BOSTON UNIVERSITY
GRADUATE SCHOOL OF ARTS AND SCIENCES

Dissertation

SCALABLE ARCHITECTURES FOR MULTICAST
CONTENT DISTRIBUTION

by

GU-IN KWON
M.A., Queens College (City University Of New York), 1998
B.A., Inha University, South Korea, 1995

Submitted in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy
2005
Approved by

First Reader

John W. Byers, Ph.D.
Assistant Professor of Computer Science

Second Reader

Ibrahim Matta, Ph.D.
Associate Professor of Computer Science

Third Reader

Azer Bestavros, Ph.D.
Professor of Computer Science
ACKNOWLEDGEMENTS

I would like to express my gratitude to my advisor, John Byers, for his support, patience, and encouragement throughout my graduate studies. His technical and editorial advice was essential to the completion of this dissertation and has taught me innumerable lessons and insights on the workings of academic research in general.

My thanks also go to the members of my committee, Azer Bestavros and Ibrahim Matta for providing many valuable comments that improved contents of this dissertation.

I am very grateful for my wife Eunhee, for her love and patience during the PhD period. One of the best experiences that we lived through in this period was the birth of our son Daniel Kwon, who provided an additional and joyful dimension to our life mission.
SCALABLE ARCHITECTURES FOR MULTICAST CONTENT DISTRIBUTION

(Order No. )

GU-IN KWON

Boston University Graduate School of Arts and Sciences, 2005

Major Professor: John W. Byers, Assistant Professor of Computer Science

ABSTRACT

IP multicast techniques were developed to provide a scalable architecture for the delivery of large volumes of content to multiple receivers. However, one of the significant challenges associated with multicast delivery to large audiences is providing a scalable congestion control mechanism that is not only compliant with the dominant Internet transport protocol, TCP, but also addresses heterogeneity in the network bandwidth across receivers since receivers may have different connection speeds and different congestion levels.

In this dissertation, we propose scalable solutions for reliable IP multicast congestion control which accommodate heterogeneous network bandwidth across receivers and provide a fair share of network resources with competing TCP flows. Despite the considerable works in the area of multicast congestion control, these schemes have some major limitations; unfairness to TCP flows, non-scalability to large audiences, coarse-grained rate change, and high design complexity. We first demonstrate that fine-grained multicast congestion control protocol can be realized and implemented with reasonable costs and complexity. At a high level, our STAIR (Simulate TCP's Additive Increase/multiplicative...
decrease with Rate-based) multicast congestion control algorithm simulates the property of TCP congestion control scheme as closely as possible while imposing minimal load on the network and minimizing the amount of control traffic. We also propose another new approach to multiple rate congestion control that leverages proven single rate congestion control methods by orchestrating an ensemble of independently controlled single rate sessions in the design of multiple rate congestion control. The main advantages of our protocol called SMCC are simplicity, scalability, and modular design.

In addition to our work on IP multicast, we also propose a new network architecture for reliable content distribution using an overlay network. There has been a significant amount of previous work in overlay multicast since unlike IP multicast, overlay multicast does not require additional network support in the current Internet. However existing approaches for providing both reliability and congestion control on overlay multicast impose fundamental performance limitations, such as dragging down all transfer rates in the system to the rate of the slowest receiver. The ROMA architecture we propose delivers a scalable solution that better accommodates a set of heterogeneous receivers.
## Contents

Acknowledgements iv  
Abstract v  
Contents vii  
List of Tables xi  
List of Figures xii  

1 Introduction 1  

2 Multicast Congestion Control 11  
   2.1 TCP Congestion Control 11  
   2.2 Single Rate Multicast Congestion Control 14  
   2.3 Multiple Rate Multicast Congestion Control 16  

3 Fine-Grained Layered Multicast With STAIR 23
3.1 Introduction ................................................................. 23
3.2 Definitions and Building Blocks for our Approach ................. 25
3.3 Components of Our Approach ......................................... 29
  3.3.1 Combining Cumulative and Non-Cumulative Layering .......... 29
  3.3.2 Introducing Stair Layers ............................................ 33
3.4 The STAIR Congestion Control Algorithm ............................ 36
  3.4.1 Additive Increase, Multiplicative Decrease ....................... 36
  3.4.2 Configuration of Stair Layers ...................................... 37
  3.4.3 RTT Estimation and STAIR Subscription ......................... 38
3.5 Experimental Evaluation .................................................. 42

4 Leveraging Single Rate Schemes in Multiple Rate Multicast Congestion Con-
trol Design .................................................................. 50
  4.1 Motivation .................................................................. 50
    4.1.1 Our Work in Context .............................................. 52
  4.2 Overview Of Our Approach ............................................. 54
    4.2.1 Setting up Layers and LRs ....................................... 56
    4.2.2 Adaptively Adjusting Layer Rates ............................... 58
    4.2.3 TCP-Friendliness .................................................. 60
  4.3 Single Rate Congestion Control ......................................... 61
    4.3.1 TFMCC Overview .................................................. 61
    4.3.2 pgmcc Overview ................................................... 64
4.4 Multicast Congestion Control from TFMCC

4.4.1 Lead Receiver Change

4.4.2 Subscribing to an Additional Layer

4.5 Multicast Congestion Control from pgmcc

4.6 Additive Increase Join Attempts

4.6.1 Introducing Binary Counting Layers

4.6.2 Extended Binary Counting Layers

4.6.3 Cost Of Additional BCLs For Join Attempt

4.7 Experiments

4.7.1 Preliminary Fairness Tests

4.7.2 Late Join by a Low-rate Receiver

4.7.3 Inaccuracy of Non-LR Estimated Target Rate

4.7.4 Responding to dynamics of competing traffic

4.7.5 Fairness With Heterogeneous Receivers

4.7.6 Comparison of SMCC and STAIR with competing TCP flows

5 Reliable Overlay Multicast with Loosely Coupled TCP Connections

5.1 Motivation

5.2 Related Work in Overlay Design

5.3 Candidate Architectures

5.3.1 Limited Buffer Space Solution

5.3.2 Unlimited Buffer Space Solution
# List of Tables

3.1  Performance of AIMD Congestion Control for Various Approaches . . . 28

4.1  Terms Description . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 56

4.2  Calculated target rate of TFMCC receivers in Fig. 4.9 . . . . . . . . . 82

5.1  End-to-End Measured Throughput . . . . . . . . . . . . . . . . . . . . . . . . 114
List of Figures

1.1 Unicast vs. IP Multicast .................................................. 2
1.2 Layered Multicast ............................................................. 6
1.3 IP Multicast vs. Overlay Multicast ........................................ 8
2.1 RLC: Sequence of packet transmission ............................... 18
2.2 FLID-DL: Rate on dynamic layers $L_0$ and $L_1$. .................... 20
2.3 WEBRC Channels, brought from the presentation slides [34] .......... 21
3.1 Dilation of a shared bottleneck link. (a) Cumulative: Dilation = 1, (b) Non-Cumulative: Dilation = 12/9. ................................. 26
3.2 Hybrid Layer Scheme : $K = \alpha^i + r$. For a target rate of 13, with $\alpha = 2$, $K = 2^3 + 5$. $C_i$ denotes the rate on cumulative layer $i$, $N_i$ denotes the rate on non-cumulative layer $i$. .................................................. 30
3.3 Maximal dilation at a link as a function of available link bandwidth .... 33
3.4 Depiction of $SL_{128}$ (a stair layer with $t = 128$ms) in isolation. $R = 1$Mbps, packet size = 1KB. ......................................................... 34
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.5</td>
<td>35</td>
</tr>
<tr>
<td>3.6</td>
<td>39</td>
</tr>
<tr>
<td>3.7</td>
<td>40</td>
</tr>
<tr>
<td>3.8</td>
<td>42</td>
</tr>
<tr>
<td>3.9</td>
<td>43</td>
</tr>
<tr>
<td>3.10</td>
<td>44</td>
</tr>
<tr>
<td>3.11</td>
<td>45</td>
</tr>
<tr>
<td>3.12</td>
<td>46</td>
</tr>
<tr>
<td>3.13</td>
<td>47</td>
</tr>
<tr>
<td>3.14</td>
<td>48</td>
</tr>
<tr>
<td>3.15</td>
<td>49</td>
</tr>
<tr>
<td>4.1</td>
<td>55</td>
</tr>
<tr>
<td>4.2</td>
<td>59</td>
</tr>
<tr>
<td>4.3</td>
<td>72</td>
</tr>
<tr>
<td>4.4</td>
<td>74</td>
</tr>
<tr>
<td>4.5</td>
<td>75</td>
</tr>
<tr>
<td>4.6</td>
<td>78</td>
</tr>
<tr>
<td>4.7</td>
<td>78</td>
</tr>
<tr>
<td>4.8</td>
<td>79</td>
</tr>
<tr>
<td>4.9</td>
<td>81</td>
</tr>
</tbody>
</table>
4.10 Throughput of TF1 and TF2 over time in Fig. 4.9. Competing TCP connections plotted in the background. ............................................. 82
4.11 Impact of dynamic competing traffic, $B_0 = 8$ Mbps .............................. 84
4.12 Topology for Many Heterogeneous Receivers, $B_0=4$Mbps .................. 86
4.13 Throughput of SMCC receivers, $B_0 = 4$ Mbps .................................. 87
4.14 Throughput Comparison of S1, S2, and S3 ........................................ 88
4.15 Dilation on the shared link (A-B) ...................................................... 89
4.16 Throughput of STAIR and SMCC receivers, $B_0 = 4$ Mbps .................. 90

5.1 Buffering is inadequate for handling rate mismatches. ............................ 92
5.2 Overview of TCP-based Content Delivery in an Overlay. ....................... 98
5.3 Adaptive reconfiguration of the overlay ............................................. 100
5.4 Overlay Node Implementation .......................................................... 106
5.5 Basic Model for Chains of TCP Connections ....................................... 108
5.6 Chains of TCP Connections .............................................................. 111
5.7 Multirate Reliable Multicast with B.U. as the sender ............................. 116
5.8 Host throughput with B.U. as the sender ............................................ 117
5.9 Multirate Reliable Multicast with UCLA as the sender ......................... 119
5.10 Host throughput with UCLA as the sender ....................................... 119
5.11 Throughput Improvement on Chains of TCP ...................................... 120
5.12 Single Source Widest-Path Tree rooted at UW ................................... 121
5.13 Host throughput with UW as the sender .......................................... 123

xv
<table>
<thead>
<tr>
<th>Section Number</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>5.14</td>
<td>Throughput Advantage From ROMA</td>
<td>124</td>
</tr>
<tr>
<td>5.15</td>
<td>End hosts with the same network conditions</td>
<td>124</td>
</tr>
<tr>
<td>5.16</td>
<td>Number of packets in the application buffer</td>
<td>125</td>
</tr>
</tbody>
</table>
# List of Abbreviations

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACK</td>
<td>Acknowledgement</td>
</tr>
<tr>
<td>AIMD</td>
<td>Additive Increase/Multiplicative Decrease</td>
</tr>
<tr>
<td>FEC</td>
<td>Forward Error Correction</td>
</tr>
<tr>
<td>FGLM</td>
<td>Fine Grained Layered Multicast Congestion Control Protocol</td>
</tr>
<tr>
<td>IP</td>
<td>Internet Protocol</td>
</tr>
<tr>
<td>IGMP</td>
<td>Internet Group Membership Protocol</td>
</tr>
<tr>
<td>LT</td>
<td>Luby Transform</td>
</tr>
<tr>
<td>NACK</td>
<td>Negative Acknowledgement</td>
</tr>
<tr>
<td>pgmcc</td>
<td>Pragmatic General Multicast Congestion Control</td>
</tr>
<tr>
<td>PGM</td>
<td>Pragmatic General Multicast</td>
</tr>
<tr>
<td>RED</td>
<td>Random Early Detection</td>
</tr>
<tr>
<td>Abbreviation</td>
<td>Description</td>
</tr>
<tr>
<td>--------------</td>
<td>--------------------------------------------------</td>
</tr>
<tr>
<td><strong>RLC</strong></td>
<td>Receiver driven Layered Congestion control</td>
</tr>
<tr>
<td><strong>RLM</strong></td>
<td>Receiver-driven Layered Multicast</td>
</tr>
<tr>
<td><strong>ROMA</strong></td>
<td>Reliable Overlay Multicast Architecture</td>
</tr>
<tr>
<td><strong>RTT</strong></td>
<td>Round Trip Time</td>
</tr>
<tr>
<td><strong>TCP</strong></td>
<td>Transmission Control Protocol</td>
</tr>
<tr>
<td><strong>TFRC</strong></td>
<td>TCP Friendly Rate Control</td>
</tr>
<tr>
<td><strong>TFMCC</strong></td>
<td>TCP-Friendly Multicast Congestion Control</td>
</tr>
<tr>
<td><strong>UDP</strong></td>
<td>User Datagram Protocol</td>
</tr>
<tr>
<td><strong>WEBRC</strong></td>
<td>Wave and Equation Based Rate Control</td>
</tr>
</tbody>
</table>
Chapter 1

Introduction

Recently the face of the Internet has seen a paradigm shift from an architecture which was purely used for basic services such as email and the World Wide Web to one which is being used for distributing popular content (such as new software and software updates), sharing media (such as movies and music), and real time video broadcast. Typically, such services are expected to support thousands of simultaneous requests from a geographically distributed set of users. The traditional method of data transport on the Internet is unicast, where a separate connection is established between one specific sender and one specific receiver. Thus present unicast-based methodologies for bulk distribution of such content to multiple users result in large overheads on both the sender and the network. In such scenarios, a server must maintain a separate connection for each receiver, thus multiple copies of identical content are delivered across the network. Figure 1.1 (a) illustrates this unicast technology, where the number of identical copies of content on the physical links
Multicast techniques were developed to provide an efficient and elegant solution for such applications while minimizing the total bandwidth usage and server load. In Internet Protocol (IP) multicasting, the sender sends only one copy of the content and addresses it to the group of receivers that want to receive it. A multicast distribution tree is constructed with only the networks and receivers that need to receive that content. A given packet, containing a portion of the content, is delivered only once on each physical link and it is replicated at each branching point in the multicast distribution tree (Figure 1.1 (b)).

