Spring 2000

Study of architecture and protocols for reliable multicasting in packet switching networks

Shiwen Chen
New Jersey Institute of Technology

Copyright Warning & Restrictions

The copyright law of the United States (Title 17, United States Code) governs the making of photocopies or other reproductions of copyrighted material.

Under certain conditions specified in the law, libraries and archives are authorized to furnish a photocopy or other reproduction. One of these specified conditions is that the photocopy or reproduction is not to be "used for any purpose other than private study, scholarship, or research." If a, user makes a request for, or later uses, a photocopy or reproduction for purposes in excess of "fair use" that user may be liable for copyright infringement,

This institution reserves the right to refuse to accept a copying order if, in its judgment, fulfillment of the order would involve violation of copyright law.

Please Note: The author retains the copyright while the New Jersey Institute of Technology reserves the right to distribute this thesis or dissertation

Printing note: If you do not wish to print this page, then select "Pages from: first page # to: last page #" on the print dialog screen

ABSTRACT

STUDY OF ARCHITECTURES AND PROTOCOLS FOR RELIABLE MULTICASTING IN PACKET SWITCHING NETWORKS

by
Shiwen Chen

Group multicast protocols have been challenged to provide scalable solutions that meet the following requirements: (i) reliable delivery from different sources to all destinations within a multicast group; (ii) congestion control among multiple asynchronous sources. Although it is mainly a transport layer task, reliable group multicasting depends on routing architectures as well.

This dissertation covers issues of both network and transport layers. Two routing architectures, tree and ring, are surveyed with a comparative study of their routing costs and impact to upper layer performances. Correspondingly, two generic transport protocol models are established for performance study. The tree-based protocol is rate-based and uses negative acknowledgment mechanisms for reliability control, while the ring-based protocol uses window-based flow control and positive acknowledgment schemes. The major performance measures observed in the study are network cost, multicast delay, throughput and efficiency. The results suggest that the tree architecture costs less at network layer than the ring, and helps to minimize latency under light network load. Meanwhile, heavy load reliable group multicasting can benefit from ring architecture, which facilitates window-based flow and congestion control.

Based on the comparative study, a new two-hierarchy hybrid architecture, Rings Interconnected with Tree Architecture (RITA), is presented. Here, a multicast group is partitioned into multiple clusters with the ring as the intra-cluster architecture, and the tree as backbone architecture that implements inter-cluster multicasting. To compromise between performance measures such as delay and

throughput, reliability and congestion controls are accomplished at the transport layer with a hybrid use of rate and window-based protocols, which are based on either negative or positive feedback mechanisms respectively. Performances are compared with simulations against tree- and ring-based approaches. Results are encouraging because RITA achieves similar throughput performance as the ring-based protocol, but with significantly lowered delay.

Finally, the multicast tree packing problem is discussed. In a network accommodating multiple concurrent multicast sessions, routing for an individual session can be optimized to minimize the competition with other sessions, rather than to minimize cost or delay. Packing lower bound and a heuristic are investigated. Simulation show that congestion can be reduced effectively with limited cost increase of routings.

STUDY OF ARCHITECTURES AND PROTOCOLS FOR RELIABLE MULTICASTING IN PACKET SWITCHING NETWORKS

by
Shiwen Chen

A Dissertation
Submitted to the Faculty of
New Jersey Institute of Technology
in Partial Fulfillment of the Requirements for the Degree of
Doctor of Philosophy

Department of Computer and Information Science

May 2000

APPROVAL PAGE

STUDY OF ARCHITECTURES AND PROTOCOLS FOR RELIABLE MULTICASTING IN PACKET SWITCHING NETWORKS

Shiwen Chen

Dr. Joseph Leung, Committee Chair — Date
Distinguished Professor of Computer and Information Science, NJIT

Dr. Bulent Yener, Dissertation Advisor — Date
Member of Technical Staff, Bell-Labs, Lucent Technologies Inc.

Dr. Ali N. Akansu, Committee Member — Date
Professor of Electrical and Computer Engineering, NJIT

Dr. Dennis Karvelas, Committee Member — Date
Director of M.S. in Telecommunications, NJIT

Dr. Marvin Nakayama, Committee Member — Date
Associate Professor of Computer and Information Science, NJIT

Dr. Yoram Ofek, Committee Member — Date
President, Synchrodyne Inc.

BIOGRAPHICAL SKETCH

Author: Shiwen Chen

Degree: Doctor of Philosophy

Date: May 2000

Undergraduate and Graduate Education:

- Doctor of Philosophy in Computer Science,
  New Jersey Institute of Technology, Newark, New Jersey, USA, 2000
- Master of Science in Electrical Engineering,
  Beijing University of Posts and Telecommunications, Beijing, China, 1991
- Bachelor of Science in Electrical Engineering,
  Beijing University of Posts and Telecommunications, Beijing, China, 1988

Major: Computer Science

Presentations and Publications:

Shiwen Chen, Oktay Gunluk, and Bulent Yener,
"The Multicast Packing Problem",
to appear in IEEE/ACM Transactions on Networking, 2000.

Shiwen Chen, Bulent Yener, and Yoram Ofek,
"Performance Trade-offs in Reliable Group Multicast Protocols",
Proceedings of IEEE INFOCOM'99, pp. 982-989, March 1999.

Shiwen Chen, Oktay Gunluk, and Bulent Yener,
"Optimal Packing of Group Multicastings",
Proceedings of IEEE INFOCOM'98, pp. 980-987, March 1998.

Shiwen Chen, Dennis Karvelas, Qicheng Ma, and Bulent Yener,
"A Simple Hierarchical Approach to Intra/Inter Domain Multicasting",
Proceedings of SPIE'97: Performance and Network Control, vol. 3231,
pp. 102-112, October 1997.

This work is dedicated to
my wife, my mom and dad.

ACKNOWLEDGMENT

Most especially, I would like to thank my advisor, Dr. Bulent Yener, who initially suggested and continuously supported the work presented in this dissertation. I am deeply grateful to his commitment of advising my study. Without his continuing advice, this dissertation could never have been finished.

Thanks to Dr. Joseph Leung, my committee chair, who has helped me so much so that I can continue my school career.

Thanks to all my committee members: Yoram Ofek, Dennis Karvelas, Marvin Nakayama, and Ali Akansu. I benefited greatly from working with them and getting their valuable advises. I am also grateful that I had the opportunity to be able to work with Oktay Gunluk.

Thanks to David Faust for proofreading my thesis. David not only has corrected my English, but has also made valuable suggestions to me and deeply impressed me with his seriousness. I also owe many thanks to Leon Jololian, Carole Poth, Ronald Kane, Sharon Gilbert, and Barbara Harris, who were always there to help me. Thanks to Jeffrey Weinick at Lucent Technologies Inc., who is helping me with a patent application and helped me clarified many details of part of the work in this dissertation. Thanks to Dr. Peter B. Lederman for helping me with the copyright issue. Thanks to Annette Damiano for reviewing the dissertation format. Thanks to my friends: Junao Xue, Xiong Wang, Feihong Chen, Jiangli Zhu, Zhiyuan Wang and Qicheng Ma, from whom I got so many encouragements, valuable suggestions and important information.

I gratefully acknowledge the support from the grant of the Office of Naval Research.

TABLE OF CONTENTS

TABLE OF CONTENTS
(Continued)

TABLE OF CONTENTS
(Continued)

LIST OF TABLES

LIST OF FIGURES

LIST OF FIGURES
(Continued)

Figure **Page**

LIST OF FIGURES
(Continued)

LIST OF FIGURES
(Continued)

Figure **Page**

GLOSSARY

ACK:	Acknowledgment control message.
AIM:	Addressable Internet Multicast routing protocol.
BN:	Bridge node.
BNMT:	Bridge node multicast transport protocol.
CBT:	Core-based tree protocol.
CTN:	Candidate transient node.
DDL:	Dominant direction load.
DVMRP:	Distance vector multicast routing protocol.
FEC:	Forward error correction.
IGMP:	Internet group management protocol.
KAM:	Keep-alive control message.
LBRM:	Log-based receiver-reliable multicast protocol.
LPM:	Loss path multiplicity problem.
LRMP:	Light-weight reliable multicast protocol.
MBONE:	Multicast backbone.
MN:	Member node.
MNMT:	Member node multicast transport protocol.
MOSPF:	Multicast extensions to OSPF (Open Shortest Path First) routing protocol.
NACK:	Negative acknowledgment control message.
OY:	A ring-based reliable group multicast protocol by Ofek and Yener.
PDU:	Protocol data unit.
PGM:	Pragmatic general multicast protocol.

GLOSSARY
(Continued)

PIM: Protocol independent multicast protocol.

RBRM: Ring-based reliable multicast.

RITA: Rings interconnected with tree architecture.

RMA: Reliable multicast architecture.

RMP: Reliable multicast protocol.

RMTP: Reliable multicast transport protocol.

RMX: Reliable multicast proxies architecture.

SRM: Scalable reliable multicast protocol.

TBRM: Tree-based reliable multicast.

TCP: Transmission control protocol.

TN: Transient node.

TSP: Traveling salesman problem.

VUE: Virtual path with unique embedding.

CHAPTER 1

INTRODUCTION

Multicast service is provided by a network to deliver information to multiple destinations [28]. Traditionally, network unicast service delivers messages to a single destination, and broadcast service sends messages all over a network. In contrast, multicast service is to deliver messages to a *group* of users in a network.

The significance of network multicast service has been widely recognized during this recent decade. Multicast technology facilitates more efficient utilization of network resources, and makes it feasible for new network applications to implement multi-point communications across the network. With the increasing power of computing and networking technology, more and more applications need to communicate among multiple network end points. Examples include, conferencing across the Internet, stock and news updates, distributed enterprise database operations, etc.

The Internet is a typical packet-switching network. One important feature of such networks is the connectionless best-effort network layer service. This brings challenges to multicasting implementations, particularly for transport and network layers in a packet-switching network. Based on the underlying routing architecture, protocols implementing transport-layer functions must, according to different multicast application needs, address issues such as performance, reliability, flow and congestion control, etc. However, in contrast to traditional unicast service, there is a rich diversity of potential multicast routing architectures. Therefore, network-layer protocols not only must be able to deliver multicast packets efficiently, but also have to set up a routing architecture that can benefit the upper-layer functions.

The focal point of this dissertation is reliable group multicasting. Although applications have different requirements for network reliability and have different levels of error tolerance, a fully reliable multicast service is always necessary to guarantee delivery of a multicast packet from its source to every multicast receiver. Having multiple receivers makes a fully-reliable implementation more difficult, as we will discuss in the following section. However, it is even more challenging for group multicasting, in which case there are multiple asynchronous multicast sources (i.e., many-to-many multicasting). Unlike most research on reliable multicasting, this dissertation not only investigates the transport protocol operations and implementations, but also studies the impacts of different network-layer architectures.

Let us first review the background and prevalent works on this topic in the following section. Pertinent works that are related to the works in this dissertation are highlighted in subsequent chapters.

1.1 Background

1.1.1 Multicast Routing: Network Layer Issues

Motivated by studies on efficient broadcast algorithms [23, 11, 37], early research started by solving one-to-many multicast routing [1, 27, 61, 60, 77, 30]. Among these pioneering work, Deering's work [27, 24, 28] made the breakthrough. Deering's major contributions is the definition of models and architecture for datagram internetwork multicast, as well as several protocols used for IP multicast. The *"host group model"* defines the relation between a multicast host and a multicast group. In this model, the set of destinations of a multicast packet is called a host group, and all destination hosts share a same group address (or multicast address). This concept of *logical addressing* in the model is widely adopted in today's Internet multicast services. Properties like non-member sources and unlimited dynamic group membership also have become the basic requirements for a multicast service. The *Distance-Vector*

Multicast Routing Protocol (DVMRP) (updated later in [67]) and *Internet Group Management Protocol (IGMP)* [29, 33] proposed by Deering became the starting protocols for IP multicast.

With the introduction of IGMP and multicast routing protocols such as DVMRP and MOSPF [61, 60], the multicast backbone (MBONE)[31] was set up as a test bed for the multicast community. Now the MBONE has evolved into a large virtual backbone on top of the Internet, spanning several countries.

The basic strategy for one-to-many multicast is routing by shortest paths. Shortest paths between a source to each receiver form the multicast tree, which is *"source based"*. Therefore, a source-based shortest-path tree provides minimal delay.

However, with group multicast applications where each member could be a source, setting up a multicast tree for each member is not efficient in cost. A low-delay strategy does not necessarily provide a low-cost solution. Later research proposed solutions based on low-cost strategy rather than the low-delay strategy.

Two important protocols were developed to use one shared multicast routing tree to delivery packets from different sources. One is the *Core Based Tree* by Ballardie [8, 7, 5, 6]. The multicast tree is built by finding a core such that the tree paths from the core to receivers are shortest paths. Packets from different senders in a group are delivered along the same tree.

CBT motivated the Protocol Independent Multicast protocol (PIM) [26, 25, 32]. PIM distinguishes two different network multicast configurations: sparse mode and dense mode. In sparse mode, multicast sites are scattered sparsely in the internetwork, and a shared tree can be built by PIM_SM to avoid too much overhead. Low network cost is thus achieved. In dense mode, PIM_DM can be used to build multicast trees. PIM also provides the mechanism for receivers switching from group-

shared tree to source-based tree. However, one major drawback about PIM is its complexity.

Some recent works on multicast routing focus on performance study [57, 86, 2, 41] and new protocols with improvements to the above works and addressed issues like resource reservation [89, 68], correctness and reliability [64, 73], and hierarchical routing [19].

In all, with the thriving of the Internet, research on multicast routing is mostly oriented toward the datagram network environment. Multicast routing generally uses a tree structure to distribute multicast messages across the network. According to different types of multicast, one-to-many or many-to-many, multicast tree can be either source based or shared to achieve low-delay or low-cost objectives.

1.1.2 Reliable Multicast Service at Transport Layer

Because connectionless and best-effort service is the essential property of packet-switching networks, other than multicast routing, multicast support from the network layer is limited. Important issues such as end-to-end multicast reliability have to be addressed by transport-layer protocols. Overall, the major issues for reliable multicasting are: feedback control, loss recovery and congestion control. Below we provide an overview of this area, including both one-to-many and many-to-many cases.

1.1.2.1 Feedback Control: Feedback control is fundamental for reliable multicasting. Early works [80, 83, 76] emulate the Transmission Control Protocol (TCP) [66] by having receivers send acknowledgments (ACKs) to the sender, in order to ensure reliability. However these *ACK-based* protocols did not pay enough attention to the ACK implosion problem [69], and acknowledgment control is weak in these protocols. Besides, it can be an overwhelming overhead for the sender(s) to keep

track of the receiver set. Therefore these protocols may not scale well for large multicast sessions.

One approach to solve the ACK-implosion problem is to introduce a hierarchical structure, as used in some one-to-many reliable multicast protocols. In this structure-based approach, intermediate nodes in the multicast tree have the responsibility to process and combine feedback informations. Therefore, ACKs are aggregated so that a sender gets feedbacks only from neighbors in the hierarchy next to it. A typical example is the Reliable Multicast Transport Protocol (RMTP, [65, 54, 71]) proposed by Lin and Paul. One similar approach is reported by Hofmann [43, 42] which divides a multicast group into subgroups. Subgroup controllers are responsible for loss recovery in their corresponding subgroups so as to reduce the burden of a sender.

Another approach is to employ negative acknowledgment (NACK) scheme. There are many works [15, 84, 58, 14] that use a representative to send ACKs to the sender, while use NACKs between the representative and other receivers. In the Log-Based Receiver-reliable Multicast (LBRM, [44]) protocol presented by Holbrook et al., a logging server sends ACKs to the sender and accepts negative acknowledgments as retransmission requests from receivers. In the work by Chang and Maxemchuck [15], a *token site* is rotated among receivers to send ACKs to the sender and process NACKs from other receivers. Motivated by this token-passing mechanism, the Reliable Multicast Protocol (RMP) [84] includes new mechanisms such as multicasting ACKs and NACKs, to speed up loss detection and avoid NACK implosion [38].

NACK implosion may occur when many receivers detect losses of the same packet. This is a problem in several NACK-based protocols, such as the above LBRM and others [15, 4, 12, 47]. Grossglauser proposed a *time-based* approach in [38], which uses the *delayed feedback* technique. A NACK is multicasted to the whole group so that other receivers can hold and postpone their NACKs if they lost the

same data packet. This scheme is also known as *NACK suppression.* Besides RMP, some other works [88, 34, 72, 59] also adopt this mechanism.

Generally, in the ACK-based approach it is the sender who detects losses and initiates loss recovery, while in the NACK-based approach it is the receivers. Whether the sender or the receivers should detect packet losses and initiate loss recovery. Comparative studies [87, 78] show that receiver-initiated protocols outperform the *sender-initiated* ones. Also in [49, 51], Levine analyzed different classes of protocols and concluded that protocols with NACK-avoidance protocols achieve better throughput than those with the rotating token site mechanism do.

Some recent research proposes to extend network-layer support to facilitate transport-layer reliability control. In [52, 50], the AIM (e.g., Addressable Internet Multicast) is proposed as the network-layer support to establish an ACK tree for reliable multicasting. Another example is the Pragmatic General Multicast protocol (PGM, a.k.a. Pretty Good Multicast) [74]. With PGM, a network-layer hop-by-hop procedure is defined so that NACKs can be forwarded with enhanced reliability back to the source reliably. Routers are required to maintain source-path state information and process NACKs from receivers. It also eliminates the necessity of multicast NACKs, therefore significantly reducing multicast traffic complexity.

1.1.2.2 Loss Recovery: Two schemes are widely adopted in most proposed reliable multicast protocols—local recovery and retransmission by multicast. Grauss-glauser proposed *receiver collaboration* in [38], which requires the receivers to buffer correctly received data so that they may respond to each other's NACKs. For example, RMTP uses designated members to perform retransmissions, and retransmissions are directed towards the downstream subtrees. In the Scalable Reliable Multicast protocol (SRM) [34], as another example, retransmissions are multicasted

by receivers. The retransmissions are scheduled according to the distance estimations to NACK requester, in order to avoid duplication.

Another recovery approach is the Forward Error Correction (FEC) scheme [70, 62]. In this approach, data is coded with redundant information, so that receivers can reconstruct data without requesting retransmissions. When retransmission is necessary, coded retransmissions can be used by receivers to compute their own lost packets, which may differ from each other.

In summary, sender-initiated multicast protocols often have to use structure-based approaches to avoid ACK implosion, while receiver-initiated protocols are mostly NACK-based. Moreover, NACK suppression and local recovery are also widely adopted in reliable multicast protocols.

1.1.2.3 Congestion Control: As another key issue in reliable multicasting [56], congestion control is also difficult. Intensive research has been focusing on various mechanisms for feedback control. However, congestion and flow control are restricted by these mechanisms. It is often hard for a sender to gather sufficient information to enforce effective congestion/flow control. Sender-initiated protocols are usually window-based, such as RMTP. However, because of the use of feedback aggregation and hierarchical structure, a sender cannot estimate network conditions such as the round trip time (RTT), and the loss status at receivers.

For receiver-initiated protocols, such as SRM, flow control is usually open-loop and rate-based, and is also incapable to perform effective congestion control.

As congestion control is one of our major topics in this dissertation, we will further discuss this issue in next section and also in later chapters.

1.1.2.4 Other Issues: More issues are identified and discussed in recent works. The multicast loss correlation is observed by Yajnik in [86]. It is observed that losses

during a multicast session often occur on leaf links. In [10], the loss path multiplicity problem was presented—multicast loss probability increases with multicast scale even when the probability is low on individual paths.

Heterogeneity is another issue. In [36], Golestani proposed to use a differentiated window mechanism for heterogeneous receivers. Also, Chawathe et al. presented a Reliable Multicast proXies (RMX) architecture [17, 16] as a solution to reliable multicast for heterogeneous networks.

1.2 Dissertation Overview

1.2.1 Motivations

Wide-area multicast is still difficult to deploy, as mentioned in [17, 16], one of the reasons is the difficulty to control and manage multicast traffic.

Flow-control schemes can be classified as either an *open-loop* scheme or a *closed-loop* scheme. In open-loop flow control, an agreement on the sender's traffic behavior is established among senders, receivers and transmitting networks. While in closed-loop flow control, senders dynamically adapt to network and receivers' condition. Feedback from receivers to senders is the critical issue.

Window protocol is an effective closed-loop flow-control scheme for reliable unicast protocols, such as the Transmission Control Protocol (TCP) [66]. Basically a window protocol requires a sender to watch feedbacks from receivers when it sends a quota of data. Without feedbacks from receivers to signal for further transmission, the sender can not send more data. Therefore, by employing a window protocol, a sender can: (*i*) control the amount of data going into the network; (*ii*) detect losses and therefore initiate of congestion control and error recovery. Basically, a window protocol has to be ACK-based.

A closed-loop flow-control scheme is difficult to apply to reliable group multicast. In a multicast session, senders usually cannot afford to manage group

membership, therefore it is hard to establish commitment between a sender and a receiver. Lack of positive acknowledgments makes a sender incapable to detect losses in a timely fashion. With NACK-avoidance strategy, it is impossible for a sender to learn who has failed receiving its data. The local recovery strategy further keeps senders from getting feedback information, such as receiving statuses of receivers and network congestion status. Therefore, most current dominant reliable multicast protocols are rate-based.

Open-loop flow control is a popular strategy for multicast. For example, the *wb* application [34] employs a token bucket to regulate the peak rate with which one may send to the network. PGM also suggests use of token bucket for flow control [74]. However, open-loop control needs resource reservation, which could bring serious scalability concerns for large-scale multicast sessions. Unlike the telecommunication network, where a session cannot access the network without first acquiring sufficient resource allocation, the datagram network allows competition for network resources among sessions. Without effective dynamic congestion and flow control, multiple multicast sessions, or even a single many-to-many multicast session, can cause network congestion. On the other hand, if we enforce conservative and rigid traffic shaping, the network can be under-utilized.

The work by Ofek and Yener [63] (referred to as the OY protocol) provides an alternative solution. Instead of using the tree structure, the OY protocol organizes multicast members into a virtual ring. The essential change is: concurrent ACKs are no longer necessary. With the ring structure, all multicast packets can return to their sources when they finish visiting all multicast members. The protocol solves the ACK implosion problem by regarding returning packets as implicit ACKs. Window-based control thus can be employed with these feedbacks. The most obvious drawback of this mechanism is the latency. As packets are received sequentially, the delay can be large when ring size is large. Again, scalability is the key difficulty.

Therefore, it is necessary to investigate the performance of OY protocol. We survey the OY protocol performance by comparing it with a tree-based approach [18, 20]. The comparative study enables us to further understand the merits and weaknesses of the two different approaches, especially what is missing in the prevalent tree-based approach for reliable multicast, and what can be done to improve the performance of reliable multicast.

1.2.2 Dissertation Outline

The purpose of the dissertation is to investigate better architectural and protocol solutions for reliable group multicasting. One major effort in the following chapters is to find out the important factors that affects reliable multicast performance. With this objective, we study different feedback mechanisms with respect to different multicast routing structures. Specifically, there are mainly two approaches in our scope: a NACK-based approach with a shared multicast tree as the dissemination structure, and an ACK-based approach with a multicast ring structure. The study of these two paradigms is comparative.

In the next chapter, we present our study of the costs of building multicast trees and rings on a network graph. By comparing the routing costs needed for trees and rings, we would like to investigate the performance of ring-based routing relative to tree-based approach. The results from the cost comparison shows that a multicast ring necessitates more network links than a multicast tree, although of the same order. Considering the propagation delay is logarithmic with tree size but linear with ring size, ring topology does not show an advantage over tree topology for network-layer multicast routing.

To further investigate the impact of routing topologies on reliable multicast performances, we compare two generic protocols in Chapter 3. One is a generic NACK-based transport protocol modeled after the SRM protocol, which is tree-

based. The other one is a generic window-based protocol emulating the OY protocol. We establish an event-driven simulation system to emulate these multicast protocols. Performance measures of the two protocols, such as multicast delay and multicast throughput, are observed and compared. The major conclusion from the study in this chapter is that ring topology significantly benefits the upper-layer reliable multicast protocol by providing implicit feedbacks to multicast sources. Multicast traffic on a multicast ring can be regulated more easily, and therefore, the risk of network congestion due to excessive multicast traffic can be significantly reduced.

In Chapter 4, we propose a hierarchical and hybrid architecture RITA (Rings Interconnected with Tree Architecture). This new architecture is designed based on the study of the previous chapters. Multicast rings and trees are both used in this architecture, but in different hierarchies. Transport-layer functions are specifically designed for different entities in this hybrid architecture. Reliable multicast throughput is maintained with the use of ring topologies and a window-based protocol at sources. Meanwhile, use of a tree structure in the backbone helps to reduce multicast latency. Simulations are conducted and performances are compared with the original two approaches (i.e. reliable multicast using tree or ring only). Results are satisfying, because RITA achieves similar throughput performance as the ring-based protocol, but with significantly lowered delay.

Effective congestion control improves multicast reliability performance by reducing the packet-loss probability. Although flow and congestion control is traditionally done with transport-layer protocols, proper routing at the network layer can help to reduce the risk of network congestion. In Chapter 5, we discuss the multicast tree packing problem. The central idea is to avoid competition for the same network resources between multiple concurrent multicast sessions. Proper re-routing of multicast trees helps to reduce congestion significantly. We show a lower bound computation for multicast tree packing. Also, simulations using a multicast

tree packing algorithm show congestion can be reduced, at the cost of increasing multicast tree sizes within a certain degree.

Finally, we conclude our dissertation in Chapter 6. Open problems and a summary of our contributions are provided at the end of this chapter.

CHAPTER 2

GROUP MULTICAST ROUTING WITH TREE AND RING: NETWORK-LAYER ISSUES

The essential task at the network layer is routing. A network can be represented as a graph $G = (V, E)$, where V is the set of network switches and E is the set of network links between switches. The multicast routing can be viewed as finding a subgraph $S = (N, L)$ for a given node set $M \subseteq V$, such that $M \subseteq N \subseteq V$ and $L \subseteq E$ is the set of links connecting all nodes in N. From this view, the *cost* of a multicast subgraph is the number of links used, i.e. $|L|$. The challenge to multicast routing is to find a connected subgraph S^* with minimum cost. Obviously, such a minimum multicast subgraph should be in the form of a tree. Therefore, a ring structure is not the optimum subgraph. However the question is, what is its cost difference versus a tree structure?

The graph cost (i.e. hop count) is important in order to estimate the actual network cost of a routing protocol. It indicates the utilization of other resources. Usually the larger the number of links used by a route, the more routers are involved and hence more use of resources from the network.

In this chapter, we first introduce related work in Section 2.1. Our approach to compare ring and tree subgraphs are based on two heuristic algorithms presented in [18]. These two algorithms are recapitulated in Section 2.2. In Section 2.3, we present the comparison based on results that are from simulations with the heuristic algorithms. Section 2.4 presents a brief summary of this chapter.

2.1 Related Work

In general, searching for a minimum-cost multicast tree is known as the Steiner Tree problem, which is an NP-complete problem. Existing protocols build near-optimal multicast trees based on two strategies: either low delay or low cost.

To connect a source node to a destination node, the shortest path between them is the most effective routing because the number of links (and also the switches visited) is minimal. Therefore, to connect a source node to multiple destination nodes, it is reasonable to use a source-based tree that is formed by shortest paths from that source. In fact, this strategy motivated early work on multicast routing [11, 23, 9, 24, 77, 60].

With shortest paths connected from source to destinations, propagation delay is shortest across a source-based tree. However, source-based trees are not necessarily low cost. Motivated by [79], center-based tree solutions were proposed to decrease cost [8, 7, 5, 6, 25, 26, 32]. Center-based trees are efficient especially when there are many senders in a multicast group (i.e., many-to-many multicast). A single tree can be shared by sources within a same group, instead of using multiple source-based trees.

To connect multicast members into a ring topology, as proposed in [63], the underlying problem is actually a variation of the Traveling Salesman problem, which also is NP-complete. In practice, such a ring solution may not exist for an arbitrary topology network. An alternative is to embed an Euler tour on links to form the multicast ring.

2.2 Heuristic Ring and Tree Construction Algorithms

The Steiner Tree problem and the Traveling Salesman problem (TSP) have attracted some attention in the integer-programming community ([21, 22, 35, 40, 39]). In [18], integer-programming algorithms are presented to investigate the lower bound of the cost difference of Steiner tree and TSP ring on a same network. In this thesis, we introduce the heuristic approaches to study relatively large networks that are difficult to obtain optimal trees or rings.

Algorithm: TreeHeu

Input: $G = (V, E)$, and $M \subseteq V$
Output: Multicast Tree (V_M, E_M), s.t. $M \subseteq V_M \subseteq V, E_M \subseteq E$

BEGIN
1. Construct a complete graph of $G' = (M, E')$, such that:
   E'=$\{(i,j)$: $i \neq j$, and $i, j \in M\}$ and $w_{(i,j)} = d_{ij}$.
2. Find a minimum spanning tree T_1 on G'
3. Let $V_M = M$, $E_M = \emptyset$. For each link $l = (i,j)$ in T_1, do:
   3.1 Find the corresponding path p on G.
   3.2 For every node x in p, if $x \notin V_M$, let $V_M = V_M \cup \{x\}$.
   3.3 For every edge e in p,
       let $E_M = E_M \cup \{e\}$ if $e \notin E_M$.

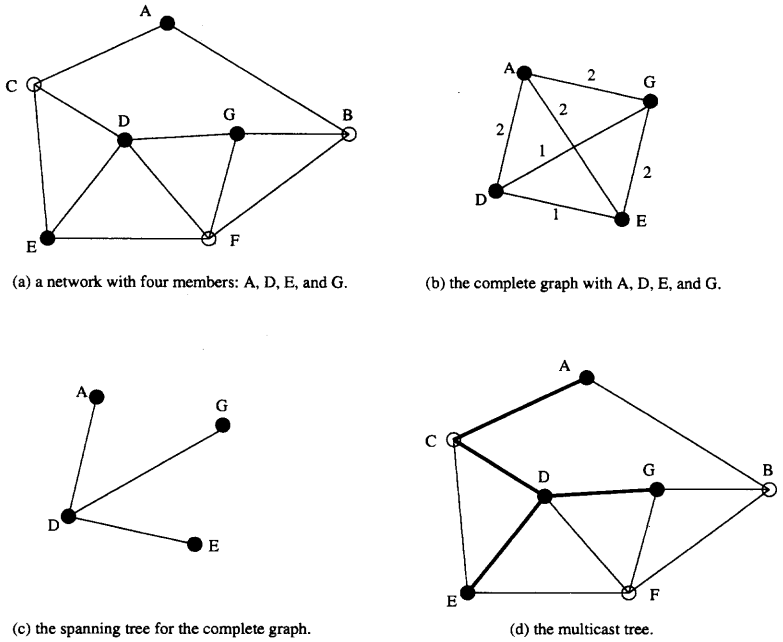
END

Figure 2.1 A heuristic algorithm for near-optimal multicast tree.

2.2.1 Heuristic Multicast Tree Construction

The heuristic, shown in Figure 2.1, enhances the distance network heuristic [48, 45] with an additional refinement step to optimize the embedding of the tree. There are two major stages: (i) generation of a virtual tree, which is formed with multicast member nodes, and virtual links that are weighted according to the distance between the corresponding member nodes. and (ii) embedding the virtual tree onto the actual graph.

In the first stage (as in the algorithm, step 1 and 2), a complete graph of the nodes in M is constructed. Each edge is associated with a cost which is the shortest path distance between its end points. Next, a minimum spanning tree of the complete graph is obtained. The cost of the virtual tree provides an approximation to the optimal solution. This tree is a virtual tree since its links may correspond to paths in the physical graph. Furthermore there can be multiple paths with the same cost between the end points of a virtual link.



