorithm. If the source node is in the upper part, then it sends a copy of the message to the highest node (with respect to $<_d$) in the lower part. Again, this node will be responsible for delivering the message to the other nodes in the lower

part using the same OPT-mesh algorithm. In addition to the data, each message carries the addresses of the destinations for which the receiving node is responsible. At each step, the source deletes from Φ the receiving node and those nodes in the part not containing the source. The source continues this process until Φ contains only its own address.

Algorithm 5.2.1 OPT-MESH

Input: Φ: dimension-ordered chain $\{x_l, x_{l+1}, \cdots, x_r\}$ for source and destinations.
x_s: the address of source node.
j_i: the size of the subtree containing the source, where $i = r - l + 1$.
Procedure
While $l < r$ **do**
if $s < l + j_i$ **then**
$rec = l + j_i$;
$D = \{x_{rec}, x_{rec+1}, \cdots, x_r\}$;
$r = rec - 1$;
else
$rec = r - j_i$;
$D = \{x_l, x_{l+1}, \cdots, x_{rec}\}$;
$l = rec + 1$;
endif
Send a message to node x_{rec} with the address field D;
endwhile

A multicast implementation using the OPT-mesh algorithm is shown in Figure 5.4. A multicast with 7 destination nodes is considered in a 6×6 2-D mesh where $t_{hold} = 20$ and $t_{end} = 55$. Table 1 shows each value j_i for $1 \leq i \leq 8$ computed by the OPT-tree algorithm. Node $(3, 2)$ is the source of a multicast message destined for 7 nodes $\{(1, 5), (2, 1), (3, 4), (4, 3), (4, 4), (5, 1), (5, 4)\}$. As shown in Figure 5.4, the 8 nodes are initially sorted into the dimension ordered chain. The source $(3, 2)$ first sends to node

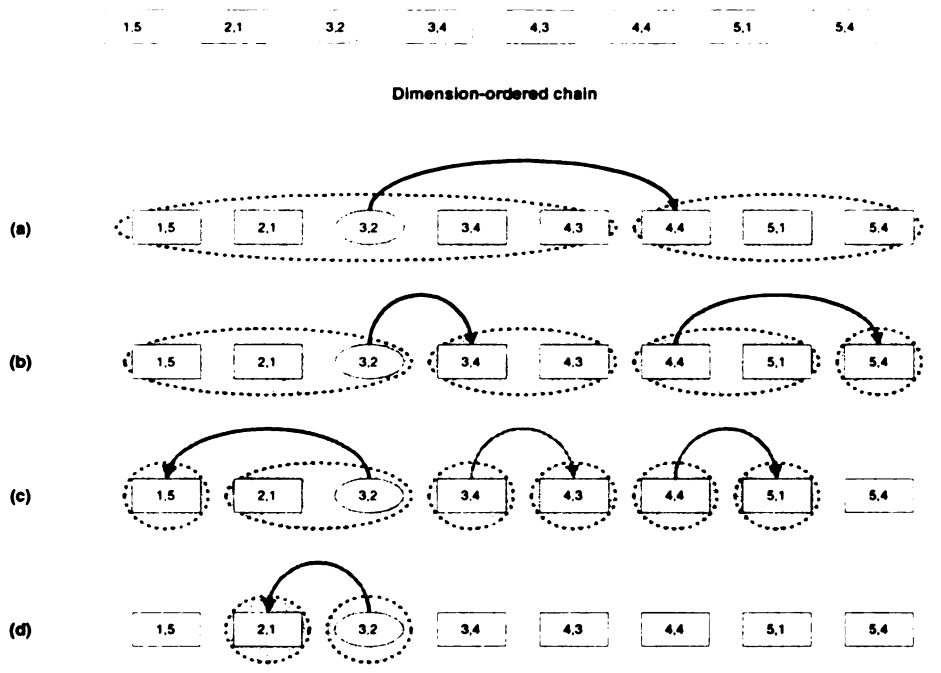


Figure 5.4: An example of using the OPT-mesh algorithm

$(4,4)$, the node with the lowest address in the upper $8 - j_8$ nodes of Φ, where $j_8 = 5$. The upper 3 nodes are deleted from Φ, and therefore the nodes remaining in Φ are $\{(1,5),(2,1),(3,2),(3,4),(4,3)\}$. Source $(3,2)$ next sends to $(3,4)$, the node with the lowest address in the upper $5 - j_5$ nodes of Φ, where $j_5 = 3$. Each of receiving nodes is like wise responsible for delivering the message to the nodes in its subtree using the same algorithm. The optimal multicast tree obtained by the OPT-mesh algorithm is shown in Figure 5.5(a). Figure 5.5(b) shows a multicast tree obtained by the U-mesh algorithm [44] based on the binomial tree construction. The multicast latency when implementing the tree in Figure 5.5(a) is at least 130 and the multicast latency when implementing the tree in Figure 5.5(b) is at least 165. As the U-mesh algorithm guarantees contention-free paths at each step, the lower bound 165 is achieved. In the following, we discuss that the OPT-mesh algorithm also guarantees contention-free paths during any time of multicasting process, which

shows that the lower bound 130 of the multicast latency can also be achieved. Figure 5.6 shows unicast paths obtained by the OPT-mesh algorithm for the above example.

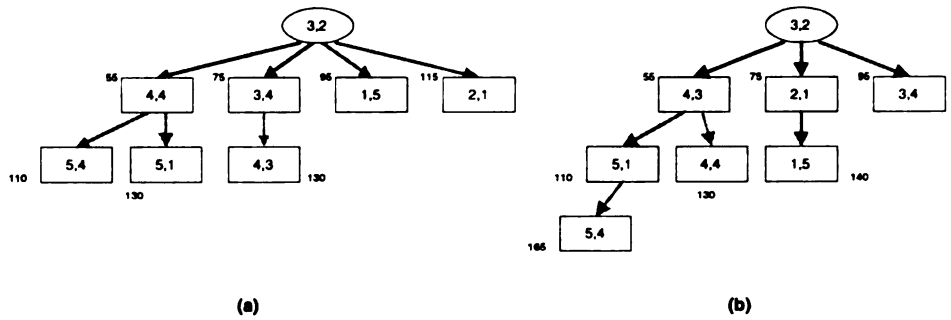


Figure 5.5: Comparison of two multicast trees in mesh network

An inspection of Figure 5.6 shows that there may exist a collision on link $((4,4),(5,4))$ or $((3,2),(2,2)$. Path $\rho((4,4),(5,4))$ and path $\rho((4,4),(5,1))$ have a common link $((4,4),(5,4))$. However, link $((4,4),(5,4))$ used in path $\rho((4,4),(5,4))$ will be available from time $t_0 + t_{hold}$, where t_0 is the time when node $(4,4)$ starts to send the message to node $(5,4)$ (i.e., link $((4,4),(5,4))$ will be available at time 75). Hence, this link can be used in path $\rho((4,4),(5,1))$ without any contention. Similarly, link $((3,2),(2,2))$ is commonly used in paths $\rho((3,2),(1,5))$ and $\rho((3,2),(2,1))$. But link $((3,2),(2,2))$ will become available from time 60 for path $\rho((3,2),(2,1))$ after being used in path $\rho((3,2),(1,5))$. The following theorem proves that the dimension ordered routes produced by the OPT-mesh algorithm are contention-free.

Theorem 5 *The implementation of parameterized multicast trees in meshes using the OPT-mesh algorithm is optimal.*

Proof: Let u, v, x, and y be four distinct nodes. For any two paths $\rho(u,v)$ and $\rho(x,y)$ used by the OPT-mesh algorithm, there are only six possible orderings of u, v, x, and y. These

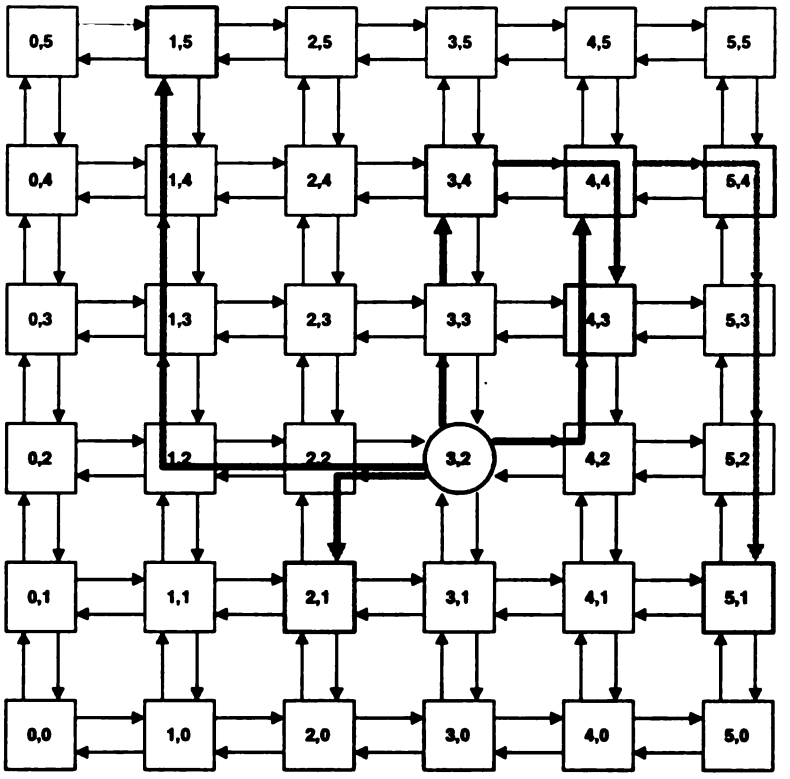


Figure 5.6: Multicast for 7 destinations in 2D mesh

are $u <_d v <_d x <_d y$, $v <_d u <_d x <_d y$, $v <_d u <_d y <_d x$, $x <_d y <_d u <_d v$, $y <_d x <_d u <_d v$, and $y <_d x <_d v <_d u$. In each case, $\rho(u, v)$ and $\rho(x, y)$ are arc-disjoint by Theorem 4.

Assume that some of u, v, x, and y may be an identical node. We then observe that if there exist two paths $\rho(u, v)$ and $\rho(x, y)$ used in the OPT-mesh algorithm such that $\rho(u, v)$ and $\rho(x, y)$ share an arc (or arcs), then u and x must be an identical node. Thus, if u sends a copy of the message to v starting at time t_0 and to y starting at time t_1 with $t_0 < t_1$, then $t_1 \geq t_0 + t_{hold}$. As the output port at u will become available at time $t_0 + t_{hold}$ after sending the message to v, arcs in the path $\rho(u, v)$ become successively available starting from the arc close to u. Therefore, there exists no collision between two unicast routes $\rho(u, v)$ and $\rho(u, y)$. This proves that the OPT-mesh algorithm produces a minimum-time

implementation.

5.3 Optimal Multicast in BMIN Networks

The issue of contentions using the software multicast in multistage interconnection network with turnaround routing was discussed by Xu and Ni[45]. They proposed the U-min algorithm which can construct a binomial multicast tree without stepwise and depth contention. As mentioned earlier, the binomial tree is guaranteed to be optimal only if it is implemented on the network with $t_{hold} = t_{end}$. Thus, the U-min tree is only a special case of parameterized multicast tree. To eliminate the overhead due to contentions, we propose an algorithm called OPT-min algorithm which tunes the parameterized multicast tree specifically for BMIN networks. The basic concept of OPT-min algorithm is based on the framework in [45].

Let each node x also denote its n-bit binary address. The ith bit of address x is denoted by $\sigma_i(x)$, $0 \leq i \leq n-1$, where $\sigma_0(x)$ represents the least significant address bit; hence, address x can be written as $\sigma_{n-1}(x)\sigma_{n-2}(x)\cdots\sigma_0(x)$. The binary relation *lexicographic order*, denoted by $a < b$, is defined between two nodes a and b such that $a < b$ if and only of the binary value of a is less than the binary value of b.

Theorem 6 *[45] In a multistage cube network with turnaround routing, if $u < v < x < y$, then (i) $\rho(u,v)$ and $\rho(x,y)$ are arc-disjoint, (ii) $\rho(y,x)$ and $\rho(v,u)$ are arc-disjoint, and (iii) $\rho(v,u)$ and $\rho(x,y)$ are arc-disjoint.*

This section describes an optimal implementation of parameterized multicast trees in multistage cube networks supporting turnaround routing. The OPT-min algorithm is given

in Algorithm 5.3.1. The source and destination addresses are first sorted into a lexicographic ordered chain. A lexicographic-ordered chain $\{d_l, d_{l+1}, \cdots, d_r\}$ is denoted as Φ, at the time when multicast is initiated by calling the OPT-min algorithm. The source successively divides Φ into two groups with j_i nodes and $i - j_i$ nodes, where $i = r - l + 1$. The source should be in the group with j_i nodes. If the source is in the lower group, then it sends a copy of the message to the smallest destination (with respect to the lexicographic order) in the upper group. That destination will be responsible for delivering the message to the other destinations in the upper group, using the same OPT-min algorithm. If the source is in the upper group, then it sends a copy of the message to the largest destination in the lower group. The source continues this procedure until Φ contains only its own address.

Algorithm 5.3.1 OPT-MIN ALGORITHM

Input: Φ: lexicographic-ordered chain $\{x_l, x_{l+1}, \cdots, x_r\}$ for source and destinations.
x_s: the address of source node.
j_i: the size of the subtree containing the source, where $i = r - l + 1$.
Procedure
While $l < r$ **do**
  if $s < l + j_i$ **then**
    $rec = l + j_i$;
    $D = \{x_{rec}, x_{rec+1}, \cdots, x_r\}$;
    $r = rec - 1$;
  else
    $rec = r - j_i$;
    $D = \{x_l, x_{l+1}, \cdots, x_{rec}\}$;
    $l = rec + 1$;
  endif
  Send a message to node x_{rec} with the address field D;
endwhile

A multicast implementation using the OPT-min algorithm is shown in Figure 5.7. A

multicast with 6 destination nodes is considered in a 16-node bidirectional butterfly multistage interconnection network with turnaround routing. Let $t_{hold} = 20$ and $t_{end} = 55$. The source begins with a lexicographic-ordered chain $\Phi = \{0001, 0011, 0110, 0111, 1010, 1100, 1101\}$. As shown in Figure 5.7, the set of nodes is partitioned into two parts of sizes j_7 and $i - j_7$, where $i = 7$ and $j_7 = 5$ (see Table 1). The source 0011 first sends to node 1100, the node with the lowest address in the upper partition of Φ. The upper part is deleted from Φ, and therefore the nodes remaining in Φ are $\{0001, 0011, 0110, 0111, 1010\}$. After t_{hold} time, the source 0011 can initiate another send operation to node 0111 since there are 5 nodes in this lower partition and these 5 nodes are further partitioned into two groups of sizes j_5 and $5 - j_5$ where $j_5 = 3$ (see Table 1). In the meanwhile, after t_{end} time, node 1100 sends the message to the node 1101. Each of the receiving nodes is likewise responsible for delivering the message to the nodes in its subtree using the same algorithm. This multicast implementation in Figure 5.7 requires $\max\{t_{hold} + 2t_{end}, 3t_{hold} + t_{end}\}$, which is 130. Figure 5.8 (a) shows a parameterized multicast tree obtained from the OPT-min algorithm and a tree with the same number of nodes obtained from U-min algorithm is shown in (b). Figure 5.3 shows unicast routes obtained by the OPT-min algorithm.

Theorem 7 *The implementation of parameterized multicast trees in multistage interconnection networks supporting turnaround routing using OPT-min algorithm is optimal.*

Based on Theorem 6, similar arguments in Theorem 5 can be applied to prove this theorem.

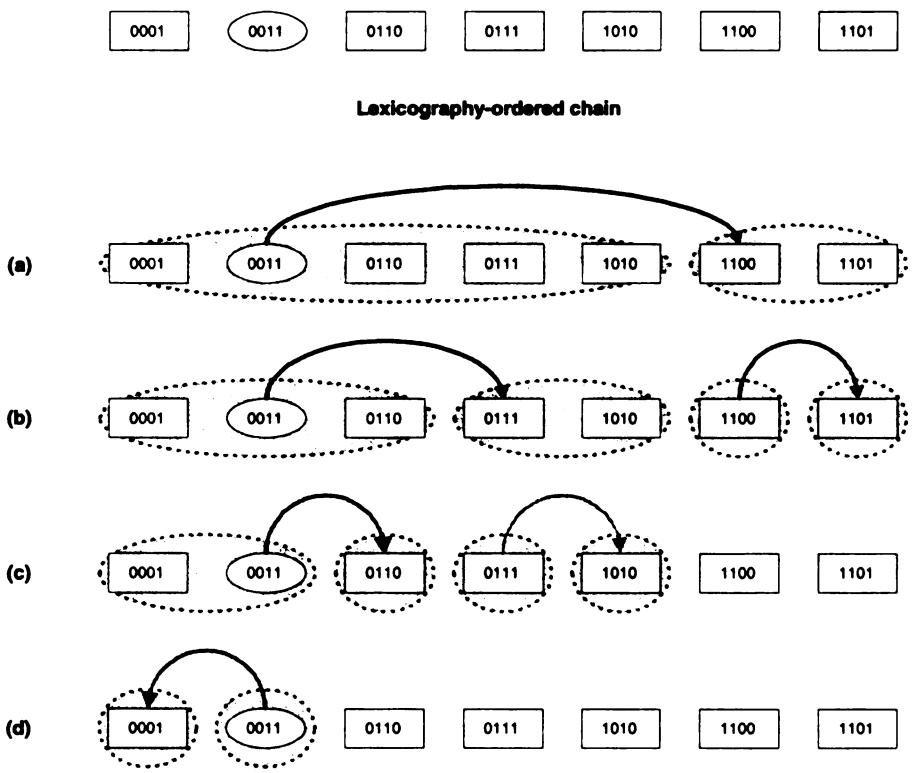


Figure 5.7: An example of using the Opt-min algorithm

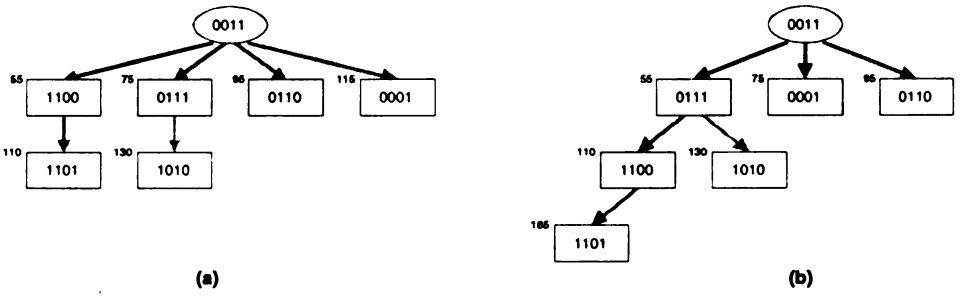


Figure 5.8: Comparison of two multicast trees in BMIN networks

5.4 Simulation Results

In order to evaluate our proposed algorithms, we implement a flit-level simulator for both wormhole-switched mesh and wormhole-switched BMIN topologies. The mesh network is based on a 16x16 topology supporting XY routing with one-port architecture. The BMIN network has 128 nodes based on 2 × 2 bidirectional switches. The network parameters of both simulators are $t_{hs} = 2000, t_{hd} = 2, t_{ns} = 1000, t_{nd} = 2, t_{rs} = 2500$, and $t_{rd} = 3$ which are similar to the network parameters of IBM/SP. As the locations of processors involving in the multicast service affect the probability of contention, we perform 16 independent experiments with the same input parameters, but different processor locations (randomly picked). Each data point presented in our results is the average of the multicast latency from all 16 experiments.