Although multicast exhibits several advantages over unicast for simultaneously delivering content to thousands of receivers, there are many challenges to deploy multicast on the Internet, where a design goal of the Internet is to apportion network resources across the users of the network without overloading the network and doing so fairly.

In today’s Internet, network paths will get congested when the demand for bandwidth
exceeds available bandwidth and a router with a congested link simply discards packets when congestion occurs. The dominant transport protocol in the Internet, the Transmission Control Protocol (TCP), was specifically designed to provide a reliable content delivery over the Internet even if packets are dropped or corrupted. The original specification of TCP included a mechanism for reliable content distribution and window-based flow control, which limits the number of sending packet to prevent overflow of the receiver’s data buffer space. TCP congestion control was introduced in the late 1980’s by Van Jacobson [24] after early TCP implementation induced a congestion collapse, which occurs when the network is so overloaded that it is only forwarding retransmissions. The basic strategy of TCP congestion control is to force the sender to reduce the transmission rate in response to detection of packet loss, where congestion is the most common reason for packet loss in the network. TCP congestion control mechanism is considered as additive increase multiplicative decrease (AIMD), where the sender increases the transmission rate linearly to take advantage of any additional bandwidth that becomes available and decreases the transmission rate suddenly and significantly in response to congestion. TCP’s congestion control allows receivers sharing the same bottleneck path with similar round-trip times to get the similar throughputs and this mechanism is one of the critical factors in the robustness and stability of the Internet, where TCP constitutes 90% of all traffic in today’s Internet.

Although TCP provides good mechanisms for reliable content distribution with congestion control, TCP’s abrupt rate change in response to a single packet loss does not
satisfy the requirements of some applications such as streaming applications, which require smooth rate changes. Such applications do not employ congestion control currently and the growing numbers of these flows will have heavy impact on the Internet. This results in the need for the design of new transport protocols to meet specific requirements for such applications while providing congestion control. These new transport protocols should employ congestion control to prevent the network being overloaded, and congestion control for these applications should not be more aggressive than TCP flows. Multicast also has to provide congestion control which shares the network resources with competing TCP flows. However, due to large numbers of receivers with heterogeneous and diverse network characteristics (for instance bandwidths, latencies, and loss rates), congestion control is one of the significant challenges associated with multicast delivery.

We enumerate issues in providing efficient and scalable solutions for reliable content distribution to multiple recipients, with a primary focus on challenges which arise due to the highly heterogeneous nature of today’s networks.

- Congestion control scheme should consider each receiver’s congestion level. While single rate congestion control schemes provide a simple solution by transmitting at a rate for the receiver with the lowest expected throughput, this approach does not scale to large sets of receivers with heterogeneous available bandwidths.

- TCP congestion control uses the receiver’s feedback to detect congestion. In multicast, a receiver may send feedbacks to the sender by either sending acknowledgments (ACKs) for received packets or sending negative acknowledgments (NACKs)
for packets which were detected to be lost. If congestion control scheme relies on receiver’s feedbacks, then a scalable solution for feedback implosion should be employed since the sender may receive simultaneous feedback messages from large numbers of receivers.

- The throughput of each receiver along a path should not exceed the throughput that TCP would achieve along that path. When the throughput of multicast receiver is similar to the throughput of TCP, this congestion control mechanism is called TCP-friendly.

- The number of control messages sent by receivers to accomplish congestion control should be minimal to reduce the load on the sender and the network.

- The congestion control scheme should not be affected by the number of receivers to provide scalability.

This dissertation proposes scalable solutions for reliable multicast congestion control which accommodate heterogeneous network bandwidth across receivers and provide a fair share of network resource with competing TCP flows. The primary set of target applications are applications requiring reliability and high bandwidth, such as the delivery of large files.

The now standard technique of layered multicast, which employs several multicast groups to transmit content at different rates, has been employed as a building block to address heterogeneity in end-to-end bandwidth across receivers [37]. This technique en-
ables each receiver to adjust its reception rate according to its current network conditions, independent of other receivers. A receiver must join a group using IGMP to receive data delivering to a particular group, and it may increase the reception rate by subscribing to additional multicast groups, often called *layers*. The decrease of reception rate is done through unsubscribing from the layers, i.e. leaving multicast groups. This technique allowing different rates across receivers is called *multiple rate* multicast congestion control.

Figure 1.2 briefly shows how the sender maintains multiple layers and each receiver receives different reception rate based on its subscription level. In this figure, receiver 3 would receive the traffic at 4Mbps by subscribing to 3 layers while the reception rate of receiver 2 would be 2Mbps. A receiver may increase the receiving rate by subscribing to a new multicast group and may decrease the reception rate by unsubscribing from a multicast group.

Several schemes for multiple rate congestion control using layered multicast have been
proposed to achieve TCP-friendliness and address the heterogeneity issue. Traditional approaches to layered multicast have advocated the benefits of cumulative layering, which can enable coarse-grained congestion control that complies with TCP-friendliness equations over large time scales. In these approaches, joining of next layer will double the reception rate. In contrast to the conventional wisdom, we demonstrate that fine-grained rate adjustment can be achieved with only modest increases in the number of layers, aggregate bandwidth consumption and control traffic. In Chapter 3, we present a new, scalable multicast congestion control called STAIR. The STAIR protocol is a multiple rate congestion control scheme that provides a fine-grained approximation to the behavior of TCP additive increase / multiplicative decrease (AIMD) on a per-receiver basis.

A significant impediment to deployment of multicast services is the daunting technical complexity of developing, testing and validating congestion control protocols fit for wide-area deployment. Protocols such as pgmcc and TFMCC have recently made considerable progress on the single rate case, i.e. where one dynamic reception rate is maintained for all receivers in the session. However, these protocols have limited applicability, since scaling to session sizes beyond tens of participants with heterogeneous available bandwidth necessitates the use of multiple rate protocols. Unfortunately, while existing multiple rate protocols exhibit better scalability, they are both less mature than single rate protocols and suffer from high complexity. In Chapter 4, we propose a new multiple rate congestion control algorithm for layered multicast sessions that employs a single rate multicast congestion control as the primary underlying control mechanism for each layer, where each
layer is controlled independently from other layers. Our new scheme combines the benefits of single rate congestion control with the scalability and flexibility of multiple rates to provide a sound multiple rate multicast congestion control policy.

In addition to our work on IP multicast, where each router inside the network has to provide multicast functionality, we consider the problem of architecting a reliable content delivery system across an overlay network. While IP multicast is an efficient transmission scheme for supporting large scale content distribution over the Internet by reducing the load on both the server and the network, IP multicast has serious deployment limitations. The application-level or end-system multicast schemes [4, 8, 17, 21, 26, 40] have received a lot of attention since this approach avoids the considerable deployment hurdles associated with providing multicast functionality at the network layer. An end-system architecture constructs an overlay topology, comprising collections of unicast connections between end-systems, in which each connection in the overlay is mapped onto a path in the underlying physical network by IP routing. End-systems also implement other multi-
cast related functionality such as packet replication. Figure 1.3 illustrates overlay topology comparing to IP multicast topology and packet replication at intermediate hosts. Overlay multicast imposes performance penalties such as duplicated packets on the physical links and increased end-to-end delay comparing to IP multicast, but it may avoid significant challenges posed by heterogeneous network conditions with IP multicast. In particular, transport-level functionality such as congestion control and reliability can be realized by employing standard unicast transport protocols. However, overlay multicast has to take into account another issue to provide congestion control and reliability: an intermediate end-system relaying data to downstream hosts may have different congestion levels or connection speeds between its upstream and downstream connections. When the intermediate end-system has a high-bandwidth upstream connection and a low-bandwidth downstream connection, existing approaches for providing both reliability and congestion control on overlay multicast impose fundamental performance limitations, such as dragging down all transfer rates in the system to the rate of the slowest receiver.

In Chapter 5, we introduce a new architecture called ROMA for a reliable content delivery system across an overlay network using TCP connections as the transport primitive. The ROMA architecture delivers a scalable solution that better accommodates a set of heterogeneous receivers: slow speed connection does not affect the performance of other receivers. The methods we develop establish chains of TCP connections, whose expected performance we analyze through equation-based methods. We validate our analytical findings and evaluate the performance of our ROMA architecture using a prototype
implementation via extensive Internet experimentation across the PlanetLab distributed testbed.

We begin this thesis by briefly summarizing related work. In Chapter 2, we describe TCP congestion control and TCP-friendly unicast congestion control. We also review the large body of work of multicast congestion control. Then in subsequent chapters, we detail our contributions.
Chapter 2

Multicast Congestion Control

We first describe TCP congestion control and briefly overview TCP-friendly unicast congestion control schemes. In Section 2.2 and 2.3, we survey the large body of relevant work in the area of multicast congestion control. We overview single rate multicast congestion control schemes in which the transmission rate is adjusted by a receiver which experiences the worst congestion, and all receivers will get the same throughput. Then, we give an overview of multiple rate multicast congestion control schemes in which each receiver can get different throughput based on its current network conditions.

2.1 TCP Congestion Control

In the previous chapter, we described the basic strategy of unicast TCP congestion control, where the sender adjusts the congestion window using the AIMD algorithm based on the
current network conditions. Since one of the goals in multicast congestion control is to provide TCP-friendliness, we overview TCP congestion control in more detail in this section. TCP provides both a reliable end-to-end byte stream over the network and congestion control. The basic mechanism to achieve reliability is that a receiver sends an ACK for each received packet to a sender. The sender monitors ACKs and retransmits lost packets. TCP maintains a congestion window in addition to the advertised window, used for flow control to prevent overflow of the receiver’s data buffer space, to limit the amount of data the sender can transmit into the network before receiving ACK. The sender increases the congestion window through an additive increase mechanism, where reception of an entire congestion window’s worth of ACKs increases the congestion window by the size of one packet. This results in sending one more packet each round-trip time (RTT).

When a new connection is established between two end systems, TCP performs slow start to reach close to the available capacity of the network quickly. During slow start, the sender increases the congestion window by one on every new ACK. Thus, the congestion window increases exponentially fast. The slow-start phase ends when the congestion window size exceeds the value of a pre-defined threshold or after the first packet loss occurs. Once the slow-start phase ends, TCP enters a congestion avoidance phase and the congestion window grows linearly rather than exponentially.