(a) a network with four members: A, D, E, and G.

(b) the complete graph with A, D, E, and G.

(c) the spanning tree for the complete graph.

(d) the multicast tree.

Figure 2.2 An example of heuristic multicast tree construction.

In the second stage (step 3), the corresponding paths are selected for each link of the tree. We call this step the *embedding* of the virtual tree onto the actual network graph.

Figure 2.2 shows an example of heuristic multicast tree construction. In this example, there are four member nodes: A, D, E and G. We first build a complete graph with these member nodes only. Next the minimum spanning tree of the complete graph is computed. Finally, the multicast tree is obtained by embedding the minimum spanning tree onto the original graph (shown as the thick lines in the graph).

The time complexity of the first step is $O(|V|^3)$ because we may use all pair shortest path algorithms to construct the complete graph. The second step can be computed in $O(|M|^2)$ steps. In step 3, there are actually $|M|-1$ repetitions, as step 3.2 and step 3.3 are of $O(1)$, the complexity depends on step 3.1. If the physical paths are embedded according to the outcome of a shortest-path computation, step

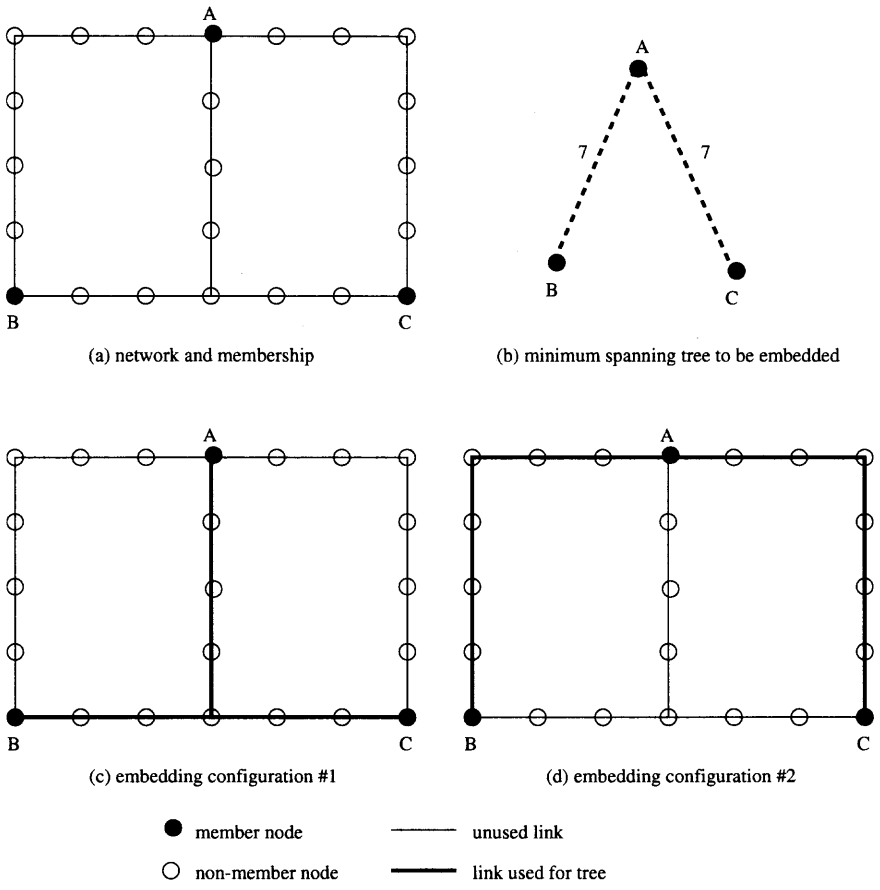


Figure 2.3 Alternative embeddings for the same minimum spanning tree.

3.1 can be done in $O(|V|)$ steps. Therefore step 3 is of $O(|M| \cdot |V|)$, and the overall time complexity of the algorithm is $O(|V|^3)$.

Note that different embeddings are possible for the same cost. Consider the example in Figure 2.3 and assume that the minimum spanning tree found with the heuristic in Figure 2.1 is (A,B), (A,C) where A, B, C are the multicast members. The tree has total cost 7+7= 14 since the corresponding paths from A to B and A to C have 7 hops each. In this example we show two possible embeddings of the tree. Each embedding preserves the cost of the heuristic tree. However, the embedding in Figure 2.3 (c) is the better one since: (i) it uses fewer links, and (ii) its diameter is smaller, which is important for propagation delay. Thus, a heuristic rule is needed to collapse the common paths (i.e., *selective-merge*).

This is the actual difficulty in Steiner tree problem. For an arbitrary network topology, the problem of finding the optimal embedding becomes intractable. In regular topology networks, it is possible to use the topology regularity of the network and find a heuristic method that can choose a low-cost embedding.

For example, in a grid network, we can label a node by its coordinates and try merging the paths. The merge can be done such that if a path from (x_1, y_1) to (x_2, y_2) is to be embedded, then it will traverse only nodes (x, y) with $min(x_1, x_2) \leq x \leq max(x_1, x_2)$ and $min(y_1, y_2) \leq y \leq max(y_1, y_2)$.

In the following we present an algorithm for an $n \times n$ grid network (see Figure 2.4).

In a grid network, let us denote a node with coordinates (x, y) as v_{xy}, and a virtual path from node v_{ij} to $v_{l,m}$ as (v_{ij}, v_{lm}). A path v_{ij} to $v_{l,m}$ has a unique physical embedding if and only if the virtual path (v_{ij}, v_{lm}) satisfies either $i = l$ or $j = m$. Such virtual paths are either horizontal or vertical in the Euclidean space. For clarity, let us call such paths as VUE (Virtual path with Unique Embedding) paths. For other paths, multiple physical embeddings exist.

Definition 1 *If a virtual path (v_{ij}, v_{mn}) has multiple physical paths that are of the same shortest-path distance, a node v which is traversed by any of these physical paths is called a* candidate transient node*(CTN).*

Consider a virtual edge in a grid network: (v_{ij}, v_{lm}), where $i \neq l$ and $j \neq m$. Any node in set $C(i, j, l, m) = \{v_{xy} : x \in [min(i, l), max(i, l)], y \in [min(j, m), max(j, m)], (x, y) \neq (i, j)$ and $(x, y) \neq (l, m)\}$ is a CTN. The number of CTN's associated with this virtual edge is in fact the cardinality of $C(i, j, l, m)$: $(|l - i| + 1)(|m - j| + 1) - 2$.

The algorithm uses an array *Flag* to record the maximum possible number of paths that traverse a node. From step 1 to 4, initializations are done and the

Algorithm: TreeEmbedding

Input:

A grid network $G = (V, E)$ of $n \times n$:
$V = \{v_{i,j} : i, j = 1, \ldots, n\}$, $E = \{(v_{ij}, v_{lm}) : i, j, l, m = 1, \ldots, n\}$.
Multicast nodes $M \subseteq V$,
A spanning tree T_v= $\{p(v_{ij}, v_{lm})$: $p(v_{ij}, v_{lm})$ is an edge of the minimum spanning tree for the complete graph G' in ''TreeHeu'' step 1$\}$.

Output: A set of multicast tree edges E_M which embed T_v on G.

BEGIN

1. Define two arrays of $n \times n$: $Flag$ and $Visited$, associate with each node on the graph.
   For each node $v_{ij} \in V$, do the following initialization:
   1.1) $Flag[i][j] = \begin{cases} n+1 & \text{if } v_{ij} \in M; \\ 0 & \text{otherwise.} \end{cases}$
   1.2) $Visited[i][j] = 0$.
2. For each arbitrary pair of nodes (v_{ab}, v_{cd}), define:
   2.1) $Total\_Flags(a, b, c, d) = \sum_{x=min(a,c)}^{max(a,c)} \sum_{y=min(b,d)}^{max(b,d)} Flag[x][y]$
   2.2) $Total\_Visited(a, b, c, d) = \sum_{x=min(a,c)}^{max(a,c)} \sum_{y=min(b,d)}^{max(b,d)} Visited[x][y]$
3. Let $E_M = \emptyset$.
4. For each virtual edge $p(v_{ij}, v_{lm}) \in T_v$, do:
   $\forall v_{xy} \in C(i, j, l, m)$, increase $Flag[x][y]$ by one.
5. Let $E_S = \{p : p$ is a VUE path in $T_v\}$.
   5.1) For any $p \in E_S$, identify its unique physical path.
   5.2) Include into E_M those edges which form the physical path.
   5.3) Increase the $Visited$ value of each node on the physical path by one.
6. For any virtual edge $p(v_{ij}, v_{lm}) \in T_v \backslash E_S$, let $x = i$ and $y = j$,
   6.1) do the following until ($x == l$ OR $y == m$):
   - Identify a neighbor of v_{xy} in $C(x, y, l, m)$, v_{uv}, with maximum $Total\_Flags$ value. If two neighboring nodes have the same $Total\_Flags$ value, pick the one with greater $Visited$ value.
   - Let $E_M = E_M \cup \{(v_{xy}, v_{uv})\}$.
   - Increase $Visited[x][y]$ by one, and let $x = u$, $y = v$.

   6.2) Include into E_M all physical edges that form the VUE path (v_{xy}, v_{lm}). Increase $Visited$ values of each node on these edges.
   6.3) For all unvisited nodes in $C(i, j, l, m)$, decrease their $Flag$ values by one.

END

Figure 2.4 Improved tree embedding on a grid network.

Flag array is initialized according to the paths found on the virtual complete graph. Step 5 maps the VUE paths (horizontal or vertical) because these paths have unique embeddings and must belong to the multicast tree. In step 6, we find paths between any pair of nodes that lie on a diagonal corner. A physical path can be found along those links that pass nodes with maximum total *Flag* values. This implies that this path will be shared effectively by the final multicast tree. Once a path is found, other candidate nodes in the corresponding diagonal region are updated by decreasing the corresponding flag value, in order to guide the next mapping correctly. The *Visited* array in the algorithm records the actual number of times a node is traversed by physical embeddings. It is used to break a tie of multiple paths having the same total *Flag* value. For example in Figure 2.3(a), the candidate path for the virtual link (A,B) has weight 10 while in 2.3(b) the path weight is 7. Hence, the heuristic will choose the former.

In this heuristic, both step 1 and 2 can be computed with $O(n^2)$ (i.e., $O(|V|)$ since $|V| = n^2$) steps. As there are $|M|-1$ virtual edges, step 4 is of $O(|M|n^2)$. Step 5 is $O(|M|n)$ since the length of VUE paths is n in maximum. Step 6.1 takes $O(n^3)$ steps, therefore step 6 can be done with $O(|M|n^3)$ because the other two sub-steps are $O(1)$. Hence this heuristic has time complexity $O(M|V|^{\frac{3}{2}})$.

2.2.2 Heuristic Multicast Ring Construction

The multicast ring construction aims to find a minimum-cost tour which traverses a subset of vertices on a graph. This is in fact equivalent to the Prize Collecting Traveling Salesman Problem, which is NP-complete. Our study is based on the assumption that the network graph does contain such a multicast ring. In the performance evaluation in the next section, the simulated networks are generated with at least one ring that traverses all network nodes.

Algorithm: RingHeu

Input: $G = (V, E)$, and $M \subseteq V$
Output: Multicast Ring (V_M, E_M), s.t. $M \subseteq V_M \subseteq V, E_M \subseteq E$

BEGIN
1. Construct a complete graph of (M, E'), such that:
   E'=$\{(i,j)$: $i \neq j$, and $i, j \in M\}$ and $w_{(i,j)} = d_{ij}$.
2. Apply two-opt algorithm to find a suboptimal solution:
   $R_1 = (M, T_1)$ of (M, E').
3. Let $V_M = M$, $E_M = \emptyset$. For each virtual link $(i,j) \in T_1$, do:
   3.1 Find the corresponding path p on G.
   3.2 For every node x listed in p, if $x \notin V_M$, let $V_M = V_M \cup \{x\}$.
   3.3 For every edge e in p, if $e \notin E_M$,
   let $E_M = E_M \cup \{e\}$.

END

Figure 2.5 A heuristic algorithm for a near-optimal multicast ring.

Like the multicast tree construction algorithm, the heuristic algorithm for multicast ring construction, "RingHeu" (shown in Figure 2.5), has also two phases. In the first phase, we generate a virtual ring, which is formed with multicast member nodes, and virtual links that are weighted according to the distance between the corresponding member nodes. In step 1 we construct a complete graph with only nodes in M. With all-pair shortest-path algorithm, this step is $O(|V|^3)$ in time. In step 2, on the complete graph, we apply the two-opt algorithm [55] to find a heuristic TSP tour, which induces a virtual ring such that each virtual link may correspond to a path in the physical network. The algorithm basically follows the following steps: (i) find an arbitary tour [1] on the complete graph, compute the total distance of the ring; (ii) swap any pair of the nodes if the ring length can be reduced, until no further improvement can be done; (iii) repeat the previous steps t times and select the minimum length ring R_1. The time complexity is $O(|M|^3)$ for this algorithm.

In the second phase, we embed the virtual ring onto the given graph. There are $|M|$ virtual links to be embedded. Hence step 3 takes $O(|M||V|)$ steps. Therefore

[1] Notice that any permutation of the vertices of the complete graph forms a valid tour.

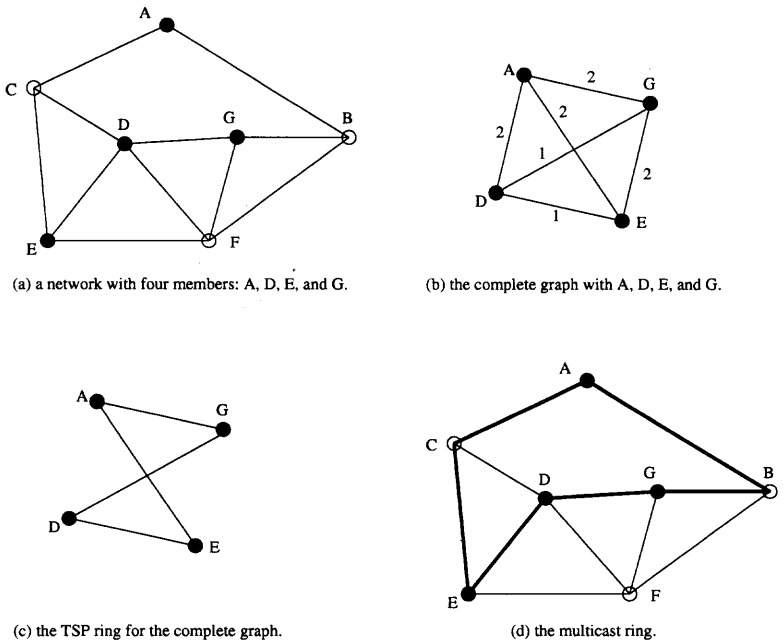


Figure 2.6 An example of heuristic multicast ring construction.

the time complexity of the whole procedure is $O(|V|^3)$. Embedding of virtual links is not trivial since mapping virtual links to corresponding paths may result in visiting some nodes more than once. Thus, the solution may not be a simple cycle. In the multicast context, this means that the same multicast packet will reach a node more than once, which is not acceptable. In order to address this problem, we first record at each node the first path that visits this node. Then we avoid any new path which visits this node again. However, if a conflict occurs between two paths such that they both must traverse the same node, then one of the paths is recomputed. Simulations in Section 2.3 show that this naive approach is quite efficient.

Figure 2.6 shows an example of heuristic multicast ring construction. The network and membership is the same as in Figure 2.2. We first build a complete graph with these member nodes only. Next the TSP ring of the complete graph is computed. Finally, the multicast ring is obtained by embedding the TSP ring on the complete graph onto the original graph.

2.3 Performance Study

2.3.1 Generating Multicast Groups and Networks

Most of our performance studies in this dissertation are conducted on a similar network environment. Usually we specify a network size (i.e. the number of nodes) first, and the network connections are generated randomly according to the network size. With these parameters defined, random network instances are generated for different simulation runs. Statistical results are then collected for further analysis.

To generate a random network of N nodes, we use a variable L to control the number of links in the network. Basically, there are N links connecting the N nodes into a simple circular path. Then LN different pairs of nodes are randomly (uniformly) picked to be connected with a link. Thus, the number of links in the network is in fact $(L+1)N$. The network generated is symmetric (i.e., each direction of a bidirectional link has same cost) for each link.

A multicast group is constructed randomly such that M out of the N nodes are chosen uniformly. In most of our simulations, $M = 2\sqrt{N}$. If K groups are to be simulated, we repeat the selection K times independently.

2.3.2 Performance of Multicast Tree Construction Heuristics

We first studied the performance of our heuristic multicast tree construction algorithm *"TreeHeu"*. We investigated two configurations with regard to different L values. To evaluate the algorithm, an Integer Programming (IP) algorithm was formulated to find the optimum tree solutions [18] for the same set of network instances.

Figure 2.7 plots the results from simulations with four different network sizes: $N = 32, 64, 128$ and 256 nodes. L is chosen to be 2 so that the number of links is $3N$. When networks are large (with 128 and 256 nodes), the IP algorithm can only determine an upper bound and a lower bound for optimum tree solutions. Therefore,

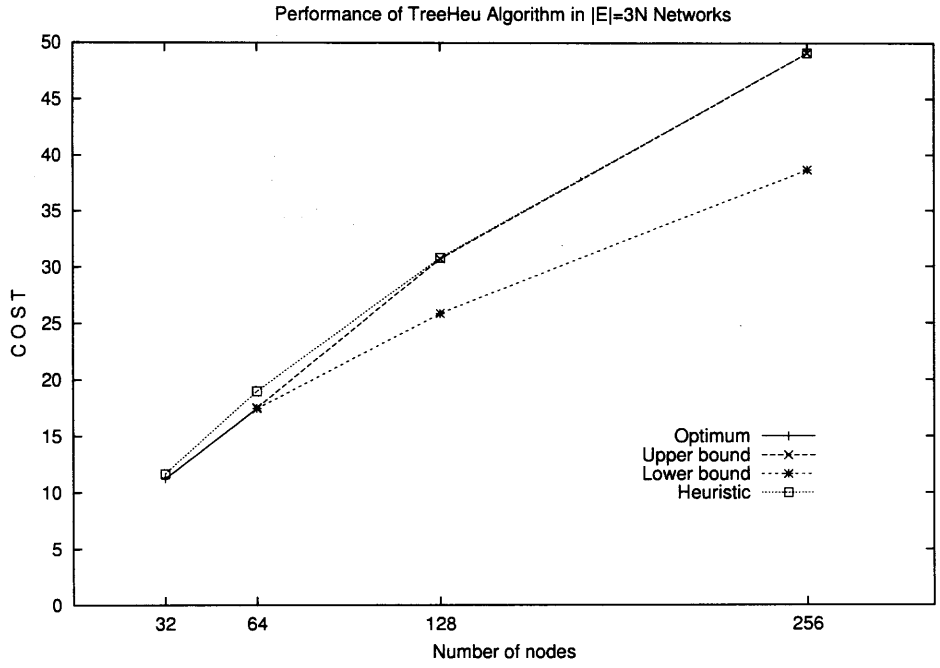


Figure 2.7 Performance of TreeHeu algorithm in $|E| = 3N$ networks.

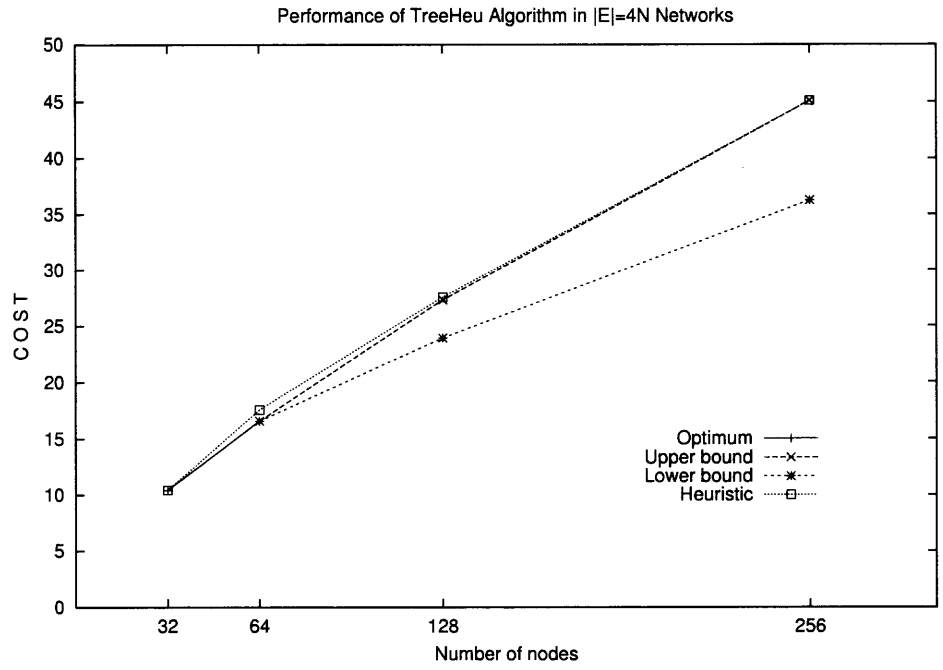


Figure 2.8 Performance of TreeHeu algorithm in $|E| = 4N$ networks.

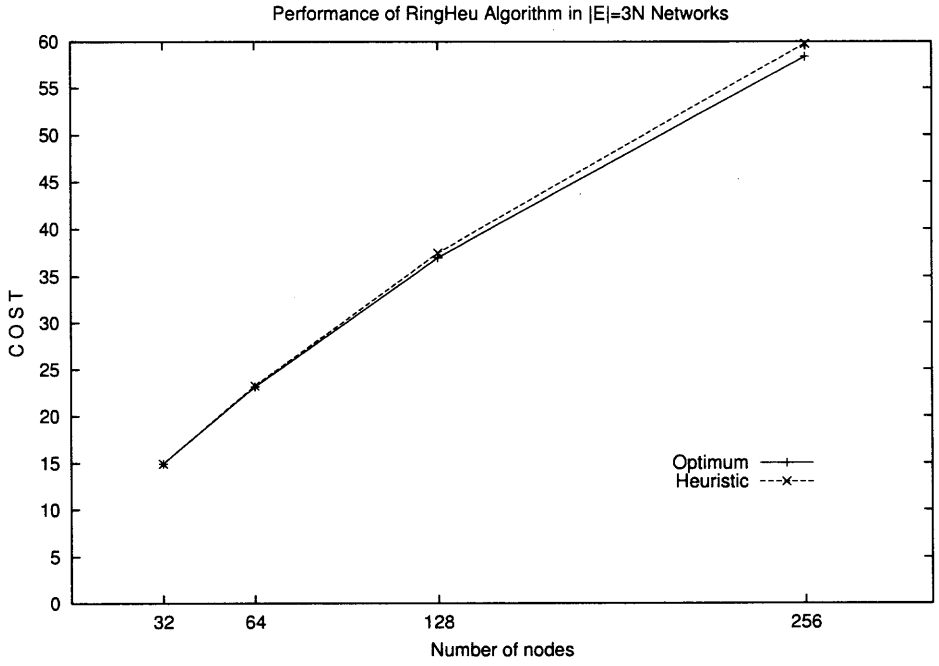


Figure 2.9 Performance of RingHeu algorithm in $|E| = 3N$ networks.

in the plotting, the "Optimum", "Upper bound" and "Lower bound" curves are results from the IP algorithm, while the "Heuristic" curve is from the results of "TreeHeu". We can tell from the figure that when the network is small, the heuristic algorithm is effective in finding near-optimum solutions. When the network is large, the difference from the optimum solution may increase.

Figure 2.8 also plots the case with $L = 3$. We find that when the connectivity increases in a network (i.e. network are connected with more links), the cost of the multicast tree slightly decreases as compared to Figure 2.7. As for the performance comparison, we see that the two figures are consistent.

2.3.3 Performance of Multicast Ring Construction Heuristics

Similar to the above tree heuristic simulation, we apply the ring heuristic to the same network instances that are used for the tree. Two configurations with different L values are again simulated, and here, another algorithm formulated with the Integer

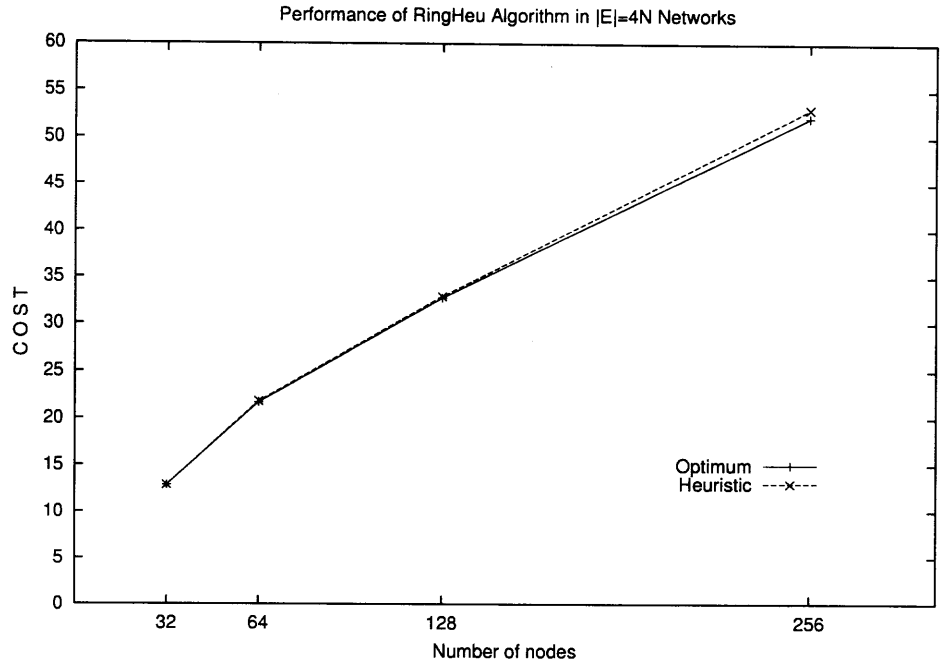


Figure 2.10 Performance of RingHeu algorithm in $|E| = 4N$ networks.

Programming approach (IP) [18] is used to obtain the optimum ring solutions for heuristic evaluation.

Figure 2.9 plots the results from simulations with four different network sizes: $N = 32, 64, 128$ and 256 nodes. L is chosen to be 2 so that the number of links is $3N$. In the plotting, "Optimum" curve is the results from the IP algorithm, while "Heuristic" curve is from the results of "RingHeu". As we can see, the heuristic algorithm results are very close to the optimum ones.

Figure 2.10 also plots the case with $L = 3$. Similar to the tree construction case, when the connectivity increases in a network, the cost of multicast ring decreases. Also, results from "RingHeu" are near-optimal.

2.3.4 Cost Comparison of Multicast Tree and Ring

With the same set of network instances, we examine the two approaches in terms of network cost. To compare the cost difference, we define a *ROT ratio* as below:

Table 2.1 Mean cost ratio (ring/tree) for 4 configurations of random networks.

number of nodes	$\|E\| = 3N$ networks		$\|E\| = 4N$ networks	
	mean	95% CI	mean	95% CI
32	1.28	(1.23, 1.33)	1.23	(1.20, 1.26)
64	1.23	(1.19, 1.26)	1.24	(1.21, 1.27)
128	1.22	(1.19, 1.25)	1.19	(1.16, 1.22)
256	1.22	(1.20, 1.23)	1.18	(1.16, 1.19)

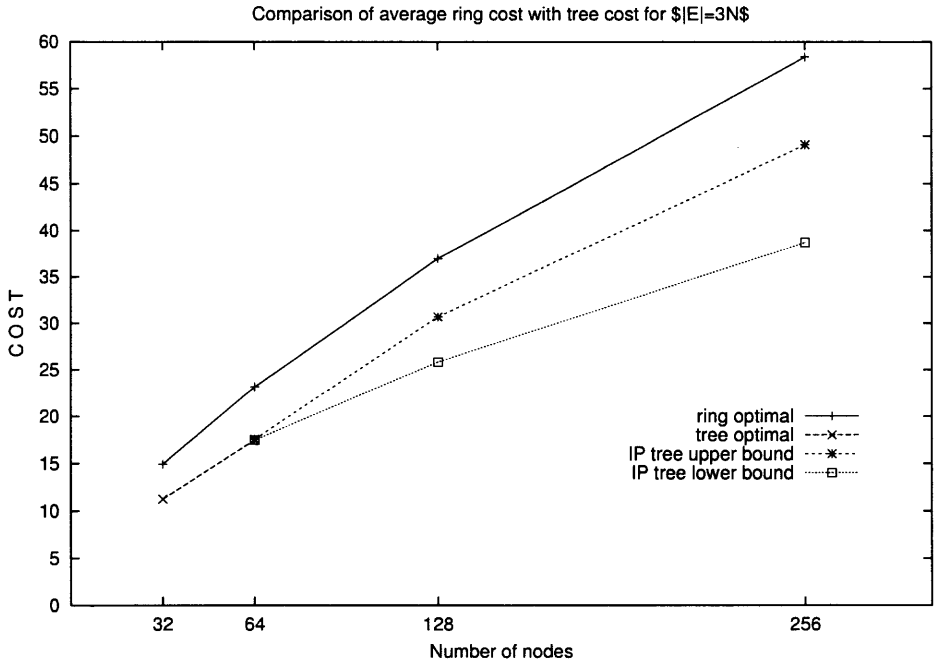


Figure 2.11 Comparison of ring and tree heuristics for $|E| = 3N$.

Definition 2 *ROT ratio is the ratio of multicast ring size over multicast tree size for a given multicast group M in a network $G = (V, E)$, where $M \subseteq V$.*

Table 2.1 shows the ROT ratios from our simulations. The cost statistics are results from heuristic simulations in Section 2.3.2 and 2.3.3. The IP solutions are plotted in Figure 2.11 for optimum ring and tree constructions in $|E| = 3N$ networks. When networks are large, the Integer Programming (IP) algorithm cannot determine the optimum tree solutions, therefore the upper and lower bounds are plotted instead in the figure. Solutions for $|E| = 4N$ networks are similar, thus we do not include the plotting here.

Comparisons indicates that: (i) Ring solutions, in the cases where connectivity is 3 or 4, cost about 1.20-1.30 times as the cost of tree solutions. The cost ratio is bounded in the interval $(1, 2]$.[2] (ii) The ratios of ring to tree for $|E| = 4N$ networks are generally smaller than for $|E| = 3N$ ones, which implies that networks with more link connections may have better ring solutions. In fact when the graph is fully connected, the size of multicast tree is $|M| - 1$ and ring size should be $|M|$. In another extreme case, there can be no TSP ring solution for a given graph and group, multicast ring can be constructed by embedding an Euler tour onto the multicast tree. Therefore in this case, the ROT ratio is 2. Thus the ROT ratio generally is in the range of $(1, 2]$. (iii) The ratios of ring to tree decrease as the total number of nodes in the network increases. This observation might result from the fact that the multicast group size we chose is relatively less dense in larger networks than in smaller networks, which means, as in the previous observation, that relatively more links are available to use.

2.4 Summary

The two heuristic algorithms have near-optimal performance when their results are compared to integer programming results. With the simulation results, we are able to investigate the performance comparisons between ring and tree-based routing approach in terms of network cost. We find that construction of a multicast ring usually uses more network links than a multicast tree solution. The cost ratio of ring over tree is about 1.20 to 1.30. One issue to notice is that ring-based multicast routing usually is uni-directional, while for tree-based group multicast, multicast traffic may flow in both directions of a tree link.

Generally, we may conclude that multicast ring does not show advantages over tree when considering multicast routing only.