5.4.1 The Performance of the OPT-Mesh Algorithm

In Figure 5.9, we measure the multicast latency of the three multicast trees with 32 nodes on our mesh simulator. The OPT-mesh tree provides the best performance while the U-mesh tree performs the worst. Based on the fact that t_{hold} does not always equal to t_{end}, the

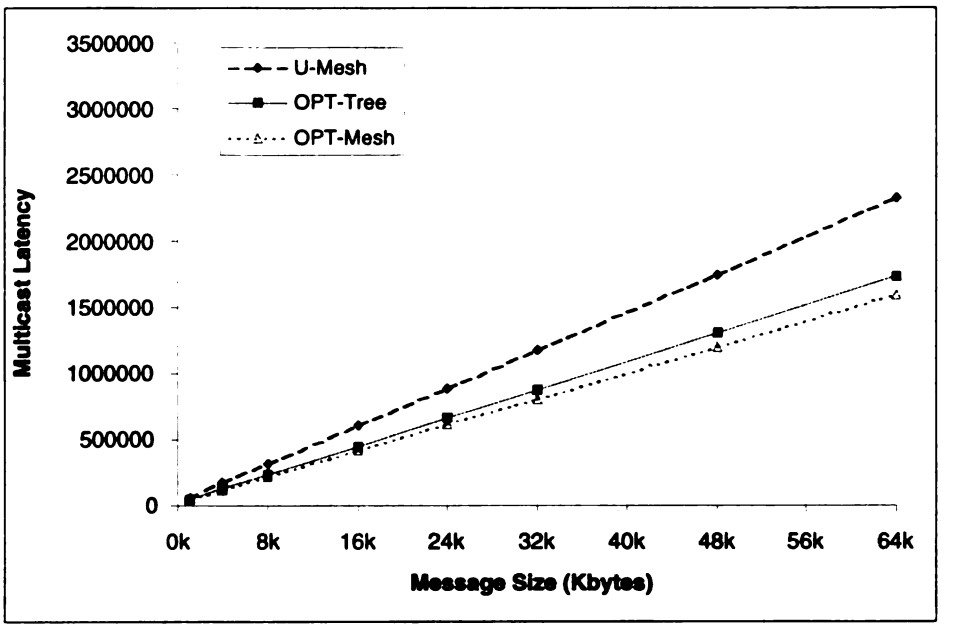


Figure 5.9: Comparison of 32-node multicast trees on a 16x16 mesh.

OPT-mesh algorithm and the OPT-tree algorithm can generate trees that are more suitable to the underlying network than the binomial tree generated by the U-mesh algorithm. Although the OPT-mesh tree and the OPT-tree tree have the same tree structure, the proper node ordering in the OPT-mesh algorithm eliminates the contention overhead which allows the OPT-mesh tree to achieve their theoretical lower bound. Figure 5.10 presents a similar experiment using 128-node multicast trees. The results are quite similar to the first experiment. However, the contention overhead of the OPT-tree tree is much higher since a large optimal multicast tree is deeper than a smaller optimal multicast tree. The deeper the multicast tree is, the more contention overhead occurs.

In both experiments, all multicast trees exhibit linear performance with respect to the message size. We can explain this phenomenon by considering the fact that the multicast latency of any tree is always dependent on the depth of the trees, the contention overhead,

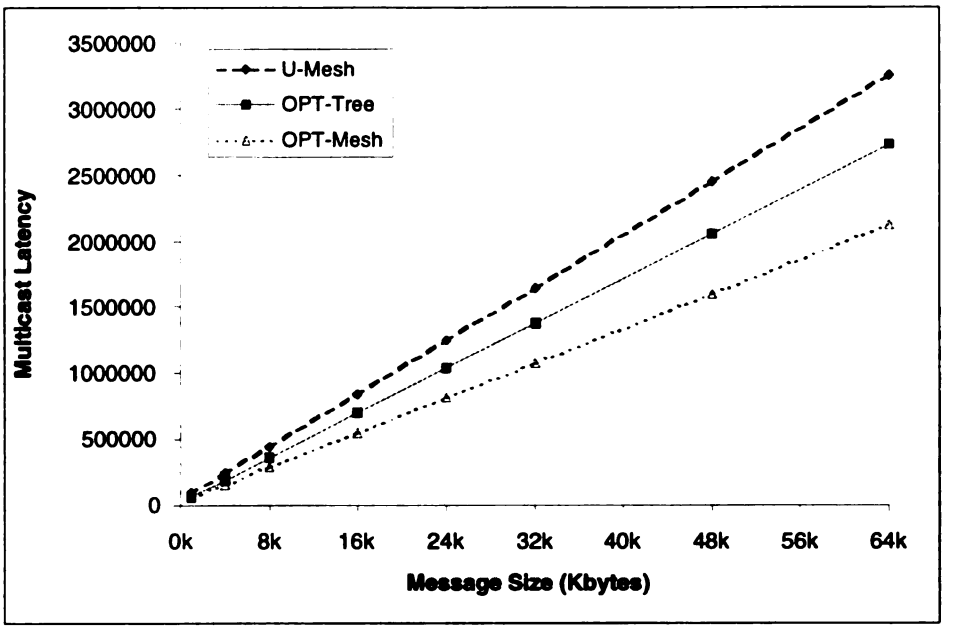


Figure 5.10: Comparison of 128-node multicast trees on a 16x16 mesh.

and the end-to-end latency of the underlying network. For a fixed multicast tree size, the U-mesh always generates a binomial tree with the same depth, regardless of the network parameters. As the U-mesh tree is contention-free, the multicast latency of the U-mesh tree depends on the end-to-end latency which also depends on the message size. Therefore, the multicast latency of the U-mesh tree is linear with respect to the message size. This is also true for both OPT-mesh tree and OPT-tree tree when the message size is large. For a large message size, the startup costs of t_{hold} and t_{end} are insignificant. In this case, the ratio of $\frac{t_{end}}{t_{hold}}$ which dictates the shape and depth of the OPT-mesh and OPT-tree trees is constant. Hence, the depth of both trees remains fixed. Thus, the performance of both trees are linear with respect to the message size.

Figure 5.11 and 5.12 demonstrate the performance of the multicast trees when the message size is 4 Kbytes and 64 Kbytes, respectively. The results confirm that the OPT-mesh

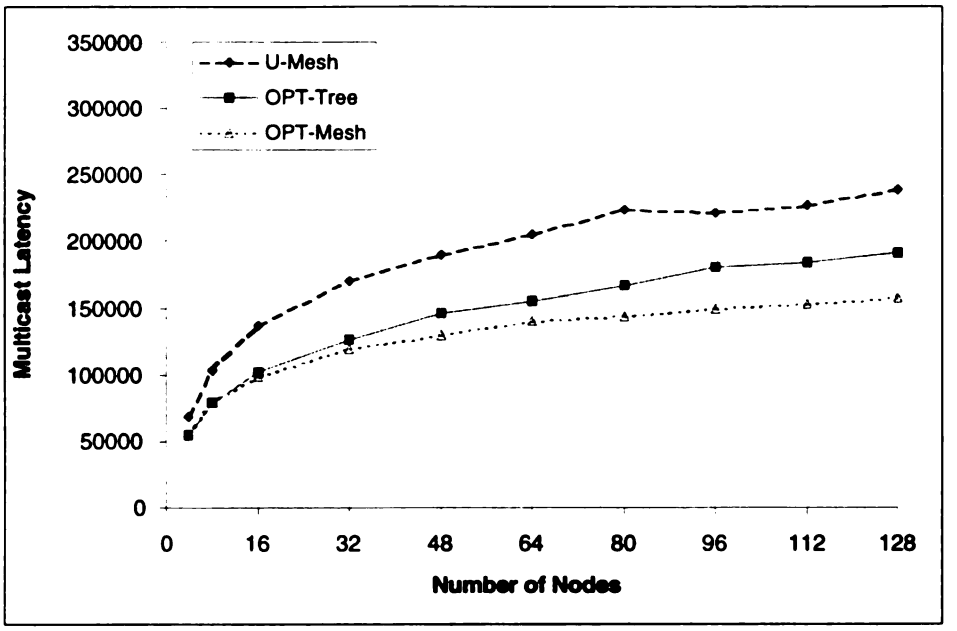


Figure 5.11: Comparison of 4-Kbyte multicast trees on a 16x16 mesh.

is more efficient than both OPT-tree and U-mesh. As the number of nodes in the multicast tree increases, the depth of the U-mesh tree increases faster than the depth of OPT-mesh and OPT-tree trees. Thus, the U-mesh tree gives the worst performance. For the OPT-tree tree, the increasing of number of nodes in the multicast tree means more messages are sent during the multicasting process. Hence, the contention probability also increases which leads to the increasing contention overhead. Moreover, the contention overhead increases quite noticeable when the depth of the multicast tree increases, for example, when the size of the multicast tree is 96 nodes in Figure 5.12.

5.4.2 Performance of the OPT-min Algorithm

We perform experiments on our BMIN simulator using the same network parameters used in the mesh experiments. The results are presented in Figure 5.13 and 5.14. As expected,

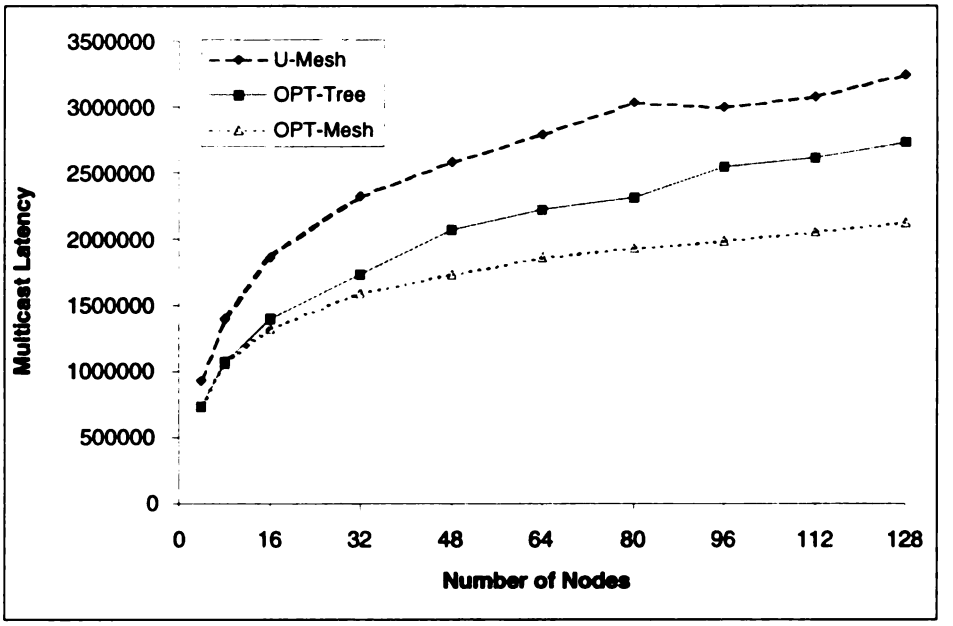


Figure 5.12: Comparison of 64-Kbyte multicast trees on a 16x16 mesh.

the multicast trees generated by OPT-min are more efficient than multicast trees generated by the other two algorithms. As message size become larger, the OPT-min tree performs much better. This is also true when the number of nodes increases. Note that the contention overhead in the OPT-tree tree is less sever. This implies that the OPT-tree tree in the BMIN network has less contention overhead than the similar trees in the mesh network, given that both networks have the same network parameters.

In order to support this claim, we define *multicast contention rate* as a metric to compare the effect of the contention overhead among different network topologies. Given a multicast algorithm A, the multicast contention rate (t_{c_A}) is the proportion of the overhead of the contention to the multicast latency of the multicast algorithm A. It is defined as:

$$t_{c_A} = \frac{(t_A - t'_A)}{t_A}$$

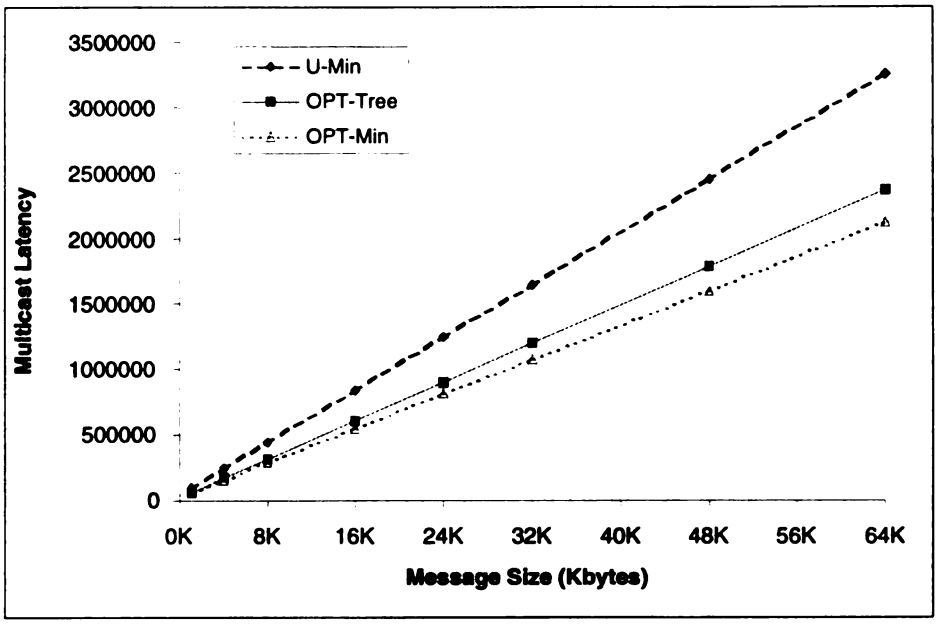


Figure 5.13: Comparison of 128-node multicast trees on 128-node BMIN.

where t_A is the multicast latency of algorithm A when contentions occur and t'_A is the theoretical lower bound of the multicast latency of algorithm A when no contention occurs.

Figure 5.15 shows the comparison of multicast contention rate of OPT-tree algorithm implemented in mesh and BMIN topologies. The results indicate that, under the same circumstances, multicasting in the BMIN network will suffer less contention overhead than the mesh network. This is because the BMIN network with turnaround routing has more communication paths between any pairs of nodes than the mesh network with X-Y routing which has only one path. These extra paths allow the BMIN network to reduce the effect of the contention.

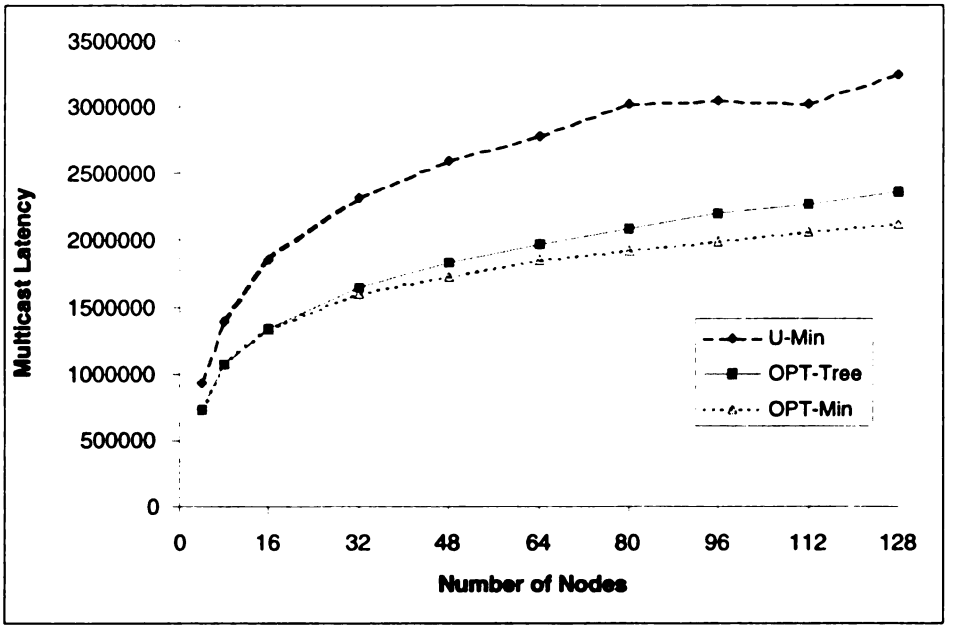


Figure 5.14: Comparison of 64-Kbyte multicast trees on 128-node BMIN.

5.5 Conclusion

In this chapter, we studied architecture-dependent tuning of the architecture-independent multicast algorithm. We have demonstrated that the effect due to network contention can prevent a parameterized multicast tree from achieving its optimal performance. We have proposed two algorithms, OPT-mesh and OPT-min, which generate contention-free parameterized multicast on wormhole-switched mesh and BMIN networks. However, our contention-avoidance technique is not restricted to these two networks. This concept can be applied to any network as long as the underlying networks can be partitioned into contention-free processor clusters.