The sender decreases the congestion window suddenly and significantly in response to congestion. TCP uses following two mechanisms for how to detect congestion and how to react to congestion. If a packet is not acknowledged for the certain amount of time, the
threshold is set to half of the current congestion window and the congestion window is then set to one, and TCP reenters the slow start phase. The fast retransmit mechanism was added to detect packet loss quickly without waiting for the retransmission timer to expire. With fast retransmit mechanism, the sender reduces the congestion window to half of its previous size. If we ignore the slow start phase, TCP uses additive increase multiplicative decrease (AIMD) to increase or decrease the transmission rate based on the presence of congestion.

Although TCP provides good mechanisms for reliable content distribution with congestion control, TCP is not suitable for some applications such as streaming and realtime audio and video since TCP’s congestion control causes significant performance degradation on these applications. Many unicast congestion control schemes have been proposed to be better suited for these applications, where one common design goal of these schemes is to provide smooth rate changes and low delay jitter. Along with this design goal, many researchers have also considered that these new congestion control schemes should share the network resources fairly with competing TCP flows. In some schemes, like TCP, the congestion window is used to control the number of outstanding packets in the network. This approach is called window-based congestion control. However, they use different rules to increase or decrease the congestion window. GAIMD [53] uses a moderate window decrease factor to reduce abrupt rate changes and an appropriate increase factor is determined to provide TCP-friendliness. SIMD [27] and the Binomial algorithm [7] propose non-linear algorithms in increasing or decreasing the congestion window; the congestion
window is adjusted by a receiver’s ACK, but increase or decrease is not linear.

The other approach, called rate-based, actually adjusts the transmission rate by controlling the inter-packet spacing. RAP [42] mimics TCP’s AIMD property by increasing or decreasing the sending rate according to the receiver’s ACK stream. TFRC [20] adjusts the sending rate using a control equation (Eqn. 2.1) derived from the model of TCP’s long-term throughput [39].

\[
T_{TCP} = \frac{s}{RTT \left( \sqrt{\frac{2p}{3}} + (12 \sqrt{\frac{3p}{8}})p(1 + 32p^2) \right)},
\]

where \(T_{TCP}\) is a function of the steady-state loss event rate \(p\), the TCP round-trip time \(RTT\), and the packet size \(s\). The measured loss rate and round-trip time are plugged in this equation to compute the next sending rate. Smooth rate changes are achieved from the mechanism used in measuring the loss rate.

Along with large body of work in unicast congestion control, many multicast congestion control schemes also have been proposed. Some unicast congestion control schemes have motivated multicast congestion control schemes and some parts of unicast schemes have been used in multicast congestion control design.

### 2.2 Single Rate Multicast Congestion Control

With single rate congestion control schemes, the sender transmits at a rate for the receiver who experiences the worst congestion. This receiver is often called the slowest receiver...
and defined as the receiver whose measured loss rate and round-trip time at a given time results in the lowest throughput in the control equation (Eqn. 2.1). All receivers in a multicast session receive data at the same rate. Thus, this approach does not scale to large sets of receivers with heterogeneous available bandwidth. Even though single rate schemes have the limitation of scalability, there have been considerable works due to the simplicity of the design and since many applications have small relatively heterogeneous groups of receivers. Principal among these approaches is to guarantee that the sender receives acknowledgment information from the receiver with the worst network congestion while avoiding feedback implosion. pgmcc [43] and TFMCC [52] are examples of well designed and tested single rate multicast congestion controls.

pgmcc (pragmatic general multicast congestion control) runs a window-based control scheme to mimic TCP congestion control between the sender and the receiver with the worst network conditions, called acker. An ACK sent by the acker for each received packet will adjust the congestion window size of the sender. Each receiver measures its loss rate and its RTT (Round Trip Time) and this information is delivered to the sender for the acker election procedure by employing randomized feedback timers to avoid implosion. The acker is then selected as the receiver with the slowest throughput by using the simplified TCP equilibrium equation: $T = \frac{1}{RRT \sqrt{p}}$, where $RRT$ is the round-trip time and $p$ is loss rate.

TFMCC (TCP Friendly Multicast Congestion Control) provides smooth rate change over time, which makes it suitable for applications such as streaming media. In TFMCC,
each receiver measures its loss rate and its RTT. While pgmcc only uses this information for the selection of acker, TFMCC takes this information to compute the expected rate in addition to the selection of the slowest receiver, called the \textit{current limiting receiver (CLR)}. The fundamental idea to compute the expected rate is brought from a TCP-friendly unicast congestion control, TFRC.

The CLR sends its expected rate to the sender once every RTT and the feedback from the CLR adjusts the transmission rate for the multicast session. The feedback suppression scheme with a randomized timer is used to avoid the feedback implosion.

In single rate congestion control schemes, a receiver on highly congested link can degrade the performance of all receivers. To avoid this case, a policy decision may be employed to force such a receiver to leave the multicast group.

\section{Multiple Rate Multicast Congestion Control}

Multiple rate congestion control schemes tend to provide better scalability by allowing a different throughput to each receiver based on its current network conditions. Principal among these approaches is to provide a mechanism where 1) each receiver must share the network resources fairly with competing TCP flows, 2) a receiver must not affect the performance of other receivers and 3) the cost for supporting multiple rate scheme should be minimized. One of the challenges associated with layered multicast is how each receiver infers the available bandwidth along the path to the sender, which is the main component
of layered multicast congestion control.

**RLM**

Receiver-driven Layered Multicast (RLM) was the first multiple rate scheme proposing a layered multicast approach for video transmission [37]. In Chapter 1, we described the basics of layered multicast, whereby different multicast groups within the multicast session transmit at different rates, and participants use IGMP messages to join and leave groups to adjust their rate. RLM enforces *cumulative layering*, which imposes an ordering on the multicast layers and requires clients to subscribe and unsubscribe to layers in sequential order. This layering scheme transmits on the base layer at normalized rate 1 and transmits across all other layers $i \leq 1$ at rate $2^{i-1}$.

An RLM receiver periodically performs a join experiment by subscribing to an additional layer for a certain period of time. If the receiver does not experience packet loss during the join experiment, it remains on the subscribed layer. Packet loss during normal transmission induces the receiver to drop the highest layer. Although RLM achieves scalability by using a receiver-driven methodology, it has the following drawbacks. The join attempt can introduce packet loss at other hosts. RLM does not take TCP-fairness into account when each receiver adjusts its reception rate by changing the subscription level. Thus, it does not exhibit fairness when it competes with TCP flows for the same bottleneck link.
These limitations of RLM motivated Vicisano et al. to propose the Receiver-driven Layered Congestion Control (RLC) algorithm [49]. Their approach cleverly synchronizes join experiments by having the sender periodically and temporarily double the sending rate on each layer in turn. Figure 2.1 shows the sequence of packet transmission of each layer. A receiver joins a higher layer only if there was no packet loss on its uppermost layer during such an experiment. Since the rates on the layers are exponentially spaced using a doubling scheme, the burst during the join experiment simulates the join of next layer and no packet loss during the burst indicates that increasing the subscription level may be safe. RLC emulates TCP’s linear increase on average over large time scales by spacing the decision time to join on the layers proportionally to the bandwidth corresponding to the subscription level. Dropping the highest layer when there is packet loss decreases the reception rate by half. While the primary goal of RLC is to be friendly to TCP traffic, RLC
does not provide true TCP-fairness under certain network conditions since this approach did not consider each receiver's RTT. Another problem in RLC is the inaccuracy of inferring available bandwidth through the join attempt. Since the buffer size of the router with the bottleneck link would affect the accuracy of join attempt, there is no guarantee that the join of the next layer is safe even if there is no packet loss during the burst. In particular, the burst may not be large enough to induce packet loss.

**FLID-DL**

Byers *et al.* propose Fair Layered Increase/Decrease with Dynamic Layering (FLID-DL), which remedies the potentially significant problems associated with large IGMP leave latencies and abrupt rate increases in earlier layered schemes [10]. In all previous layered schemes, a receiver sends an IGMP leave message to the last hop router to stop receiving content from a multicast group. However, in the current IGMP group membership protocol, the leave operation can take several seconds to stop the flow of packets. Large leave latencies induce a slow response to congestion and impose unfairness to other fast reacting congestion control schemes. FLID-DL introduces *dynamic* layers to solve this problem, where the sender decreases the transmission rate of each layer in a series of steps to zero rate (Fig. 2.2). Thus if a receiver does not take any action (i.e., it does not issue any IGMP join/leave message), the receiver can decrease its reception rate. The receiver needs to periodically join layers to maintain the same average reception rate. FLID-DL
also introduces a scheme for placing the increase signals inside packets in such a way that the average throughput is similar to a TCP flow with a fixed RTT value experiencing the same loss rate. Despite the considerable improvement over RLC, FLID-DL does have some flaws including a large amount of join and leave control traffic, no concern about heterogeneous RTT among receivers, and a lack of sensitivity to RTT.

**WEBRC**

Wave and Equation Based Rate Control Using Multicast Round Trip Time (WEBRC) proposed by Luby *et al.* provides several advantages over previous multiple rate congestion control schemes such as RLC and FLID-DL. The key advantages of WEBRC are that the reception rate of each receiver is fine-grained, it provides good fairness to TCP, and it overcomes some of RLC’s and FLID-DL’s drawbacks as we described above. WEBRC
not only employs equation based congestion control proposed by TFRC [20] to compute the target rate based on RTT and loss rate, but also uses the dynamic layering scheme proposed by FLID-DL to avoid the large IGMP leave latencies. While FLID-DL employs a step-like decreasing pattern for the transmission rate of each layer, WEBRC decreases the transmission rate of each layer, called _wave channel_ in WEBRC, exponentially. The rates of all the dynamic layers follow the same pattern but with different time shifts. Thus, in order to maintain the same rate, a WEBRC receiver joins each consecutive layer at the same point in its descent. The receiver joins each consecutive layer slightly earlier in its descent to increase its reception rate slightly. To decrease its reception rate, the receiver joins the next wave later in its descent. With this approach, the receiver can get the fine-
grained reception rate while avoiding the large IGMP leave latencies. Figure 2.3 shows how the receiver adjusts its reception rate as a function of time. The y-axis indicates the aggregate reception rate of the receiver from the subscribed layers. WEBRC introduces the multicast round trip time (MRTT), which is the time between when a join is sent for a wave and when the packet flow from that wave is received. MRTT provides ideal scalability in the measurement of round trip time since it can be measured by receivers without sending any extra messages. The receiver computes the expected target rate based on the loss rate and the MRTT using the equation-based rate control mechanism. While WEBRC exhibits several advantages over previous schemes, it suffers from rather high complexity.
Chapter 3

Fine-Grained Layered Multicast With STAIR

3.1 Introduction

Layered multicast congestion control schemes have been proposed to address the heterogeneous available bandwidth across receivers and these layered schemes tend to employ cumulative layering, which mandates that each receiver always subscribe to a set of layers in sequential order. In conventional layering schemes, the rates for layers are exponentially distributed: the base layer’s transmission rate is $B_0$, and all other layers $i$ transmit at rate $B_0 \times 2^{i-1}$. Therefore, subscription to an additional layer doubles the receiver’s reception rate. Reception rate increase granularity of those schemes is unlike TCP’s fine-grained additive-increase, multiplicative decrease (AIMD). Because of this coarse granularity, rate
increases are necessarily abrupt, which runs the risk of buffer overflow; therefore, receivers must carefully infer the available bandwidth before subscribing to additional layers.

A different approach advocating fine-grained multicast congestion control to simulate AIMD was proposed in [13]. We refer to this approach as FGLM (Fine-Grained Layered Multicast). FGLM relies on non-cumulative layering and careful organization of layer rates to enable a receiver to increase the reception rate at the granularity of the base layer bandwidth $B_0$. Unlike earlier schemes, in this scheme, all receivers act autonomously with no implicit or explicit coordination between them. One substantial drawback of this approach is a constant hum of IGMP traffic at each last hop router (1 join and 2 leaves per client at every additive increase decision point). This volume of control traffic is especially problematic for last hop routers with large fanout to one multicast session, or those serving multiple sessions. Another drawback is that this approach incurs some bandwidth dilation at links, wasted bandwidth introduced by the uncoordinated activities of the set of downstream receivers. Finally, the use of non-cumulative layers is only amenable to applications which can make use of an arbitrary (and frequently changing) subset of subscription layers over time. The most natural applications of which we are aware are those in which any packet on any layer is equivalently useful to every receiver; such a situation arises in the digital fountain approach defined in [14], which facilitates reliable multicast by transmitting content encoded with fast forward error correcting codes.