---

[2]When n is the cost of a multicast tree, a ring costs $(n + 1)$ at least and $2n$ at most.

CHAPTER 3

TREE VS. RING IN MULTICAST RELIABILITY: TRANSPORT LAYER ISSUES

This chapter presents a performance study in order to identify some tradeoffs between tree-based and ring-based reliable group multicast protocols. The scope of our study has two major focuses: reliable end-to-end transport and group (i.e. many-to-many) multicast.

We see from the previous chapter that a ring-based multicast does not show any cost advantage over the tree-based approach. We may also observe that for the tree-based approach, the propagation delay is $O(\log(N))$, where N is the tree size. In contrast, propagation delay on a ring is $O(N)$. However, the ultimate end-to-end performance is not only determined by these static factors.

The essential requirement of reliability is that a packet can be received by all group members. A successful multicast happens when every receiver receives a copy of a same packet. When a packet is dropped in network, transport protocol should detect the loss and perform retransmission. Throughput can be significantly affected, and the delay caused by these operations are much longer than the static propagation delay. With necessary control messages exchanged to ensure reliability, protocol overhead increases and efficiency decreases.

The key criteria for evaluating the performance of a reliable group multicast protocol are: (*i*) reliable multicast delay, which we define as the time period between the time when a multicast packet is first sent by a transport protocol and the time when the packet eventually is received by the last receiver in the group. (*ii*) reliable multicast throughput, which is the mean number of *successful multicasts* in a unit of time, a.k.a. "goodput". (*iii*) efficiency, namely, the ratio between the number of successful transmissions and the total number of packets that were transmitted.

These are the major issues we investigate in this chapter. By asking what are the delay, goodput and efficiency performances for both tree- and ring-based protocols, we actually aim to learn the importance of closed-loop flow control for reliable group multicast.

Below we first discuss important related work in Section 3.1. In particular, SRM[34] and the OY protocol [63] are the two major motivating works. In Section 3.2, we setup two generic protocol models: *Tree Based Reliable Multicast* (TBRM) protocol, and *Ring Based Reliable Multicast* (RBRM) protocol. In Section 3.3, the performance model is presented. Our simulation results are presented in Section 3.4. Finally, we summarize this chapter in Section 3.5.

3.1 Related Work

As introduced in Chapter 1, scalability issues, such as the ACK implosion problem and the difficulty of maintaining member state information, make reliable transport protocol hard to adopt a TCP-like window-based control mechanism. RMTP[71] is a window-based one-to-many reliable protocol. It organizes multicast receivers into multiple hierarchies. Designated receivers are assigned at each hierarchy to collect ACKs from lower hierarchies and repair packet losses in lower hierarchies. A RMTP sender transmits with a window triggered only by the ACKs from its direct children of the next hierarchy. The ACK implosion is therefore avoided. However, the rigid tree-based hierarchical structure makes it difficult to adapt to the many-to-many reliable multicast environment. When multiple sources are present in a same group, it is costly to maintain multiple RMTP tree structures for different sources.

The representative work for reliable multicast is the SRM protocol [34]. Several important features are included in the SRM framework. (i) It is receiver initiated. Receivers detect packet losses and send NACKs to request repairs. (ii) A NACK avoidance mechanism is implemented. NACKs are multicast to the group, and

receivers perform random back-off to watch for same NACK requests before sending their own NACKs. (iii) With the implementation of a distance-based back-off approach, repairs are multicast from the closest member to suppress others sending the same repair. (iv) SRM is rate-based. There is no explicit control to multicast traffic through feedbacks. The flow control relies on open loop control mechanisms, such as the token bucket control.

The ability of traffic control is a major concern of the SRM protocol. Token bucket regulation is not sufficient. Because it is for group multicast, multiple sources may generate excessive traffic to overload the multicast tree. There is no mechanism for global coordination among sources. Because of its open-loop nature, it cannot dynamically adapt to network condition changes. Other network traffic, such as TCP traffic, may be severely affected. Also, as NACKs and repairs are multicast to the group, although a random back-off mechanism is provided to inhibit this traffic, there is no strict control of this traffic. Notice that one packet loss will result in at least one NACK request and one repair, and we can see traffic may be exponentially increased in a network with heavy loss. Therefore, implosion by NACKs and repairs is likely to happen in a heavily loaded network.

As discussed in Chapter 1, the ring-based approach proposed in [63] has its unique features in multicast traffic control. As traffic flows in the same direction of the multicast ring and eventually returns to sources, the reliable transport protocol on a multicast ring can be relieved from many concerns difficult to handle on a multicast tree. For example, packets multicast on the ring may serve as implicit feedbacks from the network to their senders. By watching a homing packet, the source may quickly learn the congestion status of a network without introducing much network overhead and protocol complexity. Feedback handling is one of the key issues in a reliable multicast protocol design and performance. The ring-based multicasting approach provides excellent structural support for reliable multicast protocols. With benefits

from this built-in feedback of ring structure, a window-based protocol can be enforced at sources in order to control multicast traffic. Consequently, the ring-based reliable multicast protocol may have a short delay for recovery and retransmission, and also better goodput performance. Moreover, the window-based flow-control scheme will provide stable performance under a high network load and avoid what is known as "congestion collapse", where the "goodput" is approaching zero. Therefore, although the ring-based approach may not show its advantage in terms of network link cost and propagation delay, it is worthy to study the overall performance of a ring-based approach when end-to-end reliability is the objective.

Protocols using the rotating token site mechanism [15, 84] are classified as *"ring-based"* by Levine in [49]. However, these protocols are different from the OY protocol because multicast packets are still forwarded on a tree and receivers have to send ACKs. We clarify that "ring-based" protocols in this dissertation refers specifically to protocols similar to the OY protocol, in which packets are delivered to receivers that are arranged on a ring.

3.2 Modeling of Reliable Multicast Protocols

Two protocols are studied in this chapter one is referred to as the *Tree-Based Reliable Multicast* (TBRM) protocol, and the other is referred to as the *Ring-Based Reliable Multicast* (RBRM) protocol. The protocols were modeled in a generic manner. For each of the protocols, only the basic set of functions were included in the corresponding model. This was done in order to gain a better understanding of the basic operational properties of the two protocols.

3.2.1 Tree-Based Reliable Multicast Protocol

The tree-based reliable multicast protocol (TBRM) is assumed to use various underlying multicast routing protocols that build shared multicast trees, and is

outside the scope of this chapter. The TBRM protocol is based on a simplified model of the SRM Protocol [34] and some of its basic features were modeled. For TBRM, we modeled explicit NACKs with NACK avoidance and repairing at nearest neighbors.

TBRM is rate-based without any access-control scheme, which means that packets are multicast to the group after they are generated without considering the state of the network. Senders are not responsible for detection of packet loss. A packet is uniquely identified by its global unique source identification (ID) combined with a sequence number. The receiver detects a packet loss by finding gaps in the packet sequence numbers. When a packet loss is detected, a NACK is scheduled by the detecting receiver. Since we are modeling the network queuing, and the usual estimation of distance by hop count is not the major factor in the network delay, the NACK is scheduled with a delay equal to the delay of the newly received packet. This delay is actually an estimation of the delay to the sender, and is used for NACK avoidance. Also, it is a reflection of the network load. Longer packet delay means current load on the delivery path is higher, thus delaying NACK and repair avoids congestion and is more desirable. NACKs are multicast to the group and NACK avoidance is used in TBRM to reduce the chance of NACK implosion. When a member receives a NACK, it can send a repair packet if it has received the requested packet correctly. Repairs are also scheduled with a delay which equals the delay of the NACK. Repairs are multicast to inhibit other members from sending repairs for same retransmission requests. After forwarding repairs, TBRM ignores NACKs for the repaired data for a period which is three times the delay from the NACK requester.

The TBRM algorithm is shown in Fig. 3.2.1. Step 1 in the algorithm is basically what TBRM does as a sender. It simply sends data packets when they are generated. Step 2 is the basic function of a receiver: receiving data and detecting loss. The

```
/* Tree Based Reliable Multicast Protocol(TBRM) Algorithm */

Variables:
  o: host id (this host).
  s: the last packet sequence number ever sent.
  l_i: the last packet sequence number received from host i.
  d_i: the distance to host i.
BEGIN
  repeat
    1. If new packet generated, then
       (a) let s := s + 1.
       (b) send packet identified as PDU(o, s).
    2. If receive PDU(x, s_x), then
       (a) update d_x.
       (b) find N = {n : l_x < n < s_x}.
       (c) set NACK timer for each NACK packet:  NACK(x, n), n ∈ N.
       (d) let l_x := s_x.
    3. If NACK(x, s_x) arrive, then
       (a) if ∃ NACK timer for NACK(x, s_x),
           backoff the timer and goto loopend.
       (b) if ∃ IGNORE timer for NACK(x, s_x), goto loopend.
       (c) if PDU(x, s_x) was received, then
           (c.1) set REPAIR timer for repair packet REPAIR(x, s_x).
           (c.2) set corresponding IGNORE timer.
       (d) if s_x > l_x, then
           (d.1) find N = {n : l_x < n ≤ s_x}.
           (d.2) for each n ∈ N, if ∄ NACK timer for NACK(x, n),
                 do same as 2.(c) and 2.(d).
    4. If REPAIR(x, s_x) data arrive, then
       (a) if ∃ NACK timer for NACK(x, s_x), kill the timer.
       (b) set(update) IGNORE timer for NACK(x, s_x).
       (c) do same as 3.(d).
    5. If NACK timer expires, send corresponding NACK(x, s_x)
       and back off the timer.
    6. If REPAIR timer expires, send corresponding REPAIR(x, s_x)
       and the timer is killed.
    7. If IGNORE timer expires, the timer is killed.
  end repeat
END
```

Figure 3.1 TBRM algorithm.

remaining parts are what TBRM tries to do for loss recovery while attempting to avoid putting duplicate NACKs and repairs into the network. Loss of a continuous train of packets is hazardous to the network operation. In this case, their NACKs are scheduled to be sent at the same moment, resulting in a bursty load to the network. Attempting to avoid this by detecting losses as early as possible, TBRM tries to find out packet loss from other's NACK messages. If a received NACK packet is requesting for a packet whose loss is undetected yet at this host, the host schedules its own corresponding NACK.

There are no session messages in TBRM. In SRM, session messages are used for the purpose of distance or delay estimation and loss detection of the last packet in a page. In TBRM, we used packet delays for estimations of the distances between multicast members. In our simulation, packets are generated continuously, so packet losses can always be detected.

Because TBRM multicasts data on a tree topology, packet delay should be less than that of ring-based multicasting. However, although it makes great effort to limit the overhead for loss recovery, the protocol does not have guaranteed control on the overhead such as NACKs and repairs. This is not only an efficiency issue. When too many NACKs and repairs are put into the network, the overhead can harmfully congest the network. One purpose of this chapter is to study the effects of some key aspects of protocol overhead.

3.2.2 Ring-Based Reliable Multicast Protocol

In our study, the Ring-Based Reliable Multicast (RBRM) transport protocol, simplified from the protocol presented in [63], assumes that a ring is built by switches. Each switch on the ring should have an incoming link and an outgoing link. And the switch is assumed to be able to decide if a packet from the incoming

```
/* Ring Based Reliable Multicast Protocol (RBRM) Algorithm */
```

Variables:

o: host id (this host).
d: the ring trip delay.
w: multicast transmission window size.
f: the first packet sequence number in the sending window.
s: the last packet sequence number ever sent.
$BACK(x)$: the flag array indicating if packet with seq. number x is back.

BEGIN

repeat

1. If new packet generated then
   (a) keep in buffer.
   (b) if $w > s - f + 1$, send $PDU(o, s+1)$
   (c) let $s := s + 1$.
2. If receive $PDU(o, x)$, then
   (a) set $BACK(x) := TRUE$.
   (b) if $x \neq f$, retransmit all outstanding PDUs before x.
   (c) if $x = f$, find the first $y : f < y < s$, s.t.
   $BACK(y) = FALSE$, and let $f := y$, goto 1.(b).
3. If receive keep alive message KAM, then
   (a) update d.
   (b) resend all outstanding packets that were
   sent before this KAM.
   (c) send new KAM.
4. If KAM timer expires, then send new KAM.
5. If retransmission timer expires, then
   resend corresponding PDU.
6. If receive $PDU(o', x)$ and $o' \neq o$, then receive data.

end repeat

END

Figure 3.2 RBRM algorithm.

link should be forwarded to the outgoing link. When the packet comes back to the switch attached to the packet's origin, it should not be forwarded any more.

The major feature RBRM simulates is actually a window protocol which takes advantage of the feedbacks to multicast sources. Each sender has a predefined sending window size. After sending packets in the sending window, a sender should not send any more packets. Receivers will do nothing except receive data. Packets going back to senders work as acknowledgments (implicit ACKs) to senders and allow senders to advance their sending windows and detect any packet loss. When a packet comes back, a sender will inspect the sequence number to see if it has returned in sequence order. If packets come back in order, the window can be advanced. Otherwise, loss is detected and a retransmission should be sent. A timer is started for each retransmitting packet. If the timer expires before the retransmitted packet comes back, the packet is again retransmitted. The timer value is according to the estimation of the ring delay.

Any packet can be lost once it is put on the network, and, for the case of last packet loss, a keep-alive control message is sent periodically by a sender. Senders record the sending time of each outgoing packet. The keep-alive message is time stamped with its sending time. When a keep-alive message returns, the sender should have received all the packets sent before this keep-alive message. The keep-alive message also serves as an estimator of the ring trip delay. The estimation of the delay is used for retransmission timer control and the pacing of keep-alive messages. A timer is maintained for the outstanding keep-alive message in case it is lost.

The RBRM model is as shown in Fig. 3.2.2. Basically, step 1 through step 5 are functions of a RBRM sender. As a receiver, RBRM works simply by receiving any newly arrived data. Thus, RBRM puts more responsibilities on the senders. The sender has to send data, control the amount of data traffic, detect any packet losses and retransmit lost packets.

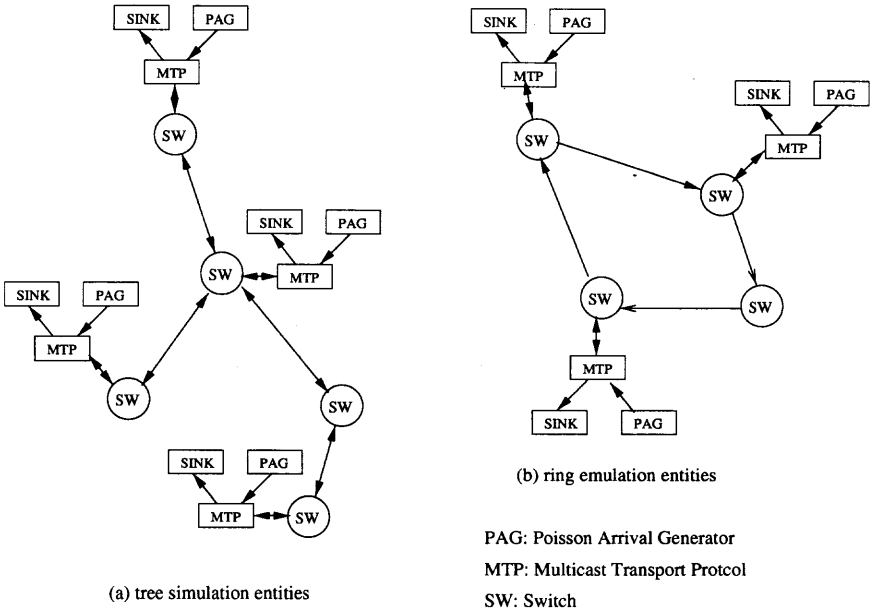


Figure 3.3 Simulation models.

In [63], a NACK scheme is designed to overcome overflow of the input buffer (i.e., PDU from network is lost when being transferred from the network interface to the transport layer). For simplicity, here we assume the input buffer is unlimited and can never overflow. Therefore, RBRM does not use any NACK messages. Note that the same assumption is applied to TBRM.

Although the protocol introduces control of traffic, RBRM is still fairly simple. The sender can detect packet loss easily. There are no NACKs in this protocol. The keep-alive messages and retransmissions are the protocol's overhead. Note, however, that this overhead is introduced into the network in a controlled way. The RBRM sender knows when this overhead must be used. The efficiency of the protocol may be reduced by the frequent keep-alive messages. However, since they are controlled as well as the data traffic is controlled, the extra burden from these KAMs is limited. This is another motivation of our study.

3.3 Performance Model

The two generic protocol models discussed in the previous section have key differences—they certainly perform differently with each having its own unique advantages.

Although we are talking about two reliable multicast protocols which appear to be at the transport layer, we have to be careful about all related network layers, above and below, in order to make the two protocols comparable. The modeling for either of the protocols can be regarded as two parts: network-layer modeling and upper-layer (i.e., transport-layer) modeling. In the network model, we build a simulation model specifying the behavior of network switches/routers and transmission links. For upper layers, i.e., application and transport layers, we should specify the packet-generation behavior and model the transport control for reliable multicasting.

In the following subsections we will present details of our simulation environment, assumptions and models.

3.3.1 Simulation Method

A multicasting system consists of network switches/routers, links and multicasting member hosts. Switches and links form the delivery paths of the multicast session. A subset of the switches are attached to multicast member nodes. Reliable multicast transport protocols are applied to these nodes with the applications layered above. It is the applications that generate data packets. In our simulations, applications are designed as Poisson arrival generators.

Fig. 3.3 shows the main simulation entities and their relations. Data packets are generated at packet generators and passed to the multicast transport protocol, which will put packets to the underlying network. When receiving a packet from the network, the transport protocol will forward to the sink, which actually does nothing but represent the receiving application.

The simulation model is based on discrete clock ticks. Each packet generator generates packet arrival events independently at random ticks according to the Poisson process. After a packet is generated, different events, such as arrival to a switch or being served at a queue, are scheduled at different ticks according to queuing disciplines of the system, until the packet finally arrives all receivers. The simulated networks are actually non-terminating systems, which means that we should observe the steady-state performance. Thus, to avoid initial transient problems, the simulation program waits for a long period before it starts to collect the statistics of various events. Then statistical data such as packet generation time, arrival time at each member, etc., are collected.

3.3.2 Arrival Process

Packet arrivals are generated according to a Poisson distribution. There are some other traffic generation models. For example, the binary Markovian asynchronous arrival model is a good one for bursty traffic simulation [85]. This model generally can generate a batch of packets in short bursts and then be idle for long periods. It is foreseeable that TBRM will not be able to handle this traffic model well. For rate-based protocols without traffic control, bursty traffic can usually fill up link buffers more quickly, and thus result in higher packet loss, hence more multicast overhead such as NACKs and retransmissions. Also, TBRM is a simplified model which does not have a periodic session progress report scheme like the session messages in SRM. With a long idle period, TBRM will take much longer time to detect loss of the last few packets of a bursty transmission. Note that in a bursty transmission, the last few packets are most likely to be discarded by switches.

For RBRM, since it has a sending window to limit the amount of traffic injecting into the network. The burstiness is shaped by the sending window. Therefore, the effect of bursty traffic on RBRM performance is limited.

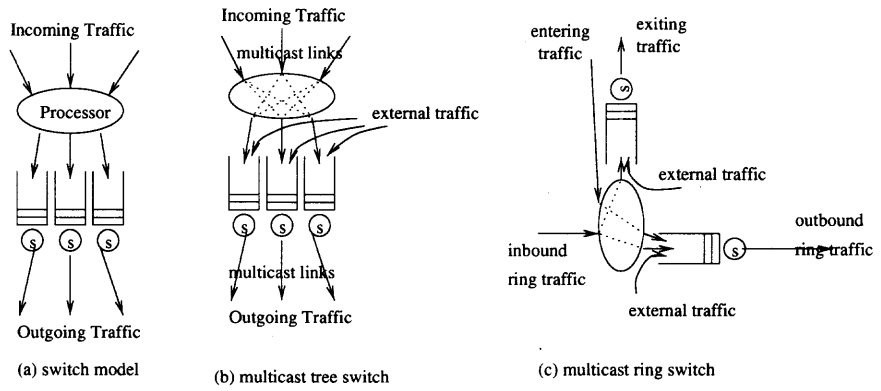


Figure 3.4 Network layer models.

3.3.3 Network-Layer Models

In our study, we assume the network environment to be multicast users separated from a cloud of networks providing network-layer services. Network topologies are generated randomly for simulation. For each random network generated, we use the algorithms in Chapter 2 to create a near-optimal shared routing tree and a near-optimal routing ring. Different transport protocols are applied on multicast users outside the network cloud. As indicated in Chapter 2, a ring uses 1.25 times as much link cost as a tree. In this way, we want to see how one protocol performs vs. the other for the same network environment.

The networks for our simulation consist of packet switches and links. A link can be viewed as a delaying device. That is, any packet/cell arriving at one end at time t of a link can be considered reaching the other end at time $t + d_p$, where the d_p is a fixed propagation delay.

A network switch can be viewed as a set of multiple single-server queues (Fig. 3.4-a). Suppose a switch is attached with n links, then each link is a queue for outgoing traffic, as well as an incoming source for the switch node. When a packet arrives from a link $i, (i = 1, 2, ..., n)$, it takes t_{proc} to process the packet and the packet will be put onto at least one of the queues. Each queue is basically a FIFO

queue, with each queue i having a limited size b. In our simulation, the service time is exponentially distributed as in M/M/1 queuing model.

When a packet is introduced into the network cloud, the total network delay on this packet is determined by three factors: propagation delay by links, processing delay at switches on delivery path, and queuing delay (include waiting time in queues and service time at queue servers).

As we are concerned with the performance of multicast protocols, we actually can focus on switches that form the delivery paths for a multicast session (these switches will be referred to as multicast switches in the following discussion). Switches that are not on delivery paths (also, links not on multicast deliver paths) will be ignored. Thus our simulation is simplified to contain only multicast switches and multicast links.

However, types of traffic other than multicast traffic also exist on the network. These should have effects on all multicast switches and multicast links. Thus, to make this simplification more realistic, an assumption will be made: each multicast switch will be introduced with some background/external traffic including unicasting and other non-related traffic. Without loss of generality, we may assume that each queue in the switch model has external traffic with Poisson distribution arrivals.

For switches on a multicast tree, multicast packets arriving from link i must be forwarded to links $j = 1, 2, ..., i-1, i+1, ..., n$. A tree switch can be modeled as shown in Fig. 3.4 (b).

For switches on a multicast ring, only three links will be considered: inbound and outbound ring links, and a link (referred to as the exit link) to host for data entering and exiting the network cloud. Traffic from the exit link will be put on the queue for the outbound ring link, while traffic from the inbound ring link will be forwarded to both queues of the exit link and outbound ring link (Fig. 3.4 (c)). Certain control has to be simulated because a packet has to be checked for its first

ring switch to avoid continuous looping around the ring. This is assumed to be accomplished by a ring routing protocol. Also, notice that some of the switches on the ring may not have multicast group members attached, i.e., they are transient nodes. Still, the ring switch model can be used with zero entering traffic. So these switches have only two ring links.

3.3.4 Simulation Parameters

Several parameters are variable and can be changed. Some parameters are common for both RBRM and TBRM simulations: the average inter-arrival time of the multicast PDU generator (denoted as $1/\lambda_m$), the average inter-arrival time of external traffic at each link (denoted as $1/\lambda_e$), the average service time $1/\mu$ of the server of each queue, the propagation delay d_p of each link, and the buffer size b for each link. Besides these, two other parameters are the simulation length in ticks for each run, and the starting tick for collecting data. Results presented in the following are run from simulations with length of 240,000 simulation ticks and the starting tick is at 120,000. The window size w is a variable parameter particular for the window-based RBRM.

We always assign a permanent link propagation delay $d_p = 1$ to simplify. The processing time t_{proc} is assumed to be 0. The average service time is assigned as 10 simulation ticks, which serves as the base for deciding other parameters. Then, most other variables are changed to see how they affect the performance measure of interests. In most simulations, link buffer size b is assigned as 20 packets to provide small packet-drop probability, and window size w is 8. However, these two parameters are also changed in some simulations, as we will see later.

Traffic of a network in our simulation consists of three parts: multicast data packets, overhead introduced by multicast protocols, and external traffic introduced at each link. As an indicator of the network offered load, we define *combined traffic*

load as below:

$$\rho = \frac{N\lambda_m + \lambda_e}{\mu}$$

where N is the number of multicast members in a group.

3.4 Performance Results

Generally, there are three measures of our interest. One is the delay performance, and, although we know that a tree path is usually much shorter than the ring path, we think it is more important to find out what the total delay would be when queuing delay is taken into consideration. The second one is the throughput performance. As the RBRM applies the window control, we want to see how much we can benefit. In the following discussion, we use packet count of successful multicast to demonstrate the throughput performance of these protocols. We define successful multicast here to be a data packet received by everyone in the same multicast group. With both service rate and the simulation period fixed, the packet count of success multicasts is actually a constant times the usual normalized throughput. The other one is the efficiency—we want to see how much overhead is incurred for each protocol. Overall, the multicast session related packets are control messages, retransmissions and data packets.

Most of our simulations are based on 16-member sessions. The random networks we generated are the same as in [18]. Each network has 64 nodes. 128 links are generated between randomly chosen pairs of nodes, with an additional 64 links linking them in a global ring to ensure there is at least one solution for the ring construction. 16 group members are uniformly picked from these 64 nodes. Then, a shared multicast tree and a multicast ring are created separately. The maximum degree in these random networks is between 5 and 3, with the average 3.8. Multicast tree sizes (i.e., number of nodes) range in $[17, 26]$, with an average of 20.1. The longest paths of these trees are 10.7 on average, in the range $[7, 17]$. Multicast ring

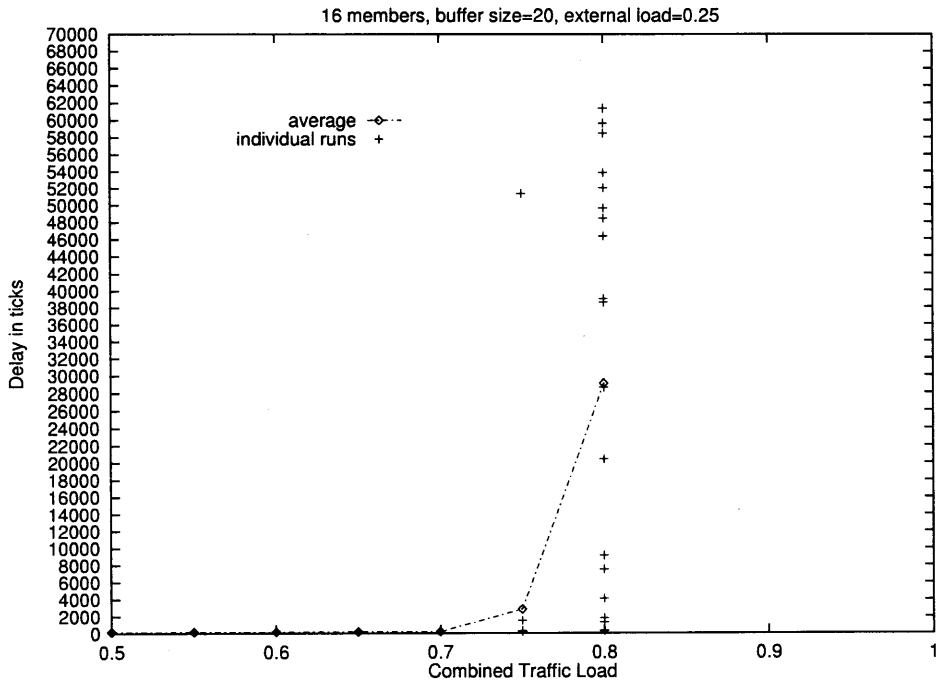


Figure 3.5 TBRM delay vs. combined traffic load.

sizes are between 19 and 28. The average is 23.1. Note this is also the average ring length.

The following results are obtained from simulations on 20 random networks. In most of the simulations, each link is assigned with a background traffic (referred to as external load) rate λ_e such that $\lambda_e/\mu = 0.25$.

3.4.1 TBRM Behavior

Figures 3.5 and 3.6 show the characteristics of the TBRM protocol. The x-axis of all the charts are the combined traffic load (i.e., ρ) of multicast traffic and external background traffic. Results of each individual simulation run are plotted as well as the average of these runs.

We can see that the delay of successful transmissions stays very low for light traffic load. When traffic grows heavy, results from individual runs start to diverge. The variance is large—delay can be small in some runs and huge in others.

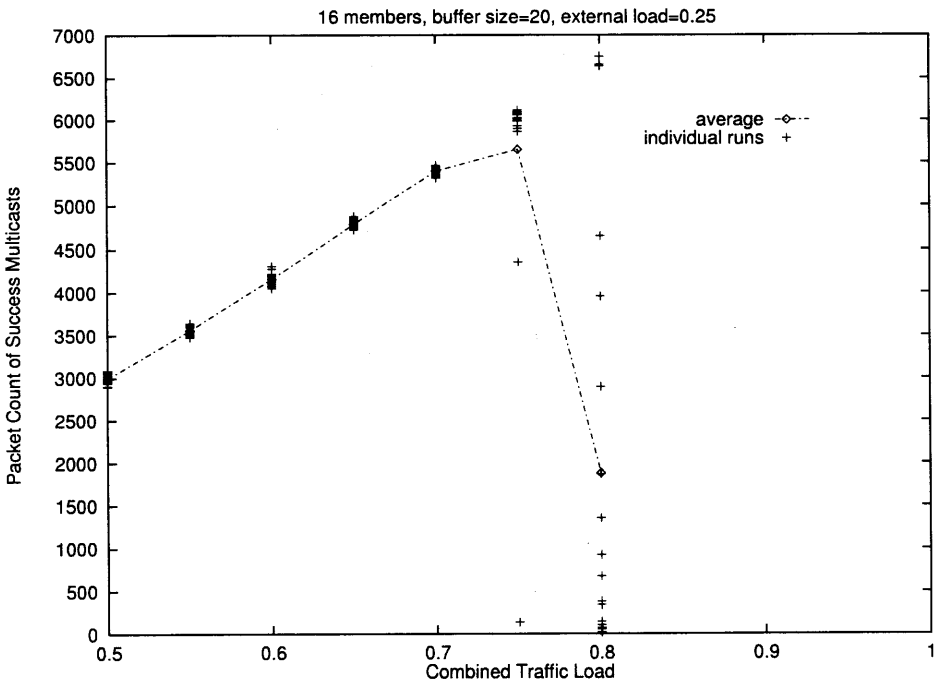


Figure 3.6 TBRM throughput vs. combined traffic load.

From Figure 3.6, we observe the same pattern. We may find that the amount of successful transmissions increases with traffic load until networks are congested. And finally, for high load session data can hardly be successfully multicasted. As we will discuss in the below subsections, the turning point is the when the networks begin to discard packets, and protocol overheads(e.g. repairs and NACKs) are involved.

This shows that TBRM does not have a good scheme of recovering from packet loss. Once the network is congested, the protocol is likely to aggravate the congestion. The average curve actually does not provide any accurate prediction of the actual network performance because of the high variability. However, it indicates the chance of congestion. With traffic load increasing, the chance of congestion soon increases enormously. Thus, we may conclude that TBRM's performance is unpredictable when multicast traffic load is sufficiently high.

Although not accurate, the average curve does show the tendency of the protocol behavior. In the following discussion of the TBRM protocol, we use the average curve instead of the individual runs to make clear charts.