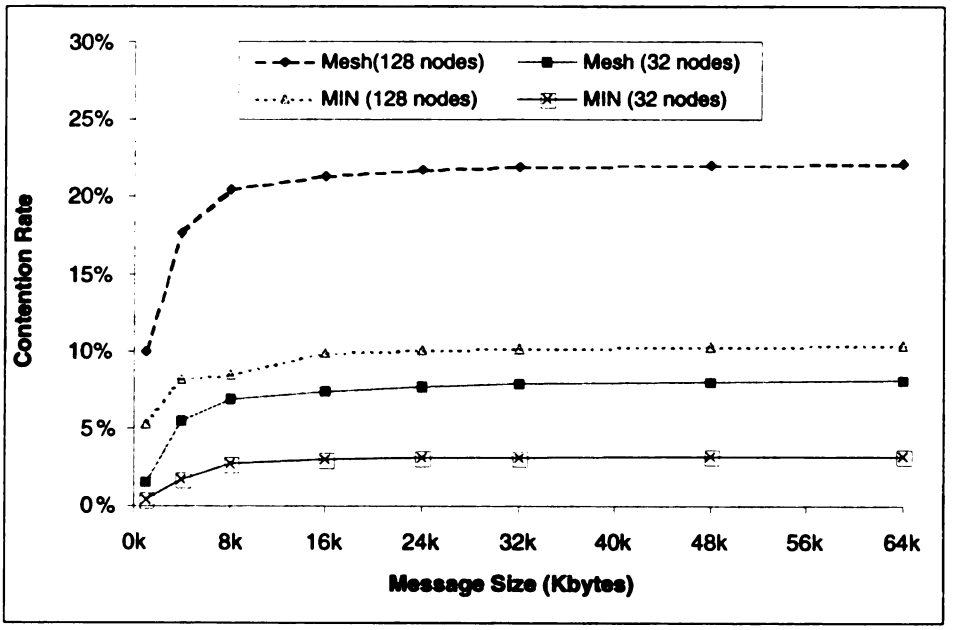


Figure 5.15: The effectiveness of topology against contention

Chapter 6

The Effect of Network Traffic on Multicast Behavior

Most multicast-related research, including our studies in the last two chapters, have been done under the ideal condition. Under this condition, it is commonly assumed that there is no other network traffic in the network, except the multicast traffic. Although some studies measured results in the actual network, the results may not actually include the effect from other network traffic. This is because, during the measurement, the processor-allocation schemes may guarantee contention-free or the measuring technique may focus only on the performance of multicast with no interference from other network traffics. For example, the effect of other network traffic can be eliminated by measuring performance of a multicast tree several times and choosing the minimum value. Thus, the performance of so-called optimal multicast algorithms when perform in the actual network may not be better or even worse than other conventional multicast algorithms such as the binomial multicast algorithm.

Typically, the actual network condition is not ideal. Besides the multicast traffic, other nodes outside the multicast group usually generate some network traffic as well. We called this kind of traffic the *background traffic*. Although sources and destinations of the background traffic do not belong to the multicast group, when the background traffic exists, they can interfere the progress of multicast operations. In addition to the background traffic, there are several factors that can effect the performance of a multicast service. During a multicast operation is taken place, nodes in the multicast group may generate more than just the multicast traffic. We called this traffic the *local traffic*. This traffic can also content with the multicast traffic. Another factor that must be taken into consideration is the inter-injection time of two consecutive multicasts at the same root node. If multicasts are injected too fast, the contention among these multicasts, even originated from the same root node, is highly possible which can result to performance degradation. This inter-injection time is often determined by the grain size at the root node.

The main focus of this study is to answer two important questions:

- How useful is an optimal multicast when the background traffic exists ?
- Can we possibly propose an algorithm to handle the dynamic environment due to the background traffic ? And how much improvement should we expect ?

6.1 Study Model

To obtain our goals, it is important to be able to control several system parameters, including the pattern of the background traffic, applied workload, etc. Obviously, this is very difficult to achieve if we perform our study on a real system. Thus, we use our flit-level

simulators to study the effects of all aforementioned factors. To make our work manageable, but not too far from realistic, our simulators are based on three models: the system model, the communication model, and the workload model.

6.1.1 System Model

In this study, we implement two simulators. Both simulators have 256 nodes connected to a wormhole-switched Bidirectional MIN-based network with turn-around routing (BMIN) and 16x16 mesh network respectively. Both systems are based on a single-port architecture which implies that each node has one incoming channel and one outgoing channel. In addition, we assume the communication subsystem to provide only basic point-to-point communication services, send and receive operations. These services are assumed to be reliable. There is no hardware support for either multicast or other collective communication services.

Actual parallel systems may implement some processor scheduling and allocation policies such as contention-free allocation, fragmentation-free allocation, and gang-scheduling in order to maintain high utilization by supporting multiple independent applications simultaneously. To simplify our study model, we assume nodes in the system are partitioned into two groups, *multicast group* and *background group*. Members of each group communicate to the others in the same group only. In other words, there is no traffic between groups.

For the multicast group, all members in the multicast group participate in the multicast operation. Among them, one member in the multicast group is selected as a root node to send multicast messages to all other nodes in the group. Other nodes in the group, upon re-

ceiving a multicast message, forward the message to others based on the multicast structure being studied. Most experiments in this study assume that the root node is the only sender in the group. In some experiments, however, all members in the multicast group, except the root node, are allowed to send point-to-point messages to other members in the group in order to study behavior of multicast when the local traffic is presented. The remaining nodes in the system form the background group. All members in this group communicate to the others via point-to-point communication services only. The main purpose of this group is to generate the background traffic.

6.1.2 Communication Model

We assume that the communication subsystem supports only two point-to-point communication service: send and receive. Both send and receive operations are assumed to be blocking operations. Our semantic of blocking operation is similar to MPI's definition which specifies that a blocking operation returns when user's buffer can be reused. However, upon returning, the sending message may not be completely delivered to the destination.

In general, most parallel systems allocate kernel memory as communication buffers to guarantee the reliable delivery and improve the CPU utilization. Recent studies suggest to move communication buffers to network interface which can significantly reduce the software overhead [63]. Regardless of its location, the communication buffer usually causes a problem called *head-of-line blocking* (HOL). A message encounters a HOL blocking when it is sent while another message is occupying the out-going channel. In this case, it will be

placed in the sending buffer (*output contention*) and have to wait until all preceding messages in the buffer are sent. The delay due to this problem is quite high, especially when the network load is heavy. To emulate this system characteristic, we assume that each node has two communication buffers, sending and receiving buffers. When an incoming channel is not available, outgoing message will be placed in the sending. If the sender tries to issue a send operation and the sending buffer is full, the sender will be blocked. The sender resumes its execution when there is enough space available in the sending buffer. On the receiving side, an incoming message will be put in the incoming buffer and wait for the receiver to pick it up. The receiver is blocked if it tries to receive and there is no message available in the receiving buffer.

The communication latency model used in this study is based on the Parameterized Communication Model presented in Chapter 2. Basically, this model consists of five parameters as follow:

- holding latency (t_{hold}) : minimum interval between two consecutive send operations
- end-to-end latency (t_{end}) : interval between the sender starts sending a message until the receiver finishes receiving.
- sending latency (t_{send}) : software latency at the sender
- network latency (t_{net}) : time required to transmit a message across the network
- receiving latency (t_{recv}) : software latency at the receiver

Theoretically, when sending a message size m, all parameters can be decomposed into two

components, startup latency and delay latency. Thus:

$$
\begin{aligned}
t_{send} &= t_{ss} + m.t_{sd} \\
t_{net} &= t_{ns} + m.t_{nd} \\
t_{recv} &= t_{rs} + m.t_{rd} \\
t_{hold} &= t_{hs} + m.t_{hd} \\
t_{end} &= t_{es} + m.t_{ed} \\
&= (t_{ss} + t_{ns} + t_{rs}) + m.(t_{sd} + t_{nd} + t_{rd})
\end{aligned}
$$

Based on techniques in Chapter 2, we can measure all five parameters and their components. Note that t_{ns} is the fixed delay incurred at the sender when it prepares to transmit a message *from its sending buffer to its out-going channel.* Moreover, we assume that t_{nd} is the delay incurred in a channel when transmitting a flit (one byte) over the channel. This includes the delay in the channels connecting a node to a switch and a switch to a switch.

In reality, user's program may not be the only application running in the system. It may have to share some resources such as processor, storage, especially network, with other applications. Thus, the network contention among different applications is possible. We define t_{cont} to be the delay due to network contention. Furthermore, since we assume that each node has sending and receiving buffers, we define t_{sq} and t_{rq} to be the waiting time of a message at the sending buffer and receiving buffer respectively.

When a message occupies the out-going channel of the sender, it may causes blocking to subsequence messages. We define t_{out} to be the interval that a message occupies the out-

going channel. In addition, a message, which is either occupying the out-going channel or waiting in the sending buffer, may cause HOL blocking to subsequence messages. We define t_{hol} to be the interval that a message causes HOL blocking delay to subsequence messages which is $t_{sq} + t_{out}$. We can summarize the relationship of these parameters as follows:

$$
\begin{aligned}
t_{send} &= t_{ss} + m.t_{sd} + t_{sq} \\
t_{net} &= t_{ns} + m.t_{nd} + t_{conf} \\
t_{recv} &= t_{rs} + m.t_{rd} + t_{rq} \\
t_{end} &= t_{send} + t_{net} + t_{recv} \\
t_{out} &= t_{ns} + m.t_{nd} + t_{cont} \\
t_{hol} &= t_{sq} + t_{out}
\end{aligned}
$$

Figure 6.1 demonstrates the timing diagram of sending a message from P_0 to P_1. Let $t_{hold} = t_{send} = 20$, $t_{out} = 15$, $t_{net} = 20$, $t_{recv} = 20$ when there is no contention. Let C_0 represent the out-going channel occupation at P_0. Let assume that the out-going channel from P_0 is not available until $t = 50$ and there is no other message in the sending queue. Assume there is no message in the receiving buffer at P_1 and P_1 is ready to receive. Suppose the contention in the network (t_{cont}) causes a delay for 15 time units. In this example, network parameters of this message become $t_{sq} = 30$, $t_{rq} = 0$, $t_{cont} = 15$, $t_{out} = 30$, $t_{hol} = 60$, $t_{send} = 50$, $t_{net} = 35$, $t_{recv} = 20$, $t_{hold} = 20$, and $t_{end} = 105$

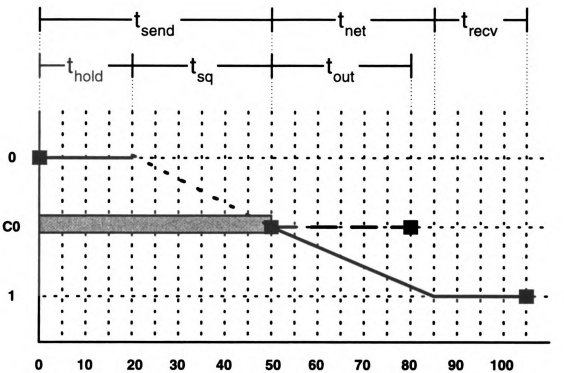


Figure 6.1: The timing diagram of sending a message from P_0 to P_1 ($t_{hold} = 20, t_{out} = 15, t_{net} = 20, t_{recv} = 20, t_{cont} = 15$, the out-going channel is available at $t = 50$)

6.1.3 Workload and Traffic Models

Throughout this study, we assume that the system implements only space-sharing policy which means there is only one application process per each node. To consider a more realistic workload, we further assume that each process generates messages to the network based on the SPMD programming model which is widely used in the development of parallel programs. Although many message-passing libraries, such as PVM and MPI, can support generic parallel programming model, most programs are still implemented based on the SPMD programming model.

Algorithm 6.1.1 SPMD MODEL

Loop forever
  Randomly choose t, d, and m;
  Compute for t time units;
  Send a m-byte messages to node d;
EndLoop

The SPMD-based workload composed of computation part and communication part which interleave through the lifetime of the program. This workload is more realistic because, in the actual parallel programs, nodes do not only send messages, but also perform some computation. The algorithm of our workload model is shown in Algorithm 6.1.1. In this figure, we assume that the computation time t is a Poisson process which is exponentially-distributed with mean t_{comp} time units. After computation, each node randomly selects a destination d and sends a message of which length m is also randomly chosen. In our study, the destination can be either uniformly-distributed or hot-spot where a special node in the group receives substantially more traffic than the others. We also assume that message size of background traffic is uniformly distributed. The following criteria will be used to determine the distribution of destination and message size:

- If the node is a designated root node, the node sends fixed-size multicast messages to other nodes in the multicast group. In most our experiments, the root node does not issue a new multicast until the previous one is completed.

- If the node is a member of the multicast group (except the root node), the distributions of destination and message size are uniform.

- If the node belongs to the background group, the distribution of the destination can

be either uniform or hot-spot and the distribution of message size is uniform.

We use *multicast latency* defined in Chapter 3 as a performance metric. To study the effect of background network traffic, we vary the applied load of the background traffic by changing the mean of computation t_{comp}. As we decrease t_{comp}, the applied load of the background traffic increases. When we further decrease t_{comp}, there are two cases:

- Destinations may not be fast enough. The incoming messages are placed in the receiving buffers.

- The network may not have enough bandwidth to handle the traffic. The contention rate in the network increases.

For both cases, the back-pressured mechanism in the network eventually causes new messages to be placed in the sending buffers. When the sending buffer is full, the sender will be blocked. In this case, the applied load decreases.

6.2 The Effect of the Network Contention

Network contention is generally considered to be a major factor in performance degradation of the network. As a collective communication service such as a multicast usually involves several point-to-point communication services, its performance can be greatly effected by the network contention. To have clear understanding of the impact of the network contention, we classify multicast-related contention into three groups, the *internal contention*, the *external contention*, and the *source contention*.

The internal contention is the contention between two multicast traffic from the same multicast operation. This kind of contention is quite simple and can be easily avoided by using algorithms proposed in Chapter 5. The contention between the multicast traffic and the local/background traffic is called the external contention. Without using the contention-free processor allocation scheme, this type of contention cannot be avoided. Note that, under the same network condition, different multicast trees can usually tolerate different levels of the external contention.

The source contention is more subtle than the other types since it is caused by the HOL blocking. Let consider the following example. Figure 6.2 demonstrates the relationship of three network parameters, holding latency, end-to-end latency, and normalized network load. When network load is heavy, the end-to-end latency increases drastically. This is because messages are delayed longer due to the contention in the network as the network load increases. When we consider the holding latency, the existence of the sending buffer keeps the holding latency constant as the sender can resume execution as soon as a message being sent is placed in the sending buffer. Figure 4.2 and Figure 6.2 lead to the conclusion that the sequential tree should perform very well when the network load is high. In contrast to our belief, Figure 6.3 shows that our conclusion is not quite correct. The sequential tree performs much worse than expected when the network load is high.

To explain this phenomenon, we have to observe the nature of software-based multicast. Consider an example of a 4-node sequential multicast tree presented in Figure 6.4. Let $t_{hold} = 20$, $t_{out} = 15$, $t_{net} = 20$, $t_{recv} = 20$ when there is no contention in the network. Let $C0$ present activities at the out-going channel of P_0. Based on these parameters, the multicast latency is 100 time units. Let assume that the out-going channel from P_0 is free

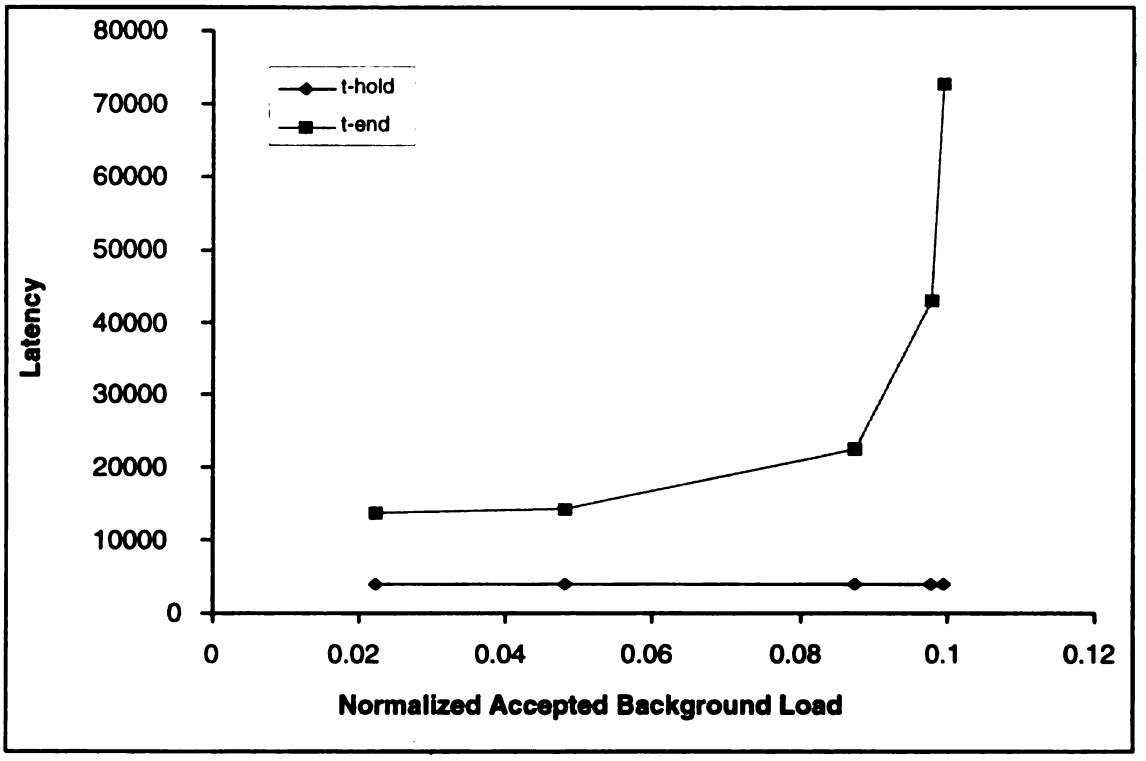


Figure 6.2: The relationship of t_{hold}, t_{end} and normalized network load

and there is no message in the sending queue. Suppose the contention in the network (t_{cont}) delays all messages for 15 time units. Thus, t_{net} becomes 35 time units. The latency of a message occupying the out-going channel (t_{out}) also increases to 30 time units. Thus, a message from P_0 to P_1 occupies the out-going channel of P_0 until $t = 50$. Although P_0 starts sending to P_2 and P_3 at $t = 20$ and $t = 40$ respectively, both messages encounter the HOL blocking and have to wait for the out-going channel. Therefore, the latency of this multicast is actually 135 time units.