Our work presents a better method for simulating true AIMD multicast congestion
control. At a high level, our STAIR\(^1\) (Simulate TCP’s Additive Increase/multiplicative
decrease with Rate-based) multicast congestion control algorithm enables reception rates
at receivers to follow the familiar sawtooth pattern which arises when using TCP’s AIMD
congestion control. We facilitate this by providing two key contributions. First, we define
a stair layer, a layer whose rate dynamically ramps up over time from a base rate of one
packet per RTT up to a maximum rate before dropping back to the base rate. The primary
benefit of this component is to facilitate additive increase automatically, without the need
for IGMP control messages. Second, we provide an efficient hybrid approach to combine
the benefits of cumulative and non-cumulative layering \textit{below} the stair layer. This hy-
brid approach provides the flexibility of non-cumulative layering, while mitigating several
of the performance drawbacks associated with pure non-cumulative layering. While our
STAIR approach appears complex, the algorithm is straightforward to implement and easy
to tune; it delivers data to each receiver at a rate that is in very close correspondence to the
behavior of a unicast TCP connection over the same path; and it does so with a quantifiable
and reasonable bandwidth cost.

\section*{3.2 Definitions and Building Blocks for our Approach}

In order to motivate our new contributions, we begin with techniques from previous work
which relate closely to our approach. In [13], four metrics for evaluating a layered mul-

\^1This work appeared in the proceedings of NGC’01 [11]. This work with FGLM [13] which motivates
STAIR will appear in ACM/IEEE Transactions on Networking [12].
Figure 3.1: Dilation of a shared bottleneck link. (a) Cumulative: Dilation = 1, (b) Non-Cumulative: Dilation = 12/9.

ticast congestion control scheme are provided. Here, we recapitulate join complexity and dilation.

**Definition 3.2.1.** The join complexity of an operation (such as additive increase) under a given layering scheme is the number of multicast join messages a receiver must issue in order to perform that operation in the worst case.

Another significant problem of non-cumulative schemes is the need for extra bandwidth to accommodate receivers, which the following example in Fig. 3.1 illustrates. Consider two receivers $R_1$ and $R_2$ who share a bottleneck link $l$ and wish to receive at rates 9 and 4, respectively. In the cumulative setting (Fig. 3.1 (a)), $R_1$ must settle for a reception rate of 8, which it can achieve by subscribing to the first four layers $[1,1,2,4]$. Meanwhile $R_2$ can achieve its target rate by subscribing to the first three layers $[1,1,2]$. Since $R_2$ subscribes to a subset of the layers that $R_1$ subscribes to, the demand on link $l$ is identical to that placed by $R_1$.

But in a non-cumulative scenario (Fig. 3.1(b)), $R_1$ can now subscribe to layers one and five to achieve its target rate exactly, while $R_2$ still subscribes to the first three layers. This
increases the end-to-end rate perceived by $R_1$ by a single unit, yet the load on link $l$ now jumps from eight to twelve. The requirement of additional bandwidth is a fundamental consequence of non-cumulative layering.

**Definition 3.2.2.** For a layering scheme which supports reception rates in the range $[1, R]$, and for a given link $l$ in a multicast tree, let $M_l \leq R$ be the maximum reception rate of the set of receivers downstream of $l$ and let $C_l$ be the bandwidth demanded in aggregate by receivers downstream of $l$. the dilation of link $l$ is then defined to be $C_l/M_l$. Similarly, the dilation imposed by a multicast session on tree $T$ is taken to be $\max_{l \in T}(C_l/M_l)$.

In the example above, the dilation of the shared link was 1 in the cumulative case and $\frac{12}{9}$ in the non-cumulative case. In general, cumulative layering enforces a guarantee that links are never dilated, i.e. have a dilation of 1. The worst-case dilation imposed by the basic non-cumulative layering scheme grows to 2. In fact, the worst case is when one receiver is subscribed to just the base layer and the highest layer, and another is subscribed to the base layer and all other layers except the highest layer. This worst case dilation can be shown to be $2 - \frac{4}{R+2}$.

Table 3.1 shows a performance comparison along these metrics of various layering schemes which attempt to perform AIMD congestion control. Briefly, one cannot perform additive increase in a standard cumulative protocol, and while non-cumulative schemes can do so, they do so only with substantial control traffic and/or bandwidth dilation per operation.

---

2Standard refers to the doubling scheme described in the introduction of this chapter.
<table>
<thead>
<tr>
<th>Sequence</th>
<th>Dilation</th>
<th>Complexity of AI</th>
<th>Complexity of MD</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ideal</td>
<td>1</td>
<td>zero</td>
<td>zero</td>
</tr>
<tr>
<td>Std. Cum</td>
<td>1</td>
<td>N/A</td>
<td>1 leave</td>
</tr>
<tr>
<td>Std. NonCum</td>
<td>2</td>
<td>$O(\log R)$</td>
<td>$O(\log R)$</td>
</tr>
<tr>
<td>FGLM [13]</td>
<td>1.6</td>
<td>2 joins, 1 leave</td>
<td>1 leave</td>
</tr>
</tbody>
</table>

Table 3.1: Performance of AIMD Congestion Control for Various Approaches

We briefly sketch one non-cumulative layering scheme used [13]. The layering scheme is defined by $B_0 = 1, B_1 = 2$, and $B_i = B_{i-1} + B_{i-2} + 1$ for $i \geq 2$. The first few rates of the layers for this scheme are 1, 2, 4, 7, 12, 20, 33, ..., where the base rate can be normalized arbitrarily. Increasing the reception rate by one unit can be achieved by the following procedure: Choose the smallest layer $i \geq 0$ to which the receiver is not currently subscribed; then subscribe to layer $i$ and unsubscribe from layers $i - 1$ and $i - 2$. A receiver can approximately halve its reception rate by unsubscribing from its highest subscription layer. While this does not exactly halve the rate, the decrease is bounded by a factor which lies in the interval from approximately 0.4 to 0.6.

One salient issue with FGLM is that the base layer bandwidth $B_0$ is fixed once for all receivers. Setting $B_0$ to a small value mandates frequent subscription changes (via IGMP control messages) for receivers with small RTTs. Setting it to be large causes the problems of abrupt rate increases and buffer overruns that FGLM is designed to avoid.
3.3 Components of Our Approach

In this section, we describe our two main technical contributions. The first contribution is a method for minimizing the performance penalty associated with non-cumulative layering by employing a hybrid strategy which involves both cumulative and non-cumulative layers. Our approach retains all of the benefits of fine-grained multicast advocated in [13], with the added benefit that the dilation can be reduced from 1.62 down to $1 + \varepsilon$ with only a small increase in the number of multicast groups. The second contribution introduces new, dynamic stair layers, which facilitate fine-grained additive increase without requiring a substantial number of IGMP control messages. Taken together, these features make the fine-grained layered multicast approach much more practical.

3.3.1 Combining Cumulative and Non-Cumulative Layering

In a conventional cumulative organization of multicast layers, only cumulative layers are used to achieve rates in the normalized range $[1, R]$.

- **Cumulative Layers (CL):** The base layer rate is $C_0 = 1$, and for all other layers $CL_i$, $1 \leq i \leq \log_\alpha R$, the rate of $C_i = C_0 \times \alpha^{i-1}$. When $\alpha = 2$, this corresponds to doubling of rates as each layer is added.

In the fine-grained multicast scheme of [13], only non-cumulative layers are used to achieve a spectrum of rates over the same normalized range.

---

3 Bandwidths can be scaled up multiplicatively by a base layer bandwidth $B_0$ in both of schemes.
Figure 3.2: Hybrid Layer Scheme: $K = \alpha^j + r$. For a target rate of 13, with $\alpha = 2$, $K = 2^3 + 5$. $C_i$ denotes the rate on cumulative layer $i$, $N_i$ denotes the rate on non-cumulative layer $i$.

- **Non-Cumulative Layers (NCL):** The non-cumulative layering scheme Fib1 presented in [13] is defined by the Fibonacci-like sequence $N_0 = 1, N_1 = 2$, and $N_i = N_{i-1} + N_{i-2} + 1$ for $i \geq 2$.

Note that both CLs and NCLs are static layers for which the transmission rate to the layer fixed for the duration of the session.

In the hybrid scheme which we propose, we will require that both a set of cumulative layers $CL_i$ and a set of non-cumulative layers $NCL_i$ are available for subscription. To attain a given subscription rate $K$, a receiver will subscribe to set of cumulative layers to attain a rate that is the next lowest power of $\alpha$, capped by a set of non-cumulative rates to achieve a rate of exactly $K$, as depicted in Figure 3.2. In particular, we let $j = \lfloor \log_\alpha K \rfloor$ and write $K = [\alpha^j] + r$, then subscribe to layers $CL_0, \ldots, CL_j$ as well as the set of non-cumulative layers that the Fib1 scheme would employ to attain a rate of $r$. As prescribed
by FGLM, fine-grained increase (adding $C_0$) requires one join and two leaves, except for the relatively infrequent case when we move to a rate that is an exact power of $\alpha$. In this case, we unsubscribe from all non-cumulative layers and subscribe to one additional cumulative layer. Multiplicative decrease now requires one leave from a cumulative layer and one leave from a non-cumulative layer. Leaving the highest CL reduces the reception rate on CLs by a factor of $\alpha$, i.e. the rate on CLs is cut in half when $\alpha = 2$. Similarly, leaving the highest NCL decreases the reception rate on NCLs by approximately half.

Comparing against a standard non-cumulative scheme, which used $\log_{1.6} R$ layers to obtain integral rates $[1, R]$, we now require $\lceil \log_{\alpha} R \rceil$ layers for the cumulative part, plus roughly $\lceil \log_{1.6}(R - (R/\alpha)) \rceil$ non-cumulative layers. This constitutes a constant factor increase in number of layers used.

An improvement of dilation in hybrid scheme is expressed as the following lemma.

**Lemma 3.3.1.** The dilation of the hybrid scheme is $1 + 1.62 \frac{(\alpha - 1)}{\alpha}$.

**Proof.** We proceed by proving an upper bound on the dilation $D$ of an arbitrary link $\ell$, which gives a corresponding bound on the dilation of the session. For each user $U_i$ downstream of $\ell$, denote the rate it obtains over the cumulative layers by $a_i$, the rate it obtains over non-cumulative layers by $b_i$, and the total rate by $u_i$. Let the user with maximal total rate be denoted by $\hat{U}$, and let its corresponding rates be $\hat{a}, \hat{b}$, and $\hat{u}$ respectively. Now consider an arbitrary user $U_i$. By definition of $\hat{U}$, and from the organization of rates, $a_i \leq \hat{a}$. If $a_i < \hat{a}$, then by the layering scheme employed, $u_i = a_i + b_i < \alpha a_i$. Adding $\alpha b_i$
to both sides gives \( u_i + \alpha b_i < \alpha a_i + \alpha b_i = \alpha(u_i) \). Simplifying yields

\[
b_i < \frac{u_i(\alpha - 1)}{\alpha} \leq \frac{\hat{u}(\alpha - 1)}{\alpha}.
\]

Otherwise, if \( a_i = \hat{a} \), then by maximality \( b_i \leq \hat{b} < (\alpha - 1)\hat{a} \). In either case, \( b_i < \frac{\hat{u}(\alpha - 1)}{\alpha} \), so \( \max_i b_i < \frac{\hat{u}(\alpha - 1)}{\alpha} \). From the dilation lemma proved in [13], a set of users subscribing to non-cumulative layers experiences a dilation of at most 1.62. Thus the total bandwidth consumed by non-cumulative layers across \( \ell \) is at most \( 1.62 \max_i b_i \). Plugging these derived quantities into the formula in Definition 3.2.2 yields

\[
D \leq \frac{\hat{a} + 1.62 \max_i b_i}{\hat{a} + \hat{b}} < \frac{\hat{a} + 1.62(\alpha - 1)\hat{a}}{\hat{a} + \hat{b}}
\]

\[
D \leq 1 + 1.62 \frac{(\alpha - 1)}{\alpha}.
\]

\[\square\]

Applying this lemma to a hybrid scheme with a geometric increase rate of \( \alpha = 1.2 \) on the cumulative layers realizes the benefits of a non-cumulative scheme, reduces the worst-case dilation in the limit from 1.62 to 1.27 (a 22\% bandwidth savings) and requires only a modest increase in the number of groups. Figure 3.3 shows the maximal dilation at a link as the link bandwidth varies as a function of \( N_0 \) for Fib1 and the hybrid scheme for two different values of \( \alpha \).
3.3.2 Introducing Stair Layers

Our next contribution is stair layers, so named because the rates on these layers change dynamically over time, and in so doing resemble a staircase. This third layer that a sender maintains is used to automatically emulate the additive-increase portion of AIMD congestion control, without the need for IGMP control traffic. Different stair layers are used to accommodate additive increase for receivers with heterogeneous RTT’s from the source. These layers also smooth discontinuities between subscription levels of the underlying CLs and NCLs, which provide rather coarse granularity. In the subsequent discussion, we assume that these underlying layers have base rates $C_0 = N_0 = 1$Mbps for simplicity.