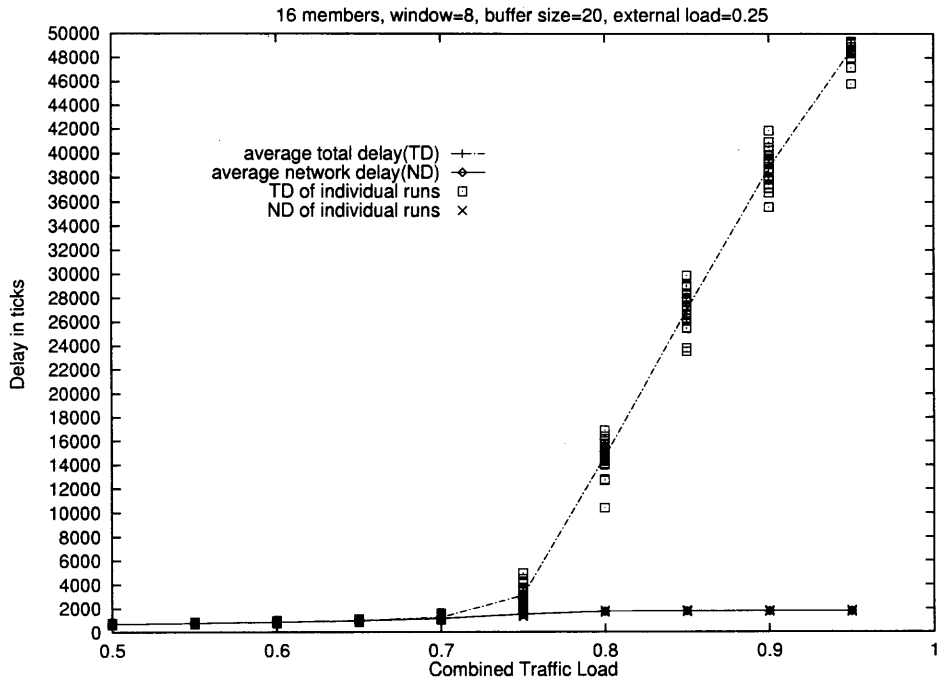


Figure 3.7 RBRM delay vs. combined traffic load.

3.4.2 RBRM Behavior

Figure 3.7 and 3.8 show the simulation results of the RBRM protocol. The simulated networks are the same and the parameters are mostly the same as in Section 3.4.1. The RBRM employs the window protocol. In these charts, the window size is 8.

The delay of a packet under RBRM protocol is made up of two parts, the accessing delay before it is put on the network and the network delay when it traverses the network. In Figure 3.7, the total delay and the network delay are both plotted. Note that RBRM's network delay increases slightly as the traffic load increases. The total delay increases significantly when data is accumulated faster than the window can send. Note that this is an effect of the window protocol. The window protocol actually controls the amount of traffic put on the network, thus avoiding overloading networks. This can be observed from Figure 3.8: the amount of successfully multicasted data remains constant at high network load.

Note that the total delay shown in the chart should also be a function of time. When data is generated faster than the protocol could send, the accessing delay is

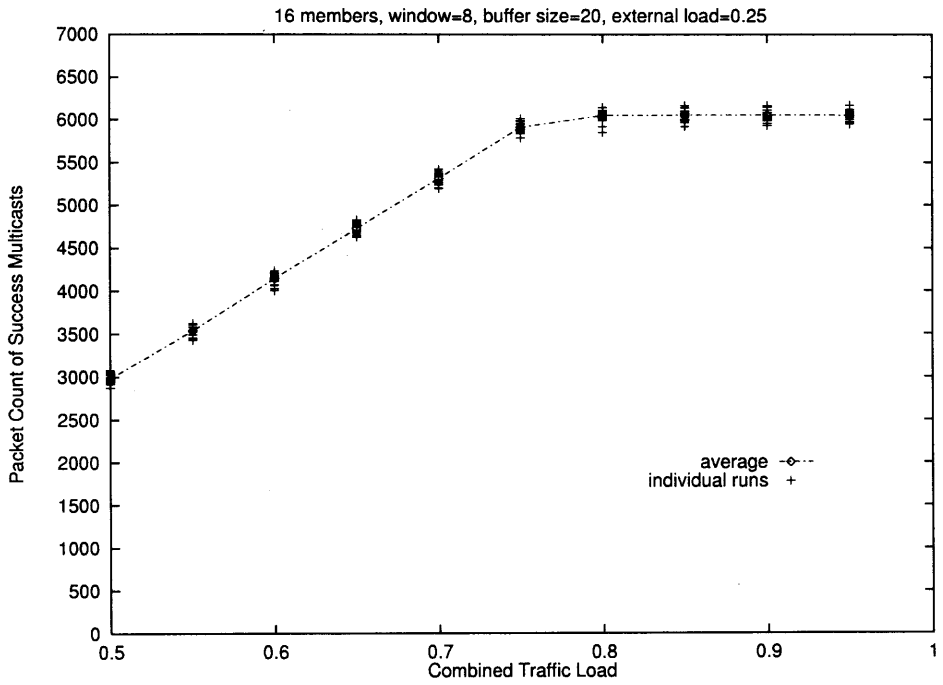


Figure 3.8 RBRM throughput vs. combined traffic load.

in fact unbounded. It is not the case in real life that a host application will keep on accumulating data in the transport buffer. The point is, RBRM provides a good scheme of traffic control. The access delay is at each host end. Blocking at hosts does no harm to the network.

Contrary to TBRM protocol, the RBRM shows that at any time, the simulation results are consistent. All results from individual runs are around corresponding average values, meaning the RBRM should yield a predictable performance.

3.4.3 Effect of Traffic Control

Fig. 3.9 is a close-up comparison of delays of both protocols. Here (and in the following discussions) for clear demonstration we use average curves only. Although not accurate, the average curve does show the tendency of a protocol's behavior. From the chart, we may see that: (*i*) TBRM can multicast packets with much smaller delay when traffic load is low (before congestion happens). Multicast trees always have the advantage of shorter delivery paths over multicast rings. (*ii*) RBRM

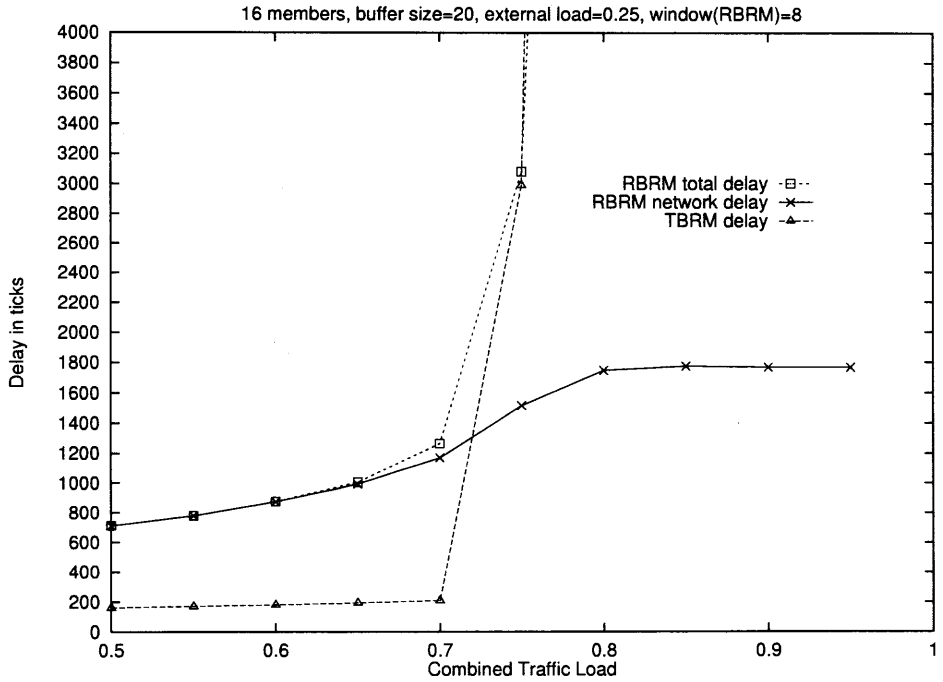


Figure 3.9 Comparison of TBRM and RBRM delay.

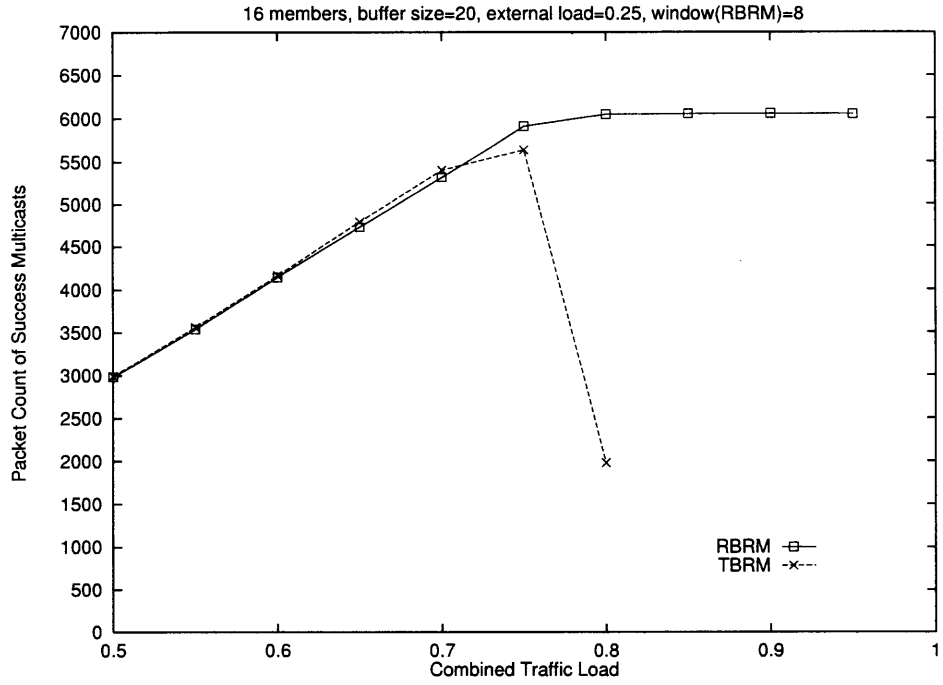


Figure 3.10 Comparison of TBRM and RBRM throughput.

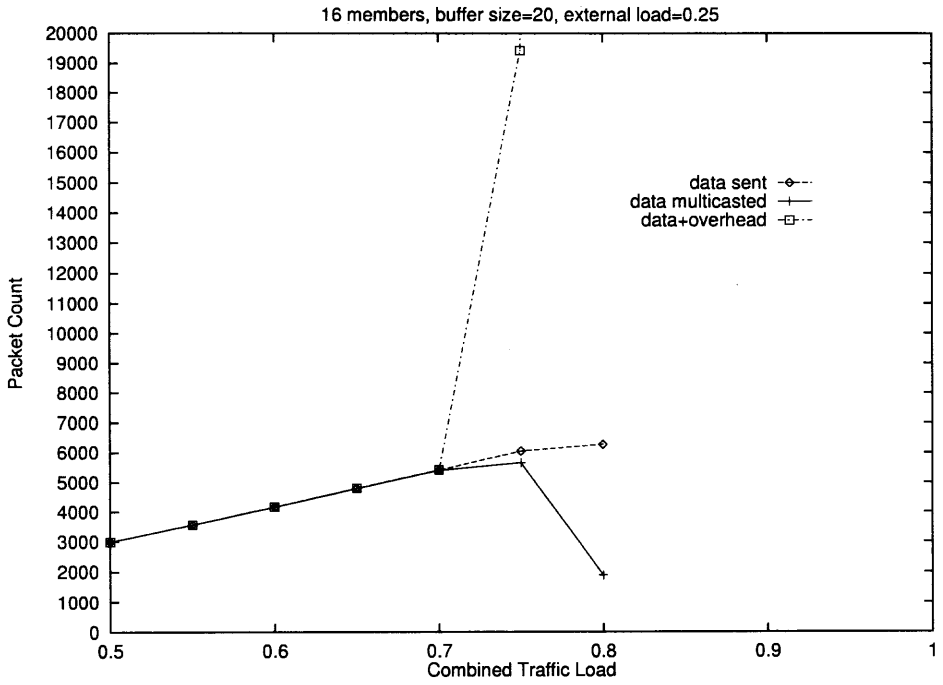


Figure 3.11 TBRM traffic composition statistics.

is able to maintain relatively constant network delay at heavy traffic load. When congestion happens, TBRM takes a significantly long time to multicast packets to every receiver. With the window control scheme, RBRM prevents overloading a network very effectively. This property is very important because a network should not be overwhelmed under any circumstances.

Fig. 3.10 shows the comparison of goodput performance between the two protocols. We may see that goodput of TBRM protocol drops quickly when load reaches a certain point, resulting from serious network congestion. RBRM can manage to keep on sending data packets constantly at very high load. One characteristic, not shown in this figure, is that individual simulation results of TBRM's goodput are also diverse when traffic is heavy, similar to TBRM's delay performance. Similarly, RBRM still shows consistent simulation results.

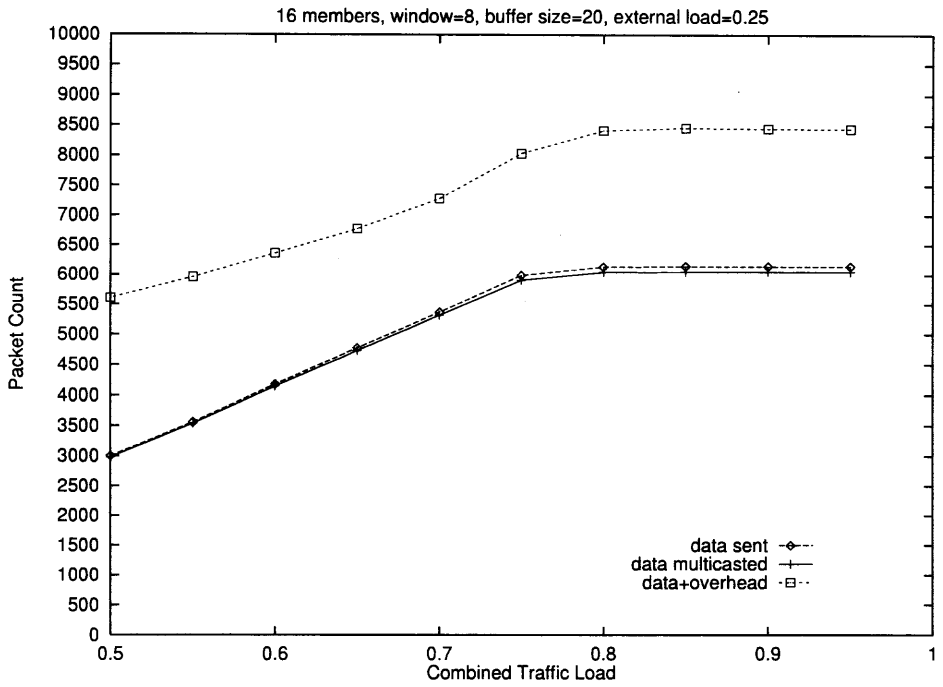


Figure 3.12 RBRM traffic composition statistics.

3.4.4 Traffic Overhead and Protocol Efficiency

Figure 3.11 shows three curves: number of data packets put in the network, number of data packets successfully multicasted, and the total number of packets involved in the session. We may see that although TBRM keeps on sending data into the network, the number of successful multicasted packets drops at high load. The total traffic increases enormously, showing that the protocol overhead is overwhelming at high load. The protocol overhead includes NACKs and repairs (i.e., retransmissions). Thus, TBRM's low-load efficiency is very high because of minimal packet loss, but it drops sharply at high load.

Fig. 3.12 describes RBRM. The same three curves are plotted. Even at low load, the protocol has overhead because of regular keep-alive messages. Due to the relatively small amount of data traffic at low load, the efficiency is low compared to TBRM. When net-wide load increases, the amount of data traffic increases, but there is not much change of the overhead. At high load, both the overall traffic and data traffic keep constant. Thus, the efficiency stays at a constant level.

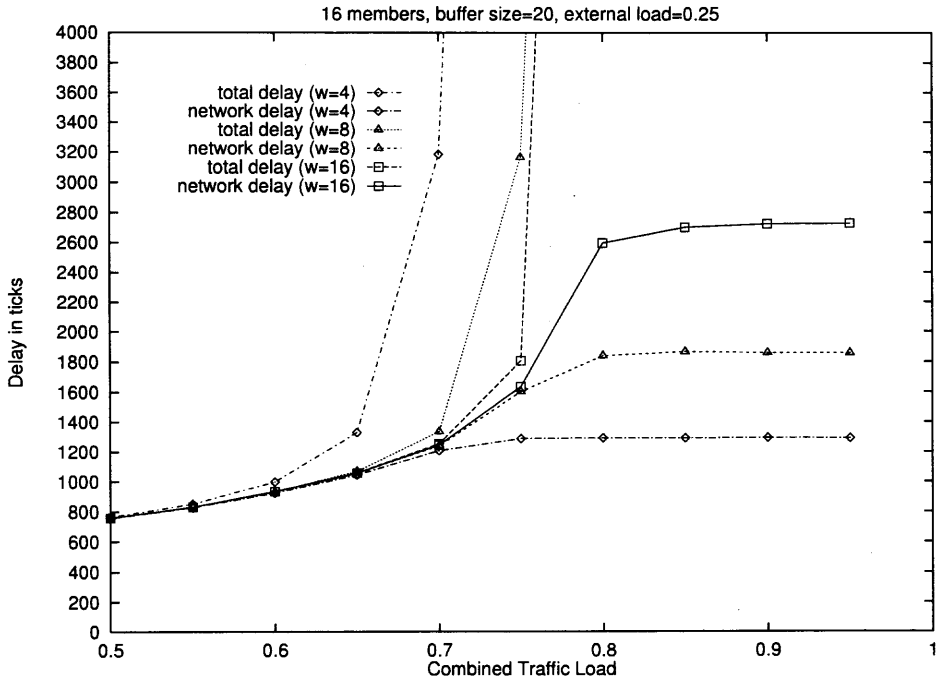


Figure 3.13 RBRM delay performance varies with window size.

Note that for RBRM, the number of successful multicasted packets is very close to the amount of data that RBRM puts on the network. This shows the effectiveness of RBRM: it keeps a very high successful rate, and most data packets put on the network can be successfully multicasted.

There is one more interesting observation of the RBRM overhead. We may observe that as the traffic load increases, the gap between the total traffic amount and the data traffic narrows in the mid area, then increases again and finally keeps constant. When the load is low, the overhead consists of only keep-alive control messages (KAMs). KAMs are ticked by the ring-trip delay. This delay increases at low load, resulting the KAMs decreasing. Hence, the overhead decreases. When packet losses begin to happen, retransmissions are involved, resulting in the overhead increasing again. When the advancing of the sending window finally balances the packet losses, the overhead stays constant.

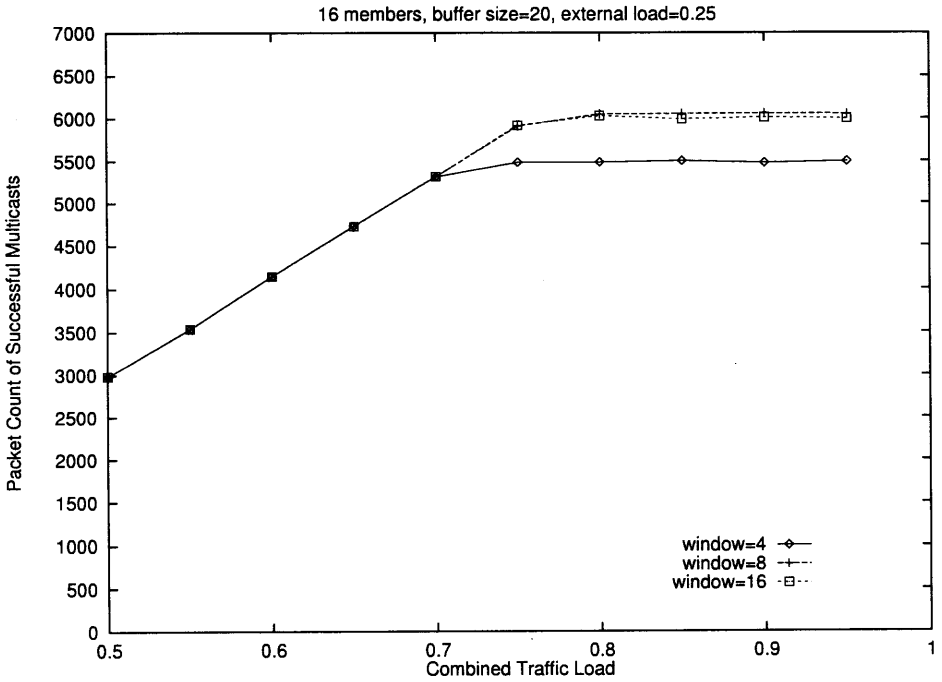


Figure 3.14 RBRM goodput performance varies with window size.

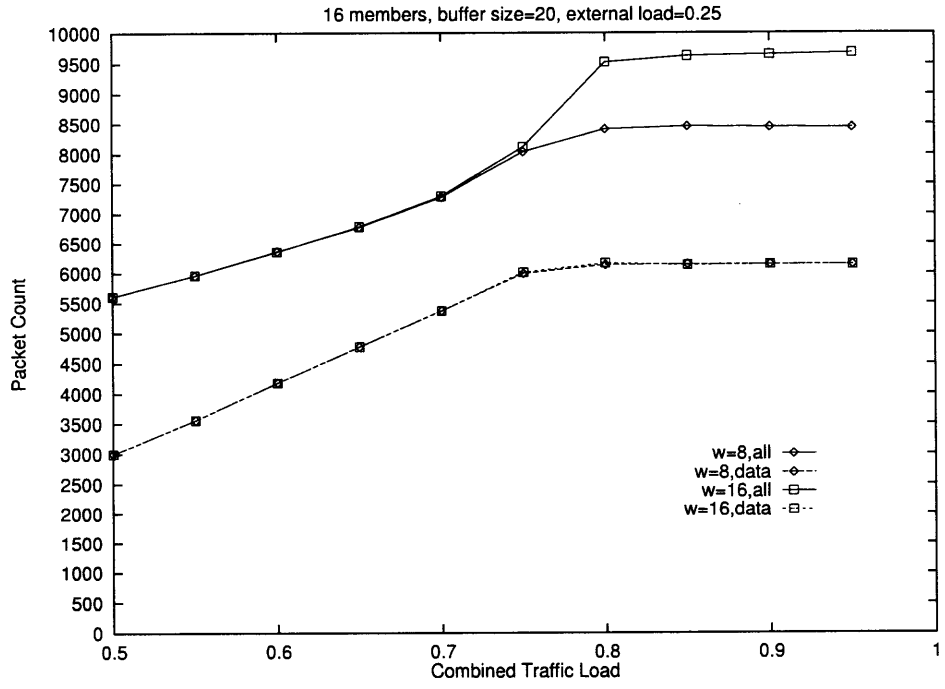


Figure 3.15 RBRM overhead comparison of different window sizes.

3.4.5 RBRM Window Size

When window size is increased, RBRM allows more outstanding packets in network. Therefore, the access delay may be decreased. Since the traffic amount is also increased, packets suffer longer network delay. Fig. 3.13 shows the delay performance changes with different window sizes.

Increasing window size may not necessarily improve RBRM's goodput. Fig. 3.14 shows that the goodput of a 16-packet window is about the same as that of an 8-packet window, although there is an increase when the window is enlarged from 4 to 8. The reason is worthy of further study. Here we compare, in Fig. 3.15, the traffic compositions of the two window configurations. We can see the amount of data sent by both configurations are about the same. With window size 16, the overhead is larger. RBRM overhead includes keep-alive control messages($KAMs$) and retransmissions. As mentioned before, the frequency of keep-alive messages is ticked by the ring-trip delay. For the larger window configuration, $KAMs$ should be less frequent due to longer ring delay. Therefore, the major difference is due to the retransmission traffic, which implies that retransmissions for a size-16 window are much more than for the size-8 window.

From these observations, we may conclude that the window size is one factor that effects RBRM goodput, but not the only one. The ring itself has a capacity for RBRM traffic. Therefore, when a window size is sufficiently large, a further increase of window size does not improve the throughput, but efficiency drops and the delay performance is worse.

3.4.6 Effect of External Traffic

In Figure 3.16, we compare two scenarios for TBRM. In one case, the background traffic load is fixed and the delay is observed with increasing multicast traffic load. In the other case, we keep multicast traffic load unchanged and watch the delay changes

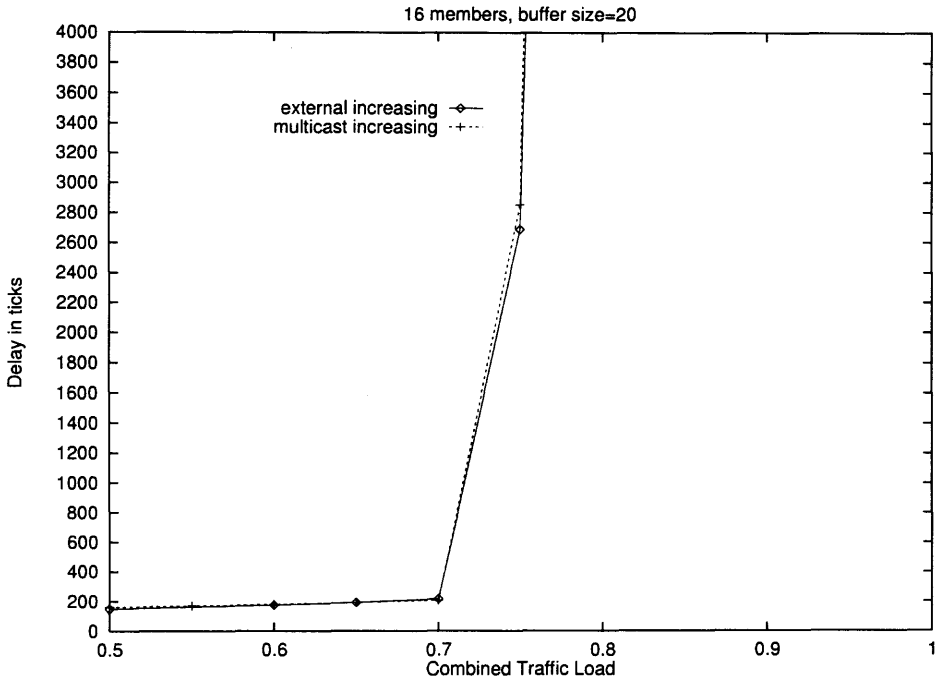


Figure 3.16 Effect of external traffic on TBRM delay.

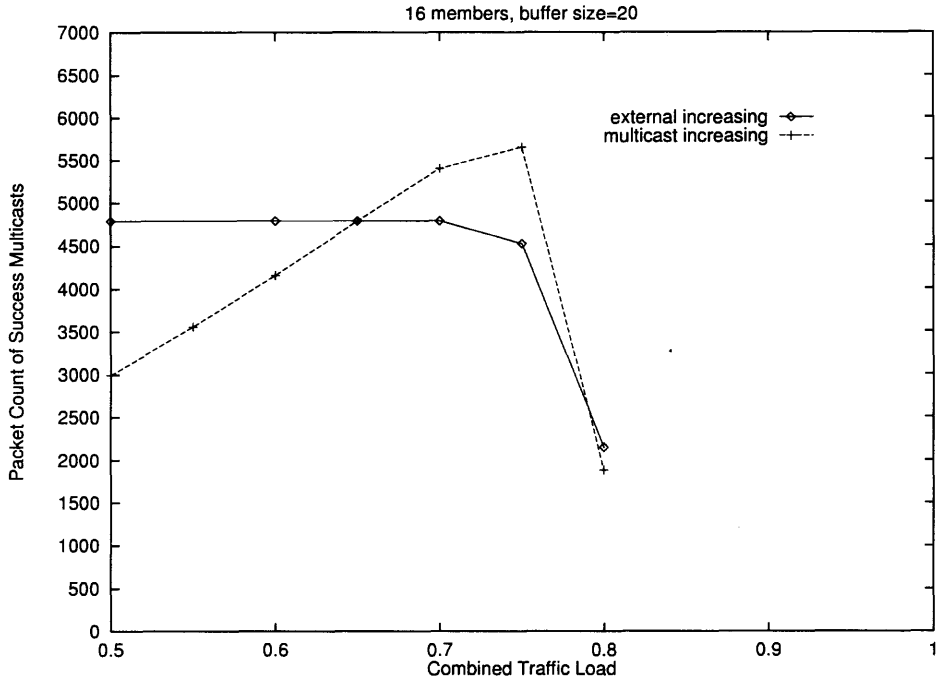


Figure 3.17 Effect of external traffic on TBRM throughput.

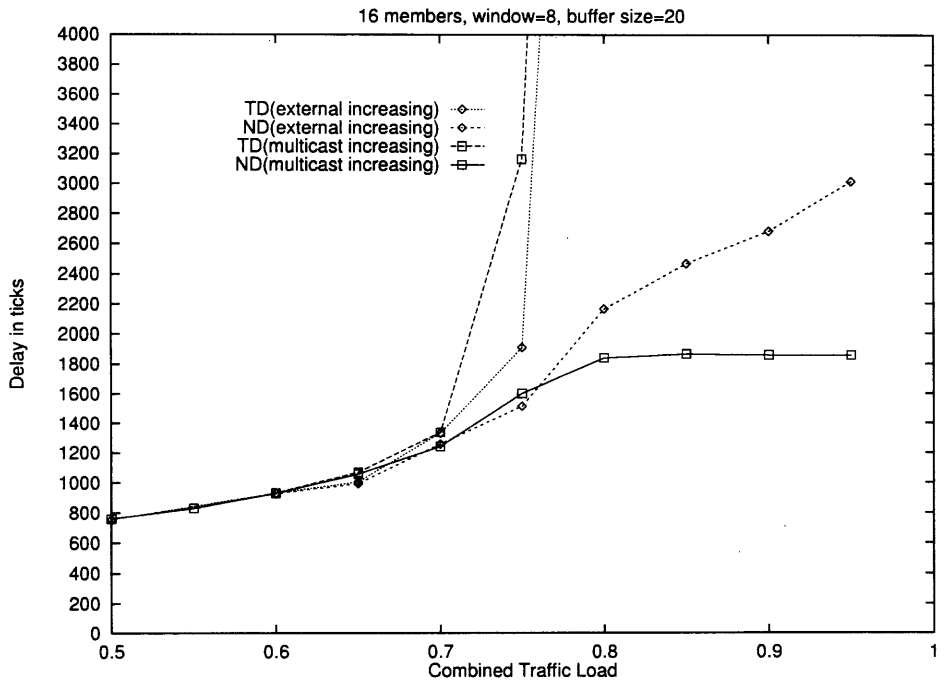


Figure 3.18 Effect of external traffic on RBRM delay.

according to background traffic. As shown in the figure, these two scenarios have the same outcome. When the combined traffic load increases to a certain point, either by increase of background traffic or multicast traffic, the network becomes congested, delay turns out to be intolerable, and the throughput is harmed (Figure 3.17).

For RBRM, we may see from Figure 3.18 that increasing the external traffic also will cause longer network delay. Different from the steady network delay at high load when external load is fixed, when external load increases, network delay at high load increases accordingly. This is because the external traffic is not controlled. The actual network load becomes heavy when external load rises. Correspondingly, the throughput (Figure 3.19) at high load decreases while the external traffic increases.

3.4.7 Link Buffer Size

Figure 3.20 compares the delay of 2 different buffer size settings for TBRM. One setting has larger link buffer sizes. Each link has a buffer size of 22 instead of 20.