As shown in the example, when the root node performs multicast, it sends duplicated unicast messages to its immediate children in the multicast tree. Thus, we can consider the root node injects a burst of unicast messages and the injection rate is limited by the holding latency at the root node. When the network load is light, the network has enough

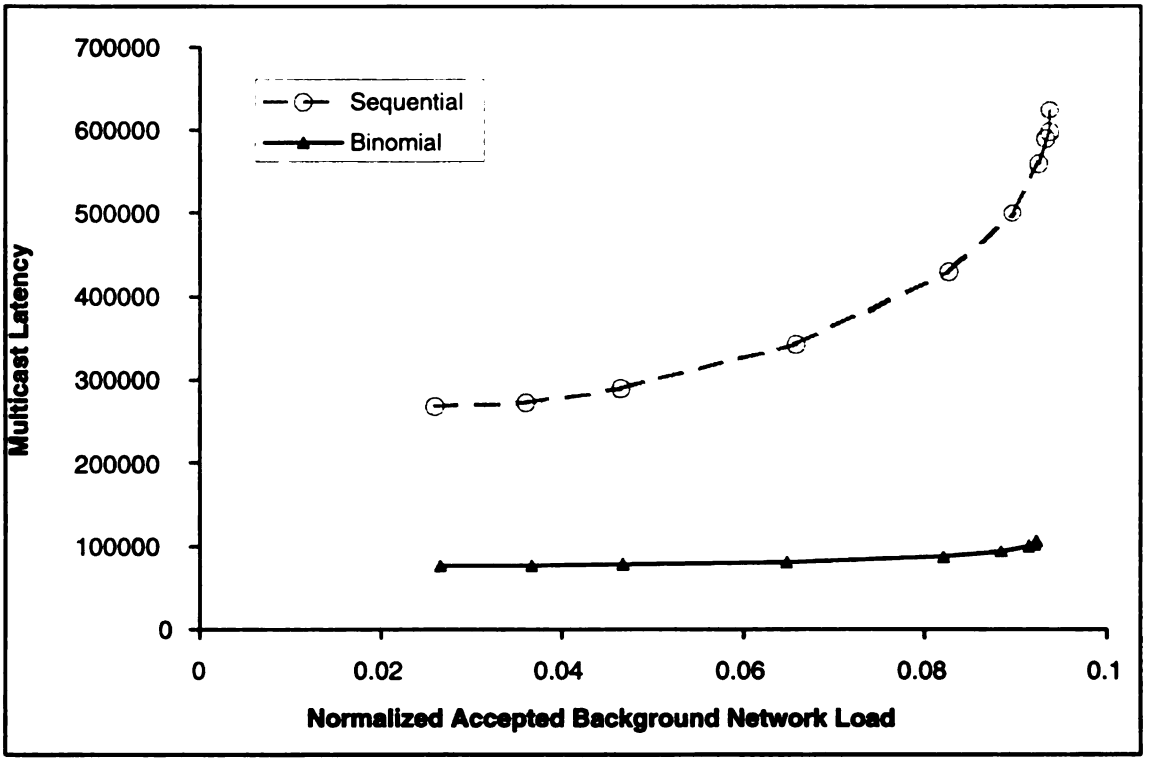


Figure 6.3: The comparison of 64-node binomial and sequential multicast trees under uniform traffic

bandwidth to consume all messages. When the network load is heavy, however, the network contention can delay messages being sent from the root node. This does not cause delay only to messages in the network, but also other messages in the sending buffer at the root node. Thus, the performance of the sequential tree which injects a large burst of unicast messages at the same time will suffer greatly due to the delay in the sending buffer. On the other hand, this delay is much smaller for a multicast tree which injects a smaller burst of unicast messages such as the binomial tree. Figure 6.5 confirms this conclusion.

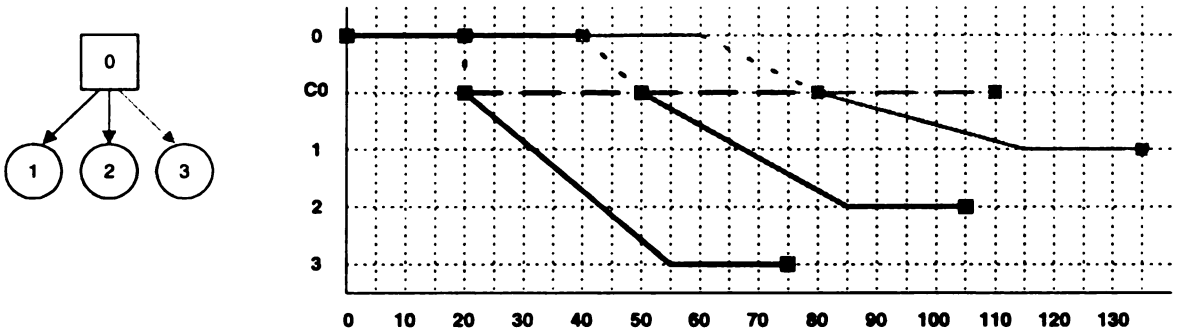


Figure 6.4: A timing diagram of a 4-node sequential multicast tree from P_0 ($t_{hold} = 20$, $t_{out} = 15$, $t_{net} = 20$, $t_{recv} = 20$, and $t_{cont} = 15$)

6.3 Dynamic Multicast

All software-based multicast algorithms discussed earlier are based on either fixed multicast tree structures such as the binomial and sequential trees or predetermined network parameters. Although the parameterized multicast algorithms discussed in Chapter 4 and 5 can adjust their trees to match the underlying network, these trees are still fixed with respect to the network condition. We consider multicast trees generated by these algorithms *static multicast trees*. As the nature of network load is very dynamic, these network parameters are constantly changed. Static multicast trees based on predetermined network parameters may not perform well as expected. This leads to an interesting question: can we develop an algorithm to generate a multicast tree based on both predetermined network parameters and current network condition? This section discusses a new class of multicast tree called the *dynamic multicast tree* which adapts its structure to fit the current network condition.

The basic idea of constructing a dynamic multicast tree is based on two observations. First, the effect of the external contention is quite great for a deep multicast tree such as the binomial multicast tree since more depth means more direct contention and hence more delay. Second, when contention occurs, it can also cause the source contention which means

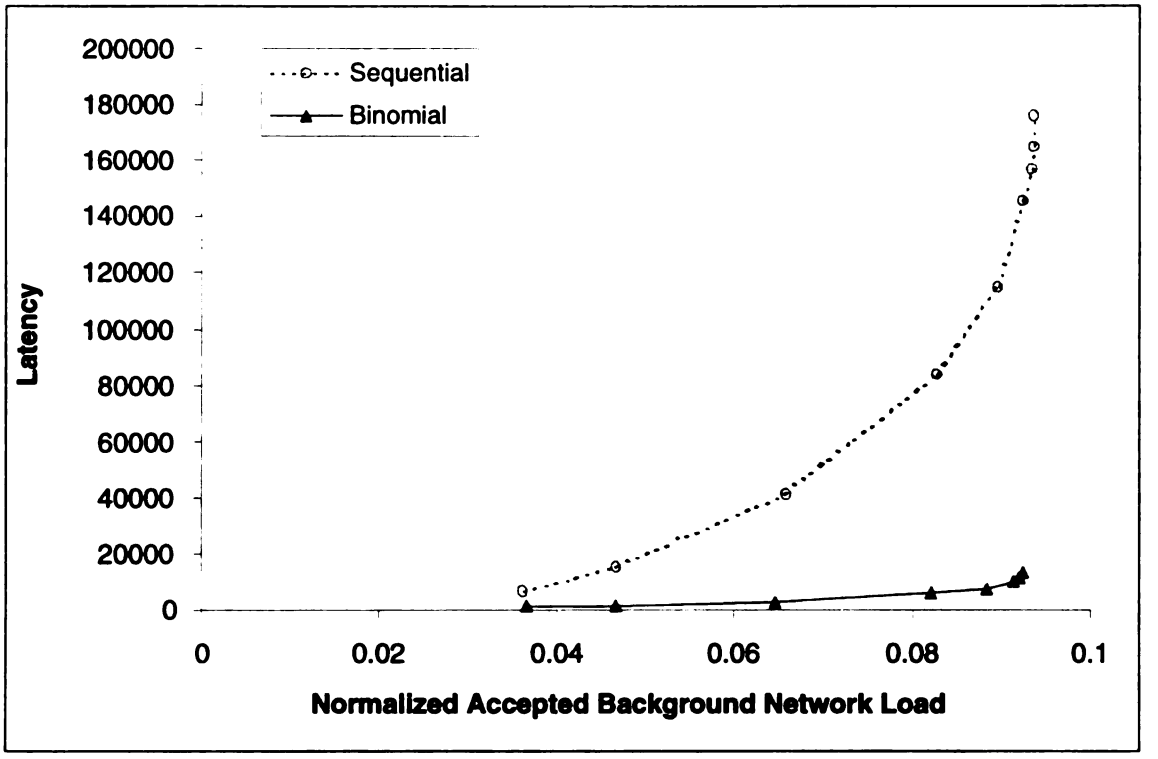


Figure 6.5: The sending queue delay at the root node of 64-node binomial and sequential multicast trees under uniform traffic

messages in the sending buffer have to wait due to the HOL blocking. As shown in the example in the previous section, this type of contention has a great impact on a multicast tree with high out-degree at the root node and the internal nodes. Based on these observations, we can construct a dynamic multicast tree using the DYN-Tree algorithm. The DYN-Tree is derived from the OPT-Tree algorithm with an assumption that communication library provides the estimation of the time-at-sender latency (t_{as}) which is an interval between the sender invokes the send operation until the whole message leaves the sender. Thus, if we measure t_{as} after sending a message size n:

$$\begin{aligned} t_{as} &= t_{hold} + t_{hol} \\ &= (t_{hs} + n.t_{hd}) + (t_{sq} + t_{ns} + n.t_{nd} + t_{cont}) \end{aligned}$$

The basic idea of DYN-Tree is to adjust the burst size of unicast messages injected by the root and all internal nodes with respect to the current network load. When the network load is low, the root node can inject many unicast messages without suffering large delay at the sending buffer. In this case, t_{hold} and t_{end} are used in the OPT-Tree algorithm to generate a multicast tree. When the network load is high, the root and internal nodes should not inject too many unicast messages. We use the estimated HOL blocking latency (t^e_{hol}) and the estimated t^e_{end} as parameters for the OPT-Tree algorithm to generate a multicast tree.

The assumption that t_{as} can be estimated allows the sender to derive both t^e_{hol} and t^e_{end}. Suppose a message size n has been sent and its time-at-sender latency is t_{as}. We have to consider the fact that a node may send messages with different sizes. Thus, after the message is sent, two parameters, t'_{hol} and t'_{end}, are calculated.

$$\begin{aligned} t'_{hol} &= t_{as} - (t_{hs} + n.t_{hd}) - n.t_{nd} \\ &= t_{ns} + t_{sq} + t_{cont} \\ t'_{end} &= t_{as} - n.t_{hd} - n.t_{nd} + t_{rs} \\ &= t_{hs} + t_{ns} + t_{rs} + t_{sq} + t_{cont} \end{aligned}$$

Suppose a multicast size m is sent next. DYN-Tree algorithm then computes t^e_{hol} and t^e_{end}:

$$\begin{aligned} t^e_{hol} &= t'_{hol} + m.t_{nd} \\ &= t_{sq} + t_{ns} + m.t_{nd} + t_{cont} \end{aligned}$$

$$
\begin{aligned}
t^e_{end} &= t'_{end} + m.(t_{hd} + t_{nd} + t_{rd}) \\
&= t_{hs} + m.t_{hd} + t_{sq} + t_{ns} + m.t_{nd} + t_{cont} + t_{rs} + m.t_{rd}
\end{aligned}
$$

To construct a multicast tree, we develop the DYN-Tree algorithm which is very similar to the OPT-Tree algorithm. The difference is that the DYN-Tree uses $\max(t_{hs} + m.t_{hd}, t^e_{hol})$ and t^e_{end} as parameters while the OPT-Tree uses t_{hold} and t_{end}. The combination of the DYN-Tree and the contention-avoidance techniques presented in Chapter 5 results to the DYN-Min algorithm (for BMIN) and the DYN-Mesh algorithm (for Mesh).

6.4 Experimental Results

We evaluate four types of multicast algorithms: the binomial-based multicast algorithms (Binomial, U-Mesh, and U-Min), the parameterized multicast algorithms (OPT-Tree, OPT-Mesh, and OPT-Min), the sequential multicast algorithm (Sequential), and the dynamic multicast algorithm (DYN-Mesh and DYN-Min). The underlying architectures of our flit-level simulators are a 16×16 node wormhole-switched mesh with X-first-Y-next routing and a 256-node wormhole-switched BMIN with turn-around routing. Each node has 64-KByte sending and receiving buffers. A mesh router does not support multi-port capability nor virtual channel. The BMIN network is based on 2×2 bidirectional switches. Both networks have the following parameters: $t_{hs} = 1915$, $t_{hd} = 2$, $t_{ss} = 1915$, $t_{sd} = 2$, $t_{ns} = 1390$, $t_{nd} = 2$, $t_{rs} = 2025$, and $t_{rd} = 3$. These values are based on the IBM/SP machine at Argonne National Lab.

The simulator will be run until 95% confidence interval on the average multicast latency

has been reached and the standard error is less than 2.5% of average multicast latency. For each run, we perform 8 independent experiments in which the locations of the multicast group are randomly chosen. This is because the locations of members of the multicast group can affect the probability of contention. In each experiment, we vary t_{comp} to vary the applied load. For each t_{comp}, the *accepted background traffic* and the multicast latency are measured. We define the accepted background traffic as the total number of bytes received at the nodes in the background group per time unit. The accepted background traffic is normalized by the maximum network throughput which is $\frac{N}{t_{nd}}$ for a network with N processors. Each data point presented in our results is the average of the multicast latency from all 8 experiments. To eliminate the start-up transient effect, all data are measured after the first 50 multicasts are completed. Unless explicitly stated otherwise, all multicast latencies are measured from 64-node multicast trees with no local traffic in the multicast communication group. Each multicast message is a fixed size of 1 KByte. The destination and message size of background traffic are uniformly distributed. The average of the message size is 1 KByte. The root node issues a new multicast when the previous one is done.

6.4.1 Performance under Uniform Background Traffic

The performances of multicast algorithms for 64-node multicast trees on the mesh network are shown in Figure 6.6. Obviously, the sequential algorithm performs very bad at any background load traffic as expected. For other algorithms, when the background traffic is low, the multicast algorithms with contention (Binomial and OPT-Tree) perform quite close to the contention-free multicast algorithms (U-Mesh and OPT-Mesh) However, as

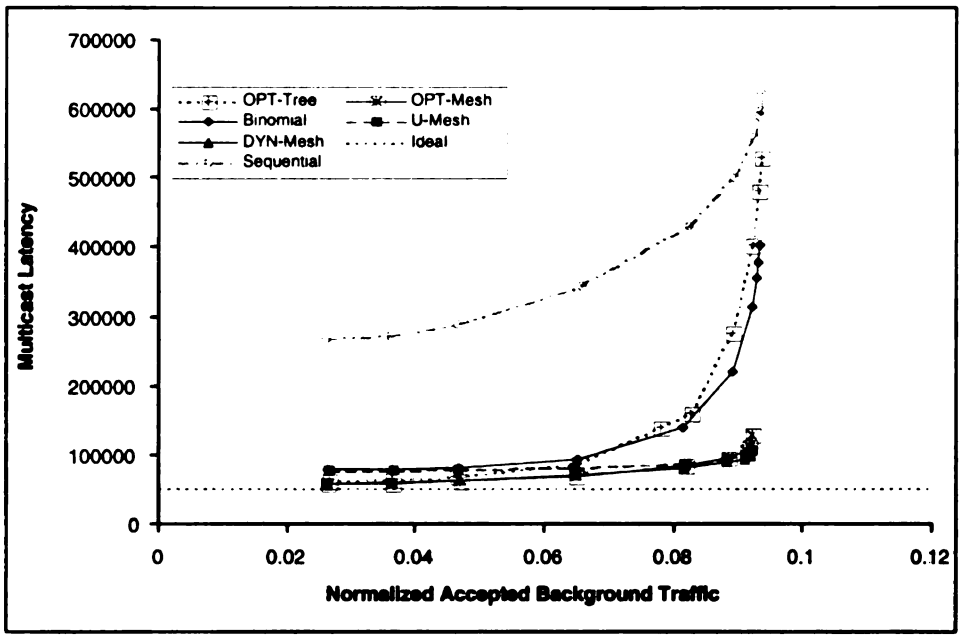


Figure 6.6: Comparison of 64-node 1-KByte multicast trees under uniform background traffic on the mesh network.

the background traffic increases, their performance degrade rapidly. This is because the external contention with the background traffic further increases the effect of the internal contention of the multicast algorithms with contention.

Figure 6.7 shows the details of Figure 6.6. Clearly, the OPT-Mesh perform much better than the U-Mesh algorithm when the network load is low. This is because the OPT-Mesh algorithm is based on an optimal multicast tree generated specifically for the underlying network given that there is no network contention. However, as the network load increases, the performance of OPT-Mesh degrades quickly. Similar to the sequential algorithm, the OPT-Mesh algorithm generates a multicast tree which have more out-degree at the root and internal nodes than the trees generated by the U-Mesh algorithm. Thus, the root node and the internal nodes inject more unicast messages each time a multicast message is sent. The increasing network contention causes longer HOL blocking latency (source contention) at

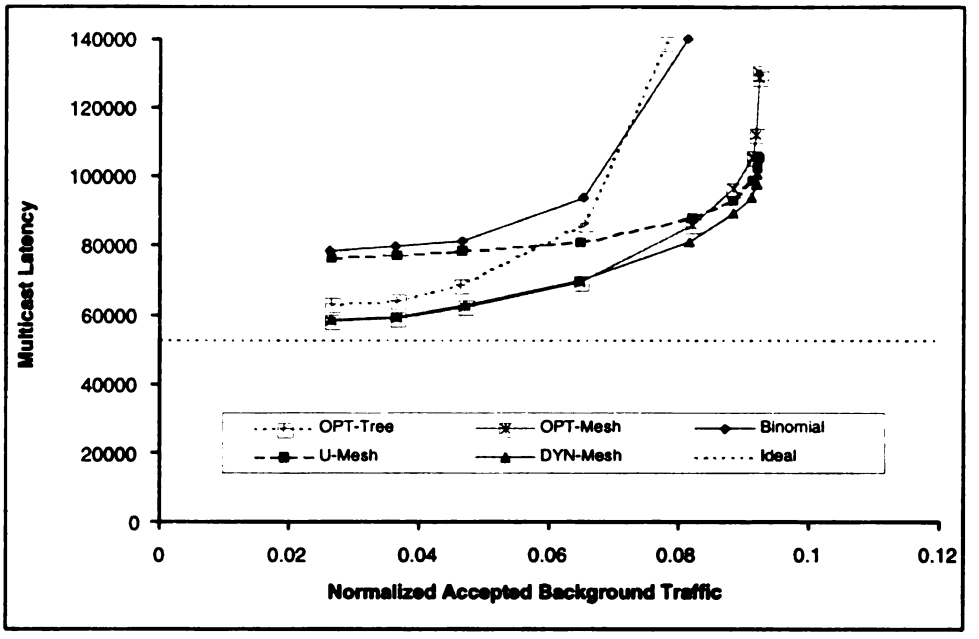


Figure 6.7: The detail comparison of 64-node 1-KByte multicast trees under uniform background traffic on the mesh network.

the root node and the internal nodes. As the results, the multicast latency of the OPT-Mesh algorithm dramatically increase and eventually it performs worse than the U-Mesh algorithm. The DYN-Mesh algorithm generates a multicast tree with respect to the current network load. When the network load is low, its tree structure is similar to the tree structure generated by the OPT-Mesh algorithm. When the network load increases, both blocking latency and end-to-end latency increase. The DYN-Mesh algorithm adjusts to the changes by decreasing out-degree at the root and internal nodes and increasing the depth of its multicast tree. Eventually, its tree will be similar to the binomial tree.