Stair Layers (SLs) are defined as follows.

- **Stair Layers (SL):** Every SL has two parameters: a round-trip time $t$ that it is designed to emulate and a maximum rate $R$. The rate transmitted on each SL is a cyclic step function with a minimum bandwidth of 1 packet per round-trip time and a maximum bandwidth of of $R$, a step size of one packet, and a stride rate of one
Figure 3.4: Depiction of $SL_{128}$ (a stair layer with $t = 128$ms) in isolation. $R = 1$Mbps, packet size = 1KB.

step per emulated RTT. Upon reaching the maximum attainable rate, the SL restarts at a rate of one packet per RTT.

Unlike CLs and NCLs, SLs are dynamic layers whose rates change over time. Dynamic layers were first used by [49] to probe for available bandwidth and later defined and used in [10] to avoid large IGMP leave latencies.

Figure 3.4 shows the transmission pattern of $SL_{128}$ (a stair layer for a 128ms RTT) with maximum rate $R = 16$ packets per RTT. Also depicted in Figure 3.4 is a third useful parameter of a stair layer:

**Definition 3.3.2.** The stair period of a given stair layer is the duration of time that it takes the layer to iterate through one full cycle of rates.

Given a stair layer with an emulated RTT $t$ and a maximum rate $R$ the stair period $p$ satisfies $p \propto Rt^2$. Typically, we will set the maximum rate $R$ of a stair layer to be the base rate of the standard cumulative scheme $C_0$ (in Mbps), in which case we substitute for $R$
and perform the appropriate conversions, assuming a fixed packet size $S$:

$$p = \left( C_0 \frac{t}{8S} \right) t.$$

We now consider the simple example depicted in Figure 3.5, which depicts the throughput of a receiver when it subscribes to NCLs and $SL_{128}$. To simplify the description of this example, we employ stair layers on top of a pure non-cumulative scheme; however, our algorithm and experiments use all three types of layers. For simplicity, let the stair period of $SL_{128}$ be 2 seconds. Let $N_0 = 1, N_1 = 2, N_2 = 4, N_3 = 7$ (all rates in Mbps). The receiver is subscribed to $NCL_1$ and $NCL_2$ at 12 seconds and has a total reception rate of 6 Mbps. By subscribing to $SL_{128}$ on top of $NCL_1$ and $NCL_2$, the receiver receives one more packet in every RTT. The sending rate of SL reaches $N_0$ at 14 seconds. SL then drops the sending rate to one packet per RTT at 14 seconds and resumes sending one more
packet in every RTT. The receiver compensates for the drop by subscribing to $NCL_0$ at 14 seconds for a total reception rate of 7 Mbps. At 16 seconds, the receiver unsubscribes from $NCL_1$ and $NCL_2$ and subscribes to $NCL_3$ to increase total reception to 8 Mbps.

Finally, we note that the addition of stair layers increases the dilation beyond that proven in Lemma 3.3.1, but when stair layers are a very small fraction of the overall bandwidth (as is typical), their contribution in aggregate to the dilation is only a small additive term.

### 3.4 The STAIR Congestion Control Algorithm

We now describe how the techniques we have described come together into a unified multiple rate congestion control algorithm. We employ a hybrid scheme as described in Section 3.3.1, from which each receiver selects an appropriate subset of layers, used in concert with one stair layer, appropriate for its RTT. The two most significant challenges to address are providing the algorithms to performing additive increase and multiplicative decrease, respectively. Two additional challenges we address are 1) incorporating methods for estimation of multicast RTTs and 2) establishing a set of appropriate stair layers.

#### 3.4.1 Additive Increase, Multiplicative Decrease

In order for a set of stair layers to complement a set of CLs and NCLs, the maximum rate of the stair layer must be calibrated to the base rate of the CLs and NCLs. The effect of
appropriately calibrated rates can be seen in Figure 3.5: at exactly those instants when
the stair layer recycles, the subscription rate on the NCL’s increases by $N_0$, to compensate
for the identical decrease on the stair layer. Now in order to conduct AIMD congestion
control, the receiver measures packet loss over each stair period, during which additive in-
crease takes place automatically. If there is no loss, then the receiver performs an increase
of $N_0$. As described earlier, this entails 1 join and 2 leaves, or $k$ leaves (where $k$ is the
number of subscribed NCLs) when the stair period is an exact power of $\alpha$. (As an aside,
we note that it may be much more efficient for a last-hop router to handle such a batch of
IGMP leave requests, rather than handling them as $k$ separate requests).

Conversely, if there is a packet loss event in a stair period (of one or more losses), then
one round of multiplicative decrease is performed. Approximately decreasing the rate by
half is straightforward – it is necessary to drop the top cumulative layer as well as the top
non-cumulative layer. We also note that there is no particular reason to wait until a stair
period terminates before conducting multiplicative decrease – it can be done any time.

### 3.4.2 Configuration of Stair Layers

As motivated earlier, to accommodate a wide variety of receivers, stair layers must be
closedly configured carefully. We choose to space the RTTs across the available stair layers expo-
nentially while noting that our methods generalize to other settings. Let the RTT in the
base stair layer be $2^i$ ms. The base stair layer increases its sending rate every $2^i$ ms and all
the other stair layers $j$ will increase the sending rate in every $2^{j+i}$ ms. The TCP throughput
rate $R$, in units of packets per second, can be approximated by the formula in [20] (derived
by applying simplifying assumptions to a formula in [39]):

$$R = \frac{1}{RTT \sqrt{q} (\sqrt{2/3} + 6 \sqrt{3/2} q(1 + 32q^2))} \quad (3.1)$$

where $R$ is a function of the packet loss rate $q$ and RTT is the TCP round trip time. Since
the throughput is inversely proportional to RTT, the receiver with a small RTT is more
sensitive to the throughput than the receiver with a large RTT, thus we recommend that
RTTs provided by stair layers be exponentially spaced. Note that with an exponential
spacing of stair layers, a receiver may subscribe to a different SL if its measured RTT
changes significantly: it can subscribe to a faster layer at the end of its current stair period,
or drop down to a slower stair layer every other stair period.

### 3.4.3 RTT Estimation and STAIR Subscription

In order to be TCP-friendly, each STAIR receiver must measure or estimate its RTT to
subscribe to appropriate stair layers. Our goal is to minimize the discrepancy between the
throughput received by TCP and by STAIR for a given RTT, using the following measure.

**Definition 3.4.1.** The throughput discrepancy of STAIR with a round-trip time $R$ is the
ratio of the throughput of TCP with round-trip time $R$ to the throughput of a STAIR receiver
with round-trip time $R$, under identical packet loss rates.

A variety of methods can be employed to measure the RTT; we describe three such
Figure 3.6: Range of RTT to subscribe a stair layer, \(RTT_i = 2^j\) for some \(i\) and \(j\).

possibilities, with the expectation that any scalable method can be employed in parallel with our approach. Golestani et al. [22] provide an effective mechanism to measure RTT in multicast using a hierarchical approach. However, their approach requires clock synchronization among the sender and receivers and depends on some router support which is not widely available. Another simple way to estimate RTT is to use one of various ping-like utilities. However, one cost associated with use of ping is that as the number of receivers increase the sender faces a “ping implosion” problem.

With an estimate of its RTT, \(E\), a STAIR receiver then subscribes to appropriate stair layers. We first describe and evaluate a simple option involving a single subscription, then describe a more complex option that reduces the worst-case throughput discrepancy.

**Simple Option:** Subscribe to the unique stair layer \(i\) satisfying \(2/3RTT_i < E \leq 4/3RTT_i\) (Figure 3.6).

This simple subscription policy emulates AIMD of a TCP experiencing a round-trip time of \(RTT_i\). When compared with a TCP experiencing the “correct” round-trip time of \(E\), it is clear that the throughput discrepancy lies in the interval \([0.66, 1.33]\).

**Complex Option:** If there exists an \(i\) such that \(RTT_i/\sqrt{2} < E \leq \sqrt{2}RTT_i\), then subscribe to stair layer \(i\) (case 1 in Figure 3.7). If \(E\) is within a \(\sqrt{2}\) factor from the geometric mean of
Figure 3.7: Range of RTT to subscribe two stair layers, $RTT_i = 2^j$ for some $i$ and $j$.

$RTT_i$ and $RTT_{i+1}$, then subscribe to stair layers $i + 1$ and $i + 2$ (case 2 in Figure 3.7).

The intuition behind this complex option is that when the measured RTT lies midway between available options, the superposition of two stair layers provides a closer approximation to the appropriate additive increase rate than any one layer can. Using this approach, the throughput discrepancy lies in the interval $[0.73, 1.19]$. Figure 3.7 shows how to subscribe the appropriate stair layers from the measured $RTT_m$ to reduce the throughput discrepancy.

In the AIMD$(a, b)$ scheme [19], the sending window is increased by $a$ packets once every $R$ seconds and cut by a factor $(1 - b)$ in case of a packet loss. The throughput of TCP with an AIMD$(a, b)$ scheme is:

$$
\hat{T} = \frac{\sqrt{2 - b}\sqrt{a}}{\sqrt{2bR}\sqrt{p}}
$$

(3.2)

Now consider the throughput discrepancy based on each option where the measured $RTT_m$ is the geometric mean of $RTT_i$ and $RTT_{i+1}$. 
• \( \hat{T}_g \): the approximate throughput of a TCP receiver with the geometric mean of \( RTT_i \) and \( RTT_{i+1} \). \( \hat{T}_g \) can be computed from AIMD(1, 1/2) with \( RTT = \sqrt{2} \times 2^j \).

• \( \hat{T}_s \): the approximate throughput of a STAIR receiver by subscribing to \( S_{i+1} \) based on the simple option 1. \( \hat{T}_s \) can be computed from AIMD(1, 1/2) with \( RTT = 2^{i+1} \).

• \( \hat{T}_a \): the approximate throughput of a STAIR receiver by subscribing to \( S_{i+1} \) and \( S_{i+2} \) stair layers based on the advanced option 2. \( \hat{T}_a \) can be computed as AIMD(3/2, 1/2) with \( RTT = 2^{i+1} \).

Plugging these quantities into the formula 3.2 yields:

\[
\hat{T}_g = \frac{\sqrt{2} - 0.5 \sqrt{1}}{\sqrt{2} \times 0.5 \times \sqrt{2} \times 2^j \sqrt{p}}, \quad \hat{T}_s = \frac{\sqrt{2} - 0.5 \sqrt{1}}{\sqrt{2} \times 0.5 \times 2^{j+1} \sqrt{p}}, \quad \text{and} \quad \hat{T}_a = \frac{\sqrt{2} - 0.5 \sqrt{1.5}}{\sqrt{2} \times 0.5 \times 2^{j+1} \sqrt{p}}.
\]

The ratio of \( \hat{T}_s \) to \( \hat{T}_g \) and the ratio of \( \hat{T}_a \) to \( \hat{T}_g \) are as follows.

\[
\frac{\hat{T}_s}{\hat{T}_g} = \frac{\frac{1}{2^{j+1}}}{\frac{1}{2 \sqrt{2} \times 2^j}} = 0.71 \quad \text{and} \quad \frac{\hat{T}_a}{\hat{T}_g} = \frac{\frac{1}{2^{j+1}}}{\frac{1}{\sqrt{2} \times 2^j}} = 0.87.
\]

By subscribing to \( S_{i+1} \) and \( S_{i+2} \) when the \( RTT_m \) is the geometric mean of \( RTT_i \) and \( RTT_{i+1} \), we can reduce the throughput discrepancy from 29% to 13%.