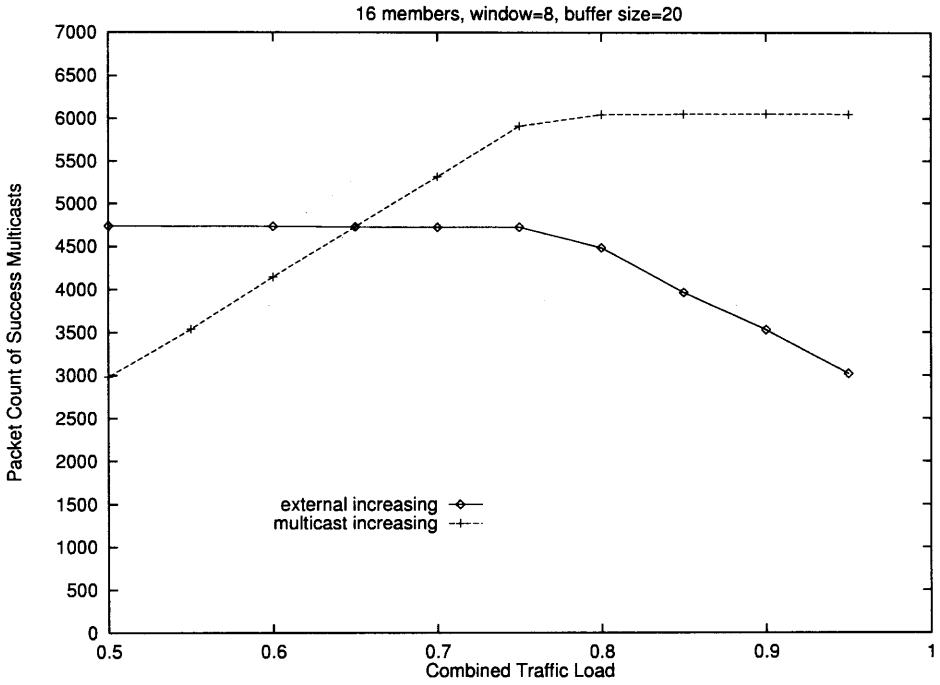


Figure 3.19 Effect of external traffic on RBRM throughput.

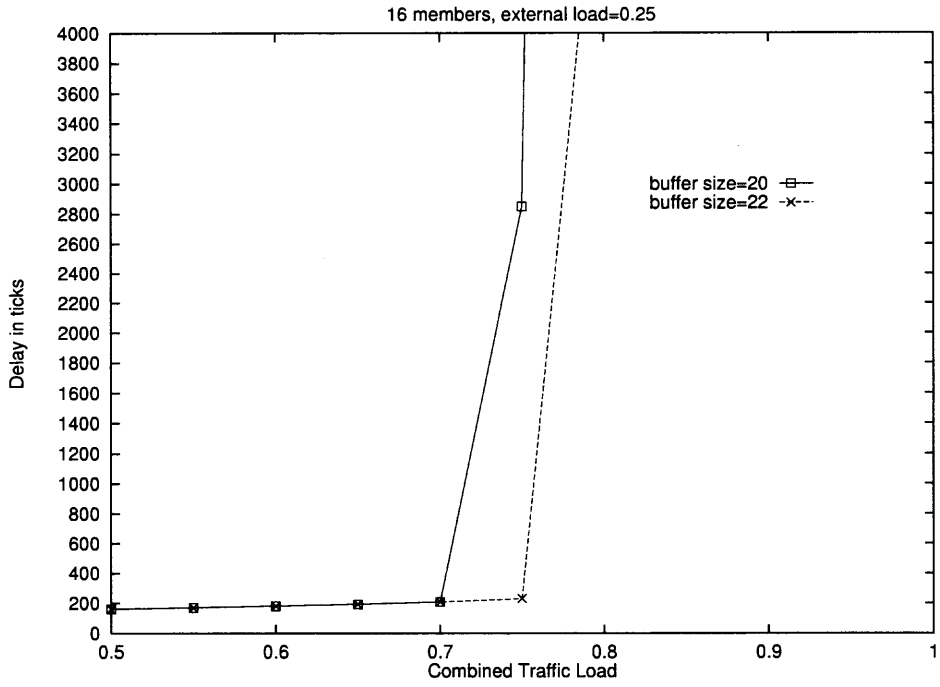


Figure 3.20 TBRM delay performance varies with link buffer size.

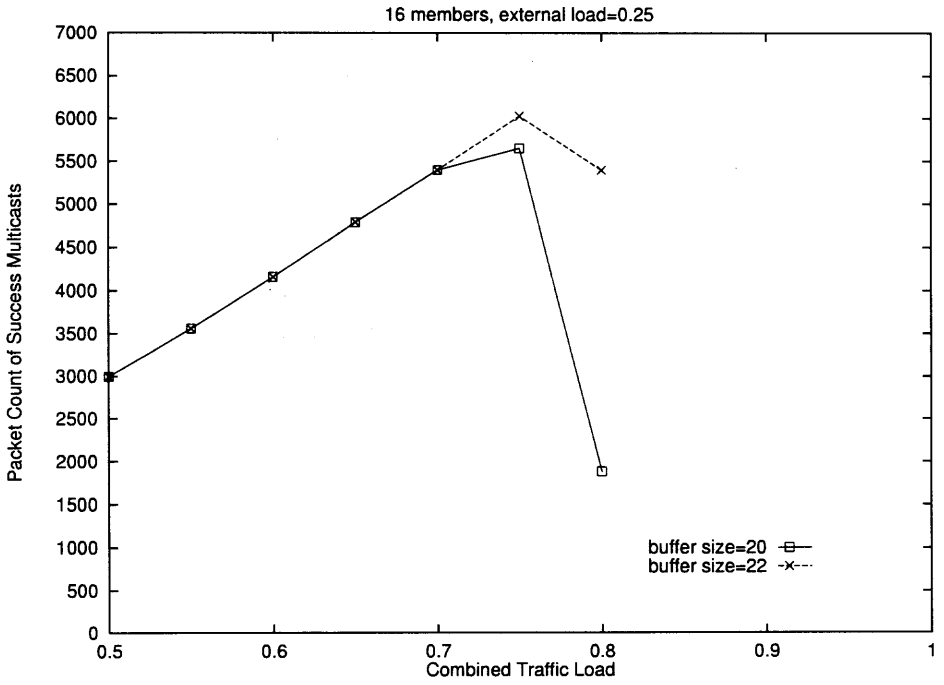


Figure 3.21 TBRM throughput performance varies with link buffer size.

As we can see from the chart, with a larger buffer size, congestion occurs at a higher load. Also, the throughput can be better with larger size (see Figure 3.21).

For RBRM, the delay performance shown in Figure 3.22 does not change much except that the access delay at load 0.75 is smaller, which is brought about by the enlarged buffer. The network delay at high load is about the same for the 2 settings. This implies that when packet dropping occurs, the network load condition is kept at a steady level by the window protocol. From the throughput chart (Figure 3.23), we can see the enlarged buffer results in better throughput at high load.

Increasing link buffer size can help both protocols. With an enlarged link buffer, TBRM can bear the higher traffic load without performance collapse, and RBRM can maintain a larger goodput.

3.4.8 Session Size

We generated 22-node groups from 128-node networks. Shared trees and rings are built in the same way as for 16-member simulations. The maximum degree of a

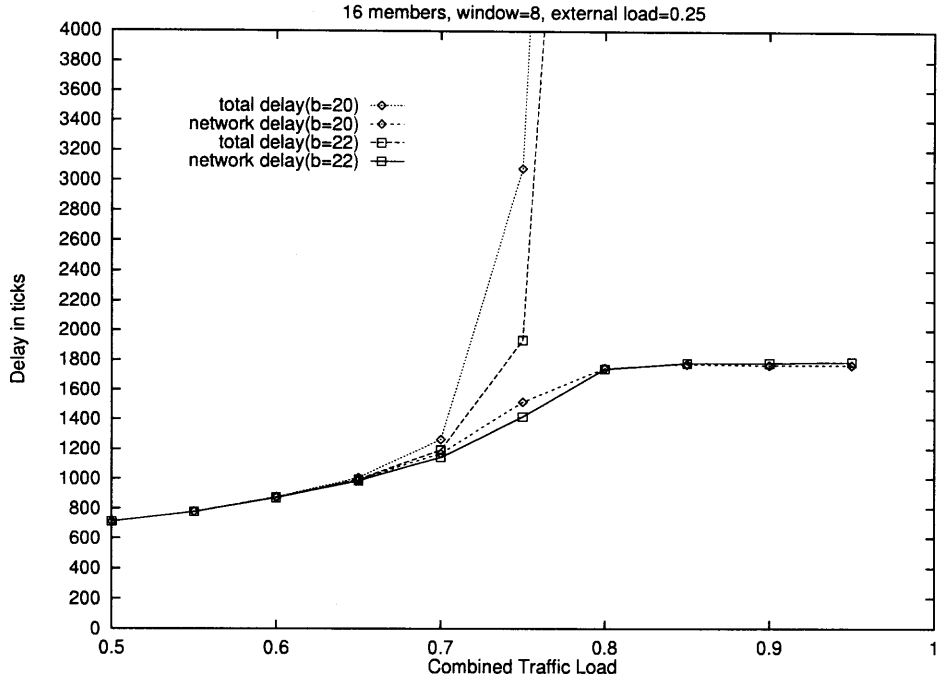


Figure 3.22 RBRM delay performance varies with link buffer size.

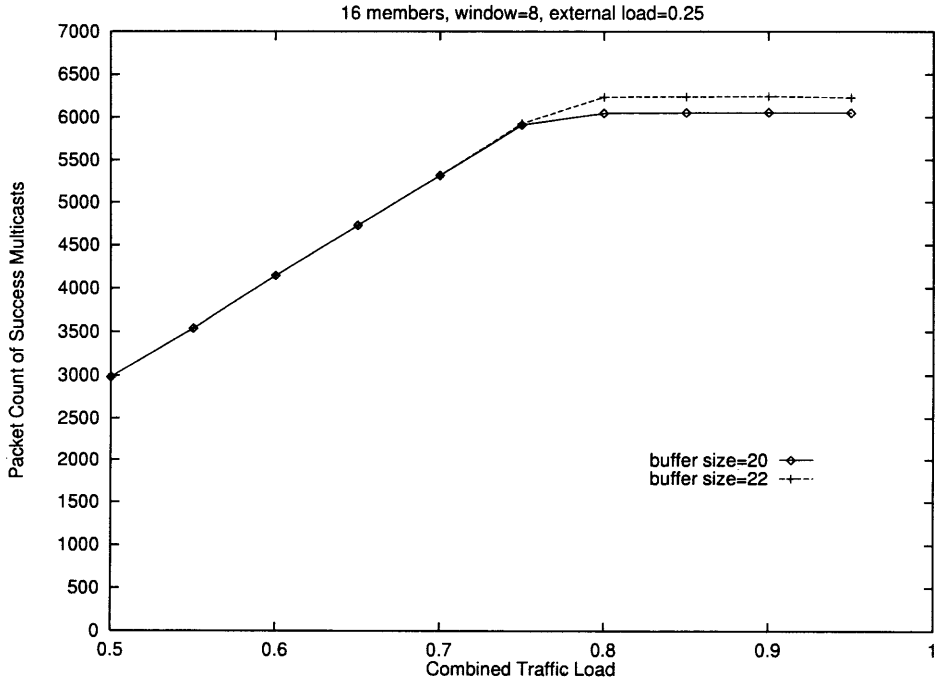


Figure 3.23 RBRM throughput performance varies with link buffer size.

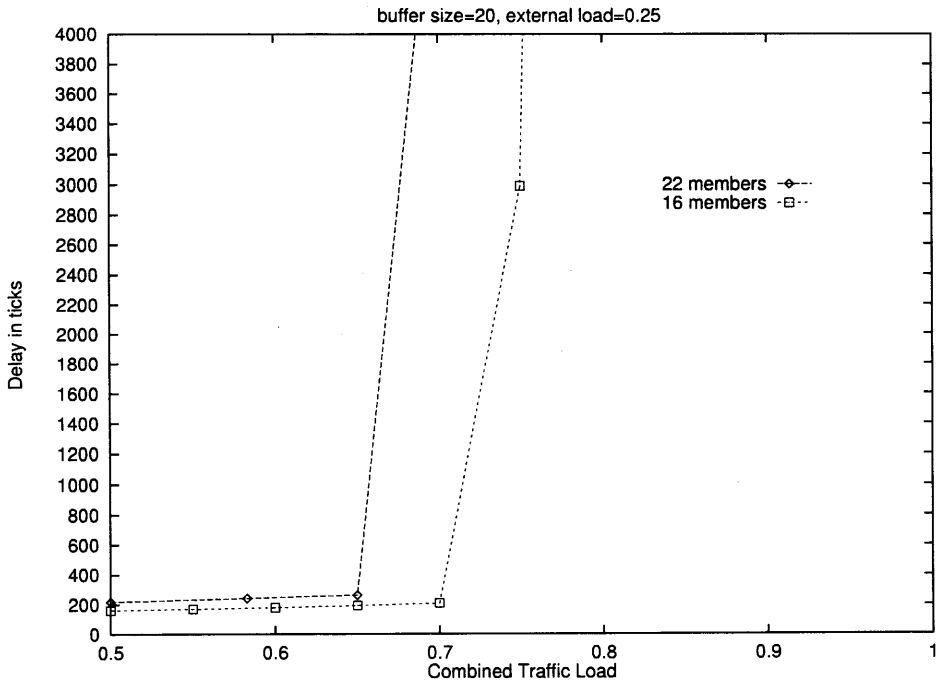


Figure 3.24 TBRM delay performance varies with session size.

node is between 3 and 6 with the average 4.3. Multicast tree sizes are in the interval $[27, 40]$, with an average of 32. The longest tree path is 14 on the average, between 9 and 25. Multicast ring size is in $[31, 44]$, 37.5 on the average.

Results show that a larger session has degraded performance. Although the scale of sessions is changed and the statistical values are changed accordingly, the basic patterns of these two protocols stay the same.

Figure 3.24 is the TBRM delay comparison between these two session sizes. The chart shows that a larger session with larger tree sizes tends to congest networks earlier. This should be true, because the probability of packet loss increases with the increase of tree path lengths. Similarly, Figure 3.25 shows smaller sessions have higher data throughput.

The same goes for RBRM with larger sessions — the host end accessing delay rises up earlier, and the network delay is longer because of more ring hops. The throughput of a smaller session is better at high-load area (Figure 3.26 and 3.27).

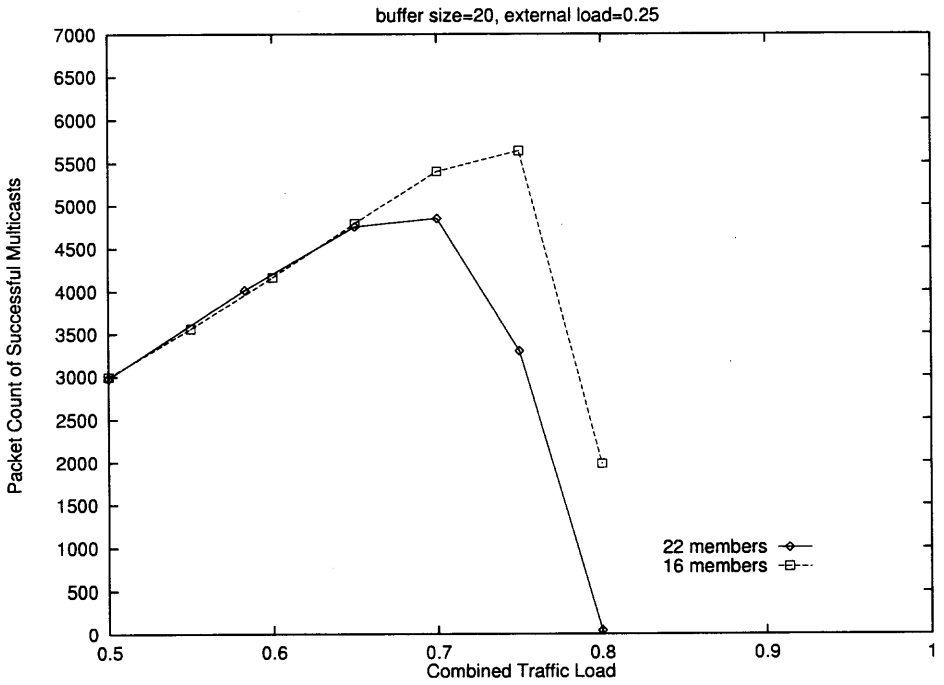


Figure 3.25 TBRM throughput performance varies with session size.

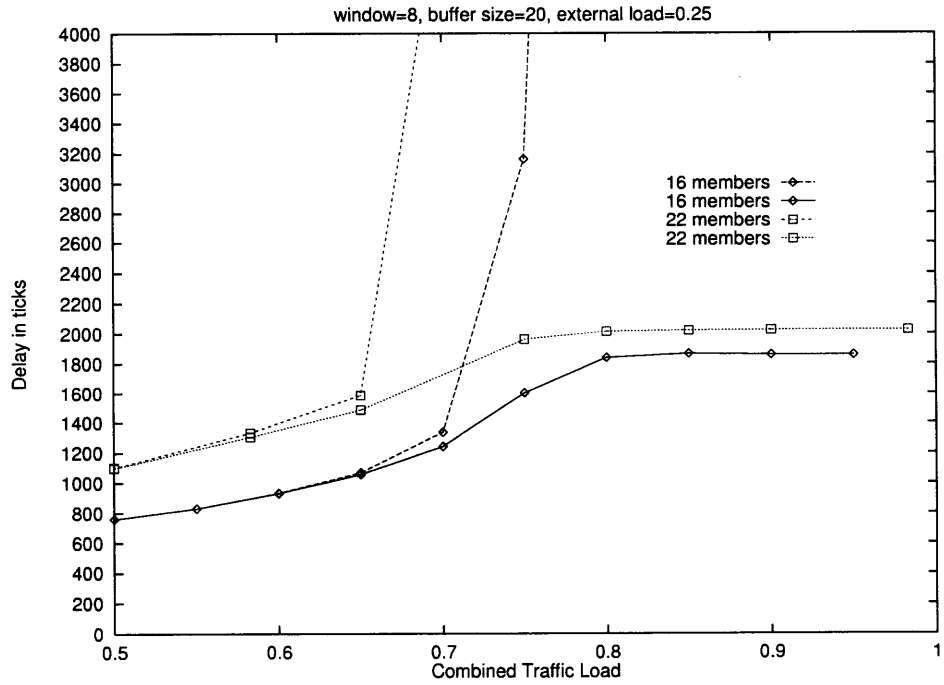


Figure 3.26 RBRM delay performance varies with session size.

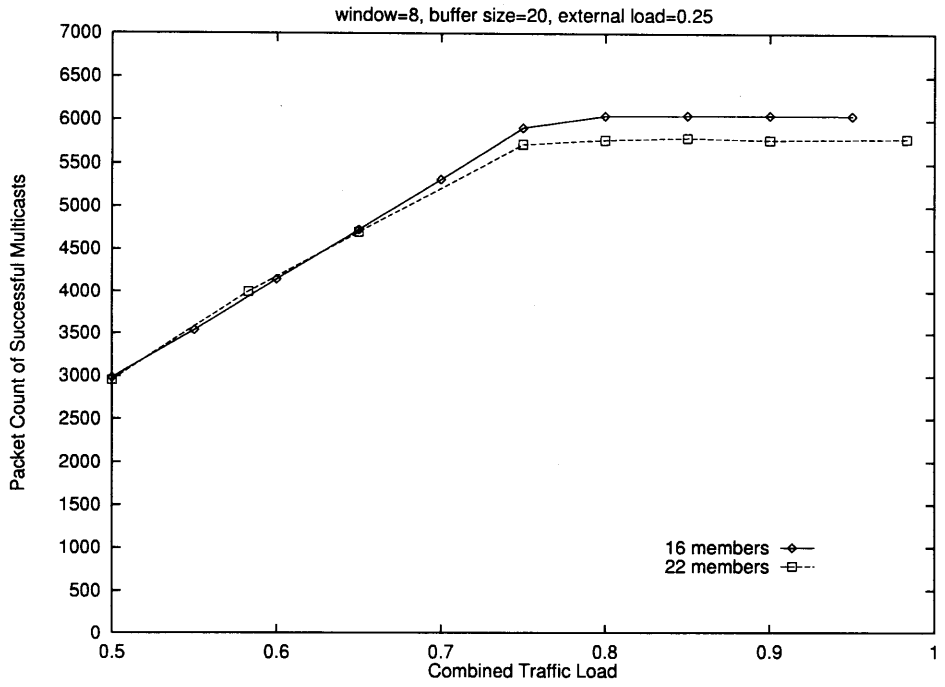


Figure 3.27 RBRM throughput performance varies with session size.

3.5 Summary

This chapter compares two different approaches to reliable group multicast protocols — tree-based (TBRM) and ring-based (RBRM) — and illustrates some trade-offs of these two approaches. From the simulation results we presented, some ideas are verified and some observations are made and should be further studied.

In summary, TBRM and RBRM behave quite differently and each has its advantages. TBRM shows very good delay performance and high efficiency at low network load. However, TBRM is very sensitive to packet loss and has poor throughput performance at high network load. Furthermore, it can experience what is known as *"congestion collapse"* (i.e., a situation in which almost no multicast transmission is completed successfully). In addition, TBRM aggravates network congestion by its uncontrollable transmission overhead.

RBRM has a longer delay at low network load than TBRM due to its long routing path. The window-based flow control facilitates simple feedback from the ring, therefore showing its important control mechanism, since at high network load

it ensures that *"congestion collapse"* will not occur. The window-based flow control regulates the traffic as it enters the network. Furthermore, if there is congestion inside the network, packets will enter the network in a reduced rate. Consequently, the ring-based reliable group multicast has stable throughput at high network load.

We also explored some variables that may affect the performances of the TBRM and RBRM protocols. In short,

- Both protocols' goodput performance benefits from increased link buffer size, but the network delay also increases.
- Performances of both protocols, can be negatively affected by increasing network background traffic. Hence, some resource reservation and control may be necessary to ensure reliable group multicast sessions.
- Both protocols show performance decreasing when size of the multicast group increases.
- For RBRM, window size is not the final determining factor of its throughput. The throughput may be limited by both the ring size and the link buffer size.

CHAPTER 4
HIERARCHICAL AND HYBRID ARCHITECTURE

The previous chapters investigate the performance impacts on reliable group multicasting from different feedback mechanisms, as well as from different routing topologies. Although NACK-based reliability control is efficient at low network load, it is difficult to prevent network congestion due to excessive multicast traffic. Although ACK-based reliability control is effective in regulating multicast traffic, it requires complex protocol operations if the multicast is tree-based. Although the multicast tree provides minimum delay performance, it is difficult to enforce reliability control from upper layers. Although multicast ring facilitates upper-layer reliability operations, its scalability is unsatisfying due to the latency performance.

Our objective in this section is to explore a solution that integrates the following: window-based traffic control with the implicit feedback from ring structure, low latency with the tree-based multicast, and high efficiency of the NACK-based approach at low network load. We propose a hierarchical architecture for reliable group multicast called RITA (Rings Inter-connected with Tree Architecture). RITA is distinguished from other hierarchical models with two essential features: (i) It is hierarchical. We divide a multicast group into several clusters, and clusters are interconnected by a backbone. The task of reliability is therefore also divided into smaller independent tasks to ensure reliabilities within separate clusters and the backbone. We hope this "divide and conquer" strategy may help reliable multicast protocol to scale to large multicast groups. (ii) It is hybrid. Tree and ring structures are both used as sub-structures for clusters and the backbone. Furthermore, it is necessary to employ both ACK- and NACK-based protocols along with the respective ring and tree structures within the hybrid architecture.

This chapter is organized as follows. In Section 4.1 we discuss considerations of RITA design, and define basic components of the architecture. Different operations

are necessary for different components. We define two protocols in Section 4.2: a dynamic window-based protocol for intra-cluster reliable multicast, and a light weight NACK-based protocol for inter-cluster reliable multicast. The interaction and cooperation of these two protocols is another major focal topic. In Section 4.3 we investigate the performance of RITA. We show results of RITA simulations and compare these with results of pure tree- or ring-based protocol simulations. We show that RITA has the ability to maintain similar throughput and congestion control performance, while gaining from significantly lower delay, as compared to ring-based protocol. Section 4.4 is the summary of this chapter.

4.1 Hybrid Multicast Architecture: RITA

A hierarchical structure is useful as a common strategy to solve the scalability issue. RMTP [71] is an example of using hierarchies. By aggregating ACKs of the same hierarchies, RMTP successfully solves the ACK implosion problem. But RMTP is difficult to adapt to many-to-many multicast, because the hierarchies are source-based. To maintain different hierarchical structures for different sources is costly and may have scalability problems. In Lorax [52] a shared ACK tree mechanism is proposed for concurrent multicasting with multiple sources. It needs sophisticated operations and participation from routers in order to aggregate ACKs to different sources. LRMP [53] divides a group into subgroups for error recovery, but it still does not address multicast flow control. RMX [17, 16] uses a hierarchical structure so that multicast can be done in subgroup domains, while using unicast for inter-domain reliable dissemination.

RITA is a two-hierarchy structure. Basically, we partition a group into subgroups. Each subgroup forms a small multicast cluster. These clusters form the first hierarchy. In the second hierarchy, a backbone is necessary to interconnect

the clusters. This structure allows concurrency: copies of a same packet can be disseminated concurrently in different clusters.

Concurrency is important for our effort to improve the delay performance of ring-based multicast. In a ring-based protocol, a packet visits receivers in a sequential order. No two receivers can receive the same packet at the same time. This is the essential reason for the weakness of ring-based multicast.

Although a tree structure may help to significantly achieve better concurrency, introducing a tree topology into the hierarchical structure should not weaken the control on multicast traffic gained in the ring-based approach. There are two basic types of hybrid structures: (i) a backbone tree interconnecting multiple rings, and (ii) a backbone ring interconnecting trees. As its name suggests, RITA adopts the first choice, and for the following reasons. First, a backbone ring may still be lengthy when spanning a wide area to reach domains, hence it may not decrease multicast delay significantly. Second, the strength of ring-based protocol lies in its capability of access control of multicast traffic. The backbone ring design loses the access control of multicast sources.

Besides these two considerations, our observation of loss correlation on a multicast tree suggests multicast flows across backbone tree links are less congested than in clusters. With the ring-based protocol's control at each cluster, the tree-based protocol can be sufficient for reliability control. Furthermore, RITA provides a solution to avoid the *Loss Path Multiplicity* problem [10]. In the following sections, we discuss these issues in detail.

4.1.1 Loss Correlation on Multicast Tree

The decision to use a tree structure in the backbone hierarchy is based on one essential property of the group multicasting. Multicast flow condition on tree links is different in the two cases of group multicasting and single-source multicasting, i.e. many-

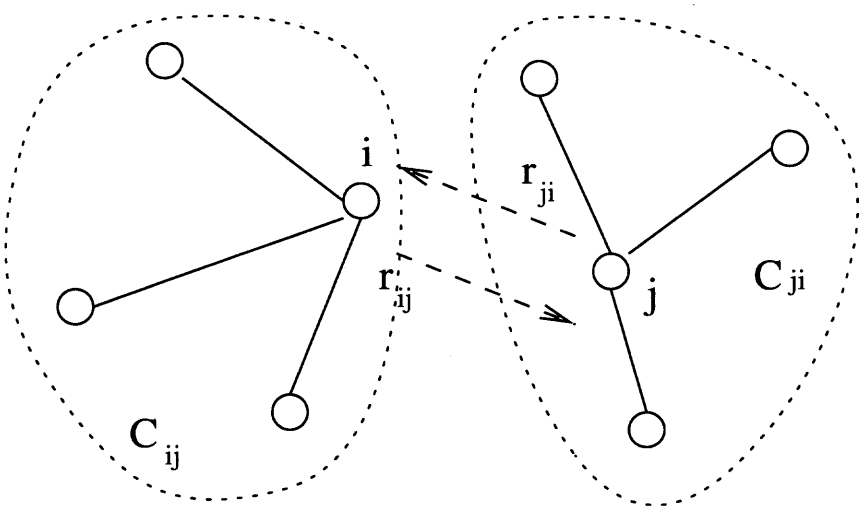


Figure 4.1 Any tree link connects two subtrees with different load flow in two directions.

to-many and one-to-many respectively. In case of a single-source session, multicast traffic flows in only one direction on each tree link. The multicast traffic load is uniform over all tree links. With many-to-many multicasting on a shared tree, traffic flows in different directions asymmetrically. The asymmetry results in uneven load correlation in different places of a tree.

4.1.1.1 Dominant Direction Load: Consider a tree $T = (M, E^T)$, where every node $i \in M$ is considered a potential multicast source with average sending rate $\lambda_i \geq 0$. Let $e_{ij} \in E^T$ denote the link between node i and j. Multicast flows in the two directions on any tree link are asymmetric, and here we notate r_{ij} as the traffic flowing from node i to node j on link e_{ij}, and r_{ji} as the traffic flowing in the other direction.

Any link e_{ij} in fact bridges two subtrees of T. Let $C_{ij} = (V_{ij}, E_{ij})$ be the subtree of T and $i \in V_{ij}$ and $C_{ji} = (V_{ji}, E_{ji})$ be the other subtree, where $j \in V_{ji}$ (Figure 4.1). We can thus obtain the traffic load on link e_{ij} with the following:

$$r_{ij} = \sum_{k \in V_{ij}} \lambda_k$$

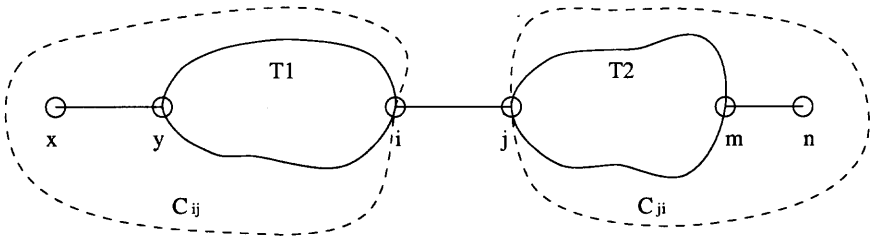


Figure 4.2 For an intermediate link e_{ij}, we may find an arbitrary leaf link e_{xy} in C_{ij}, and an arbitrary leaf link e_{mn} in C_{ji}. Suppose T_1 is the subtree connecting y and i, while subtree T_2 connects j and m.

$$r_{ji} = \sum_{k \in V_{ji}} \lambda_k$$

Notice that $M = V_{ij} \uplus V_{ji}$ and $V_{ji} = M \setminus V_{ij}$. Therefore the group's aggregated traffic Γ is divided into two directions according to the link's position in the tree:

$$\Gamma \equiv \sum_{k \in M} \lambda_k = r_{ij} + r_{ji}, \forall i, j$$

where r_{ij} and r_{ji} are not necessarily equal.

This asymmetry usually is most obvious at those links attaching to leafs (referred to as leaf links hereafter, and intermediate links for other links). A leaf x multicasts with rate λ_x and receives traffic with rate:

$$r_x = \sum_{i \in M, i \neq x} \lambda_i = \Gamma - \lambda_x$$

Therefore the leaf link at x has flows of r_x and λ_x in its two directions. Let us make the following definition.

Definition 3 *The* dominant direction load *(DDL) is the greater multicast load of the two opposite flows of a link* e_{ij}*:*

$$DDL_{ij} = max(r_{ij}, r_{ji})$$

Consider Figure 4.2, for intermediate link e_{ij}, we may find an arbitrary leaf link e_{xy} in C_{ij} with x as the leaf, and an arbitrary leaf link e_{mn} in C_{ji} with n as the

leaf. Suppose T_1 is the subtree connecting y and i, and subtree T_2 connects j and m. We may derive the following:

$$r_{yx} = \sum_{k \in T_1} \lambda_k + r_{ji}$$

$$r_{mn} = \sum_{k \in T_2} \lambda_k + r_{ij}$$

Therefore,

$$DDL_{ij} \leq max(r_{yx}, r_{mn})$$

and since $r_{yx} \leq DDL_{xy}$, and $r_{mn} \leq DDL_{mn}$:

$$DDL_{ij} \leq max(DDL_{xy}, DDL_{mn})$$

Therefore we conclude the following: for any intermediate link e, there exists at least one leaf link whose DDL is no less than the DDL of e.