In Figure 6.7, we also notice that the performance of OPT-Mesh, U-Mesh, and DYN-Mesh remain unchanged when the normalized accepted background traffic is approximately 0.095. At that point, the senders in the background group become saturated and, hence, cannot generate more traffic. As the members of the background group and the

multicast groups reside in different nodes, the effects of the network contention from the uniform background traffic are limited, even though the allocation policy is not contention-free.

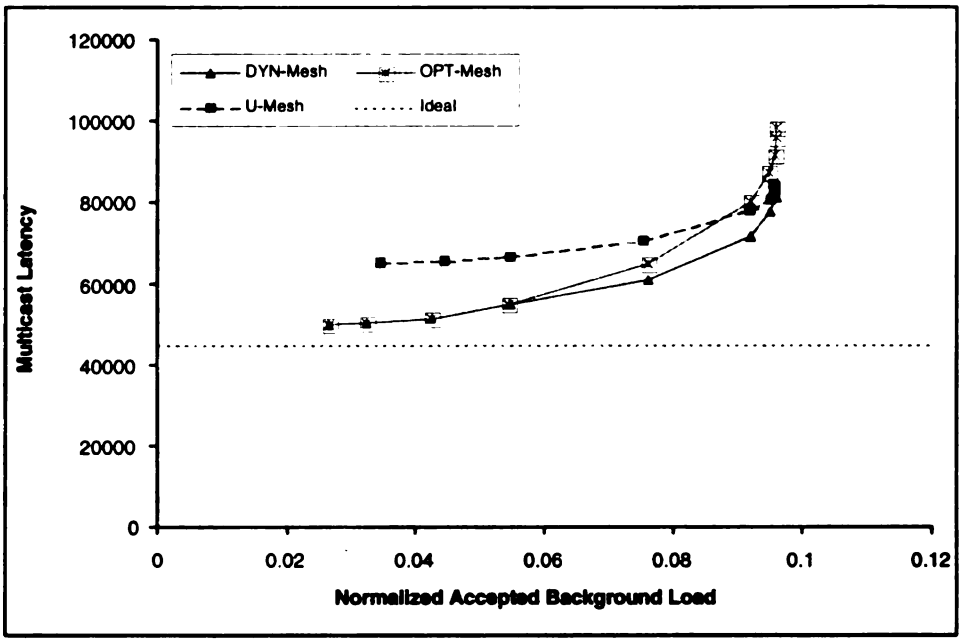


Figure 6.8: Comparison of 32-node 1-KByte multicast trees under uniform background traffic on the mesh network.

The results of multicast trees with 32 nodes on the mesh network are shown in Figure 6.8. Note that we present only the results of the OPT-Mesh algorithm, the U-Mesh algorithm, and the DYN-Mesh algorithm since the results of other algorithms are quite similar to the results in Figure 6.6. In Figure 6.8, the performance of all three algorithms are quite similar to the results in Figure 6.7. However, the DYN-Mesh algorithm perform better than the OPT-Mesh even when the traffic is less than 0.06. When we consider multicast trees with 128 nodes as shown in Figure 6.9, the improvement of the DYN-Mesh algorithm over the OPT-Mesh is insignificant even at the high background traffic load. When the size of the multicast tree is large, the binomial tree is quite deep. In this case, the effects of

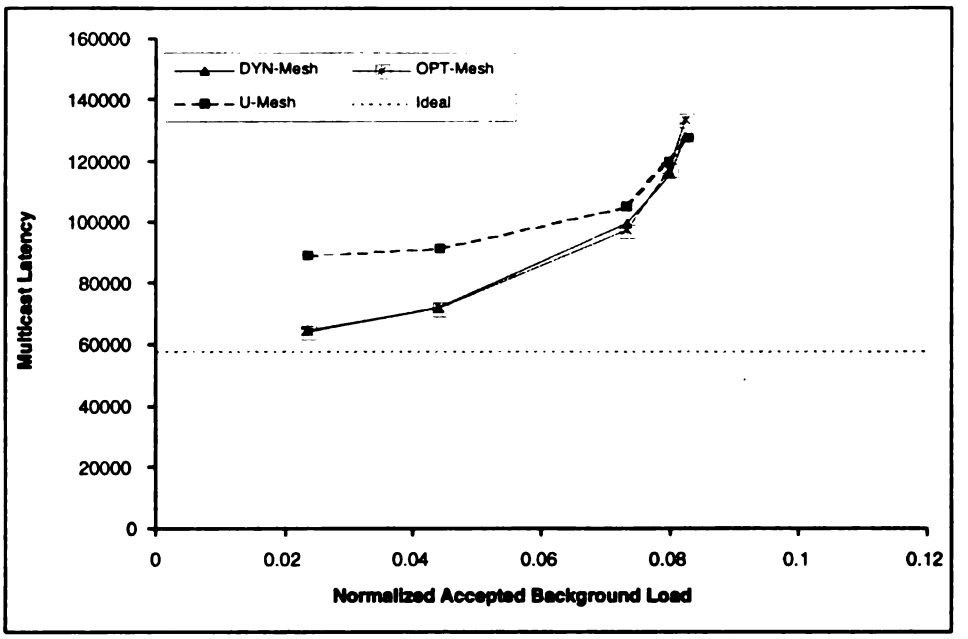


Figure 6.9: Comparison of 128-node 1-KByte multicast trees under uniform background traffic on the mesh network.

the external contention become more significant than the effects of the source contention. Thus, the U-Mesh algorithm does not perform well when the size of the multicast tree is large.

The performance of multicast algorithms for 64-node multicast trees on the BMIN network are shown in Figure 6.10. The results look very similar to Figure 6.6. However, as we study in more details, as shown in Figure 6.11, the parameterized multicast algorithms perform better than the binomial multicast tree even when the background traffic load is high. This is because our BMIN simulator uses the turn-around routing scheme, which is an adaptive-routing scheme. Thus, the impact of the network contention is not significant. Based on the same reason, the multicast algorithms with contention (OPT-Tree and binomial) also perform reasonably well in this network. We present the performance of 128-node multicast trees in Figure 6.12. The results from 64-node and 128-node multi-

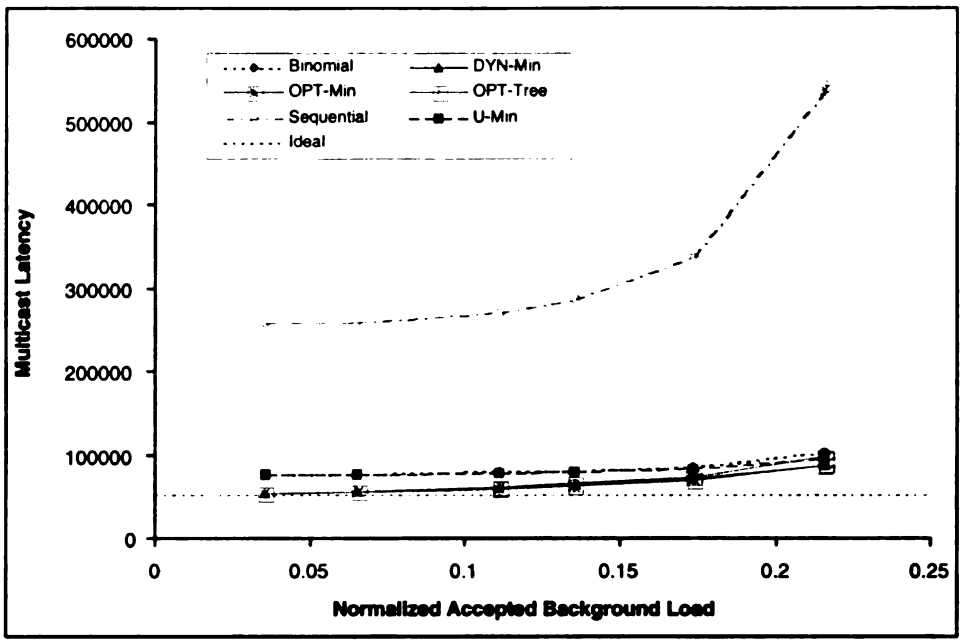


Figure 6.10: Comparison of 64-node 1-KByte multicast trees under uniform background traffic on the BMIN network.

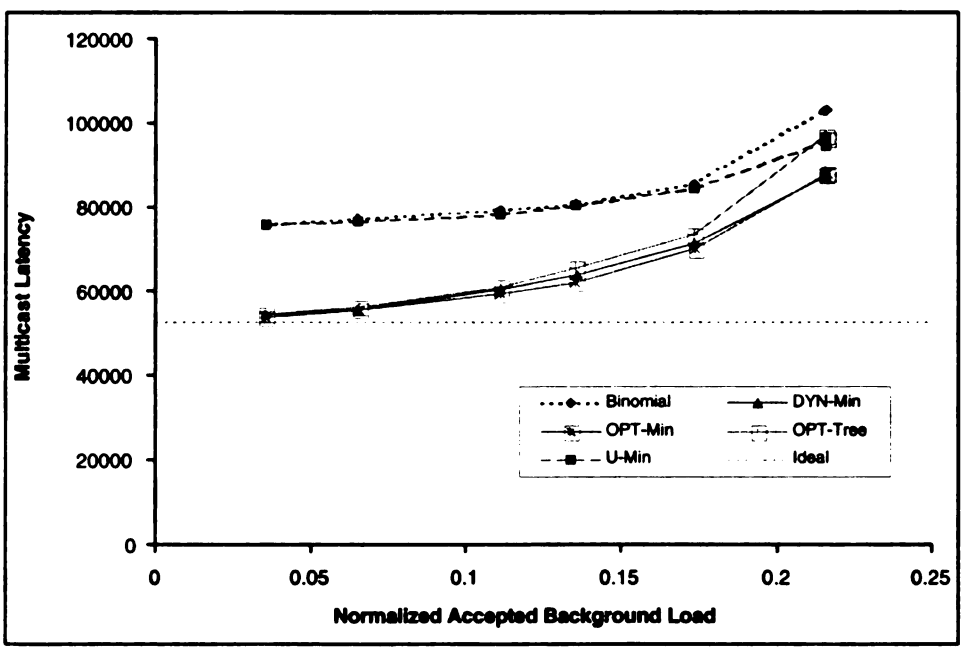


Figure 6.11: The detail comparison of 64-node 1-KByte multicast trees under uniform background traffic on the BMIN network.

cast trees clearly show that the dynamic multicast algorithm does not provide significant improvement on the BMIN network.

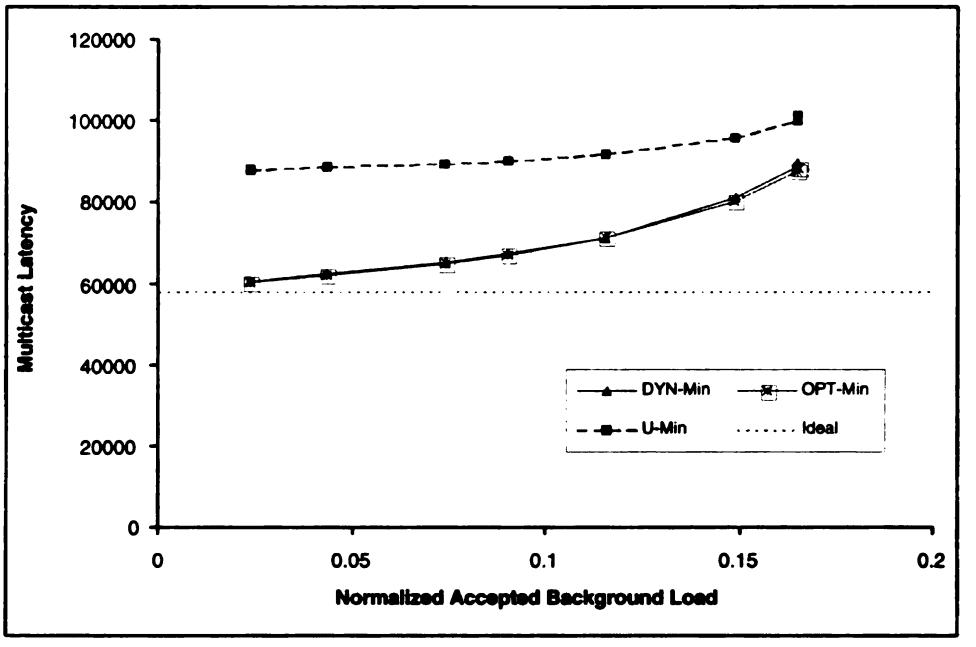


Figure 6.12: Comparison of 128-node 1-KByte multicast trees under uniform background traffic on the BMIN network.

6.4.2 The Effect of Hot-Spot Traffic

In general, the destination of the background traffic is rarely uniform. Some communication patterns such as bit-reversal and perfect-shuffle and most collective communication patterns create a problem called the *hot-spot* traffic. This problem, similar to the hot-spot memory problem, occurs when one or more nodes receive more traffic than the others. In our study, we assume there is only one hot-spot node in the background group.

Figure 6.13 shows the performance of multicast algorithms under hot-spot traffic on the mesh network. In this figure, 1% of the background traffic are sent to a hot-spot node in the background group. Comparing to the uniform traffic, the 1% hot-spot background

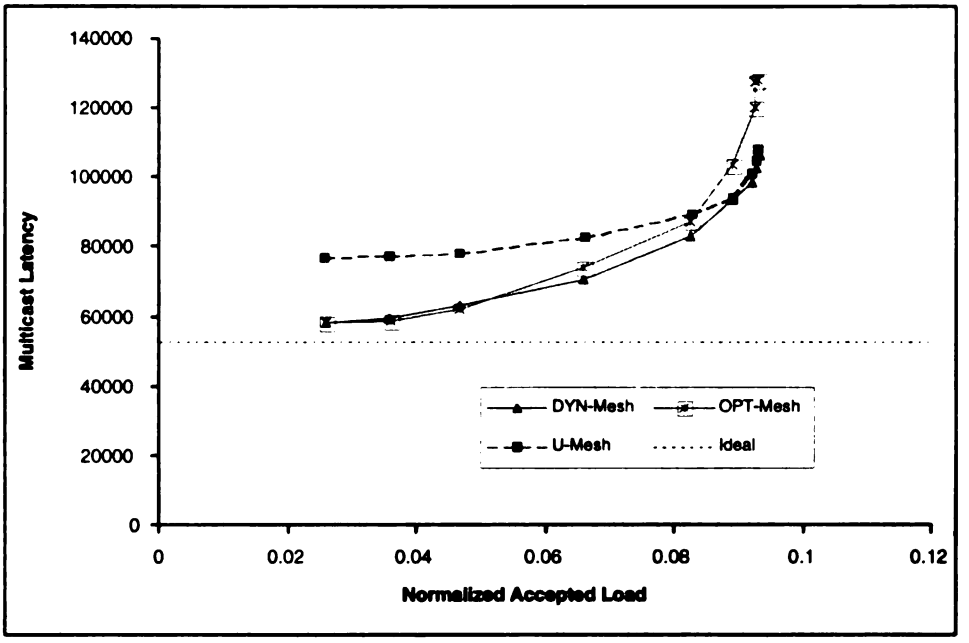


Figure 6.13: Comparison of 64-node 1-KByte multicast trees under 1% hot-spot background traffic on the mesh network.

traffic have very little impact to all algorithms. However, the DYN-Mesh algorithm clearly performs better than the OPT-Mesh when the traffic load is high. We further increase the hot-spot traffic to 2% as presented in Figure 6.14. The network congestion causes all multicast algorithms to perform very bad. In Figure 6.15, we study the effect of 1% hot-spot background traffic on the BMIN network. Comparing to the uniform traffic, the 1% hot-spot background traffic have quite serious impacts, especially when the traffic load is slightly more than 0.15.

When the hot-spot problem occurs, the receiver cannot consume fast enough and all traffic to this node is blocked. In the case of the BMIN network, due to the turn-around routing scheme, messages, whose destination is the hot-spot node, will use alternative routes. This can increase the external contention rate to the multicast traffic. Thus, the network bandwidth available for the multicast traffic gradually decreases. When there is not enough

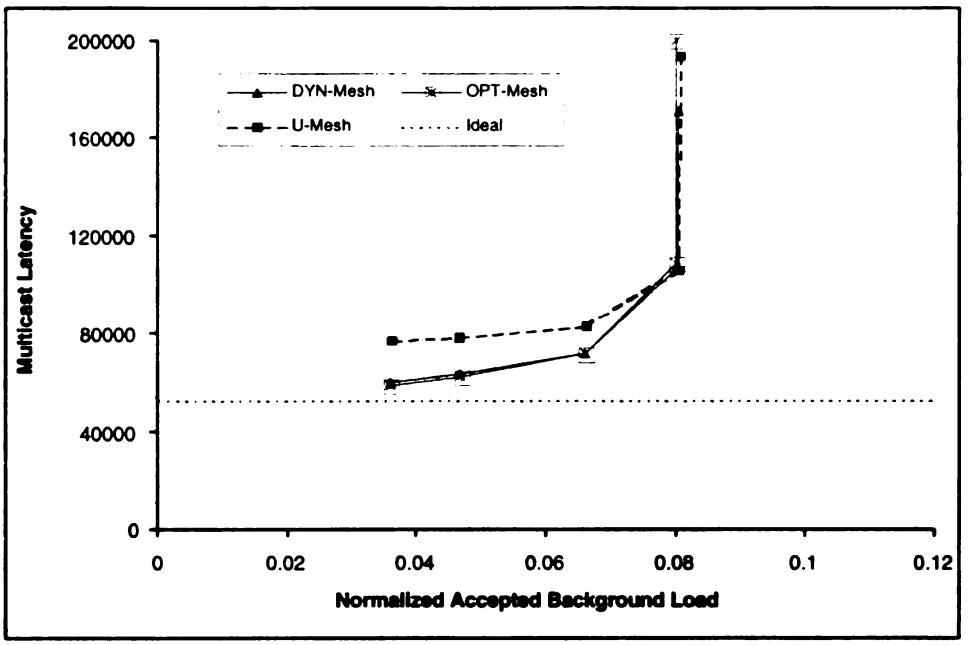


Figure 6.14: Comparison of 64-node 1-KByte multicast trees under 2% hot-spot background traffic on the mesh network.

bandwidth available, the multicast latency increases rapidly. In the case of the mesh network, messages cannot use alternative routes when the congestion occurs. Although the bandwidth available for the multicast traffic decreases, the background traffic do not use up all the bandwidth. Thus, for the 1% hot-spot traffic, all algorithms still perform reasonably well. However, for the 2% hot-spot traffic, much more traffic are sent to the same destination. The congestion is much worse. Thus, the performance of the multicast algorithms degrade severely.