Figure 3.8 shows the throughput discrepancy of each option. The range of discrepancy in option 1 is [0.66, 1.33], while option 2 has the range of discrepancy [0.73, 1.19]. The worst case throughput discrepancy of the option 2 occurs when the \( RTT_m \) is \( \sqrt[4]{2} \times 2^j \) while
Figure 3.8: Throughput Discrepancy of Each Option

the throughput discrepancy of option 1 is 1.19:

$$\frac{T_s}{T_g} = \frac{1}{2^j} = 1.19$$  and  $$\frac{\mu}{T_g} = \frac{\sqrt{2} + 1}{\sqrt{2}e^{2j}} = 0.73.$$

Alternatives exist to further reduce the throughput discrepancy, albeit with additional complexity.

### 3.5 Experimental Evaluation

We have tested the behavior of STAIR using the *ns* simulator [38]. Results of extensive simulations show that STAIR exhibits good inter-path fairness when competing with TCP traffic in a wide variety of scenarios. The general network configuration we consider is depicted in Figure 3.9. This topology generalizes both the standard “dumbbell” topology and the tree topology, and it allows for heterogeneous receivers (both bandwidth and delay) and multiple bottlenecks. On top of this topology, each of *k* STAIR receivers competes
with $X$ TCP flows that experience the same network conditions as that receiver. To test TCP-friendliness, our experiments compare the behavior of each STAIR receiver with the behavior of the $X$ competing TCP flows.

By setting link bandwidths and delays appropriately (for example, see Figure 3.13), we can establish situations with multiple target rates across the STAIR receivers. Typically, we configure the A-B link to have ample bandwidth; alternatively, we can establish it as an additional bottleneck link. With this configuration, we can tune the cross-traffic multiplexing level by varying the values of $X$ and $K$, we can vary the link bandwidths and latencies, and we can scale the queue sizes. Throughout our experiments, we set $C_0 = N_0 = 512$ Kbps and set $\alpha = 2$, i.e. the rate $c_i = 2^{i-1} \times 512$ Kbps for $i > 0$, and employ a fixed packet size of 512B throughout. We use stair layers emulating exponentially spaced RTTs starting at 16ms. Also, while there is theoretical justification for smaller settings of $\alpha$, we did not observe worst-case dilation often in our simulations.

Figure 3.9: Our network configuration.
In the experiments we describe here, we follow recommended guidelines for conducting multicast congestion control simulations [9]. In particular, we use RED gateways, primarily as a source of randomness to remove simulation artifacts such as phase effects that may not be present in the real world. Use of RED vs. drop-tail gateways does not appear to materially affect performance of our protocol. The RED gateways are set up in the following way: we set the queue size to twice the bandwidth-delay product of the link, set \texttt{minthresh} to 5\% of the queue size and \texttt{maxthresh} to 50\% of the queue size with the \texttt{gentle} setting turned on. Our TCP connections use the standard TCP Reno implementation provided with \textit{ns}.

We begin with a simple scenario to test TCP-fairness in which one STAIR receiver flow is competing with seven TCP Reno flows (i.e. \( K = 1 \) and \( X = 7 \)). The RTT is set to 32ms and the bottleneck link is set to 50Mbps. We plot the throughput of STAIR flow and the throughput of a representative competing TCP flow in Figure 3.10. Throughout this section, we choose representative TCP connections to avoid excess clutter in the plots. Our

![Figure 3.10: TCP flows and One STAIR with RED](image-url)

In the experiments we describe here, we follow recommended guidelines for conducting multicast congestion control simulations [9]. In particular, we use RED gateways, primarily as a source of randomness to remove simulation artifacts such as phase effects that may not be present in the real world. Use of RED vs. drop-tail gateways does not appear to materially affect performance of our protocol. The RED gateways are set up in the following way: we set the queue size to twice the bandwidth-delay product of the link, set \texttt{minthresh} to 5\% of the queue size and \texttt{maxthresh} to 50\% of the queue size with the \texttt{gentle} setting turned on. Our TCP connections use the standard TCP Reno implementation provided with \textit{ns}.

We begin with a simple scenario to test TCP-fairness in which one STAIR receiver flow is competing with seven TCP Reno flows (i.e. \( K = 1 \) and \( X = 7 \)). The RTT is set to 32ms and the bottleneck link is set to 50Mbps. We plot the throughput of STAIR flow and the throughput of a representative competing TCP flow in Figure 3.10. Throughout this section, we choose representative TCP connections to avoid excess clutter in the plots. Our
methodology is to choose the TCP connection whose mean rate was closest to the average of all TCP flows. Both the average rate across all TCP flows and the standard deviation are depicted in the plots. In this example, the average throughput attained by the STAIR receiver vs. the TCP flows was 4.87Mbps vs. 4.95Mbps, demonstrating fair sharing of the bottleneck link.

In the first experiment, the RTT was favorably set to a value (32ms) that provided an exact match to a stair layer. Next, we vary the RTT between the sender and the receiver on the link to see the presence of throughput discrepancy induced by rounding to the nearest stair layer as described in section 3.4.3. We consider one STAIR receiver competing with ten TCP flows on the bottleneck link while varying the RTT by 10ms from 20ms up to 140ms, and using the simple subscription option. The measured throughput of STAIR receiver and TCP receivers are depicted in Figure 3.11 with a 98% confidence interval over 100 trials. We also plot the expected STAIR throughput, which is computed by multiplying average the TCP throughput by the ratio of the rounded RTT to the actual RTT on the link.
As we expect, the throughput of STAIR is closest to that of TCP when the RTT is close to the power of two, and the measured throughput discrepancy in our experiments lies very close to the predicted value.

We next consider topologies with considerable receiver heterogeneity, to verify the ability of STAIR to perform fine-grained multiple rate control. We consider a single STAIR session with $K = 3$ and $X = 10$, but with different RTTs to the three receivers. The bandwidth on all links is set to 200 Mbps, and the RTT of STAIR receiver 1, S1, is 32ms, the RTT of S2 is 64ms, and the RTT of S3 is 128ms. This experimental set up makes the A-B
Figure 3.13: Topology for Many Heterogeneous Receivers

link (see Figure 3.9) the bottleneck for all STAIR and TCP receivers. In our experimental set up, each receiver periodically samples the RTT using ping. Since the throughput of TCP is inversely proportional to RTT, the receiver S2 should have approximately half of S1’s average throughput. The throughput of each of the flows is plotted in Figure 3.12. All of the STAIR flows share fairly with the parallel TCP flows with the same RTT and the receiver with large RTT gets the small throughput as expected. In this experiment, the average throughput attained by S1, S2, and S3 was 8.34Mbps, 3.95Mbps, and 1.98Mbps respectively. Each measured STAIR value was well within a standard deviation from the mean value across competing TCP connections.

Finally, we used a topology with multiple bottlenecks (Figure 3.13) to test the performance of STAIR with a set of heterogeneous reception rates. We consider a single STAIR session with $K = 6$ and $X = 10$, but each STAIR receiver is not behind the same bottleneck link. S1 competes with 10 TCP connections on a 33Mbps link, giving a fair rate of 3 Mbps and the fair rates of the other STAIR receivers (S2 to S6) are 4Mbps, 5Mbps,
Figure 3.14: Throughput of STAIR receivers and TCP flows

6Mbps, 7Mbps, and 10Mbps respectively. We plot the throughput of each STAIR flow and the throughput of one of the competing TCP flows in Figure 3.14. Again, the level of
fairness between each of the six STAIR receivers and its competing TCP connections is excellent.

We measured the dilation as a function of time of link (A-B) in Figure 3.13 and plot this dilation in Figure 3.15. Recall that the dilation of a link was defined and bounds on the worst case dilation of STAIR were proven in Section 3.3.1. Figure 3.15 shows that in this example, the measured dilation is much smaller than the worst case bound of 1.81 when $\alpha = 2$. 

Figure 3.15: Dilation on the shared link (A-B)
Chapter 4

Leveraging Single Rate Schemes in Multiple Rate Multicast Congestion Control Design

4.1 Motivation

There have been considerable effort and numerous technical advances in multicast congestion control such as layered scheme (RLM), TCP-friendliness (RLC), fine-grained rate changes (FLID-DL), true AIMD multiple rate scheme (STAIR), and equation-based multiple rate scheme with MRTT (WEBRC), where all schemes have been surveyed in Chapter 2. However, existing methods lead to very complex multiple rate congestion control designs that are difficult to evaluate. The daunting complexity associated with delivering
different TCP-friendly rates to different participants within the session would be the main reason of an impediment to deployment of multicast services. Another evidence of the technical hurdles associated with multiple rate schemes is given by promising recent advances in single rate multicast congestion control, notably pgmcc [43] and TFMCC [52]. While these protocols are not designed to scale to large sets of receivers with heterogeneous available bandwidth, there is building consensus that these protocols are sufficiently mature and well-tested for Internet deployment. In this next chapter of this dissertation, we seek to leverage these advances. In particular, we explore a new direction in multiple rate multicast congestion control, namely building a multiple rate scheme from an ensemble of single rate sessions, each of which has their own independent control. The major advantage of this method is that it leverages proven single rate congestion control mechanisms to provide an effective multiple rate scheme with relatively little additional complexity. This is in contrast to all existing multiple rate congestion control schemes including STAIR, which provide only an integrated control mechanism across layers, and do not attempt to take advantage of control mechanisms within layers. As a result, these integrated controls are often extremely complex, and are difficult to test and validate.

---

1 Portions of this chapter appeared in the proceedings of IEEE INFOCOM’03 [28] and this work will appear in IEEE J-SAC Special Issue on Design, Implementation and Analysis of Communication Protocols 2004 [29].
4.1.1 Our Work in Context

Previous work in multiple rate multicast congestion control can be categorized as either using static or dynamic layers. In static schemes, e.g. [31, 37], the sending rate of any given layer remains fixed over time, and all adjustments to the reception rate are exclusively receiver-driven. This approach has some drawbacks, most notably that the receiver may have insufficient information to accurately conduct join attempts, as well as necessitating abrupt rate changes. Many other schemes use dynamic layers, or layers whose transmission rate changes over time according to a predetermined pattern. Dynamic layers have been used in a variety of clever ways, including implicit coordination of receivers behind a bottleneck [49], reduction of IGMP leave messages [10], simulation of additive increase [11], and to achieve equation-based congestion control [34]. However, implementations of these dynamic layering schemes typically have a great deal of embedded complexity to realize these benefits in practice.

One feature shared by most existing multiple rate methods is that the layer rates are non-adaptive, i.e. the schedule of packet transmissions on each group (whether fixed-rate or dynamic) is known to the sender and to the receivers in advance. A limitation of non-adaptive schemes is their inflexibility: there are typically only a small constant number of feasible control actions that may be taken by a receiver at a given time step. For example, in many non-adaptive schemes, the magnitude by which a receiver may instantaneously increase or decrease its rate is fixed a priori, and the times at which a rate increase can be performed are widely separated. Our work differs in this regard, since each of the
individual layers adaptively and asynchronously adjust their rates to the limiting receivers in the session, as we will describe.

Two methods for adaptive, layered multiple rate multicast were proposed in SAMM [50] and HALM [32]. However, the methods proposed in SAMM predated current notions of TCP-friendliness and were not evaluated in that context; moreover, extra router support to monitor the available bandwidth is required to achieve the best performance. The work in HALM is most similar to our own, in that they advocate periodic, adaptive reallocation of layer rates in a multiple rate multicast session and build upon single rate congestion control methods. However, their emphasis is on periodic optimization of the layer rates at coarse time scales (tens of seconds) that is not suitable for fine-grained congestion control on the Internet.

The remainder of this chapter is organized as follows. In Section 4.2, we outline a general approach to building a multiple rate congestion control from a single rate scheme. We then review the two single rate congestion control mechanisms in more detail that we subsequently leverage, TFMCC and pgmcc, in Section 4.3. In Section 4.4, we specify SMCC, which orchestrates an ensemble of TFMCC sessions to build a multiple rate congestion control scheme. We also build an alternative multiple rate scheme by employing the underlying pgmcc congestion control in Section 4.5. In Section 4.6, we propose a new additive increase join attempt which is performed by each receiver before joining the next layer. In Section 4.7, we give the results from ns simulations to demonstrate the fairness of SMCC with competing TCP flows.
4.2 Overview Of Our Approach

We describe a new multiple rate congestion control algorithm for cumulative layered multicast sessions that employs a single rate congestion control as the primary underlying control mechanism for each layer. The high-level features of our approach are as follows:

- Each receiver subscribes to a set of cumulative layers. We refer to a receiver as being an *active* participant in the uppermost layer to which it subscribes, and a *passive* participant in all other layers.