The minimum DDL should be half of the total traffic from all multicast sources. When a link has the minimum DDL, sources at both of its ends generate an equal amount of traffic. When there is a difference in the two opposite flows, DDL is greater than the minimum. When there is no source at one end of a link, the DDL reaches the maximum, which is the total traffic Γ, i.e.

$$\frac{\Gamma}{2} \leq DDL \leq \Gamma$$

Generally, if multicast sources are homogeneous, multicast traffic is more likely to be evenly divided at a link close to the center of multicast tree, because sources are more likely to be evenly divided at such links. Thereby, there is a basin-like syndrome: outskirts of a multicast tree (i.e. links close to leafs) have relatively higher DDL, while around the middle of the tree DDL's are lower. The depth of the basin depends on many factors such as the number of multicast sources, the space distribution of multicast sources (positions on the tree), and the rates of all multicast sources.

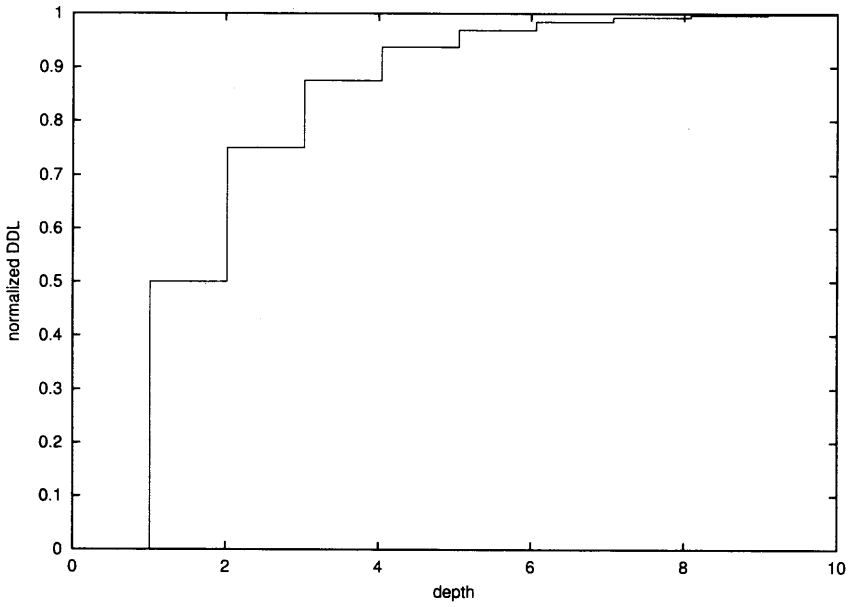


Figure 4.3 Multicast traffic load in correlation with binary tree layers.

To illustrate how uneven the DDL can be on a many-to-many multicast tree, let us use the binary tree as an example. Suppose a binary tree of depth n (i.e. $2^{n+1}-1$ nodes), where each leaf node is a source with the same mean traffic rate λ. If a link is at the ith layer from the root, its DDL should be:

$$DDL_i = [(2^n - 1) - (2^{n-i} - 1)] \cdot \lambda = 2^n \lambda \cdot [1 - \frac{1}{2^i}]$$

Since the overall traffic of the group is $\Gamma = (2^n - 1)\lambda$, when n is sufficient large such that $2^n >> 1$,

$$\frac{DDL_i}{\Gamma} \approx [1 - \frac{1}{2^i}]$$

Therefore we may have the multicast load correlation with tree depth shown in Figure 4.3. In this figure, the normalized DDL is plotted as a function of the depth of links. A link directly connects the root (i.e., depth = 1) and has half of the whole group's traffic as its DDL. When a link is farther from the root, its DDL is closer to the total traffic of the whole group. Notice that this relation is also true even when all nodes are sources.

Heavy load may cause packet switch buffer overflow, which is the major reason for packet losses. Generally, packet-loss probability at a link is an increasing function of the load of the link. Therefore, the significance of DDL is that it indicates the possibility of congestion due to multicast traffic. We may conclude from the above discussion that leaf links are most likely to be congested by multicast traffic.

RITA addresses the reliability problem in the following way. With its hierarchical structure, the task is actually divided into two sub-tasks: reliable multicast in each cluster and reliable multicast on the backbone. By embedding multicast rings to links around leafs, we may use a ring-based protocol to take care of high load links. More importantly, all sources can watch multicast load conditions of the whole group, by getting feedbacks from their local rings. Hence, multicast traffic can be regulated. With high load links being taken care of in clusters, other links with relatively low load are left as backbone links. In the backbone, the reliability task is only to ensure delivery among clusters.

4.1.1.2 Avoiding The Loss Path Multiplicity Problem: If the loss probability from a source to one of its multicast destinations is p and loss events on different links are all independent, then a packet can be successfully multicast to all m destinations with probability $(1-p)^m$. When m is large enough, the successful multicast probability can be much smaller than the successful unicast probability. In other words, the chance that at least one receiver fails to get the packet increases rapidly with small increase of single path loss probability p (see Figure 4.4). This is identified as the *Loss Path Multiplicity (LPM)* problem in [10].

RITA provides a solution to this problem because it helps to reduce the two parameters that affect the overall loss probability of multicast. First, it reduces the number of destinations (i.e. m) on a multicast tree. While reliability in each cluster is a task of the ring-based protocol, the reliability across the backbone tree only

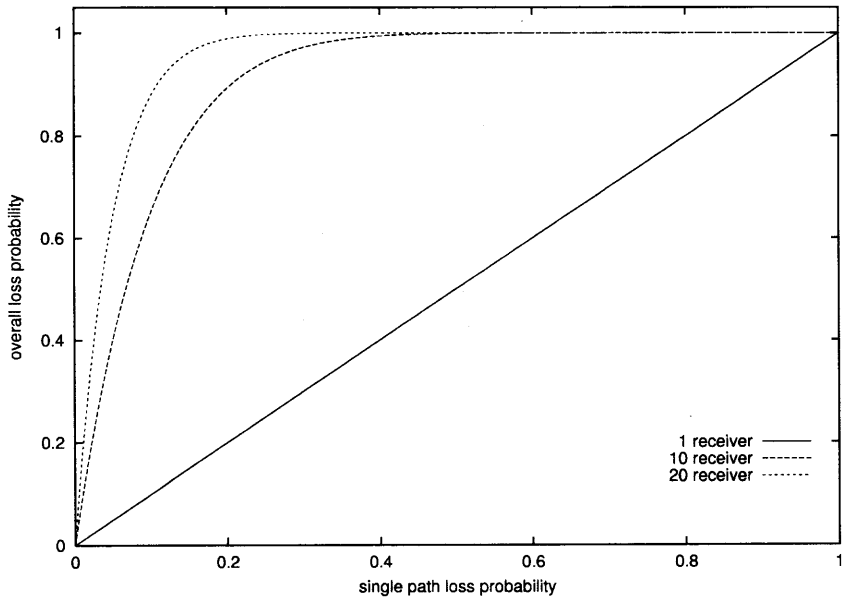


Figure 4.4 The loss path multiplicity problem.

requires delivery be guaranteed between clusters. Notice that the number of clusters is much smaller than the member set size m. Hence, the overall loss probability on the backbone can be effectively reduced. Second, RITA reduces the single path loss probability (i.e. p) on the backbone tree. A path between two clusters should be shorter than a path between two member nodes. Moreover, as we have discussed in above, because the backbone tree is formed with links that are more lightly loaded, the loss probability of a backbone path can be significantly lower than a path between a pair of members.

With m and p reduced, packets can be multicasted with a higher success probability using a tree-based protocol.

Let us look at an example to illustrate how RITA can avoid this LPM problem. Consider the binary tree shown in Figure 4.5. The 16 leaf nodes ($s1$ to $s16$) are multicast sources, each multicasts with the same rate λ. If the multicast is performed directly with this tree, the longest path between two members (e.g. from $s1$ to $s16$)

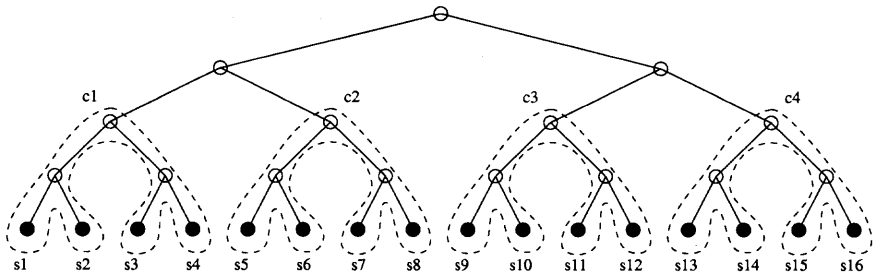


Figure 4.5 A binary tree with leaf nodes as multicast sources, RITA can be constructed by embedding rings on lower layers.

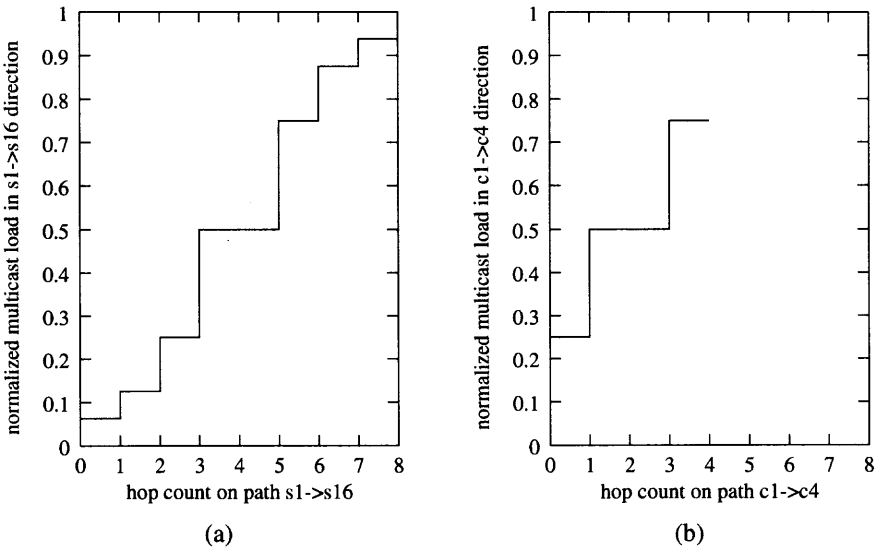


Figure 4.6 Multicast load increase pattern on a tree path from source to one destination.

has 8 hops. On the first link of this longest path, in the direction from $s1$ to $s16$ [1], multicast flow is λ, which is comprised of traffic from $s1$ only. Since the total group traffic (Γ) from these 16 sources is 16λ, the load on the first link of the path from $s1$ to $s16$ is therefore $\frac{\Gamma}{16}$. On the second link, as $s1$ and $s2$ both send packets in this direction, the load steps up to 2λ, which is $\frac{1}{8}$ of the total group traffic. Therefore, as shown in Figure 4.6.(a) on this path from $s1$ to $s16$, the multicast load steps from $\frac{\Gamma}{16}$ up to $\frac{15\Gamma}{16}$ in the consecutive links.

[1] In the other direction on this link, multicast flow is the mainstream because it contains traffic from all other sources

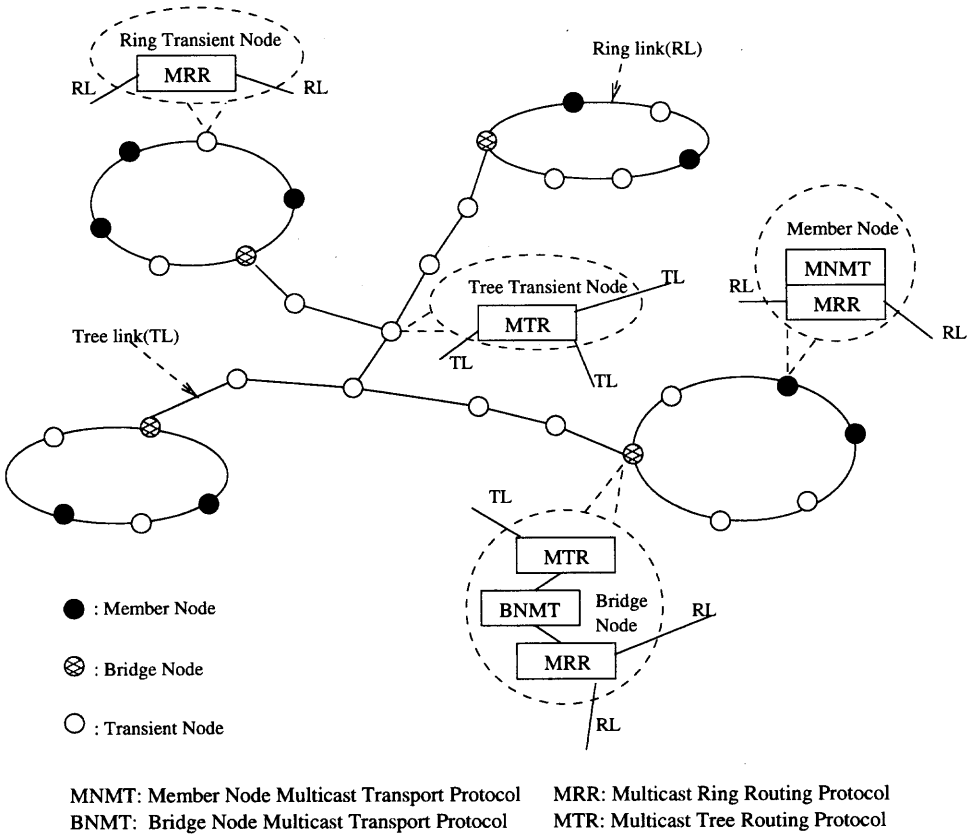


Figure 4.7 RITA structure and components.

If RITA is embedded on the same tree (as shown in Figure 4.5) with dashed lines as cluster rings [2], the backbone tree is smaller. Between two clusters, the longest path has 4 hops only (e.g. from $c1$ to $c4$). From $c1$ to $c4$, the multicast flows in the four subsequent links are $\frac{\Gamma}{4}$, $\frac{\Gamma}{2}$, $\frac{\Gamma}{2}$, and $\frac{3\Gamma}{4}$ respectively (shown in Figure 4.6 (b)). It is obvious that the loss probability of such a path should be lower than that of the path from $s1$ to $s16$. Considering the fact that the backbone has only four clusters as the destinations, instead of 16 members, we can expect significant improvement to the multicast success probability on the tree. This is verified with our simulation based performance study, which we present in a later section.

4.1.2 RITA Components

RITA is built with two hierarchies (Figure 4.7). In the first one, a multicast group is partitioned into small clusters. [3] In each cluster, a multicast ring is built to connect local members. In the second hierarchy, a backbone multicast tree is set up to interconnect these rings. Each leaf of this tree is correspondingly also connected on a cluster ring.

There are three types of nodes in RITA:

Member Node (MN): A *member node* is a multicast group member and is also a potential source. Its transport-layer protocol Member Node Multicast Transport (MNMT) is a ring-based reliable multicast protocol. At the routing layer, the routing protocol should support ring-based multicast routing;

Bridge Node (BN): A *bridge node* interconnects one cluster ring and the backbone tree. It handles the traffic from and to the backbone tree. Thus it acts like a gateway between the local ring and the backbone. The transport-layer protocol performed by a BN is called Bridge Node Multicast Transport (BNMT). A bridge node supports, at the network layer, both tree- and ring-based multicast routings. These routing protocols are not specified in this work (any one of the previously proposed routing protocols can be deployed), since the focus is on the transport protocols for reliable multicasting.

Transient Node (TN): *Transient nodes* are not the members of the multicast session, but used to form the backbone tree and ring paths. A transient node can be either a ring switch or a backbone tree switch.

---

[2]Embedding a tour on the subtree is the simplest way to construct multicast ring in a cluster. It is sufficient for our example, although a TSP ring can be more efficient.

[3]The partitioning of a multicast group can be a topic for further study. The Internet can usually be partitioned into domains. Therefore it is natural to partition a group based on participating domains. However in subsequent discussion of this chapter, we may also see that the partitioning is in fact an optimization problem w.r.t. cluster size.

For convenience, we also refer to nodes on rings as ring nodes, and to nodes on the backbone tree as tree nodes. Notice that a RITA node may not necessarily be a single host. Routing and transport protocols may reside on separate devices.

This dissertation focuses on the transport protocols that ensure multicast reliability. MNMT and BNMT protocols are discussed in the subsequent section. Existing protocols and technologies can be used for the underlying routing.

In RITA, each cluster can be viewed as a super source node, with the bridge node enforcing the reliability and each cluster/ring having a unique ID. The transient nodes simply forward multicast packets. If a multicast packet arrives, the node should forward the packet to other links associated with the same group. An intermediate ring node always has only one link to forward to, while an intermediate tree node may forward a packet to more than one link.

A sender may easily detect packet losses, by watching returning packets (i.e., implicit ACKs). Early congestion on certain cluster rings can be noticed by local members and bridge nodes quickly, and senders thereby may adjust their rate accordingly. In this way, further congestion on backbone tree is avoided. Meanwhile, losses can be recovered quickly within the congested ring without bothering other areas of the same group.

We also can see that there is a trade-off for cluster sizes. As the ring size increases, so does the delay for the feedback and the loss probability $P_R(loss)$ in the ring [4]. Therefore, we identify an optimization problem — to find the optimal number of rings and ring size to maximize the performance of the multicast protocol. Here we focus on the protocols and their performance study and leave the formal

[4]Let p_l be the probability with which a ring link drops packets under traffic load Γ. Because there are $|R_i|$ hops on ring R_i,

$$P_R(loss) = 1 - (1 - p_l)^{|R_i|}$$

which increases as the size of the ring gets larger.

treatment of the optimization problem for future work. However, we establish the following definition of optimality:

Given a multicast group M in network N, if a RITA can be found according to a partition $C = \{C_i : \uplus C_i = M\}$, this RITA can be denoted as $RITA(M, C)$.

Definition 4 *An optimal $RITA(M, C^*)$ for a given multicast M should have (1) no packet loss due to congestion, and (2) contains the maximum number of rings.*

Thus a multicast tree T with all members as its leaves can be perceived as a RITA with number of rings equal to the number of multicast members, i.e., $RITA(M, M)$. Our simulations show that packet losses on tree links can be reduced virtually to zero by employing the rings (see Figure 4.20). Since the loss in backbone tree links is negligible, the packet loss for the many-to-many multicast session is minimized. While maintaining this congestion-control property, we increase the number of rings to a maximal value (i.e., ring sizes are minimal).

As mentioned earlier, reliability in RITA is realized if the following tasks are accomplished: (i) a source is able to reliably multicast its packets on its local cluster ring; (ii) a bridge node is able to reliably multicast any packets from its local sources to other bridge nodes; (iii) a bridge node is able to reliably multicast packets from outside to its local cluster. We can see that these responsibilities should be enforced by protocols running in each multicast member node and bridge node. In the next section, we discuss the design of these protocols.

4.2 RITA Based Reliable Group Multicast Protocols

Member nodes and bridge nodes have different tasks to accomplish the multicast reliability. Transport protocol at a member node (i.e. MNMT) should do the following: (i) send data packets and make sure they reach every node on the same

ring; (ii) retransmit once its packets are found to be lost; (iii) adjust rate when network congestion is noticed, or when more bandwidth is available.

The basic tasks for a bridge node are: (i) forward ring traffic to multicast tree and perform necessary retransmission when requested; (ii) forward external traffic to its local ring and perform necessary retransmissions when losses are detected; (iii) inform member nodes about congestion detected at elsewhere in RITA. It is not the bridge node's task to cache traffic from either the local ring or the backbone tree to regulate traffic rate. Multicast sources are the only places that enforce traffic regulation.

Next we present the transport protocols, MNMT and BNMT, respectively for member and bridge nodes of RITA.

4.2.1 Member Node Multicast Transport Protocol

MNMT is an enhanced version of the RBRM from the previous section. Although RBRM is able to perform the first two tasks above, RBRM does not specify a dynamic window size adjustment mechanism. Such a mechanism is essential for RITA, because backbone congestion avoidance depends on congestion control within each individual ring. We now introduce two MNMT versions, based on different adaptive window control schemes, and their effectiveness is studied subsequently.

4.2.1.1 MNMT-1: The first version (MNMT-1) adopts a scheme similar to TCP [66] to adjust sender windows. A sender starts with a small window size, initially in probing status. Every time a packet successfully returns, the window is enlarged by a small value dw_{up}. When a packet is lost, the sender makes transition to congestion avoidance state, and the window is decreased by dw_{down}. Generally, dw_{up} and dw_{down} should be functions of the following parameters: current window size $wsize$, number of multicast senders m, and current protocol status $stat$.

```
if ( packet returns with no loss )
    if ( stat == PROBING )
        wsize := wsize + 1/wsize
    else wsize := wsize + 1/(wsize * m)
if ( one packet is lost OR LOST received )
    resend the lost packet
    wsize := max(wsize − 1, 1)
    stat := CONG_AVOID
if ( NACK received )
    wsize := max(wsize/2, 1)
    stat := PROBING
```

Figure 4.8 MNMT-1 window adjustment algorithm.

We do not intend to justify optimal window adjustment in this dissertation. There are intensive research works and arguments on TCP's window adjustment [75, 3, 46, 82, 13]. There are also works on window-based control for one-to-many multicasting, such as [36]. Window adjustment in the context of many-to-many multicasting is more complex, hence requires further exploration.

In our performance study presented in next section, we chose $dw_{up} = 1/wsize$ when protocol is in the probing state, $dw_{up} = 1/(wsize*m)$ for congestion avoidance, and $dw_{down} = 1$. The reason for these choices is that the protocol is for many-to-many multicasting, and a small adjustment at every sender can aggregate into a big overall impact. Therefore, unlike TCP, dw_{up} is relatively small for each multicast sender. Suppose a sender's window at a certain moment is w. If there are x packet losses during one ring circulation time t, then there are at least $(w - x)$ packets which still can pass through the network. Hence, $dw_{down} = 1$ for every packet loss can decrease the window from w to $(w - x)$ in a period of t. The algorithm outline is shown in Figure 4.8.

```
if ( KAM packet returns )
  update circulation time t
  if ( t > t_max ) t_max := t
  if ( t < t_min ) t_min := t
  for every four KAM returns
    if (t < alpha * t_min + (1 - alpha) * t_max)
       wsize := wsize + 1/wsize
    else wsize := wsize - 1/wsize
if ( one packet is lost OR LOST received )
    resend the lost packet
    wsize := max(wsize - 1, 1)
    stat := CONG_AVOID
if ( NACK received )
    wsize := max(wsize/2, 1)
    stat := PROBING
```

Figure 4.9 MNMT-2 window adjustment algorithm.

4.2.1.2 MNMT-2: Studies [46, 81, 82] show that the sender's window can be adjusted before congestion happens, instead of using packet losses to detect congestion and trigger a window decrease, as in MNMT-1. The idea is to consider round-trip delay change or throughput change with regard to window changes.

MNMT-2 emulates the algorithm presented in [82]. It is based on delay-based adjustments to the window control. The protocol uses round-trip delay as an indication of congestion status. Window size is regulated so that the round-trip delay is maintained under a certain threshold.

In Figure 4.9 we show the main steps of the algorithm tailored for many-to-many multicasting. In the figure, parameter *alpha* is used for performance tuning and is set as 0.5 in our simulations.

In our simulation study, we compared the performance of MNMT-1 and MNMT-2 to see the impact of delay-based adjustment to the window size in group reliable multicasting.

4.2.2 Bridge Node Multicast Transport Protocol

4.2.2.1 Forwarding Local Traffic to Backbone: For efficiency and simplicity of loss-recovery operations, each bridge node keeps a global unique cluster identification *cid* to represent its local cluster. A cluster table is maintained to record other participating clusters. When a packet is forwarded from ring to backbone, it is prepended with the local *cid* by the responsible BN. Also prepended with the *cid* is a sequence number *cseq*, which is locally unique in the cluster. This *cseq* is used to mark each packet arriving at the BN from ring independent of their origins. Thus they provide a global ordering and sequencing of the multicast packets leaving a cluster.

A copy of each sent packet is stored in a buffer (referred as t_buf), with the corresponding *cseq* as the index for retransmission. Since a member on a local ring may send the same packet more than once due to loss on the local ring, a bridge node may receive duplicate packets. Thus the bridge node must check duplicates before forwarding a newly arrived packet to the backbone. Therefore, t_buf should be also hashed with packet source ID's and packets' original sequence numbers. Below is an outline of the sending procedure, when a BN receives a packet from a local source. The packet is notated as $p[src, seq]$ to mean it is from src with sequence number seq.

In order to further prevent a BN from becoming a bottleneck, a BN's operations are minimal. Forwarding local ring traffic and external traffic to and from the backbone tree is immediate. No flow control and transmission buffering are performed at bridge nodes so that delay at a BN is hence minimized.

```
if ( p[src, seq] arrives from ring AND
     p is not found in t_buf )
  prepend [cid, cseq] to p
  send p to tree
  t_buf[cseq] := p
  cseq := cseq + 1
```

Figure 4.10 Algorithm for BNMT to handle local traffic.

```
send_to_ring( p[ci, cs] )
for x between last[ci] and cs
   send NACK[ci, x] to tree
   nack[ci, x] := TRUE
   set timer T_NACK[ci, x]
```

Figure 4.11 Backbone traffic handling with BNMT.

4.2.2.2 Backbone Loss Detection and Recovery: A NACK-based approach is used for loss detection and recovery. The interaction between bridge nodes over the backbone tree is similar to the SRM protocol [34] and TBRM. However, a simpler loss-recovery mechanism can be used at a bridge node because of the congestion control deployed at the local rings.

Upon receiving a data packet from a particular cluster, a BN checks if there is a sequence gap between the new packet and the last seen packet from the same bridge node. A gap indicates losses and the recovery procedure is started. A BN makes use of the cluster id and the cluster sequence number (i.e. *cid* and *cseq*). Since *t_buf* of a BN keeps copies from its local sources, retransmission can be performed at the BN instead of involving original sender. In Figure 4.11 we show the outline of the receiving procedure for $cid = ci$ with cluster sequence $cseq = cs$, and how the NACK procedure is started.

The bridge node forwards the received packet immediately to its local ring. It looks up the cluster table to check the last received sequence ($last[ci]$) from *ci*. A

NACK message is sent to the backbone for every lost packet with sequence x between $last[ci]$ and cs. A flag $nack[ci, x]$ is kept for further loss recovery operations. More details of procedure *send_to_ring*() are discussed later in this sub-section.

The NACK serves as three purposes: (i) to request retransmission from the BN who sends the missing packet; (ii) to warn every multicast sender that a packet is lost on tree so that senders may reduce their traffic accordingly; (iii) to suppress NACKs from other BNs for the same missing packet.

NACKs are sent immediately after detection of loss. This is intended not only to speed loss recovery, but also to speed senders' response to adjust their sending rates. Because a NACK message indicates backbone congestion, senders should be informed as soon as possible. Therefore upon receiving a NACK message, a BN forwards it to the local cluster ring. A multicast source responds to a NACK by half its window size in order to recover tree congestion.

It is possible that multiple NACKs are sent from different bridge nodes to request the same packet. Since only bridge nodes, which are much less than member nodes, generate NACKs, and loss on the backbone tree is low, the NACK implosion is not a concern.

When a NACK is sent, the sending BN starts a timer in case NACK or the corresponding repair is lost. When this timer expires, another NACK is sent. The timer is initially given a value three times as the one way distance (i.e., one round trip + extra) from the origin bridge nodes. The value is doubled each time the timer expires. The timer will be canceled once the repair packet arrives. Also, if a NACK requesting the same data is heard from the backbone, the timer is reset as if it just expired.

Loss recovery needs cooperation among bridge nodes. Upon receiving a NACK request, a bridge node sends a repair only if the requested packet originates from its own cluster. After a repair is sent, the bridge node will set a timer, which is called the

```
if ( NACK[ci, cs] received )
  if ( ci == cid AND ignr[cs] == FALSE )
    send_to_tree(REPAIR[cid, cs])
    set IGNORE timer T_IGNR[cs]
    set ignr[cs] := TRUE
  else if ( nack[ci, cs] == TRUE )
    reset timer T_NACK[ci, cs]
  send_to_ring( NACK[ci, cs] )
if ( REPAIR[ci, cs] arrives )
  if ( nack[ci, cs] == TRUE )
    send_to_ring(REPAIR[ci, cs])
    cancel T_NACK[ci, cs]
    nack[ci, cs] := FALSE
if ( timer T_NACK[ci, cs] expires )
  send_to_tree( NACK[ci, cs] )
  reset timer for T_NACK[ci, cs]
if ( timer T_IGNR[cs] expires )
  set ignr[cs] := FALSE
```

Figure 4.12 BNMT backbone error recovery algorithm.

ignore timer. Before this timer expires, the bridge node should ignore any duplicate NACKs requesting the same packet. The timer value can be twice the delay from the NACK sender. The NACK and repair algorithm is illustrated in Figure 4.12.

4.2.2.3 Reliable Delivery of Backbone Traffic to a Local Cluster: A bridge node performs the following operations on its ring: it (i) forwards packets from other BNs to the ring immediately without any flow control, (ii) detects loss (using the implicit feedback mechanism), (iii) retransmits lost packets, and (iv) removes the packets from the ring. Its main difference from the local member nodes is that window-based flow control is not enforced.

We note that there is no need to re-assign a sequence number to each packet from the tree. Instead, a bridge node maintains another buffer (referred as r_buf). Once a packet p comes from the backbone, the procedure $send\_to\_ring(p)$ does the following: appends a copy of p to tail of r_buf, and sends p to the ring. In this way, packets from the backbone are kept in r_buf according to the order in which they are put on the ring. If the returning packet is the first in r_buf, there is no loss and the buffered copy can be removed from the head of r_buf. Therefore, a BN can detect packet losses if the returning packet is not the first in r_buf. For those packets that are in front of the returning packet in r_buf, they are removed from the r_buf and re-sent with the $send\_to\_ring()$ procedure.

4.2.2.4 Heterogeneous Rings: The window-based adaptive flow control enables a sender to increase its window size to fully utilize the ring capacity. However, available bandwidth for a cluster ring changes dynamically and can be different from that of the other rings. Loss detection and retransmission of packets from the backbone is the responsibility of the BN for this ring. Suppose some of the links used by a cluster ring get overloaded (say by the unicast traffic) and packets are

Table 4.1 Summary of Protocol Functionality.

	routing basis	flow control	reliability scope	loss detection & recovery
MNMT	MN, ring	window based	local ring	sender initiated with implicit ACKs
BNMT	BN, tree + ring	no control	local ring + backbone	tree: receiver initiated with NACKs ring: sender initiated with implicit ACKs

dropped. If there is no feedback to the source nodes that reside on the other cluster rings, the BN will have buffer overflow. Thus, a feedback mechanism is needed to inform the remote sources to adjust their rates accordingly.

Once a BN detects a packet loss on its local ring, it multicasts a control message (called LOST) for such feedback. Upon receiving the LOST message, a bridge node forwards this LOST message to its local ring and informs the members. A member node adjusts its window size as if the loss occurred on its ring.

In this way, ring congestion information is shared among rings and multicast senders may adjust their window sizes even if there is no local congestion. This LOST control message is simulated in the performance study of next section.

Fairness and throughput can be increased if each BN maintains a window for all other rings. However, such a solution will increase complexity at the BNs and may cause bottlenecks.