6.4.3 The Effect of Inter-Injection Time

In some applications, the root node may issue a new multicast before the previous one is actually completed. If multicasts are injected too fast, the contention among these multicasts, even originated from the same root node, is highly possible. Figure 6.16 shows the

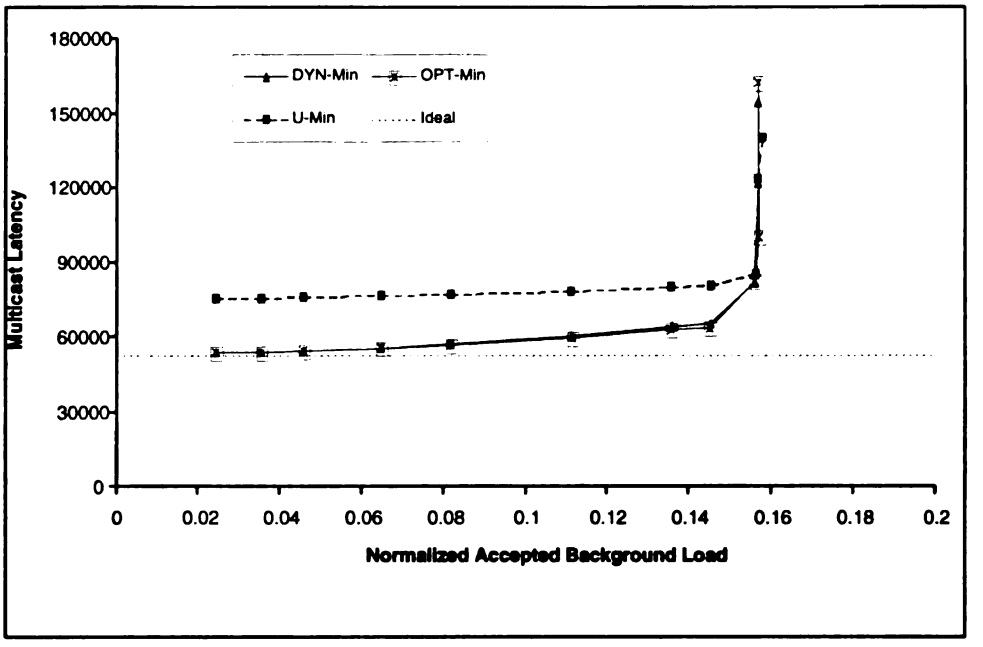


Figure 6.15: Comparison of 64-node 1-KByte multicast trees under 1% hot-spot background traffic on the BMIN network.

performance of multicast algorithms on the mesh network when the inter-injection time of multicast messages at the root node is 100,000 time units. This is about twice the multicast latency when there is no contention. Even at this injection rate, the OPT-Mesh algorithm does not perform well. In this case, the source contention degrades the performance of the OPT-Mesh algorithm. As the DYN-Mesh algorithm considers the effect of this contention, it adjusts its the tree structure. Thus, the DYN-Mesh algorithm performs very well.

In Figure 6.17, we study the performance of all algorithms on the BMIN network when the inter-injection time is 50,000 time units. At even the higher injection rate, all algorithms still perform very well. The turn-around routing, which is adaptive, reduces the contention rate between two consecutive multicasts. As the results, the source contention is low. Thus, the performance of the OPT-Min algorithm is better than the previous experiment.

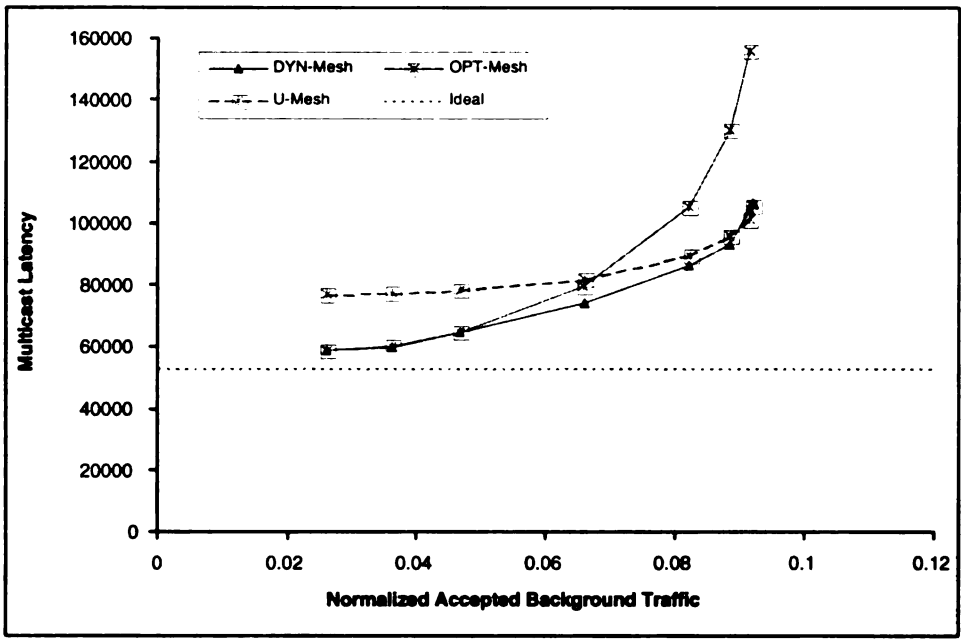


Figure 6.16: Comparison of 64-node 1-KByte multicast trees on the mesh network when the inter-injection time of multicast messages at the root node is 100,000 time units.

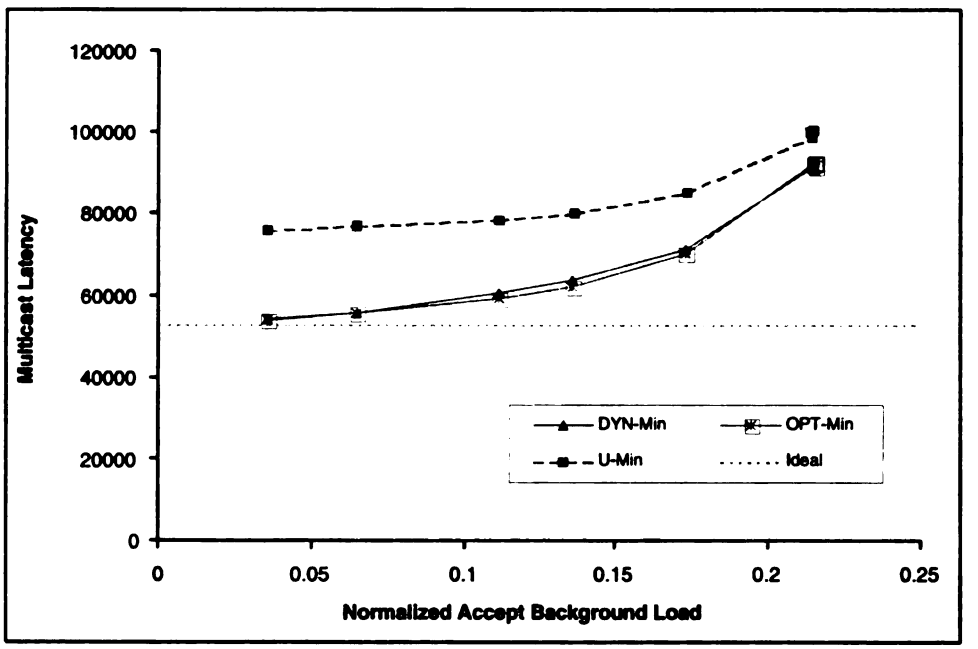


Figure 6.17: Comparison of 64-node 1-KByte multicast trees on the BMIN network when the inter-injection time of multicast messages at the root node is 50,000 time units.

6.4.4 The Effect of Local Traffic

In SPMD model, each node executes at its own pace. Some nodes in the communication group may perform computation while others perform communication. Thus, when the root node starts sending a multicast message, some nodes in the multicast group may still communicate with others. Therefore, multicast traffic may content with other traffic generated by nodes in the same communication group. We call traffic which are not multicast traffic and generated by nodes in the multicast group the local traffic.

To study the effect of the local traffic, all nodes follow the SPMD workload model presented in Algorithm 6.1.1. Besides participating in multicasting by forwarding multicast messages, all nodes in the multicast group, except the root node, are allowed to send point-to-point messages to other nodes in the multicast group. Similar to nodes in the background group, both message size m and destination d are uniformly distributed and the computational time t is exponentially distributed.

Figure 6.18 presents the performance of multicast algorithms on the mesh network when the average inter-injection time of the local traffic is 50,000 time units. Obviously, all algorithms perform worse than the experiment without local traffic. The presence of the local traffic creates contention in the network and at the receiving nodes in the multicast group. In contrast to the results from the mesh network, as shown in Figure 6.19, the results of the same experiment on the BMIN network are entirely different. Again, the turn-around routing allows all algorithms to perform well even with the presence of the local traffic.

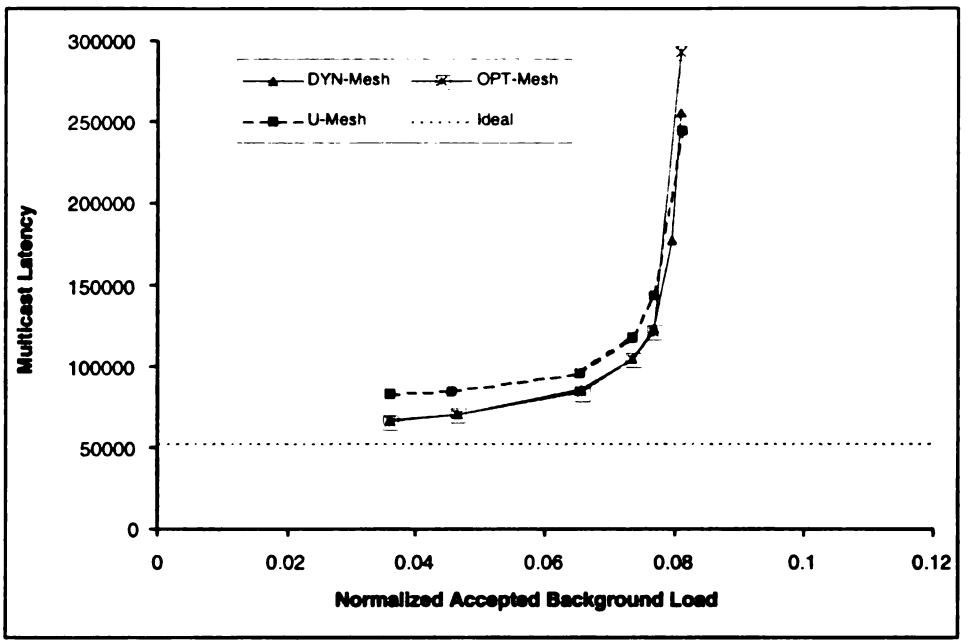


Figure 6.18: Comparison of 64-node 1-KByte multicast trees on the mesh network with local traffic. The average of inter-injection time of the local traffic is 50,000 time units.

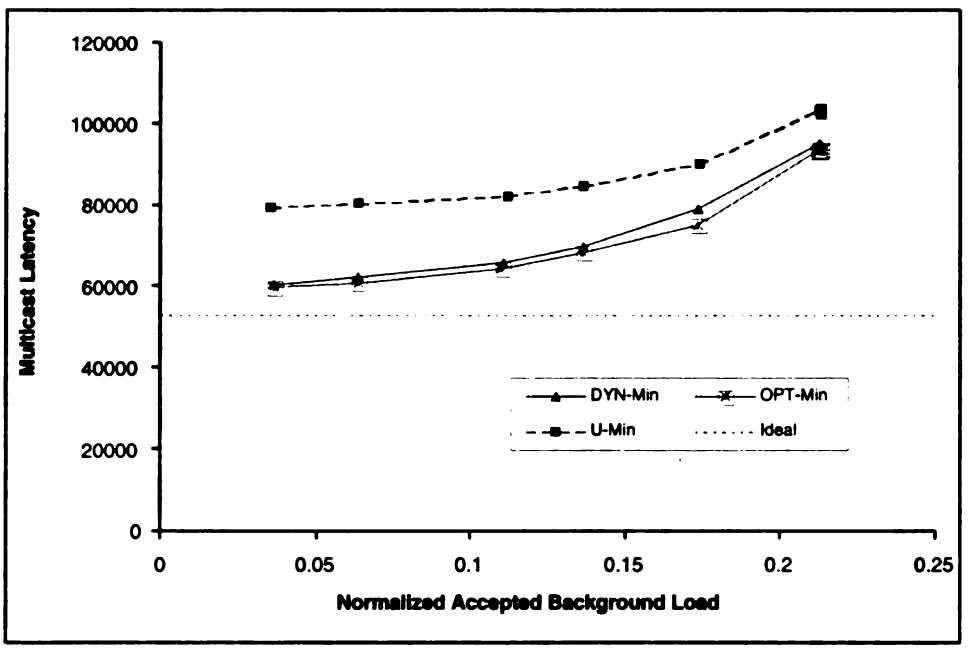


Figure 6.19: Comparison of 64-node 1-KByte multicast trees on the BMIN network with local traffic. The average of inter-injection time of the local traffic is 50,000 time units.

6.4.5 Multiple Multicasts

The traffic outside the multicast group may not always be point-to-point messages. It is possible that other communication groups may also perform multicasting as well. In Figure 6.20, we study the behavior of multicast on the mesh network when there are two groups performing multicast at the same time. We divide nodes into three groups: two 64-node groups are the multicast groups and one 128-node group is the background group. The root nodes of both multicast groups inject 1-KByte multicast messages and wait until the multicast messages are completely delivered before they inject new multicast messages. The background traffic is uniform. Note that the multicast latencies of both multicast groups are about the same.

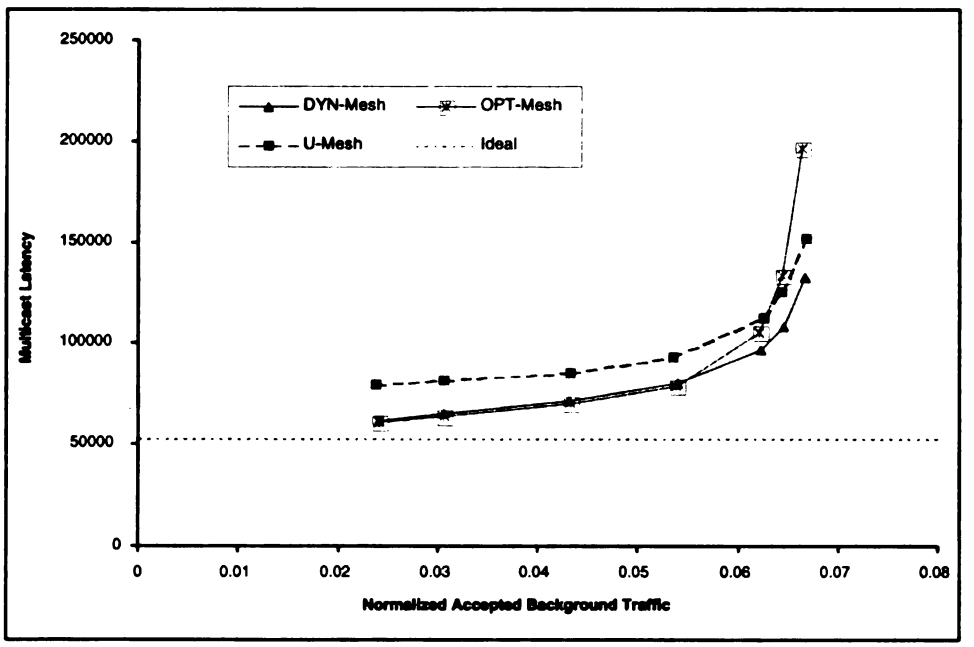


Figure 6.20: The performance of two 64-node 1-KByte multicast trees running at the same time on the mesh network.

The results in Figure 6.20 show that when there are more than one group performing multicast at the same time, the OPT-Mesh algorithm does not perform well. Each time a

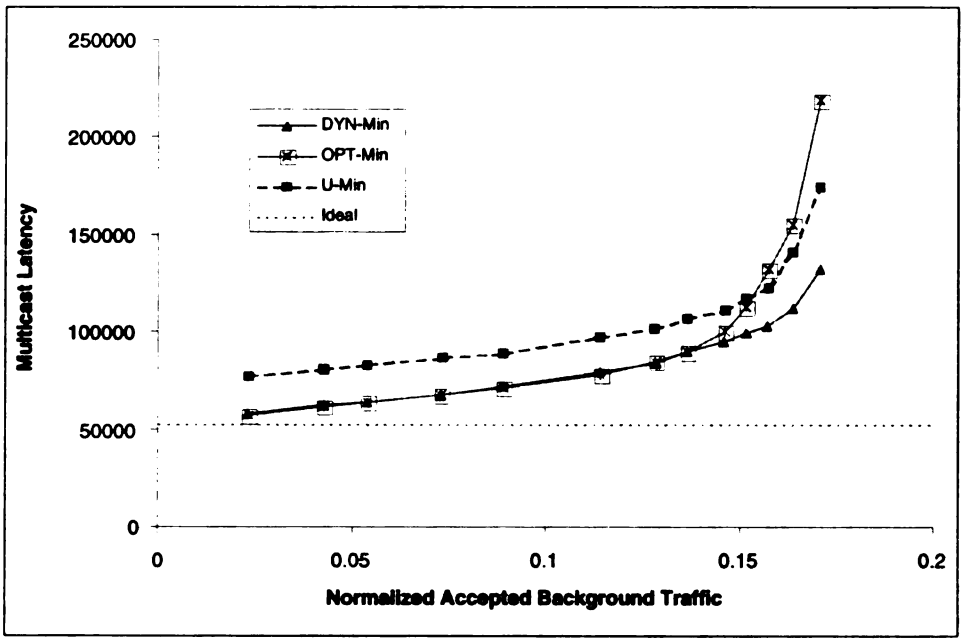


Figure 6.21: The performance of two 64-node 1-KByte multicast trees running at the same time on the BMIN network.

multicast message is sent, the root node and the internal nodes send multiple duplicated unicast messages. Thus, the injection rate of these messages at the root node and the internal nodes is quite high. When compared to background nodes, the injection rate at the node in the multicast group is much higher. This can cause much more contention in the network. Hence, the OPT-Mesh algorithm performs poorly. Although the tree structure generated by the DYN-Mesh algorithm is quite similar to the binomial tree when the traffic load is high, this structure is adjusted at every internal nodes. Thus, it can perform better than the U-Mesh. Similar results can be obtained when we perform the experiment on the BMIN network, as shown in Figure 6.21.

6.4.6 The Performance of Multicast on Different Architecture

The network parameters of the underlying architecture of all experiments we have discussed so far are based on the actual measurement from the IBM/SP at Argonne National Laboratory. Let call this architecture the *baseline architecture*. In reality, other architectures have different network parameters.