- Each layer $i$ transmits at a rate within a designated interval and the rate floats within that interval according to a single rate congestion control regulated by active participants in that layer.

- The *lead receiver* (LR) for each layer is defined as the receiver with lowest throughput on that layer. The LR for each layer is selected from among the active participants of that layer to adjust the sending rate.

- Each receiver joins the next layer to increase the reception rate when its target rate is larger than the maximum rate available from its currently subscribed layers.

- Each receiver leaves its highest layer to decrease the reception rate when its target rate is lower than the minimum rate available from its currently subscribed layers.

Figure 4.1 briefly depicts a hypothetical configuration of layers in which layer rates follow a conventional doubling scheme. Time elapses on the x-axis, while the y-axis indi-
cates the cumulative sending rate from the subscribed layers. Here, receivers subscribing only to the base layer (layer zero) experience a rate which dynamically fluctuates between zero and 1Mbps; this rate is adjusted by the LR for Layer 0. Receivers subscribing to both Layer 0 and Layer 1 receive a cumulative rate which dynamically fluctuates between 1Mbps and 2Mbps. In addition, receivers may employ transitional layers to transition between cumulative subscription levels. With our methods, adding a layer is done conservatively using additive increase, thus the diagonal transitions to add an additional layer depicted in Figure 4.1. Decrease is achieved by dropping the topmost layer (not depicted) — a practice common to most multiple rate schemes. The specifics of all of these procedures depend on the underlying single rate congestion control, and we detail these in Sections 4.4 and 4.5.

Our approach is quite general, and is applicable to many of the single rate congestion control schemes that have been proposed. However, there are several requirements that a single rate scheme must satisfy in order for our generalized multiple rate
methods to leverage it. We enumerate these needed properties from a single rate congestion control on any given layer below:

- The sending rate is controlled by the receiver with lowest throughput. This rate control removes the possibility of improper aggregation of feedback which may cause the so-called *drop-to-zero* problem [51].

- Since our methods use one LR for each layer in the multicast session, it is imperative that the single rate methods for LR selection are as efficient as possible.

- Each receiver should be able to estimate its target rate to guide the decision of subscription level changes.

### 4.2.1 Setting up Layers and LRs

We now discuss the layer rate organization in more detail. Recall that we employ a cumulative layering scheme so that each receiver subscribes and unsubscribes to layers in sequential order. For simplicity, in the following discussion and in the remainder of the
dissertation, we will assume that the maximum cumulative sending rates through layer \( i \), which we denote by \( B_i \), follow the natural 1, 2, 4, 8, \ldots progression. Our approach is amenable to other multiplicative layer rate increases, as advocated in [10], or to finer-grained rates of increase. Table 4.1 describes the terms we use in the following sections to describe our scheme.

We define the maximum cumulative sending rates of the layers formally as follows: We let \( B_0 \) be the maximum sending rate of the base layer, and we set \( B_i = 2^i \times B_0 \) for \( i \geq 1 \). From this setting of the rates, we can associate each desired reception rate with a set of subscription layers: a receiver \( j \) desiring rate \( r_j \) should subscribe to all layers \( i \) such that \( B_i \leq 2r_j \). In addition, a receiver which has a computed throughput in the range \([0,B_0]\) always subscribes to the base layer \( L_0 \). In this sense, we can map each receiver to the layer on which they are active. We say that layer \( L_i \) is responsible for receivers with rates in the range \([B_{i-1}, B_i]\). Equivalently, we define the subscription level \( S_j \) of receiver \( j \) to be the layer responsible for that receiver. Therefore, the subscription level of receiver with expected throughput \( x \) is: \( S_j = \lceil \log_2 \frac{x}{B_0} \rceil \). For example, a receiver with expected throughput 6 Mbps where \( B_0 = 1 \) Mbps has a subscription level of 3 (i.e. it subscribes to \( L_0, L_1, L_2, \) and \( L_3 \)) and \( L_3 \) is responsible for this receiver. At any instant in time, we let \( LR_i \) denote the slowest receiver of a given layer \( i \), i.e. the active receiver \( j \) that has the lowest expected throughput in the range \([B_{i-1}, B_i]\).
4.2.2 Adaptively Adjusting Layer Rates

Next we consider dynamic, adaptive rate adjustment at the sender. It is first important to draw a distinction between rate-based control and window-based control, both of which are potentially applicable to our methods, provided they meet the additional assumptions stated above. In a rate-based scheme, the sender regulates the traffic by adjusting the transmission rate according to some network feedback mechanism. In a window-based scheme, a congestion window size is computed either at the sender or at the receiver(s). The sender can then send as many packets as the congestion window size allows, while the size of the congestion window changes dynamically in the presence of congestion.

Our methods are simplest when the underlying single rate control is rate-based, and our following discussion assumes this. However, with somewhat more careful consideration, it also applies to window-based schemes, and we will discuss this case in Section 4.5 in the context of leveraging pgmcc.

Consider the setting of the layer rates, starting with the base layer, \( L_0 \). The sender adjusts the sending rate of the base layer based on the feedback sent by \( LR_0 \), the LR for the base layer, and we denote the actual sending rate on the base layer that results from this process by \( R_0 \). Receivers with expected throughput in the range of \([B_0, 2B_0]\) subscribe to \( L_1 \) as well as \( L_0 \). Let \( T_1 \) denote the total aggregate rate requested by the lead receiver subscribing to \( L_1 \). Then, the actual sending rate \( R_1 \) on layer 1 is set to the difference
between $T_i$ and $R_0$. In general, the same principle is used to set the rate $R_i$ on layer $i$:

$$R_i = T_i - \sum_{j=0}^{i-1} R_j,$$  \hspace{1cm} (4.1)

where $T_i$ is the aggregate target rate requested by $LR_i$, the lead receiver on $L_i$. From this setting, it is easy to show the following bounds on $R_i$:

$$\forall i : 0 \leq R_i \leq B_i - B_{i-2}.$$ \hspace{1cm} (4.2)

Equation 4.1 also shows that the actual sending rate on Layer $i$ depends on the actual sending rate on lower layers to make the cumulative sending rate equal to $T_i$. Figure 4.2 depicts an example of the actual sending rate on Layer 0 and Layer 1 to see the effect of rate changes on lower layers. At a given time, the sending rate for $L_0$ is controlled by
the feedback sent by $LR_0$. Also the cumulative sending rate for $L_0$ and $L_1$ is adjusted by $LR_1$. Hence, the actual sending rate on $L_1$ is the difference between $T_1$ and $R_0$ (Eqn. 4.1).

In Figure 4.2, during the time period from 80 seconds to 90 seconds the cumulative rate requested by $LR_1$ decreases while the requested rate from $L_0$ increases. Recall that $T_0$ and $T_1$ are independent. The increase of $T_0$ and decrease of $T_1$ necessitates a decrease of the actual sending rate on $L_1$. The actual sending rate on $L_i$ is bounded in Eqn. 4.2 and $R_1$ could be less than $R_0$ as depicted in Fig. 4.2.

### 4.2.3 TCP-Friendliness

In TFMCC [52] and pgmcc [43], the sending rate is adjusted by a slowest receiver and this rate adaptation also ensures TCP-friendliness. Our scheme employs a single rate congestion control as the primary underlying control mechanism for each layer. In our scheme, all active receivers on $L_i$ will get the same reception rate. This rate is $\sum_{j=0}^{i} R_j$, which is the same as $T_i$ (Eqn.4.1). The aggregate sending rate for $L_i$ is controlled by $LR_i$ using TCP-friendly single rate congestion control. Hence, as long as the underlying provides TCP-friendliness our multiple rate scheme also provides TCP-friendliness for receivers on all layers.

At this point, we have provided a high-level sketch of a general method to leverage a single-rate control in the design of a multiple rate multicast congestion control. But before describing the details of our protocols, we first give some additional description of the two main single rate congestion control protocols that we consider.
4.3 Single Rate Congestion Control

In the related work chapter (Chapter 2), we described TFMCC and pgmcc briefly. Now we go into more detail on these two protocols that we leverage.

4.3.1 TFMCC Overview

TFMCC [52] is a single rate multicast congestion control protocol designed to provide smooth rate change over time. TFMCC extends the basic equation-based control mechanisms of TFRC [20] into the multicast domain. The fundamental idea is to have each receiver evaluate a control equation (Eqn. 4.3) derived from the model of TCP’s long-term throughput [39], repeated from Section 2.1, then use this to directly control the sender’s transmission rate.

\[
T_{TCP} = \frac{s}{RTT \left( \sqrt{\frac{2p}{3}} + (12 \sqrt{\frac{3p}{8}}) p (1 + 32p^2) \right)},
\]

(4.3)

where \(T_{TCP}\) is a function of the steady-state loss event rate \(p\), the TCP round-trip time \(RTT\), and the packet size \(s\).

An overview of TFMCC functionality is as follows:

- Each receiver measures the packet loss rate.

- The receiver measures or estimates the round-trip time to the sender.

- The receiver uses the TCP control equation (Eqn. 4.3) to derive an acceptable trans-
mission rate from the measured loss rate and round-trip time.

- The receiver sends the calculated transmission rate to the sender.

- A feedback suppression scheme (additional details below) is used to prevent feedback implosion while ensuring that feedback from the slowest receiver always goes to the sender.

- The sender adjusts the sending rate from the feedback information.

In TFMCC, the receiver that the sender believes currently has the lowest expected throughput of the group is selected as the current limiting receiver (CLR). The CLR sends continuous, immediate feedback to the sender without any suppression, so the sender can use the CLR’s feedback to adjust the transmission rate. In addition, any receiver whose expected throughput is lower than the sender’s current rate sends a feedback message, and to avoid feedback implosion, biased feedback timers in favor of receivers with lower rates are used.

### 4.3.1.1 Measuring the Loss Event Rate

One crucial detail of TFMCC is the method it uses to measure packet loss. In TFMCC, a receiver aggregates the packet losses into loss events, defined as one or more packets lost during a round-trip time. The number of packets between consecutive loss event is called a loss interval. The average loss interval size can be computed as the weighted average of
the $m$ most recent loss intervals $l_k, ..., l_{k-m+1}$:

$$l_{\text{avg}}(k) = \frac{\sum_{i=0}^{m-1} w_i l_{k-i}}{\sum_{i=0}^{m-1} w_i}.$$ 

The weights $w_i$ are chosen so that very recent loss intervals receive the same high weights, while the weights gradually decrease to 0 for older loss intervals. The loss event rate $p$ used as an input for the TCP model is then taken to be the inverse of $l_{\text{avg}}$. The interval since the most recent loss event is incomplete, since it does not end with a loss event, but it is conservatively included in the calculation of the loss event rate if doing so reduces $p$:

$$p = \frac{1}{\max(l_{\text{avg}}(k), l_{\text{avg}}(k-1))}.$$ 

### 4.3.1.2 Round-trip Time Measurements

Each receiver starts with an initial RTT estimate that is used until a real measurement is made. A receiver measures the RTT by sending timestamped feedback to the sender, which then echoes the timestamp and receiver ID in the header of a data packet. An exponentially weighted moving average (EWMA) is used to prevent a single large RTT measurement from greatly impacting the sending rate:

$$t_{RTT} = \beta \cdot t_{\text{init}}^{RTT} + (1 - \beta) \cdot t_{RTT}.$$
The recommended value of $\beta$ for the CLR is 0.05 while all other receivers are recommended to use $\beta = 0.5$ due to their less infrequent RTT measurements. For further details of TFRC and TFMCC, we refer the reader to [20] and [52].

### 4.3.2 pgmcc Overview

pgmcc [43] is a single rate congestion control using a window-based controller that closely resembles the control used by TCP based on positive ACKs sent by a multicast group’s representative. The high-level features of pgmcc are as follows:

- Each receiver measures its own loss rate.

- Loss rate information is periodically delivered to the sender inside negative acknowledgments (NACKs).

- The sender selects a group’s representative, acker, as the receiver with the worst throughput according to the RTT and the reported loss rate.

- The acker sends an ACK for each received packet to the sender. The sender runs the window-based control scheme to mimic TCP congestion control.

- NACK suppression (optional) can be performed with randomization and routers which support the PGM multicast control traffic protocol [43].