4.2.3 Protocol Summary

Overall, reliable multicasting on RITA is done with the cooperation among member nodes and bridge nodes. Table 4.1 summarizes the major functionality of the two protocols for RITA. Member nodes are responsible for reliable delivery in their own cluster. Once a packet is delivered in the local ring, the source assumes that the local BN will handle the reliable delivery of the packet to all other rings. If no NACK is

received, a bridge node assumes that other BNs received the packet correctly and forward it to their clusters. Finally, when a packet is received from backbone, a BN keeps track of it and makes sure it reaches everyone in its cluster.

Loss recovery is restricted either within a cluster ring or inside the backbone. Meanwhile, most packet losses are notified to the whole group so that senders may take action to reduce their multicasting rates. A sender is aware of three types of packet losses: losses of its own packet on the ring, by implicit ACK scheme; losses in the backbone tree, by arrivals of NACK messages; and losses in other clusters, by arrivals of LOST messages. All these types of knowledge help a sender to adapt its window size to the wide area network conditions.

The overall processes of MNMT and BNMT are illustrated with Figure 4.13 and Figure 4.14.

4.3 RITA Performance Study

With the same simulation procedure as in the previous chapters, we simulated networks with three clusters. In each cluster there are 32 nodes, and 11 multicast group members are randomly chosen from each cluster, making a session with 33 senders in total. Cluster rings are built separately. For wide-area, we use a star-like backbone tree to connect these three clusters. The center node of the tree is 5 hops away from each cluster. In other words, clusters are 10 hops away from each other.

One of our interests is to compare RITA with tree- and ring-based reliable multicasting. We construct a global multicast tree and also a global multicast ring to run TBRM and MNMT respectively. The global tree is constructed by building small trees within these clusters and concatenating these small trees with the backbone tree. Similarly, the virtual ring embedded on the backbone tree is merged with the 3 cluster rings into one large ring.

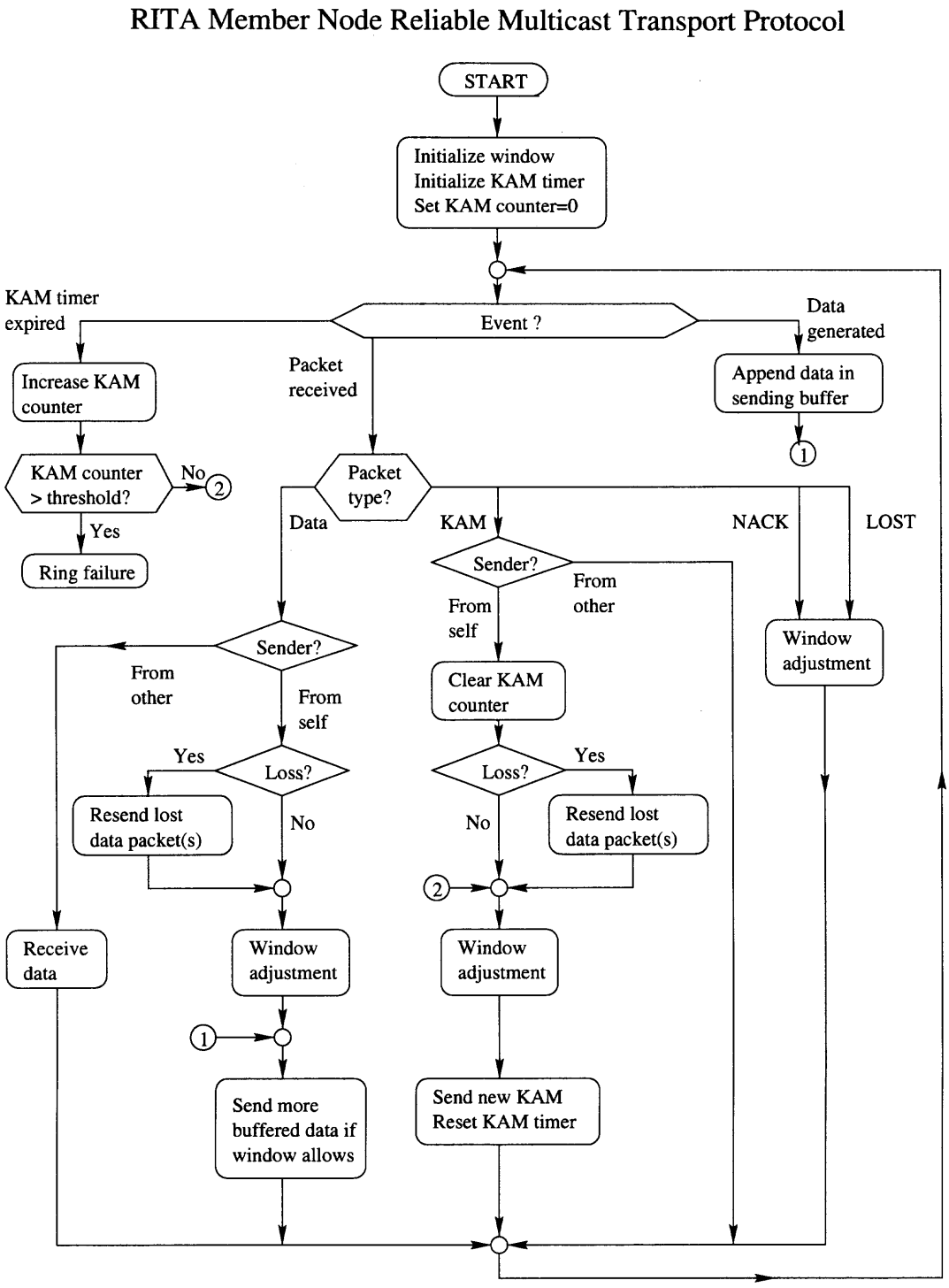


Figure 4.13 Procedure of RITA member node protocol.

RITA Bridge Node Reliable Multicast Transport Protocol

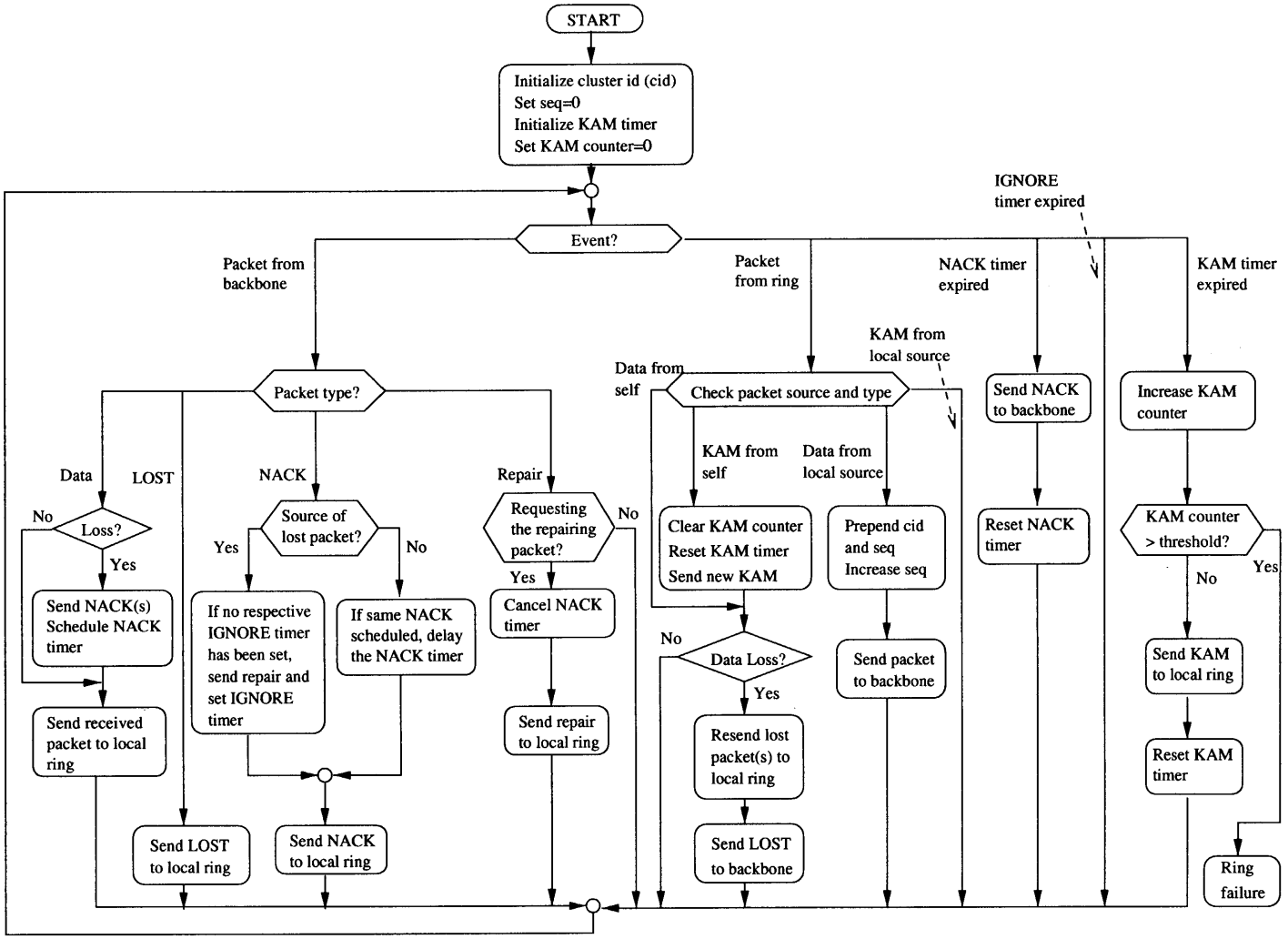


Figure 4.14 Procedure of RITA bridge node protocol.

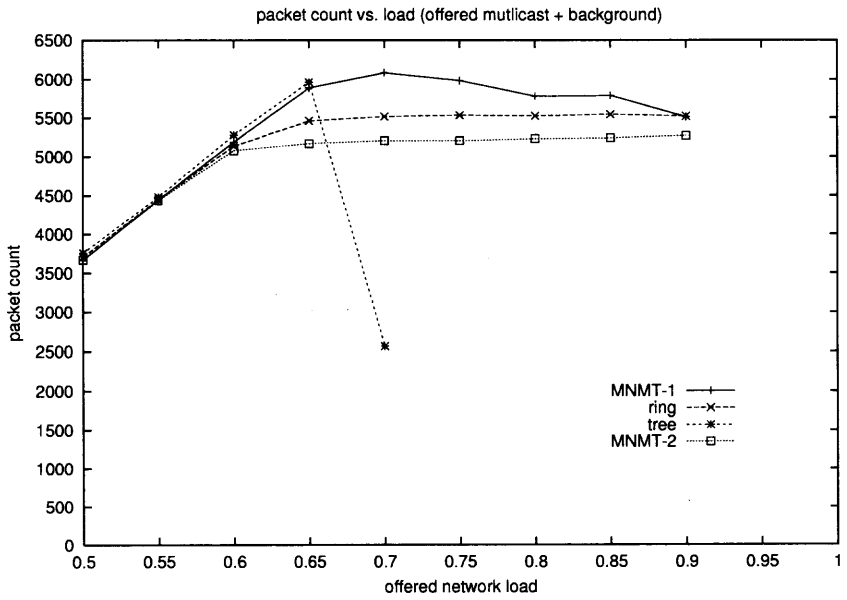


Figure 4.15 Goodput performance comparison: tree, ring, MNMT-1 and MNMT-2.

By changing cluster networks, we may derive different network instances. We used 20 different network instances in our simulations.

4.3.1 Goodput Performance

Figure 4.15 plots the goodput performance of MNMT-1 and MNMT-2. Recall that MNMT-1 senders use a basic adjustment algorithm, and MNMT-2 senders use a delay-based approach to adjust window sizes. Respective performances of ring-based and tree-based protocols are also plotted for comparison. The x-axis represents the network's offered load. Besides the average aggregated multicast traffic offered to reliable multicast protocols, the external traffic load is simulated in the same fashion as the performance study in the previous chapter. The external load is 0.25 in our experiments. The y-axis represents the number of successfully multicasted packets during the simulation period.

As shown in the figure, both MNMT-1 and MNMT-2 presents stable and good performance. There is no indication of congestion collapse. This means that introducing the backbone tree is successful. RITA uses a lightweight protocol for backbone

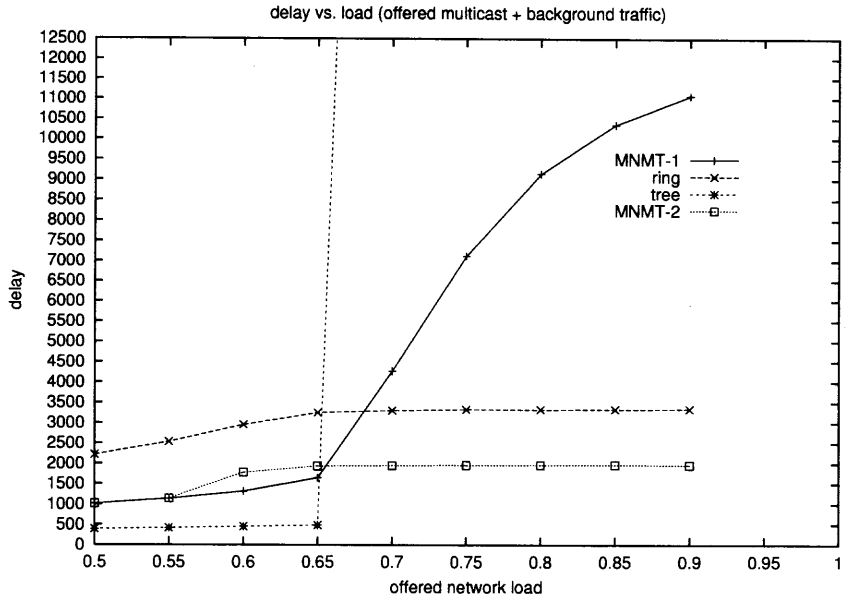


Figure 4.16 Delay performance comparison: tree, ring, MNMT-1 and MNMT-P2.

multicast, and it is sufficient for backbone reliability control. MNMT-2 has smaller goodput than MNMT-1, and this is because MNMT-2 does not encourage sender probing for maximum throughput.

4.3.2 Delay Performance

Figure 4.16 shows the delay performance comparison. Again, the x-axis is the network offered load, including external traffic. The y-axis represents the average multicast delay, which is the time needed for a successful multicast. At low network load, we may observe that MNMT-1 and MNMT-2 both show better performance than the ring-based protocol. When network load gets higher, delay of MNMT-1 begins to exceed the delay of ring-based protocol, while MNMT-2 is able to maintain its advantage over ring-based protocol. This shows that MNMT-1 is more aggressive than MNMT-2 in putting more packets into the network, resulting in more queuing delay and increasing loss probability. Because of losses, multicast delay of MNMT-1 is longer, and it increases with load.

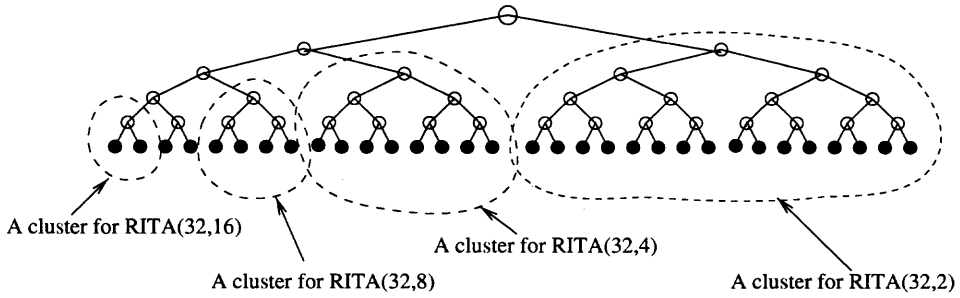


Figure 4.17 The four types clusters respective to RITA(32,16), RITA(32,8), RITA(32,4), and RITA(32,2).

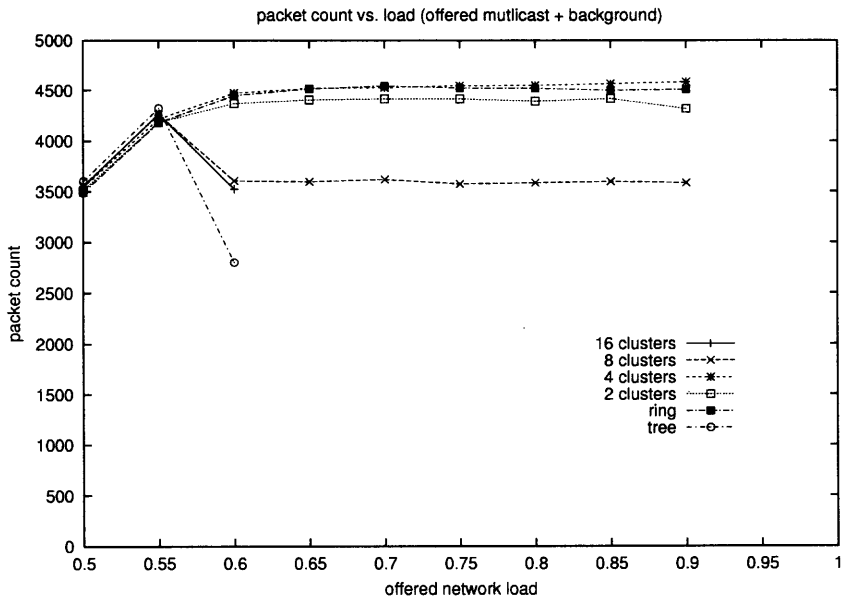


Figure 4.18 Goodput performances of different group partitionings.

It is obvious that MNMT-2 has better delay performance at high network load. It appears to show that the MNMT-2 window control algorithm is very effective in preventing network congestion. This is at a price because MNMT-2 shows lower goodput than MNMT-1.

We also surveyed the effect of external traffic. Generally, RITA performance shows the same pattern with varying external traffic. Increasing external traffic results in decreased multicast goodput and increased multicast delay.

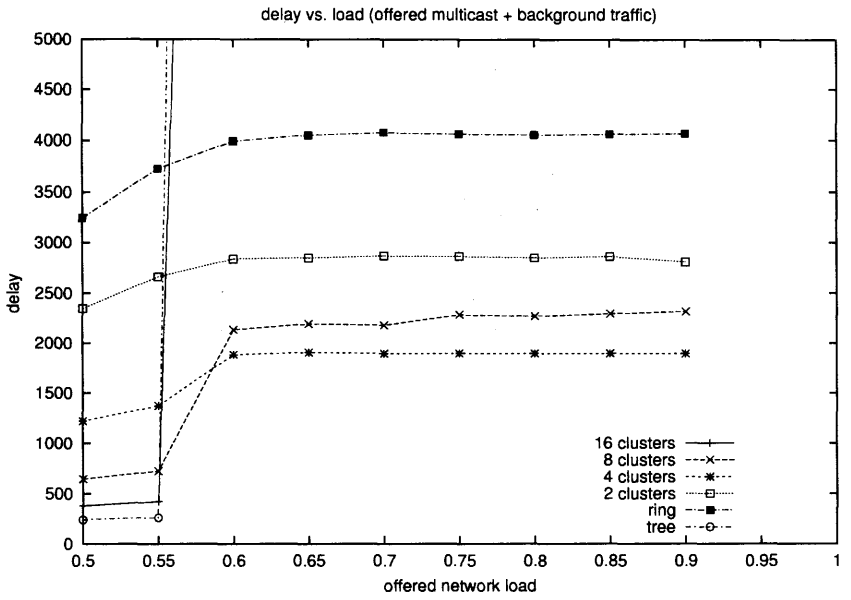


Figure 4.19 Delay performances of different group partitionings.

4.3.3 Effects of Group Partitioning

Since an optimal partitioning C^* may exist for $RITA(M, C)$, as mentioned above, we studied different RITA embeddings on a same binary tree with all leaf nodes as the multicast members. The height of the binary tree is 6. Therefore, the size of multicast group $|M|$ is 32. Just for clarity, we use $RITA(|M|, |C|)$ to denote the different configurations with which we experimented. For $RITA(32, 16)$, each cluster is formed by two member nodes which share a same parent node on the original binary tree, and this parent node as well. $RITA(32, 16)$ is constructed by embedding a virtual ring on each subtree, and connecting these 16 virtual rings with the rest of the binary tree. In a similar fashion, we also constructed $RITA(32, 8)$, $RITA(32, 4)$, and $RITA(32, 2)$. In figure 4.17, we show the four types of clusters respective to these four partitionings.

We simulated MNMT-2 on these four configurations. For comparison, we also simulated TBRM on the binary tree, which we notate as $RITA(32, 32)$, and MNMT-2 on the ring, which is embedded on the tree and notated as $RITA(32, 1)$.

Figures 4.18 and 4.19 show the result comparisons. We may observe that, $RITA(32,4)$ (four cluster configuration) shows both better goodput and delay performance than other partitionings. With too many clusters, RITA presents performance similar to tree-based multicasting. As shown in both figures, the 16-cluster configuration ($RITA(32,16)$) suffers severe congestion with combined network load 0.60. The delay with this load is very long, and the goodput starts to drop also. Further increase of the multicast load brings congestion collapse. With fewer clusters, RITA's performance is close to ring-based multicasting. Among the four RITA configurations, $RITA(32,2)$ has the longest multicast delay, although still significantly lower than the delay of the pure ring-based case.

Note that in Figure 4.19, $RITA(32,8)$ shows lower delay at low load than $RITA(32,4)$. However, as the load increases, the congestion control becomes more important. Thus, as $RITA(32,8)$ starts recovering from increasing packet losses, which is due to congestion, the multicast delay also increases and exceeds the delay of $RITA(32,4)$.

We now turn to statistics of packet losses, in order to verify our explanation of the delay.

4.3.4 Packet Loss Statistics

We observed packet losses on backbone trees by watching leaf link losses [5] of trees. The following configurations are observed: $RITA(32,32)$, $RITA(32,16)$, $RITA(32,8)$, $RITA(32,4)$, and $RITA(32,2)$. In Figure 4.20 the y-axis shows the average number of packets discarded per leaf link. We may see that configurations with more and smaller clusters are closer to $RITA(32,32)$, the configuration with a pure tree. We note that for $RITA(32,2)$ and $RITA(32,4)$ the impact of ring congestion control is remarkable, and packet losses are very few and have no

[5] Let us refer the backbone tree links ending at clusters of a RITA as leaf links.

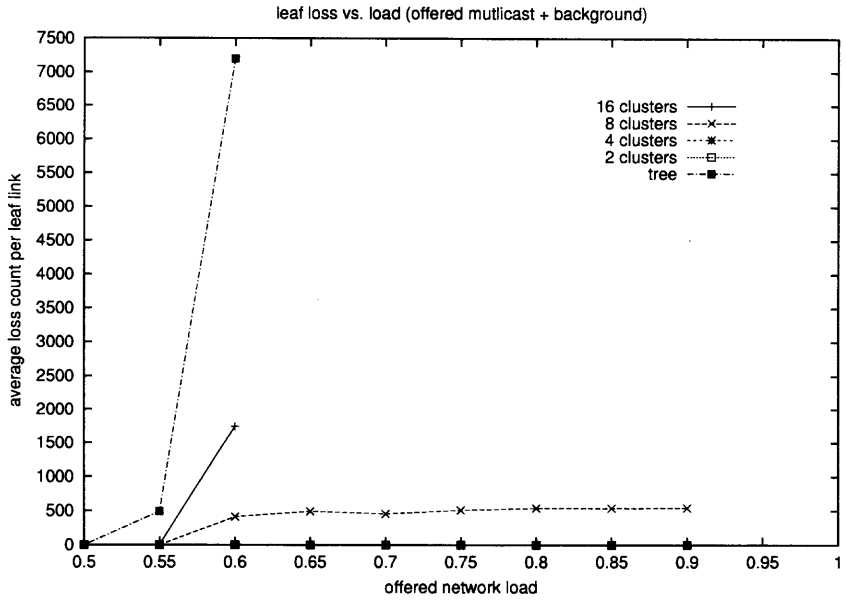


Figure 4.20 Different group partitioning effects backbone leaf link losses.

significant increase with the load increase. In general, when partitioning with a fewer number of clusters (and therefore, more members on a single ring), losses on backbone leaf links are fewer. Thus the optimality criteria for RITA is to have the maximal number of clusters for which no loss is due to congestion.

Furthermore, we compared the loss statistics of MNMT-1 and MNMT-2 under high-load conditions. With the RITA configuration used in the delay and goodput study (Figures 4.15 and 4.16), we observed the loss condition under multicast load 0.65 and external load 0.25. Numbers of packet losses in the backbone and clusters are recorded separately. Table 4.2 shows the average results only. We may conclude the following:

- The number of tree losses are much less than the ring losses in RITA. For MNMT-1, the ratio of ring losses over tree losses is 48.5; while this ratio is 505.3 for MNMT-2. This seems to verify that the low loss property can be achieved on backbone with RITA. Therefore, this can significantly reduce the complexity of packet loss handling on the backbone.

Table 4.2 Summed ring and tree losses with MNMT-1 and MNMT-2.

	number of tree losses (packets)	number of ring losses (packets)
MNMT-1	647.95	31407.3
MNMT-2	4.65	2349.7

- Furthermore, the performance of congestion control within clusters significantly affects the backbone congestion control. We may see that losses are more effectively inhibited with MNMT-2. We may see the ratio of ring losses with MNMT-1 over MNMT-2 ring losses is 13.4, while the ratio of tree losses with MNMT-1 over MNMT-2 tree losses is 139.3. Hence, the performance improvement over MNMT-2 is significant. With stricter traffic control, MNMT-2 not only reduces the ring losses very effectively, but also suppresses backbone congestion significantly.

4.4 Summary

The objective of this chapter has been to present a new architecture for reliable group multicasting. Our previous study shows that there are trade-offs between tree- and ring-based reliable multicasting. To exploit advantages from both paradigms, we introduce a hybrid multicast architecture (RITA). In RITA, multicast members are partitioned into clusters, rings are built within these clusters, and cluster rings are inter-connected by a backbone tree. This design facilitates multicast senders in their deployment of adaptive congestion controls, as in ring-based multicasting. Using a tree structure as the backbone, RITA may present better delay performance than pure ring-based multicasting.

Unlike protocols such as Lorax [52], RMA [50] or PGM [74], RITA is protocol independent. It does not require special design in the routing protocols to support reliable multicasting.

We discussed reliable multicast protocols for member nodes and bridge nodes in RITA. An adaptive window- and ring-based protocol was presented for ring congestion control. With this protocol plus the structural support of RITA, we successfully manage to control wide-area congestion through local congestion control. The multicast backbone can be relieved from congestion. Therefore, a simple lightweight tree and NACK-based protocol is sufficient for wide-area reliable multicasting.

With RITA and the associated protocols, the global multicast reliability is broken down into smaller regional reliabilities, and each node (member node or bridge node) is responsible for partial reliability only. With reliability ensured in each region, the global reliability is also ensured.

Simulation results show that RITA is effective for reliable multicast. It facilitates the use of less complex yet reliable multicast protocols. Compared to pure tree-based multicasting, RITA can achieve better goodput performance at high network load. Compared to pure ring-based one, RITA shows that with proper design of dynamic window adjustment, multicast delay can be shortened without sacrificing too much throughput.

We observed that cluster partitioning has significant effects on RITA's performance. To minimize delay, it is desirable to partition a group into more and smaller clusters. On the other hand, with fewer clusters, multicast traffic is much easier to control and therefore better throughput can be achieved. The optimal RITA should reach a compromise between throughput and delay by choosing proper cluster sizes.

CHAPTER 5

MULTICAST TREE PACKING

Congestion is the major cause of unreliability in a packet-switching network. Besides transport-layer congestion-control mechanisms, proper routing can also help reduce the risk of congestion. Generally, routing decisions made by multicast protocols ignore the fact that multiple concurrent multicast sessions may be present in a network.

With multiple concurrent multicast sessions in a network, competition for the same network resources between multicast groups may cause bottlenecks in the network, and therefore cause congestion. Optimizing multicast routing with cooperation between multiple multicast groups needs more attention and should be investigated.

In this chapter, we study the problem of optimally accommodating multiple multicast requests on a same network.

5.1 Overview

Let graph $G = (V, E)$ represent a network, and K be the set of multicast groups. Suppose t^k is the overall traffic demand of each multicast group $k \in K$. Let us define the congestion (or load) of an edge to be the total traffic demand summed over the

$$
\begin{aligned}
&\text{Min } \lambda \\
&\text{Subject to:} \\
&x_e^k \in ST^k \text{ for all } k \in K \\
&\sum_{k \in K} t^k x_e^k \leq \lambda \text{ for all } e \in E
\end{aligned}
$$

where λ denotes the maximum congestion and ST^k is defined as:

$$ST^k = \{x^k \in \{0,1\}^{|E|} : x^k \text{ induces a Steiner tree spanning } M^k\}$$

Figure 5.1 The formulation for the multicast tree packing problem.

K:	the set of multicast groups.
$k \in K$:	the kth multicast group.
M^k:	the set of member nodes of group k.
ST^k:	the Steiner tree spanning M^k.
t^k:	traffic demand from group k.
z_e:	the load (i.e., congestion) on link e.
λ:	the maximum congestion in the network.
$short^k$:	the size of the multicast tree for group k that uses the minimum number of edges.
x_e^k:	a binary variable noting whether $e \in E$ is used by group k for its multicast tree.

Figure 5.2 Notations for the multicast tree packing problem.

multicast groups using that edge. Our optimality objective is the minimization of the total load of the most congested edge in the network.

The integer programming formulations for this problem are presented in [18]. Here we recapitulate the problem defined in [18] as shown in Figure 5.1. The notations used in this chapter are listed in Figure 5.2.

In the formulation, binary variable x_e^k represents if link e is used in the multicast subgraph for group M^k. The congestion on link e is $z_e = \sum_{k \in K} t^k x_e^k$. To reduce λ means to reduce the maximum congestion over all links (i.e., $max_{e \in E}\{z_e\}$). Our approach is to re-configure ST^k for all $k \in K$ so that they avoid competing for a same link. Certain compromises may have to be made to meet this goal, because some ST^k may turns out to be non-optimum.

In this dissertation, we mainly present the multicast tree packing heuristics and the searching of lower bounds for the associated optimization problems. More details can be found in [18].

5.2 Heuristic Multicast Tree Packing Algorithm

Our approach to solve multicast tree packing includes two phases: (i) Given a graph $G = (V, E)$ as the network and K as the set of multicast groups, we first find the

Algorithm: TreePacking

Preprocessing:
For each multicast group construct the multicast tree independently.
Input: T and Δ^*
T: set of optimal multicast trees, $T = \{T_x : x = 1, \ldots, k\}$.
Δ^*: bound on the tree size.
Output:
a new set of revised multicast trees: $T' = \{T'_x : x = 1, \ldots, k\}$

BEGIN
1. Compute congestion values of each edge.
2. Sort all edges e by z_e into array LE in decreasing order.
3. Choose an edge e' with maximum congestion Z.
4. Try each tree $T_x \in T$ until a tree T_y can be found such that
   4.1 each edge in T_y has the congestion value no greater than $Z - 1$, AND,
   4.2 tree T_y can be **rebuilt** without exceeding size limit Δ^*.
5. Update z_e values in LE and go to step 2.
6. Stop if LE cannot be updated.

END

Figure 5.3 Outline of a heuristic for reducing congestion.

multicast tree solutions for each group $k \in K$. (ii) Let δ^k to be the ratio between number of multicast members in group k and the size of the corresponding multicast tree T^k. Define Δ to be the maximum δ over all multicast trees. Given a size limit Δ^*, we try to reduce the maximum congestion of the network by reconstructing the multicast trees so that z is reduced, while the new Δ satisfies: $\Delta \leq \Delta^*$.