Suppose that we enhance the baseline architecture by doubling the bandwidth of all physical channels in the network and the network start-up cost is reduced by half. Thus, the network parameters become: $t_{hs} = 1915$, $t_{hd} = 2$, $t_{ns} = 695$, $t_{nd} = 1$, $t_{rs} = 2025$, $t_{rd} = 3$. Figure 6.22 presents the performance of all multicast algorithms on the mesh network using the new network parameters. The results indicate that the effect of the background traffic is not significant. In this case, the software overhead dominates. Thus, the external contention due to the background traffic causes lesser delay to the multicast latency comparing to its

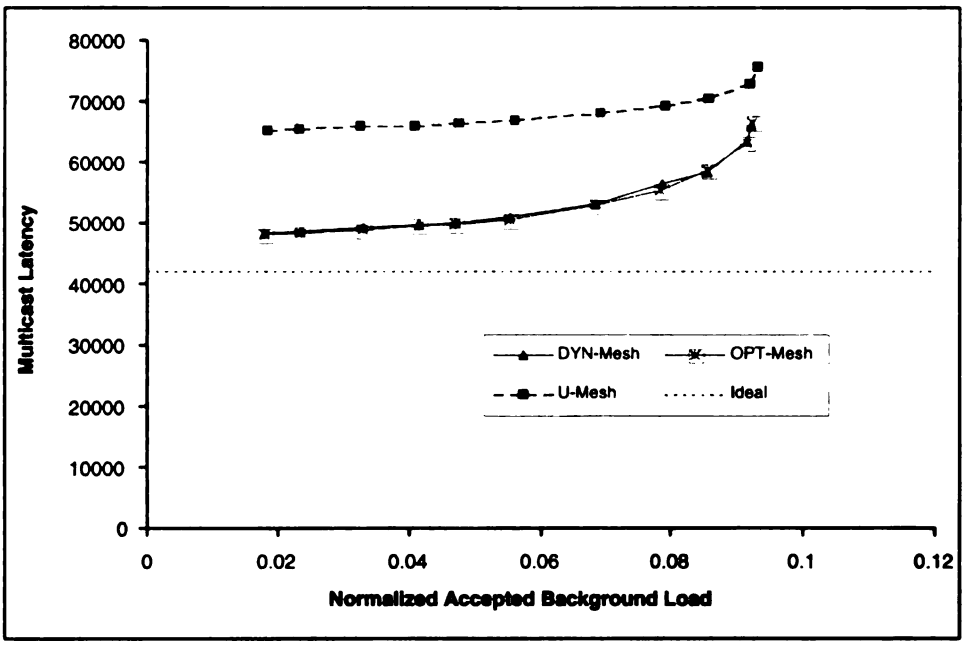


Figure 6.22: Comparison of 64-node 1-KByte multicast trees on the mesh network with: $t_{hs} = 1915$, $t_{hd} = 2$, $t_{ns} = 695$, $t_{nd} = 1$, $t_{rs} = 2025$, and $t_{rd} = 3$.

large software overhead.

Suppose we improve the baseline architecture by reducing the software overhead such that the network parameters become: $t_{hs} = 957$, $t_{hd} = 2$, $t_{ns} = 1390$, $t_{nd} = 2$, $t_{rs} = 1012$, $t_{rd} = 3$. Figure 6.23 presents the performance of all multicast algorithms on this architecture. Although the network overhead becomes more significant, the results from this architecture are quite comparable to the results from the baseline architecture (Figure 6.6). However, the DYN-Mesh algorithm performs slightly better in this experiment than in the baseline experiment. This is because its ability to adjust its own structure allows the DYN-Mesh algorithm to adjust to the changes.

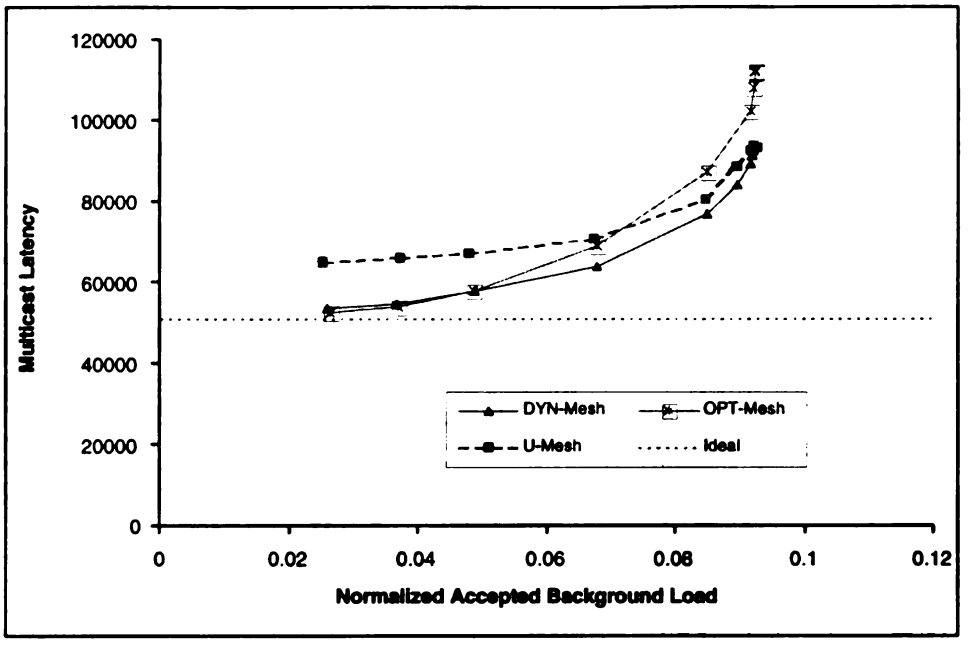


Figure 6.23: Comparison of 64-node 1-KByte multicast trees on the mesh network with: $t_{hs} = 957$, $t_{hd} = 2$, $t_{ns} = 1390$, $t_{nd} = 2$, $t_{rs} = 1012$, and $t_{rd} = 3$.

6.5 Conclusion

Software-based multicast has been studied extensively as it is heavily used in many parallel applications. Most studies have been done in the ideal condition where the effects of network traffic from other node are not included. In this chapter, we study the effect of network traffic on multicast behavior. Several parameters such as the type of background traffic, the size of the multicast group, and the network parameters are considered. We focus our study on four multicast groups including the binomial-based multicast algorithms (Binomial, U-Mesh, and U-Min), the parameterized multicast algorithms (OPT-Tree, OPT-Mesh, and OPT-Min), the sequential multicast algorithm, and the dynamic multicast algorithm (DYN-Mesh and DYN-Min).

Based on the results, the parameterized multicast algorithms perform very well when the background traffic load is not too high. They also perform well when the underlying

network is based on the adaptive routing scheme, such as the turn-around routing. In addition, when the software overhead dominates, the effect from the background traffic is not significant.

Under heavy load, the binomial-based multicast algorithm performs better than the parameterized multicast algorithm. However, this can happen if the load is very high which is usually unacceptable in the real working system. Moreover, the difference of the performance of the parameterized multicast algorithm and the binomial-based multicast algorithm is noticeable when the load is light. Thus, the parameterized multicast algorithm is very useful in most working conditions.

The understanding of the effects of external and source contention allows us to introduced the dynamic multicast algorithm. Although we cannot predict the future network condition, we can still adjust the multicast tree structure based on the current network condition. To avoid high overhead, the dynamic multicast algorithms use only the local network information which are assumed to be provided by the system. Based on these local network information, the dynamic multicast algorithms adjust the network parameters, t_{hold} and t_{end}, with some simple calculation. Then they use the parameterized multicast algorithm, such as OPT-Mesh and OPT-Min, to construct a multicast tree based on the new parameters. The results from the experiments indicate that, in spite of the unavailability of global network information, the dynamic multicast algorithm still performs reasonably well, especially when the network load is high and when there is more than one multicast group performing multicast simultaneously.

Chapter 7

Related Works

As mentioned earlier, multicast communication is a frequently used communication pattern in many parallel applications. Thus, it is quite common to see special hardware supports for efficient multicast communication in many parallel systems, such as the TMC CM-5 [3] and the NEC Cenju-3 [4], as well as several high-speed switches, such as DEC GIGAswitch [64] and Myricom's Myrinet [8]. In addition, new message-passing libraries, such as MPI [12] and PVM [9], specify a multicast service as one of their standard collective communication primitives.

Our works have focused on several aspects of efficient software-based multicast communication. The topics include modeling, benchmarking, and fine-tuning. In this chapter, we concentrate on those works related to our topics.

7.1 Communication Models

Recent parallel models such as the BSP model [65], the postal model [51], and the LogP model [52] emphasize the importance of communication overhead. By taking the cost of communication into consideration, these models can address the limitation of other simple models, notable the PRAM model [66]. Although all three models are intended to address similar problems, they are based different communication approaches. The BSP model uses synchronous-based communication while the other two models use asynchronous-based communication. As our model also uses asynchronous-based communication, we discuss only the postal model and the LogP model.

The Postal Model

The postal model was proposed by Bar-Noy and Kipnis [51]. The model is based on the postal service system where a source can *send-and-forget* its message and eventually the message is delivered at the destination. Since the postal model is asynchronous-based, it is quite suitable for recent parallel machines whose networks are based on packet switching techniques.

The postal model has only one parameter, λ. λ is defined to be the ratio between the time it takes to deliver a message from a source to its destination and the time it takes a source to send a message. In the parameterized model:

$$\lambda = \frac{t_{end}}{t_{hold}} \tag{7.1}$$

Although the postal model is quite simple, it suffers from several drawbacks. First, λ is assumed to be an integer and its value is fixed for each system. As shown in Chapter 2, these are not true. As both t_{hold} and t_{end} depend on the message size, Equation 7.1 suggests that λ may not be an integer and it is not fixed for each system. Second, the original model does not discuss how to measure λ. Bruck et al. [37] address solutions for these drawbacks.

The LogP Model

The LogP model, proposed by Culler et al. [52], has recently become a widely used parallel model. Based on a reasonable abstract machine, the LogP model is flexible, realistic, and yet simple at the same time. In this model, a system can be characterized by four parameters including:

L the *latency* or delay incurred in communicating a message containing a single unit (or a word) from its source to its destination.

o the *overhead* when a processor is sending or receiving a message. The processor cannot perform other operations during this time.

g the *gap* between consecutive sends or receives at a processor. This parameter defines the available per-processor communication bandwidth.

P the number of processor/memory modules in the system.

Figure 7.1(a) shows the timing diagram of sending a message from P_0 to P_1 based on the LogP model. Obviously, this diagram is very similar to the timing diagram of the parameterized model since the parameterized model is derived from the LogP model ($t_{hold} = g$

and $t_{end} = L + 2o$). In addition, the postal model is a special case of the LogP model where $g = 1$, $o = 0$, and $L = \lambda$. Note that the LogP model works particular well for short messages. However, it is not adequate to model long messages. To overcome this limitation, Alexandrov *et al.* [54] propose the LogGP model. The LogGP model has one additional parameter, G, which represents the gap per byte for long messages. The diagram of the LogGP model is shown in Figure 7.1(b).

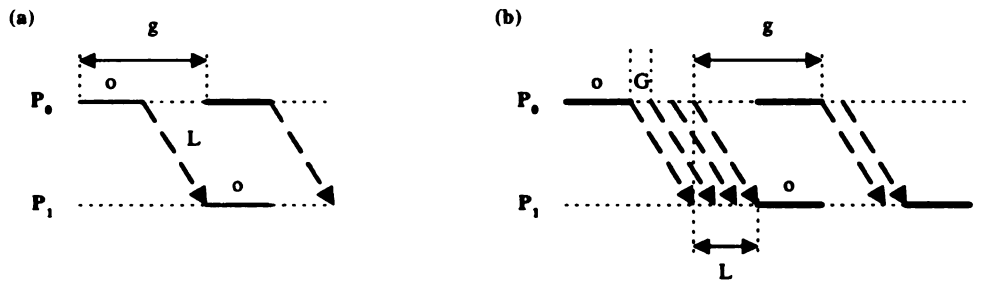


Figure 7.1: The timing diagram of (a) the LogP model, (b) the LogGP model.

Culler et al. [56] address several problems of the LogP model. They propose some small changes to the LogP model in order to increase its effectiveness in model a system. They distinguish between the send overhead and the receive overhead. To model large messages, they assume parameters L, o, and g to depend on message size. In addition, they propose a set of communication microbenchmarks to measure latency, overhead, and bandwidth. These microbenchmarks are based on Active Messages paradigm.

7.2 Multicast Benchmarks

There are several approaches to measure the latency of multicast communication. In a system which has a global clock, we can measure the multicast latency by using time-

stamp. Before the root node performs multicasting, it records the starting time. Upon receiving, all receivers record the time. Based on these records, we can find the starting time and the time that the last node receives the message. However, this approach is not applicable if there is no global clock or if the clock resolution is not fine enough.

Another approach is based on the knowledge of the multicast algorithm [60]. In this approach, an arbitrary node is picked as the root node. Given the root node, the last node can be predicted based on the multicast algorithm and the communication overheads. After the first multicast is completed, the last node becomes the root node and the whole process repeats again. Since the last node of the current multicast operation is always the root node of the next multicast, it can be guaranteed that the next multicast always starts immediately after the previous one is completed. The average time of this benchmark is the multicast latency. The main drawback of this approach is that the multicast algorithm must be known.

Bal *et al.* [67] propose a benchmark to evaluate a multicast operation on high-speed networks in clusters of workstations. To measure the multicast latency, they pick two arbitrary nodes to be senders. Each sender sends a multicast message when it receives a multicast message from the other sender. Note that during the measurement, the remain nodes only receive messages. The average multicast latency over all combinations of two nodes in the cluster is used as the multicast latency. This approach suffers from two problems. First, the multicast messages from both senders may create network contention since a new multicast may be issued before the previous one ended. If the network contention happens, the results can be skewed. Second, it is possible that, for some multicast algorithms, given any combinations of two nodes in the system, one node is not the last node in the multicast tree when the other node is the root node and vice versa. In this case, the latency is not the

multicast latency.

7.3 Platform-Independent Multicast

A platform-independent multicast algorithm is a software-based algorithm which uses only point-to-point communication services. To achieve both portability and high performance, the algorithm is usually based on an abstract parallel architecture model, such as the postal model and the LogP model. Typically, these abstract models use a small set of parameters to characterize the underlying network architecture. Based on these network parameters, the platform-independent algorithm can construct a multicast tree which matches the underlying network.

Bar-Noy and Kipnis [51] propose an optimal multicast algorithm for the postal model. To construct an optimal multicast tree, their algorithm use the *Fibonacci numbers* and divide-and-conquer method. Since λ is assumed to be an integer, their algorithm may not generate an optimal multicast tree when used in the real system. Bruck et al. [37] address this problem and propose the λ-tree algorithm. However, their algorithm is too complicated to implement. Furthermore, the shortcomings of the postal model discussed earlier limit the applications of the optimal multicast algorithms based on this model.

As the LogP model is a general-purpose model and its parameters are well-defined, constructing an optimal multicast tree based on this model is not difficult. Karp et al. [53] propose an efficient $O(k)$ algorithm based on the special labeling scheme. The algorithm starts from the root node whose label is 0 and adds more nodes until all nodes are added. When added, each node is labeled based on the time it expected to receive the multicast

message from its parent. By selecting the location to add a new node such that the new label is the smallest, it can be proven that a multicast tree constructed based on this criteria is optimal. Figure 7.2 shows an optimal multicast tree with its timing diagram for $P = 8$, $L = 6$, $g = 4$, and $o = 2$.

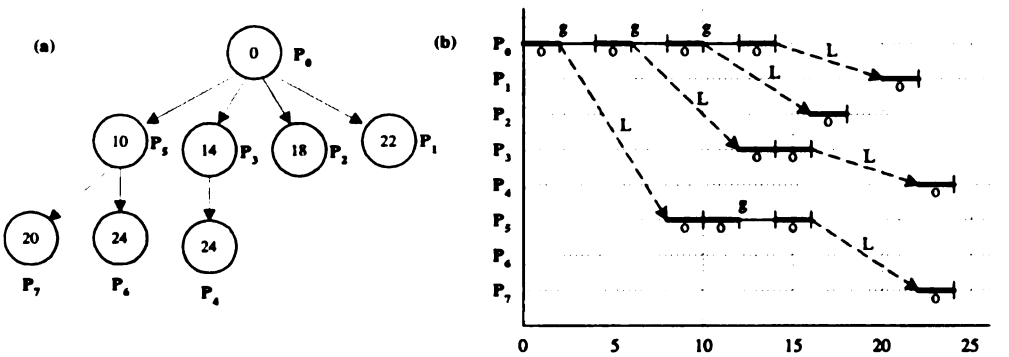


Figure 7.2: (a) An optimal multicast tree for $P = 8$, $L = 6$, $g = 4$, and $o = 2$. (b) The timing diagram.

In [54], Alexandrov et al. discuss the problem of modeling long messages with the LogP model. They also address the problem of optimal algorithms for multicast and scatter. Using the dynamic programming technique similar to our approach discussed in Chapter 4, their algorithm can construct a optimal multicast tree in $O(k^2)$ time. To improve its performance, they propose an approximate algorithm which can construct a nearly-optimal multicast tree in $O(k)$ time.

7.4 Platform-Dependent Multicast

The design of multicast communication based on the system characteristics, or platform-dependent multicast, have been quite well-studied. One popular technique is to design a multicast algorithm to efficient utilize communication channels based on the underlying

network topology. Research based on this technique include the spanning binomial tree (SBT) algorithm [41], the edge-disjoint spanning tree (EDST) algorithm [42], and the edge disjoint fences (EDF) algorithm [43]. Based on the network topology, we can also construct a contention-free multicast algorithm to further improve its performance. The examples are U-cube, U-mesh [44], and U-Min [45]. The details of these works have been discussed in Chapter 5.

In some systems, each node may have multiple channels, or multi-port interfaces, connecting to the network. With this multi-port capability, a node can send/receive several messages simultaneously. As multicast involves multiple receivers, utilizing the multi-port feature can greatly improve its performance. To demonstrate this technique, let consider Figure 7.3 which presents the EDN broadcast in a 16 $\times$ 16 mesh [49]. For 16 $\times$ 16 mesh, nodes are classified into four levels, including level-3 EDN, level-2 EDN, level-1 EDN, and level-0 node. When a broadcast is issued, the root node sends the broadcasting message to the highest level, level-3 EDN, nodes. Then the message is propagated to the lower level until it reaches all nodes. Based on the concept of extended dominating nodes, the EDN algorithm is contention-free. Other multi-port algorithms include the Double Tree (DT) algorithm [46], Ho and Kao's algorithm [47], and the W-sort algorithm [48].