5.2.1 Packing: Reducing the Maximum Congestion

Figure 5.3 describes the heuristic approach for packing multicast trees with minimum congestion. The heuristic takes as input a set of optimal multicast trees computed independently. The objective is to decrease the maximum value of congestion with minimum increase in the maximum tree size.

The first step is to calculate the congestion values on all links. In the next step, a sorted array LE is maintained to order the edges according to their congestion value.

From step 2 to step 5, the algorithm starts to search alternative tree solutions to avoid using the most congested link. Starting from the most congested link, heuristic chooses a link e and finds a tree which uses this link. The heuristic proceeds by removing a congested link from a multicast tree and thus disconnecting the tree. If an alternative tree can be found without using e and with δ value no more than Δ^*, this tree replaces the original tree. To achieve better performance, the alternative tree is searched in following way: if the most congested link (congestion value is Z) is used by tree T_x for group x, identify a set E_z of all edges which are not used by T_x, and whose congestion values are Z or $Z-1$. Re-calculate an alternative tree for group x by using edges in $E \backslash E_z$ only.

After all edges are attempted, if some improvement has been done, all edges are re-sorted. The algorithm repeats until no further improvement can be done.

Later in this section we show that this naive heuristic can perform quite well to reduce the maximum value of congestion.

5.2.2 Lower Bound for Multicast Tree Packing

Basically, we may choose a subset S of V and let $\bar{w}$, which is the vector of link weights to be assigned, be such that:

$$\bar{w}_e = \begin{cases} 1/|\delta(S)| & e \in \delta(S), \\ 0 & \text{otherwise.} \end{cases}$$

where $\delta(S)$ denotes the set of edges separating S from $V \setminus S$. According to *Lemma 1* in [18], a valid lower bound can be obtained with the equation which we cite as below:

$$\lambda^* \geq \frac{1}{|\delta(S)|} \sum_{k \in K(S)} t^k$$

where $K(S) = \{k \in K$ such that $M^k \cap S \neq \emptyset$ and $M^k \not\subset S\}$.

Furthermore, it can be extended to partitions of V involving more than two subsets. When $\Pi = \{S_1, S_2, \ldots, S_{|\Pi|}\}$ is the partition of V, we may obtain a valid lower bound with the equation below:

$$\lambda^* \geq \frac{1}{|\Delta(\Pi)|} \sum_{k \in K} t^k \cdot (s^k(\Pi) - 1) \tag{5.1}$$

where $\Delta(\Pi)$ is the set of edges bridging two subsets, and $s^k(\Pi)$ is the number of S_i's that contain at least one member of multicast group k [1]. Also notice that, when $\Pi = \{S, V \backslash S\}$, $s^k(\Pi)$ is 2 when $k \in K(S)$, otherwise it is 1.

With equation 5.1, we define below a procedure to find these partitions.

For $S \subset V$ define

$$\kappa(S) = \sum_{k \in K(S)} t^k$$

which, in some sense, measures the contribution of a subset to the partition lower bound, (recall that the definition of $K(S) = \{k \in K$ such that $M^k \cap S \neq \emptyset$ and $M^k \not\subset S\}$).

Given a partition $\Pi = \{S_1, S_2, \ldots, S_{|\Pi|}\}$ of V, for each pair of subsets (S_m, S_n) with at least one edge connecting them, define

$$\beta_{mn} = \frac{\sum_{l \in K(S_m) \cap K(S_n)} t^l}{\delta_{mn}(\Pi)}$$

where $\delta_{mn}(\Pi_i)$ is the subset of edges between S_m and S_n.

Notice that with partition Π,

$$\lambda_\Pi = \frac{\sum_{k \in K} t^k (s^k(\Pi) - 1)}{|\Delta(\Pi)|}$$

When S_m and S_n are merged to make a new partition Π', the lower bound can be updated as:

$$\lambda_{\Pi'} = \frac{\sum_{k \in K} t^k (s^k(\Pi) - 1) - \sum_{l \in K(S_m) \cap K(S_n)} t^l}{|\Delta(\Pi)| - |\delta_{mn}(\Pi)|}$$

[1] Therefore, the multicast tree k would contain at least $s^k(\Pi) - 1$ edges from the multicut $\Delta(\Pi)$

Algorithm: LBSearch

Input: Graph $G = (V, E)$ and multicast groups K.
Output: Multicast tree packing lower bound λ^*.

BEGIN
1. Let $\Pi_0 = S_1, S_2, ..., S_{|V|}$ where S_i is a singleton.
2. Let λ_0 be the value according to equation (5.1).
   This is the initial best bound λ^*.
3. Let $i = 0$
4. Repeat as long as $|\Pi_i| > 2$:
   - 4.1 compute α's for each neighboring subset pair.
   - 4.2 identify a pair (S_m, S_n) s.t. α_{mn} is the least.
   - 4.3 merge S_m and S_n, which results a new partition Π_{i+1}.
   - 4.4 compute λ_{i+1}, update the best bound if necessary.
   - 4.5 i=i+1

END

Figure 5.4 An algorithm for tree packing lower bound searching.

Therefore, if we choose S_m and S_n such that, $\beta_{mn} \leq \lambda_i$, we will get $\lambda_{\Pi'} \geq \lambda_{\Pi}$.

Furthermore, let us define:

$$\alpha_{mn} = \beta_{mn} + u \cdot (\kappa(S_m) + \kappa(S_n)) + v$$

where u and v are two random perturbations.

Our algorithm is therefore shown in Figure 5.4. It starts with the partition where each node forms a subset. With the computation of α values, the algorithm tries to merge two subsets to search for a better lower bound. The algorithm stops when there are only two subsets left.

5.3 Performance Results

We simulated the algorithm in Figure 5.3 on random networks with $N = 128$ nodes and $|E| = 3N$. There are two set of results. First, we studied the performance as a function of number of multicast groups while keeping groups size the same size (e.g., 22 member nodes in each group). Figure 5.5 shows the simulation results obtained

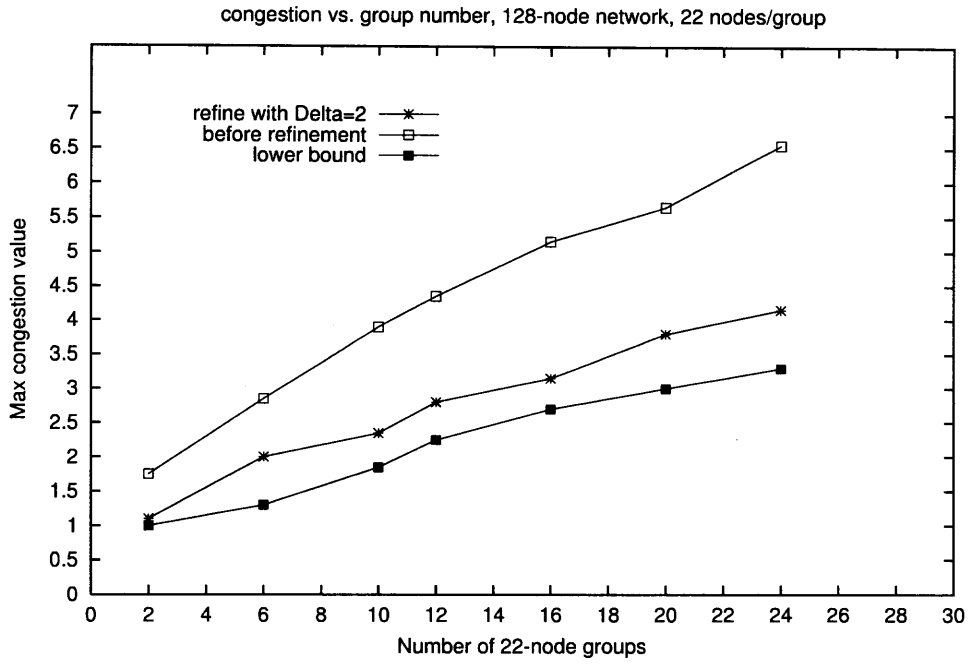


Figure 5.5 Performance of packing heuristic as a function of number of groups.

over 7 configurations, each with a different number of multicast groups. The value of parameter Δ^* (defined in 5.2.1) is set to 2. The number of groups are increased from 2 to 24. As we can see, when the number of groups are small, the impact of the heuristic is relatively small. This is reasonable because when there are not many groups in a network, the chance of a link being multiplexed by many trees is relatively small. Thus the maximum congestion before reduction is low and there is not much to reduce. As the number of groups increases, the difference increases meaning that the algorithm effectively decrease the maximum congestion of the network. Lower bounds are also computed with the "LBSearch" algorithm in Figure 5.4. Comparing the packing results to corresponding lower bounds, we can see that the packing algorithm does effectively reduce the congestion in the network.

Second, we tested the performance as a function of group size. Figure 5.6 shows how congestion varies with different group sizes while the number of groups is fixed (e.g., 20 groups). We chose group sizes ranging from 5 to 25. In the plotting, we observed also the impact of different Δ^* values. What we can see is that Δ^* value is

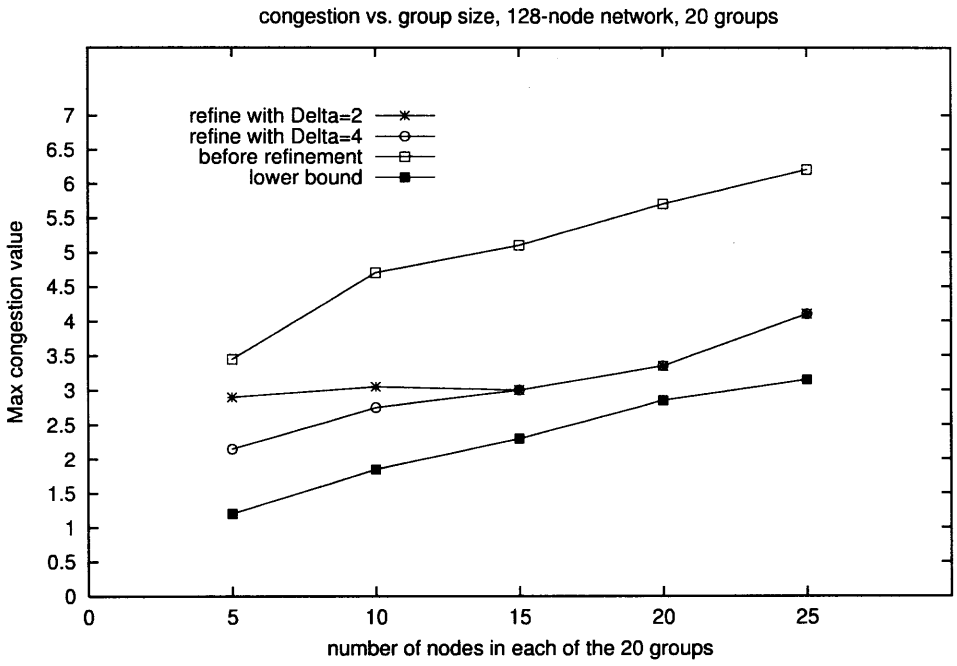


Figure 5.6 Performance of packing heuristic as a function of group size.

more significant when group sizes are relatively small. The plotting shows that when Δ^* is increased from 2 to 4, the congestion value can be reduced more significantly when group sizes are 5 and 10. Δ^* value has little effects when group sizes are large. This is reasonable. When group sizes are small, members are more sparsely distributed across the network. Therefore, when looking for alternative paths to reduce Z value (max. congestion value of all edges), it is harder to find a short path without going over the $\Delta^* = 2$ limit.

5.4 Summary

The multicast tree packing problem makes us reconsider the optimality of multicast routing. From the view of global congestion control, it may not be optimal to construct individually "optimal" multicast trees independently for multiple concurrent multicast sessions in the same network.

The study in this chapter shows that by using detours to avoid too many groups sharing a same link, we may reduce congestion risk from multicast traffic. This strategy may have to compromise the optimality of individual trees. We investigated the performance of such strategy. Also in this chapter, we provide an algorithm to compute lower bound for multicast tree packing.

In simulations, we first build multicast trees individually by using a near-optimum heuristic algorithm. Then we re-configure multicast trees within a certain size bound to reduce the maximum link congestion. Results show that this can help in reducing link congestion significantly.

CHAPTER 6
CONCLUSIONS

The focus of this dissertation is reliable group multicasting. We started with comparative studies of tree- and ring-based multicast approaches. The purpose is not to prove one is better than the other, but rather to identify the factors that affect reliable multicast performance. Use of a ring-based approach is one of the many efforts to address problems with tree-based reliable multicast (e.g. ACK implosion). But multicast by ring has its own problems. As established in our survey of the routing cost difference between these two approaches, we saw that multicast ring does not show gains at the routing layer. In fact, the propagation delay of multicast ring is worse. This obvious weakness is the major reason that prevents the ring-based approach from being widely accepted.

In addition to multicast latency, other issues such as throughput and efficiency are also important, especially to ensure reliability for group multicast. Multicast is still not widely deployed on the Internet, even though there has been more than a decade's worth of research. One main concern is that mechanisms to manage and control multicast traffic are still premature. Our modeling of tree-based reliable multicast showed that this lack of multicast control not only affects reliable multicast throughput, but also may bring heavy congestion to the network. The ring-based model, however, showed stable performance regardless of the offered multicast load. Group multicast is a resource-demanding application. If not gracefully controlled, it can develop into the major congestion cause in a network.

Based on the understanding of the two basic approaches, we further presented a new architecture RITA for reliable group multicast. RITA is a hybrid architecture which exploits the advantages from both ring and tree structures. By getting support from the tree structure, we improved the multicast delay performance at high network load. With the support of multicast rings, we achieved similar steady performance

with tolerance to heavy load. The RITA is hierarchically designed, and therefore it can scale well for wide-area group multicasting.

Finally, we investigated the multicast tree packing problem. Although it is about reducing competition for the same resources, it is actually congestion control at the network layer. By optimizing multicast routing, the risk of network congestion can be significantly reduced. This indeed can complete the reliability task for the upper layers.

6.1 Summary of Main Contributions

The main contributions of this dissertation are the following:

- We presented the RITA architecture, and protocols on top of this architecture, for reliable group multicast in datagram networks. This new architecture offers considerable performance improvements over existing reliable group multicast protocols.

- We conducted a comparative study of the tree- and ring-based reliable group multicast protocols. This study helps us to better understand the strengths and weaknesses of both approaches, and provides the primary motivation for the design of RITA architecture and the associated protocols.

- We studied the optimizing of multicast tree construction in the presence of multiple multicast groups. We investigated the performance improvements from multicast tree packing and found that tree packing can effectively help to reduce multicast traffic congestion.

6.2 Outstanding Problems for Further Study

We have identified several issues worthy of further study:

- Ring-based routing protocol for datagram networks. So far the ring-based routing in this dissertation is assumed to be provided at the network layer. In practice, almost all multicast protocols propose tree solutions. We have shown in this study the merit of ring based multicast. It is useful for small size, or delay-insensitive reliable multicast. It is also a prerequisite for construction of RITA.

- Window optimization of ring-based reliable group multicast protocol. We used a dynamic window control mechanism similar to the TCP control. However, how to converge to optimal window size and how to respond quickly to network changes remain problems for performance tuning.

- The optimization of clustering. RITA performance can be improved by optimizing between the number of clusters and cluster sizes. As ring structure is used for intra-cluster multicasting, cluster size has a negative effect on both delay and throughput. On the other hand, a large number of clusters can degrade the performance of the backbone tree.

- Further study of the multicast tree packing problem. In this dissertation, we measured the degree of a link's congestion by counting the number of groups sharing this link. However, multicast traffic can flow in both directions on a link. We argued in Chapter 4 that dominant direction load is different depending on many factors, such as the location of a link in multicast trees, the locations of multicast sources, etc. These additional factors introduce new levels of complexity and challenge into the multicast tree packing problem.

REFERENCES

1. L. Aguilar, "Datagram routing for Internet multicasting," *ACM Trans. on Computer Systems*, pp. 58–63, September 1984.

2. E. Aharoni and R. Cohen, "Restricted dynamic steiner trees for scalable multicast in datagram network," *Proc. of IEEE INFOCOM'97*, pp. 877–884, April 1997.

3. M. Allman, V. Paxson, and W. Stevens, "TCP congestion control," *RFC2581*, April 1999.

4. S. Armstrong, A. Freier, and K. Marzullo, "Multicast transport protocol," *RFC1301*, Febuary 1992.

5. A. Ballardie, "Core based tree (CBT) multicast routing architecture," *Internet Draft, draft-ietf-idmr-cbt-arch-04.txt*, 1997.

6. A. Ballardie, "Core based tree (CBT) multicast routing, protocol specification," *Internet Draft, draft-ietf-idmr-cbt-spec-07.txt*, 1997.

7. A. J. Ballardie, *A New Approach to Multicast Communication in a Datagram Internetwork*, PhD thesis, University College London, London, UK, May 1995.

8. T. Ballardie, P. Francis, and J. Crowcroft, "Core based trees (CBT): An architercure for scalable inter-domain multicast routing," *Proc. ACM SIGCOMM'93*, pp. 85–95, October 1993.

9. K. Bharath-Kumar and J. M. Jaffe, "Routing to multiple destinations in computer networks," *IEEE Trans. on Comm. COM-31*, pp. 343–351, March 1983.

10. S. Bhattacharyya, D. Towsley, and J. Kurose, "The loss path multiplicity problem in multicast congestion control," *Proc. of IEEE INFOCOM'99*, pp. 856–863, March 1999.

11. D. Boggs, *Internet Broadcast*, PhD thesis, Electrical Engineering Dept., Stanford University, Stanford, California, January 1982.

12. C. Bormann, J. Ott, H. Gehrcke, T. Kerschat, and N. Seifert, "MTP-2: Towards achieving the sero properties for multicast transport," *ICCCN'94*, September 1994.

13. L. Brakmo, S. O'Malley, and L. Peterson, "TCP Vegas: New techniques for congestion detection and avoidance," *SIGCOMM'94*, pp. 24–35, 1994.

14. H. Chang, H. Nam, C. Lee, Y. Choi, S. Chung, and C. Kim, "A reliable multicast protocols using round-robin ack and selective retransmission," *APCC'95*, pp. 148–152, 1995.

15. J.-M. Chang and N. F. Maxemchuk, "Reliable broadcast protocols," *ACM Transactions on Computer Systems*, vol. 2, no. 3, pp. 251–273, August 1984.

16. Y. Chawathe, S. Fink, S. McCanne, and E. Brewer, "A proxy architecture for reliable multicast in heterogeneous environments," *Proceedings of ACM Multimedia '98*, pp. 151–159, September 1998.

17. Y. Chawathe, S. McCanne, and E. A. Brewer, "RMX: Reliable multicast for heterogeneous networks," to appear in *Proc. of IEEE INFOCOM'2000*, July 2000.

18. S. Chen, O. Günlük, and B. Yener, "Optimal packing of group multicastings," *Proc. of IEEE INFOCOM'98*, pp. 980–987, 1998.

19. S. Chen, D. Karvelas, Q. Ma, and B. Yener, "A simple hierarchical approach to intra/inter domain multicasting," *Proc. SPIE, Performance and Control of Network Systems*, vol. 3231, pp. 102–112, October 1997.

20. S. Chen, B. Yener, and Y. Ofek, "Performance tradeoffs in reliable group multicast protocols," *Proc. of IEEE INFOCOM'99*, pp. 982–989, 1999.

21. S. Chopra and M. R. Rao, "The steiner tree problem I: Formulations, compositions and extension of facets," *Mathematical Programming*, vol. 64, pp. 209–229, 1994.

22. S. Chopra and M. R. Rao, "The steiner tree problem II: Properties and classes of facets," *Mathematical Programming*, vol. 64, pp. 231–246, 1994.

23. Y. Dalal and R. Metcalfe, "Reverse path forwarding of broadcast packets," *Communications of the ACM*, vol. 21, no. 12, pp. 1040–1048, December 1978.

24. S. E. Deeering and D. R. Cheriton, "Multicast routing in datagram internetworks and extended lans," *ACM Trans. on Computer Systems*, vol. 8, no. 2, pp. 85–110, May 1990.

25. S. Deering and et al., "The PIM architecture for wide-area multicast routing," *IEEE/ACM Transactions on Networking*, vol. 4, no. 2, pp. 153–162, 1996.

26. S. Deering and et al., "Protocol independent multicast-sparse mode (PIM-SM): Motivation and architecture," *Internet Draft, draft-ietf-idmr-pim-arch-04*, October 1996.

27. S. E. Deering, "Multicast routing in internetworks and extended lans," *ACM Symposium on Communication Architectures and Protocols, ACM SIGCOMM*, pp. 55–64, August 1988.

28. S. E. Deering, *Multicast Routing in a Datagram Internetwork*, PhD thesis, Stanford University, Stanford, California, December 1991.

29. S. Deering, "Internet group management protocol," *RFC1112*, August 1989.

30. L. Delgrossi and L. Berger, "Internet stream protocol version 2 (ST2) protocol specification - version ST2+," *RFC1819*, August 1995.

31. H. Eriksson, "Mbone: The multicast backbone," *Comm. of ACM*, vol. 37(8), pp. 54–60, August 1994.

32. D. Estrin and et al., "Protocol independent multicast-sparse mode(PIM-SM): Protocol specification," *Internet Draft, draft-ietf-idmr-PIM-SM-spec-10*, March 1997.

33. W. Fenner, "Internet group management protocol, version 2," *Internet Draft,draft-ietf-idmr-igmp-v2-06.txt*, January 1997.

34. S. Floyd, V. Jacobson, S. McCanne, C.-G. Liu, and L. Zhang, "A reliable multicast framework for light-weight sessions and application level framing," *ACM Computer Communication Review (SIGCOMM'95)*, pp. 342–356, August 1995.

35. M. Goemens, "The steiner tree polytope and related polyhedra," *Mathematical Programming*, vol. 63, pp. 157–182, 1994.

36. S. J. Golestani, "Fundamental observations on multicast congestion control in the internet," *Proc. of IEEE INFOCOM'99*, pp. 990–1002, March 1999.

37. C. Graff, "IPv4 option for sender directed multi-destination delivery," *RFC1770*, March 1995.

38. M. Grossglauser, "Optimal deterministic timeouts for reliable scalable multicast," *Prodeedings of IEEE INFOCOM*, vol. 3, pp. 1425–1432, March 1996.

39. M. Grötchel, A. Martin, and R. Weismantel, "Packing steiner trees: a cutting plane algorithm and computation," *Mathematical Programming*, vol. 72, pp. 125–145, 1996.

40. M. Grötchel, A. Martin, and R. Weismantel, "Packing steiner trees: polyhedral investigations," *Mathematical Programming*, vol. 72, pp. 101–123, 1996.

41. A. Helmy and D. Estrin, "Simulation-based 'STRESS' testing case study: A multicast routing protocol," Tech. Rep. USC-CS-TR 98-674, University of Southern California, Los Angeles, California, July 1998.

42. M. Hofmann, "Adding scalability to transport level multicast," *Proceedings of 3rd COST 237 Workshop: Multimedia Telecommunications and Applications*, pp. 41–55, November 1996.

43. M. Hofmann, "A generic concept for large-scale multicast," *Proceedings of International Zurich Seminar on Digital Communications*, pp. 95–106, Febuary 1996.

44. H. Holbrook, S. K. Singhal, and D. R. Cheriton, "Log-based receiver-reliable multicast for distributed interactive simulation," *Proc. ACM SIGCOMM'95*, pp. 328–341, August 1995.

45. F. K. Hwang, D. S. Richards, and P. Winter, *The Steiner Tree Problem*, Annal of Discrete Math. 53, North-Holland, 1992.

46. R. Jain, "A delay-based approach for congestion avoidance in interconnected heterogeneous computer networks," *ACM Computer Communication Review*, vol. 19, no. 5, pp. 56–71, October 1989.

47. A. Koifman and S. Zabele, "RAMP: A reliable adaptive multicast protocol," *Proc. of IEEE INFOCOM'96*, vol. 3, pp. 1442–1451, 1996.

48. L. T. Kous, G. Markowsky, and L. Berman, "A fast algorithm for steiner trees," *Acta Inf. 15*, pp. 141–145, 1981.

49. B. N. Levine and J. Garcia-Luna-Aceves, "A comparison of known classes of reliable multicast protocols," *Proc. of the 1996 International Conf. on Network Protocols*, pp. 112–121, November 1996.

50. B. Levine and J. Garcia-Luna-Aceves, "Improving Internet multicast with routing labels," *Proc. of the 1997 International Conference on Network Protocols (ICNP '97)*, pp. 112–121, October 1997.

51. B. N. Levine, "A comparison of known classes of reliable multicast protocols," Master's thesis, University of California, Santa Cruz, California, June 1996.

52. B. N. Levine, D. B. Lavo, and J. Garcia-Luna-Aceves, "The case for reliable concurrent multicasting using shared ack trees," *Proceedings of ACM Multimedia*, pp. 365–376, November 1996.

53. T. Liao, "Light-weight reliable multicast protocol specification," *Internet Draft, draft-liao-lrmp-00.txt*, October 1998.

54. J. C. Lin and S. Paul, "RMTP: A reliable multicast protocol," *Proc. IEEE INFOCOM'96*, pp. 1414–1425, March 1996.

55. S. Lin, "Computer solution of the traveling salesman problem," *The Bell System Technical Journal*, vol. 44, no. 10, pp. 2245–2269, 1965.

56. A. Mankin, A. Romanow, S. Bradner, and V. Paxson, "IETF criteria for evaluating reliable multicast transport and application protocols," *RFC2357*, June 1998.

57. D. Mitzel, D. Estrin, S. Shenker, and L. Zhang, "An architectural comparison of ST-II and RSVP," *Proc. of IEEE INFOCOM'94*, pp. 716–725, June 1994.

58. T. Montgomery, B. Whetten, and J. R. Callahan, "The reliable multicast protocol specification," *ftp:// research.ivv.nasa.gov/ pub/ doc/ RMP/ RMPpackets.txt*, October 1995.

59. R. Morris, "Bulk multicast transport protocol," *Proc. IEEE INFOCOM'97*, pp. 455–462, April 1997.

60. J. Moy, "Multicast extensions to OSPF version 2," *RFC1584*, March 1994.

61. J. Moy, "Multicast routing extensions for OSPF," *Communications of the ACM*, vol. 37, no. 8, pp. 61–66, August 1994.

62. J. Nonnenmacher, E. W. Biersack, and D. F. Towsley, "Parity-based loss recovery for reliable multicast transmission," *ACM SICOMM'97*, pp. 289–300, September 1997.

63. Y. Ofek and B. Yener, "Reliable concurrent multicast from bursty sources," *IEEE JSAC*, vol. 14, no. 3, pp. 434–444, April 1997.

64. M. Parsa and J. Garcia-Luna-Aceves, "A protocol for scalable loop-free multicast routing," *IEEE Journal on Selected Areas in Communications*, vol. 15, no. 3, pp. 316–331, April 1997.

65. S. Paul, K. Sabnani, and D. M. Kristol, "Multicast transport protocols for high speed networks," *Proceedings of International Conference on Network Protocols*, pp. 4–14, May 1994.

66. J. Postel, "Transmission control protocol," *RFC793*, September 1981.

67. T. Pusateri, "Distance vector multicast routing protocol," *Internet Draft,draft-ietf-idmr-dvmrp-v3-04.txt*, February 1997.

68. S. B. R. Braden, L. Zhang, "Resource reservation protocol (RSVP) – version 1 functional specification," *Internet Draft, draft-ietf-rsvp-spec-14.txt*, November 1996.

69. B. Rajagopalan, "Reliability and scaling issues in multicast communication," *Conference Proceedings on Communications Archtectures and Protocols*, pp. 188–198, 1992.

70. L. Rizzo, "Effective erasure codes for reliable computer communication protocols," *ACM Computer Communication Review*, vol. 27, no. 2, pp. 24–36, April 1997.

71. J. L. S. Paul, K. K. Sabnani and S. Bhattacharyya, "Reliable multicast transport protocol(RMTP)," *IEEE JSAC*, vol. 15, no. 3, pp. 407–421, April 1997.

72. B. Sabata, M. J. Brown, B. A. Denny, and C. Heo, "Transport protocol for reliable multicast: TRM," *Proc. of the IASTED International Conference on Networks*, pp. 143–145, January 1996.

73. C. Shields and J. Garcia-Luna-Aceves, "The ordered core based tree protocol," *Proc. of IEEE INFOCOM'97*, pp. 884–891, April 1997.

74. T. Speakman, N. Bhaskar, R. Edmonstone, D. Farinacci, S. Lin, A. Tweedly, L. Vicisano, and J. Gemmell, "PGM reliable transport protocol specification," *Internet Draft, draft-speakman-pgm-spec-03.txt*, June 1999.

75. W. Stevens, "TCP slow start, congestion avoidance, fast retransmit, and fast recovery algorithms," *RFC2001*, January 1997.

76. R. Talpade and M. H. Ammar, "Single connection emulation (SCE): An architecture for providing a reliable multicast transport service," *Proc. of the 15th IEEE Intl. Conf. on Distributed Computing Systems*, pp. 144–151, June 1995.

77. C. Topolcic, "Experimental internet stream protocol, version 2 (ST-II)," *RFC1190*, October 1990.

78. D. Towsley, J. Kurose, and S. Pingali, "A comparison of sender-initiated and receiver-initiated reliable multicast protocols," *IEEE Journal on Selected Areas in Communications*, vol. 15, no. 3, pp. 398–406, April 1997.

79. D. Wall, *Mechanisms for Broadcast and Selective Broadcast*, PhD thesis, Electrical Engineering Dept., Stanford University, Stanford, California, June 1980.

80. J. L. Wang and J. A. Silvester, "Delay minimization of the adaptive Go-Back-N ARQ protocols for point-to-multipoint communication," *Proc. IEEE INFOCOM'89*, pp. 584–593, April 1989.

81. Z. Wang, *Routing and Congestion Control in Datagram Networks*, PhD thesis, University College London, London, UK, Jan. 1992.

82. Z. Wang and J. Crowcroft, "Eliminating periodic packet losses in the 4.3-Tahoe BSD TCP congestion control algorithm," *ACM Computer Communication Review*, vol. 22, no. 2, pp. 9–16, April 1992.

83. A. C. Weaver, "Xpress transport protocol version 4," *Proc. IEEE Comp. Com. Systems*, pp. 165–174, 1995.

84. B. Whetten, T. Montgomery, and S. Kaplan, "A high performance totally ordered multicast protocol," *Theory and Practice in Distributed Systems,* Springer Verlag LCNS n. 938, pp. 33–57, 1994.

85. H. Wu, Y. Ofek, and K. Sohraby, "Integration of synchronous and asynchronous traffic on the MetaRing architecture and its analysis," *ICC'92*, pp. 147–153, 1992.

86. M. Yajnik, J. Kurose, and D. Towsley, "Packet loss correlation in the mbone multicast network," *Proc. of the 1996 IEEE Global Telecommunications Conf.*, pp. 94–99, November 1996.

87. M. Yamamoto, J. F. Kurose, D. F. Towsley, and H. Ikeda, "A delay analysis of sender-initiated and receiver-initiated reliable multicast protocols," *Proc. of IEEE Infocom'97*, pp. 481–489, April 1997.

88. R. Yavatkar, J. Griffioen, and M. Sudan, "A reliable dissemination protocol for interactive collaborative applications," *Proc. ACM Multimedia*, pp. 333–344, 1995.

89. L. Zhang, S. Deering, D. Estrin, S. Shenker, and D. Zappala, "RSVP: A new resource ReSerVation protocol," *IEEE Network*, vol. 7, no. 5, pp. 8–18, September 1993.