7.5 Conclusion

In this chapter, we have reviewed several related works and presented the difference between our work and theirs. Modeling multicast based on the postal model can be quite difficult because the model has several drawbacks. The generic model such as the LogP

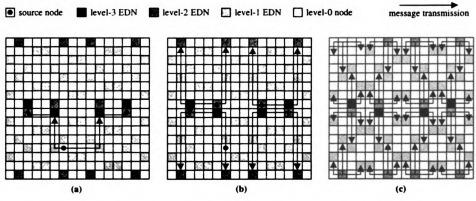


Figure 7.3: EDN broadcast in a 16 × 16 mesh.

model is very powerful. However, having many parameters makes multicast modeling to be quite complicated. Besides choosing a suitable basic communication model, evaluating multicast is also very crucial when modeling a multicast. Although there are several works addressing this issue, they fail to either define their metrics clearly or use correct measurement techniques.

Both platform-independent and platform-dependent multicast algorithms have been extensively investigated, Based on different models, these algorithms have their own limitations. In addition, these works have been done under the ideal condition where there is no other traffic in the network. As discussed in Chapter 6, the presence of other traffic can effect the performance of a multicast.

Chapter 8

Conclusion and Future Work

Efficient multicast communication is critical to the performance of parallel programs on scalable parallel computers. This thesis studies each several aspects of implementing efficient software-based multicast communication. In this chapter, we summarize the contributions made by this research and present interesting issues for possible future investigation.

8.1 Research Contributions

Software-based multicast usually facilitates point-to-point communication services to propagate a message from the root node to all destinations. Understanding the cost of these point-to-point communication services is very crucial in modeling software-based multicast communication. Although the point-to-point communication is very simple, there are several parameters which can be used to describe its cost. We discuss the assumptions and the criteria when choosing only necessary and sufficient parameters for modeling multicast. Based on our criteria, we propose the parameterized communication model derived from

the LogP model. The thesis also discusses how to measure each parameter in the model.

To fairly compare two multicast implementations, two issues are needed to be addressed. The first issue is to identify the most representative metrics. The chosen metrics must be well-defined and must be able to characterized the actual overhead of a multicast service. The other important issue is to use correct measurement techniques. Most measuring techniques are either inaccurate or having some limitations. We choose the multicast latency as the most representative metric for a multicast communication service. We provide its formal definition and propose its measuring technique. Our proposed technique is quite simple, more accurate, and have less limitation than other techniques. This thesis also discusses how to verify the correctness of our technique.

A key issue in designing a software-based multicast algorithm is to consider the trade-off between performance and portability. The main drawback of a platform-independent multicast algorithm is that its performance is typically varied significantly from one platform to another. To overcome this problem, this thesis proposes the parameterized multicast algorithm. Our algorithm is portable, but still performs well on various platforms. The main concept of our algorithm is to characterize the underlying network using the parameterized model. Based on the model, our algorithm constructs an optimal multicast tree. The original algorithm is based on the dynamic programming technique which is not efficient enough. We further discuss how to improve its runtime to be $O(n)$. The results from the IBM/SP indicate that our algorithm is significantly better than other well-known algorithms, including the implementation provided by the underlying communication library.

The optimality of the parameterized algorithm is based on the assumption that the underlying network has no communication contention. For a real network, this assumption

is not practical. The contention in the network can prevent our algorithm from achieving its optimal performance. Therefore, it is necessary to consider the architecture-dependent characteristics of a system, such as network topology and switching mechanism. We study the platform-dependent tuning of the parameterized algorithm based on two contention-free multicast algorithms, U-Mesh and U-Min. We propose two algorithms which can generate contention-free parameterized multicast trees on wormhole-switched mesh and BMIN networks.

Most multicast-related research have been done under the ideal condition in which the effect of other traffic in the network is ignored. Typically, the actual network condition is not ideal. Based on the simulation studies, we investigate the performance of several multicast algorithms on two different networks which are the wormhole-switched mesh network with the X-first-Y-next routing, and the wormhole-switched BMIN network with the turn-around routing. The main focus of our study is to examine how useful is the parameterized multicast algorithm when the background traffic exists. The results indicate that our algorithm performs reasonably well in most traffic conditions on both networks. We also propose a new class of multicast algorithm called the dynamic multicast algorithm. Our dynamic multicast algorithm handles the dynamic environment due to the background traffic by adjusting its own structure dynamically based on the local network information. The results from the simulation studies show that the dynamic multicast algorithm performs as well as the parameterized multicast algorithm in most cases and performs better when the network load is high or the irregular traffic, such as multiple multicast, exists.

8.2 Directions for Future Research

In this thesis, we have concentrated our effort on the software-based multicast implementation. The framework presented in this work establish the foundation for future study but need to be extended in several ways.

The communication flow model is proposed to be a basis for our multicast benchmarking technique and can be applied to evaluate other one-to-all communication services as well. However, most parallel applications also utilize other collective communication services, such as reduce and barrier synchronization. More research can be done to extend the communication flow model to be a framework for evaluating all-to-one and all-to-all communication services.

The concept of platform-dependent tuning can be applied to any network as long as the underlying networks can be partitioned into contention-free processor clusters. In some networks, such as a butterfly unidirectional MIN, this partitioning may not be possible. In this case, the best one can do is minimizing the internal contention as much as possible. Instead of preventing a common communication channel used by different senders at any time, some channels are allowed to be shared. However, the senders who share the same communication channels are ordered such that they are unlikely to send at the same time. More research is needed for such networks. In addition, other system characteristics, such as multi-port capability, are also good candidates for fine-tuning our parameterized multicast algorithm.

Finally, a particularly challenging direction is to further study the multicast behavior in the real network. There are several challenges. The first one is to improve the study model

to be as close as the actual condition in the real network. The current study is based on the steady-state of the network. However, in the real network, its condition does not remain at one particular state all the time. Thus, the multicast behavior can be varied. The second one is to study the multicast behavior on other networks such as irregular network. In addition, the proposed dynamic multicast algorithm is still based on simple network information. Some additional network information, such as the estimated latency from the root node to other nodes, may be useful to determine not only the structure of the multicast tree, but also the order of the send at the root node. Utilizing additional network information effectively becomes another challenging issue. However, out-of-date information and poor prediction can result to constructing a multicast tree with bad performance.

BIBLIOGRAPHY

Bibliography

[1] Cray Research, "CRAY T3E-900." [Online] Available http://www.cray.com/news/9611/crayt3e900.html, Nov. 1996.

[2] D. Pountain, "TeraFLOPS POWer," *BYTE Magazine*, Mar. 1998.

[3] C. E. Leiserson *et al.*, "The network architecture of the Connection Machine CM-5," in *Proceedings of the ACM Symposium on Parallel Algorithms and Architectures*, (San Diego, CA.), pp. 272–285, Association for Computing Machinery, 1992.

[4] N. Koike, "NEC Cenju-3: A micorprocessor-based parallel computer," in *Proceeding of the 8th International Parallel Processing Symposium*, 1994.

[5] Cray Research, Inc., Chippewa Falls, Wisconsin, *CRAY T3D System Architecture Overview*, 1993.

[6] T. E. Anderson, D. E. Culler, D. A. Patterson, and the NOW team, "A Case for NOW (Networks of Workstations)," *IEEE Micro*, Feb. 1995.

[7] C. Partridge, "GIGABIT NETWORKING," Oct. 1993.

[8] N. J. Boden, D. Cohen, R. E. Felderman, A. E. Kulawik, C. L. Seitz, J. N. Seizovic, and W.-K. Su, "Myrinet: A gigabit-per-second local area network," *IEEE Micro*, pp. 29–36, Feb. 1995.

[9] V. S. Sunderam, "PVM: A framework for parallel distributed computing," *Concurrency: Practice and Experience*, vol. 2(4), pp. 315–339, Dec. 1990.

[10] R. Alasdair, A. Bruce, J. G. Mills, and A. G. Smith, *CHIMP Version 2.0 User Guide*. University of Edinburgh, Mar. 1994.

[11] G. Burns, R. Daoud, and J. Vaigl, *LAM: An Open Cluster Environment for MPI*. Ohio Supercomputer Center, May 1994.

[12] M. P. I. Forum, *MPI: A Message-Passing Interface Standard*, Mar. 1994.

[13] N. Nupairoj and L. Ni, "Performance Evaluation of Some MPI Implementations on Workstation Clusters," in *Scalable Parallel Library Conference '94*, (Mississippi State University, MS), pp. 98–105, Oct. 1994.

[14] M. J. Lewis and J. Raymond E. Cline, "PVM Communication Performance in a Switched FDDI Heterogeneous Distributed Computing Environment," in *Proceedings of the IEEE Workshop on Advances in Parallel and Distributed Systems*, Oct. 1993.

[15] J. Kay and J. Pasquale, "Measurement, Analysis, and Improvement of UDP/IP Throughput for the DECstation 5000," in *Proceedings of the Winter 1993 USENIX Conference*, Jan. 1993.

[16] C. Thekkath and H. Levy, "Limits to Low-Latency Communication on High-Speed Networks," *ACM Transactions on Computer Systems*, vol. 11, pp. 179–203, May 1993.

[17] High Performance Fortran Forum, "High Performance Fortran Language Specification (version 1.0, draft)," Jan. 1993.

[18] H. Xu, P. K. McKinley, and L. M. Ni, "Efficient implementation of barrier synchronization in wormhole-routed hypercube multicomputers," *Journal of Parallel and Distributed Computing*, vol. 16, pp. 172 – 184, October 1992.

[19] D. Dai and D. K. Panda, "Reducing Cache Invalidation Overheads in Wormhole DSMs Using Multidestination Message Passing," in *Proceedings of the 1996 International Conference on Parallel Processing*, pp. 138–145, Aug. 1996.

[20] D. E. Comer, *Internetworking with TCP/IP*, vol. 1. Englewood Cliffs, New Jersey: Prentice-Hall, Inc., second ed., 1991.

[21] V. Kompella, J. Pasquale, and G. Polyzos, "Multicasting for multimedia applications," in *Proceedings of the IEEE INFOCOM'92*, Mar. 1992.

[22] D. R. Cheriton and S. E. Deering, "Host groups: A multicast extension to the Internet protocol," Tech. Rep. RFC-966, SRI Network Information Center, Dec. 1985.

[23] S. E. Deering, "Multicast Routing in Internetworks and Extended LANs," in *Proceeding of the ACM SIGCOMM*, pp. 55–64, Aug. 1988.

[24] S. E. Deering and D. R. Cheriton, "Multicast routing in datagram internetworks and extended LANs," *ACM Transactions on Computer Systems*, vol. 8, pp. 85–110, May 1990.

[25] H. Eriksson, "MBONE: The multicast backbone," *Communications of the ACM*, vol. 37, pp. 54–60, Aug. 1994.

[26] C. Perkins and J. Crowcroft, "Real-Time Audio and Video Transmission of IEEE GLOBECOM'96 over the Internet," *IEEE Communications Magazine*, Apr. 1997.

[27] H. Jinzenji and K. Hagishima, "Real-Time Audio and Video Broadcasting of IEEE GLOBECOM'96 over the Internet Using New Software," *IEEE Communications Magazine*, Apr. 1997.

[28] K. A. Hua and S. Sheu, "Skyscraper Broadcasting: A New Broadcasting Scheme for Metropolitan Video-on-Demand Systems," in *Proceeding of the ACM SIGCOMM*, pp. 89–100, Sept. 1997.

[29] A. Rosenstein, J. Li, and S. Y. Tong, "MASH: The Multicasting Archie Server Hierarchy," *Computer Communication Review*, July 1997.

[30] S. Mittra, "Iolus: A Framework for Scalable Secure Multicasting," in *Proceeding of the ACM SIGCOMM*, pp. 277–288, Sept. 1997.

[31] J. E. Donnelley, "WWW Media Distribution via Hopwise Reliable Multicast," in *Proceedings of the Third International World-Wide Web Conference*, (Darmstadt, Germany), Apr. 1995.

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

[33] V. P. Kompella, J. C. Pasquale, and G. Polyzos, "Multicast Routing for Multimedia Communication," *IEEE Transactions on Networking*, vol. 1, June 1993.

[34] L. Wei and D. Estrin, "The Trade-offs of Multicast Trees and Algorithms," in *Proceedings of 1994 International Conference on Computer Communications and Networks*, 1994.

[35] L. M. Ni, "Should Scalable Parallel Computers Support Efficient Hardware Multicast," in *Proceedings of the 1995 ICPP Workshop on Challenges for Parallel Processing*, Aug. 1995.

[36] R. van Renesse, T. M. Hickey, and K. P. Birman, "Design and Performance of Horus: A Lightweight Group Communications System," Tech. Rep. TR94-1442, Department of Computer Science, Cornell University, Aug. 1994.

[37] J. B. et al., "Efficient Message Passing Interface (MPI) for Parallel Computing on Clusters of Workstations," in *Seventh Annual Symposium on Parallel Algorithms and Architectures*, (Santa Barbara, CA), pp. 64–73, July 1995.

[38] B. B. Lowekamp and A. Beguelin, "ECO: Efficient Collective Operations for Communication on Heterogeneous Networks," in *Proceeding of the 10th International Parallel Processing Symposium*, 1996.

[39] J.-Y. L. Park, H.-A. Choi, N. Nupairoj, and L. M. Ni, "Construction of Optimal Multicast Trees Based on the Parameterized Communication Model," in *Proceedings of the 1996 International Conference on Parallel Processing*, pp. 180–187, Aug. 1996.

[40] J. Bruck, L. de Coster, N. Dewulf, C.-T. Ho, and R. Lauwereins, "On the Design and Implementation of Broadcast and Global Combine Operations using the Postal Model," in *Proceedings of the 1994 Symposium on Parallel and Distributed Processing*, pp. 594–602, Oct. 1994.

[41] H. Sullivan and T. R. Bashkow, "A large scale, homogeneous, fully distributed parallel machine," in *Proceedings of the 4th Annu. Symp. Comput. Architecture*, vol. 5, pp. 105–124, Mar. 1977.

[42] S. L. Johnsson and C.-T. Ho, "Optimum broadcasting and personalized communication in hypercubes," *IEEE Transactions on Computers*, vol. C-38, pp. 1249–1268, Sept. 1989.

[43] M. Barnett, D. G. Payne, R. A. van de Gejin, and J. Watts, "Broadcasting on Meshes with Worm-hole Routing," Tech. Rep. TR-93-24, Department of Computer Science, University of Texas at Austin, Nov. 1993.

[44] 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*, vol. 5, pp. 1252–1265, Dec. 1994.

[45] H. Xu, Y. Gui, and L. M. Ni, "Optimal software multicast in wormhole-routed multistage networks," in *Proceedings of Supercomputing'94*, pp. 703–712, Nov. 1994.

[46] P. K. McKinley and C. Trefftz, "Efficient broadcast in all-port wormhole-routed hypercubes," in *Proceedings of the 1993 International Conference on Parallel Processing*, vol. II, (St. Charles, IL), pp. 288–291, Aug. 1993.

[47] C.-T. Ho and M.-Y. Kao, "Optimal broadcast in all-port worm-routed hypercubes," in *Proceedings of the 1994 International Conference on Parallel Processing*, pp. 167–171, Aug. 1994.

[48] D. F. Robinson, D. Judd, P. K. McKinley, and B. H. Cheng, "Efficient collective data distribution in all-port wormhole-routed hypercubes," in *Proceedings of Supercomputing'93*, Nov. 1993.

[49] Y.-J. Tsai and P. K. McKinley, "An Extended Dominating Node Approach to Collective Communcaition in Wormhole-Routed 2D meshes," in *Proceedings of the Scalable High Performance Computing Conference*, pp. 199–206, 1994.

[50] P. K. McKinley, Y.-J. Tsai, and D. F. Robinson, "Collective Communication in Wormhole-Routed Massively Parallel Computers," *IEEE Computer*, Dec. 1995.

[51] A. Bar-Noy and S. Kipnis, "Designing Broadcasting Algorithms in the Postal Model for Message-Passing Systems," in *Fourth Annual Symposium on Parallel Algorithms and Architectures*, (San Diego, CA), pp. 13–22, June 1992.

[52] D. Culler *et al.*, "LogP: Towards a realistic model of parallel computation," in *Proc. of the 4th ACM SIGPLAN Sym. on Principles and Practice of Parallel Programming*, May 1993.

[53] R. Karp, A. Sahay, and E. Santos, "Optimal Broadcast and Summation in the LogP Model," in *Fifth Annual Symposium on Parallel Algorithms and Architectures*, pp. 142–153, June 1993.

[54] A. Alexandrov, M. F. Ionescu, K. E. Schauser, and C. Scheiman, "LogGP: Incorporating Long Message into the LogP Model," in *Seventh Annual Symposium on Parallel Algorithms and Architectures*, (Santa Barbara, CA), pp. 95–105, July 1995.

[55] H. Franke, P. Hochschild, P. Pattnaik, and M. Snir, "MPI-F: An Efficient Implementation of MPI on IBM-SP1," in *Proceedings of the 1994 International Conference on Parallel Processing*, (St. Charles, IL), pp. 197–201, Aug. 1994.

[56] D. E. Culler, L. T. Liu, R. P. Martin, and C. Yoshikawa, "LogP Performance Assessment of Fast Network Interfaces," *IEEE Micro*, Feb. 1996.

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

[58] A. S. Tanenbaum, *Modern Operating Systems*. Englewood Cliffs, New Jersey: Prentice-Hall, Inc., 1992.

[59] N. Nupairoj and L. M. Ni, "Performance Metrics and Measurement Techniques of Collective Communication Services," in *First International Workshop on Communication and Architectural Support for Network-Based Parallel Computing (CANPC'97)*, (San Antonio, Tx), pp. 212–226, Feb. 1997.

[60] H. Franke, "Private communication," Mar. 1995.

[61] T. H. Cormen, C. E. Leiserson, and R. L. Rivest, *Introduction to Algorithms*. Cambridge, Massachusetts: The MIT Press, 1992.

[62] N. Nupairoj and L. Ni, "Benchmarking of Multicast Communication Services," Tech. Rep. MSU-CPS-ACS-103, Department of Computer Science, Michigan State University, Apr. 1995.

[63] P. A. Steenkiste, "A Systematic Approach to Host Interface Design for High-Speed Networks," *IEEE Computer*, Mar. 1994.

[64] R. J. S. et al, "Gigaswitch system: A high-performance packet-switching platform," *Digital Technical Journal*, vol. 6, no. 1, pp. 9 – 22, 1994.

[65] L. G. Valiant, "A Bridging Model for Parallel Computation," *Communications of the Association for Computing Machinery*, vol. 33, pp. 103–111, Aug. 1990.

[66] S. Fortune and J. Wyllie, "Parallelism in random access machines," in *Proceedings of the 10th Annual Symposium on Theory of Computing*, pp. 114–118, 1978.

[67] H. Bal, R. Hofman, and K. Verstoep, "A Comparison of Three High Speed Networks for Parallel Cluster Computing," in *First International Workshop on Communication and Architectural Support for Network-Based Parallel Computing (CANPC'97)*, (San Antonio, Tx), pp. 184–197, Feb. 1997.