In pgmcc, each receiver measures the loss rate by interpreting the packet arrival pattern as a discrete signal and passes it through a discrete-time linear filter. This loss rate
(p) is used in the acker election procedure by employing a simplified TCP equilibrium equation: \( T = \frac{1}{RTT\sqrt{p}}. \)

In \texttt{pgmcc}, a receiver may not be able to obtain an accurate round-trip time estimate, and thus a different RTT estimate is employed. Since the RTT is used only for the purpose of acker selection, the number of packet in flights is used instead of the real RTT. Each receiver sends the highest known sequence number in each NACK and this information can be used to compute the number of packets in flight. Even though this number of packets will vary depending on the actual sending rate, ordering receivers by packets in flight is equivalent to ordering them by RTT. Thus, acker selection can safely proceed by identifying the receiver with the lowest TCP equilibrium throughput using the method described.

### 4.4 Multicast Congestion Control from TFMCC

The primary protocol we develop is SMCC (Smooth Multiple rate Multicast Congestion Control) employing TFMCC as the underlying protocol. In Section 4.5, we compare and contrast the SMCC design with a design using \texttt{pgmcc} as the underlying control.

The high level features of SMCC follow the general features of deriving a multiple rate scheme from a single rate scheme described in Section 4.2. The additional specific features for SMCC are as follows:

- Each receiver calculates its expected throughput as in TFMCC.
• If the expected throughput calculated from the equation is above the maximum sending rate of its current subscription level, the receiver performs a join attempt using additive increase methods.

• If a receiver’s computed throughput is below the minimum receiving rate of the layer \(i\), it drops its highest layer \(i\). (Note that this bounds the extent to which an LR can drag down the rate of a single TFMCC session).

### 4.4.1 Lead Receiver Change

As in the TFMCC approach, the active participants in \(L_i\) do not send feedback unless their calculated rate is less than \(T_i\), thus avoiding feedback implosion. The LRs are permitted to send immediate feedback without any form of suppression, so the sender can use the LRs’ feedback to adjust the transmission rate (upward or downward) for each layer.

The LR for a layer can change in one of two ways: either a new receiver becomes the LR or the existing LR leaves the group. Each of these cases is relatively easy to handle. If a receiver whose subscription level is \(i\) sends feedback that indicates a rate that is lower than the current rate of \(LR_i\), but still larger than \(B_{i-1}\), the sender will set \(LR_i\) to that receiver and immediately reduces its rate for \(L_i\) to the requested rate in the feedback message according to Equation (2). If a receiver on \(L_i\) has a calculated rate which is less than \(B_{i-1}\), it unsubscribes from layer \(L_i\). The receiver needs to issue one IGMP leave message to drop the layer. While dropping the highest layer does not guarantee a particular amount of multiplicative decrease, on average, the reception rate is decreased by half.
If the departing receiver is the LR on $L_i$, a new LR for layer $i$ must be elected. To accomplish this, a departing LR first sends a control message to the sender notifying it of the departure. Upon receipt of this signal, the sender multicasts a control message to the group asking active participants to select a new LR. As in TFMCC, each receiver which is an active participant on layer $L_i$ will set a random timer before sending feedback to the sender. To avoid feedback implosion, biased feedback timers in favor of receivers with lower rates are used.

If there are no active participants on layer $i$ (which can happen when other participants are active on other layers $j$ such that $j > i$), no LR is assigned to layer $i$. The actual sending rate of layer $i$ is then set to $(B_i - \sum^{i-1}_{j=0} R_j)$ and the rates on higher layers are adjusted according to Equation (4.1). If any receiver which is active in layer $j > i$ subsequently drops layers $i+1$ through $j$ and becomes active in layer $i$, this receiver will become the LR in layer $i$, as will a receiver who joins layer $i$ from below. The sending rate of layer $i$ is then adjusted by this active receiver’s feedback rate.

### 4.4.2 Subscribing to an Additional Layer

Even though the receivers in the same group have similar calculated throughput, they may not share the same congested links. So, measured packet loss events across active receivers in a layer will vary. Often, some receivers may compute a calculated throughput value which is in the range of the next layer, and those receivers will attempt to join the next layer. As motivated in the related work chapter (Chapter 2), naive join attempts using
a single IGMP join request are problematic, as they introduce a sudden rate increase along a network path. Such a spurious join attempt may cause significant packet loss prior to the time at which the attempt is rescinded [10]. In severe cases, this substantial increase on the bottleneck link may drive many competing TCP flows into timeout. For this reason, join attempts which mimic fine-grained additive increase are preferable [13, 11]. Here, instead of joining the next layer, the receiver increases the receiving rate slowly and additively, i.e. by one more packet per RTT, during the join attempt.

Another compelling reason for proceeding to the next layer slowly is due to inaccuracies in estimating the target throughput when it differs substantially from the current reception rate using TFMCC methods. As described earlier in Section 4.3.1.1, the loss rate is computed from the loss interval, which is defined as the number of received packets since the last loss event. Hence, the loss interval clearly depends on the sending rate. But since the sending rate is controlled by the LR’s feedback, the loss rate currently measured by a non-LR is not the same as if the sending rate adjusted to its feedback. In Section 4.7, we show simulation results demonstrating that the loss rate measured by non-LR is not a sufficiently accurate estimate to conclusively determine whether or not to join the next layer. In practice, depending on the specific scenario considered, the calculated throughput can either be an overestimate or an underestimate. Therefore, we recommend conservative additive increase scheme as we describe in Section 4.6 to conduct join attempts before subscribing to an additional layer.

Our methods for performing additive increase joins are the glue that holds an ensemble
of TFMCC sessions together, and constitute the key additional feature needed to provide a sound multiple rate congestion control scheme. As such, we describe them fully in Section 4.6.

4.5 Multicast Congestion Control from pgmcc

We next describe a multiple rate scheme employing pgmcc as the underlying control protocol to offer a contrasting perspective to the use of TFMCC. Two main differences impacting the designs are the use of window-based vs. rate-based control, and the differing responses to packet loss. The multiple rate scheme using pgmcc follows the property of AIMD (Additive Increase Multiplicative Decrease) in pgmcc, so a receiver with even one packet loss will drop its highest layer to follow multiplicative decrease. This is guaranteed to cause a subscription level change; whereas with SMCC, occasional packet loss can often be absorbed without rate reduction.

All receivers start by subscribing to the base layer, and as in pgmcc, the lowest throughput receiver will be selected as LR and the congestion window will be controlled by the ACKs and NACKs sent by this LR on the base layer. The sender maintains one congestion window per layer and this window is increased by the ACKs sent by the LR for that layer. The non-LRs on the base layer need to decide whether they should join the next layer or not since they may have the higher expected target rate than the maximum sending rate of base layer. Since in pgmcc, the loss rate and RTT measurement are used
only foracker selection, we employ the loss rate and RTT measurement mechanisms used in TFRC and TFMCC to compute the target rate. Using these methods, any non-LR will join the next layer if the target rate is larger than the maximum sending rate on the base layer. If there is no LR on the subscribed layer \(i\), this receiver will be selected as \(LR_i\) and its initial window is set to \(B_{i-1} \times RTT / S\) where \(RTT\) is \(LR_i\)'s RTT and \(S\) is the packet size.

Since the cumulative window-based scheme does not provide the expected additive increase like TCP when the receiver subscribes to multiple layers, we employ the following mechanism to provide the proper increase across multiple layers. The sender computes the average sending rate from the window size and the round trip time. This sending rate on layer \(i\) is set to \(T_i = W_i \times S / RTT\), where \(W_i\) is the congestion window size on layer \(i\), and \(S\) is the packet size. \(T_i\) is the aggregate target rate for layer \(i\) and it is used to set the actual sending rate \((R_i)\) described in Section 4.2.2.

The LR for a layer \(L_i\) can change in one of three ways: 1) a receiver who joins layer \(L_i\) from below becomes the LR, 2) a receiver who leaves layer \(L_{i+1}\) becomes the LR, or 3) the existing LR leaves the layer \(L_i\). Whenever there is a LR change on layer \(i\), the window size for layer \(i\) will be readjusted by the target rate reported by \(LR_i\). If the window size of layer \(i\) increases up to the maximum sending rate of layer \(i\) and there is a subscribed receiver on the next layer \(i + 1\), all active participants on layer \(i\) will join the next layer \(i + 1\) through the additive join attempt. If there is no receiver on layer \(i + 1\), \(LR_i\) will become \(LR_{i+1}\) and the actual sending rate of layer \(i\) is then set to \((B_i - \sum_{j=0}^{i-1} R_j)\). All
receivers on layer $i$ for $i > 0$ will leave their highest layer at the detection of packet loss and this leave will reduce the reception rate by approximately half.

### 4.6 Additive Increase Join Attempts

We now describe the final technical component of our methods: a new additive increase scheme to conduct join attempts between successive layers in our multicast session. Although other work has proposed the use of additive increase in multiple rate multicast congestion control, such as FGLM [13] and STAIR in Chapter 3, those methods are designed as an integral part of complex, non-cumulative multicast layering schemes, and have technical limitations which make them unsuitable for this application. In contrast, the layers we propose for additive increase are *only* used when a receiver wishes to attempt to join the next successive layer. Our scheme has the following properties.

- True additive increase with respect to end-to-end bandwidth consumption.
- Employs no IGMP leave messages (which can be slow to take effect).
- Uses only a small number of additional IGMP join messages.

Once a receiver performing a join attempt from layer $L_i$ attains a total reception rate equal to $T_{i+1}$, the target rate sent by $LR_{i+1}$, it joins layer $L_{i+1}$ and drops the special additive increase layers. If, however, there is a packet loss during the join attempt, the receiver terminates the join attempt. We incorporate the information gained from both
successful or failed join attempts into loss interval and loss rate calculations. The sender sends the next layer rate information in the packet header.

### 4.6.1 Introducing Binary Counting Layers

The key to our additive increase methods are binary counting layers, so named because the rates on the layers mimic aspects of counting in binary.

- *Binary Counting Layers (BCL):* The rate transmitted on $BCL_i(x)$ is an ON/OFF function with a sending rate of $2^i$ packets during each ON time, and where the duration of each ON and OFF time is $x \cdot 2^i$.

In TCP, the rate of additive increase is a function of the round-trip time: the window opens by one additional packet per RTT. The set of $BCL(x)$ layers provide the same functionality as TCP’s additive increase with a measured RTT of $x$ seconds. BCLs accommodate asynchronous join attempts for different receivers in the multicast session, and accommodate receivers with different target rates for the join attempt.
All layers are initially synchronized at time zero, which corresponds the beginning of an off time for all layers. Figure 4.3 shows how each Binary Counting Layer is organized, assuming a 1 second RTT, which we use throughout this discussion for simplicity.

To achieve additive increase starting at time zero, the receiver simply subscribes to BCL$_i$ at $2^i \times$ RTT seconds. In Figure 4.3, where the RTT is 1 second, the receiver subscribes to BCL$_0$, BCL$_1$, BCL$_2$, and BCL$_3$ at $1s$, $2s$, $4s$, and $8s$ respectively. Once the receiver subscribes up through BCL$_i$, the number of receiving packets per RTT has increased by $2^{i+1} - 1$ with only $i$ IGMP joins and no additional IGMP leaves. Avoidance of IGMP leaves is crucial, since in current versions of IGMP, it often takes a number of seconds before the leaves actually take effect.

In Section 3.2, we recapitulated the definition of join and leave complexity defined in [13], i.e. the number of IGMP joins and leaves per operation. For SMCC, the notion of operation does not map cleanly onto the additive increase process, so we will consider the complexity of $N$ successive additive increases. From the description above, it is clear that this process requires $\log N$ joins (and no leaves). In other approaches to additive increase, such as [11], the receiver periodically increases its rate by a constant amount $c$ using a constant number of operations (typically 1 join and 2 leaves). Thus the complexity of $N$ successive additive increases in these schemes is $N/c$, i.e. linear in $N$. 
4.6.2 Extended Binary Counting Layers

One limitation of the basic binary counting layer scheme is that the receiver has to wait until certain specific times to join the BCLs. Suppose the receiver wants to increase its rate from 1 to 14 packets per RTT in Figure 4.3. If the receiver wants to join BCLs at 5 seconds, it has to wait until the next cycle (time 17) to initiate additive increase. One solution is to allow receivers to jump-start their additive increase with an initial set of joins (i.e., an immediate increase of 5 packets per RTT in the example above). However, this can induce sudden rate increase, and in the worst case, reduces to a naive join attempt. An alternative is the following improvement.

- **Extended Binary Counting Layers:** The rate transmitted on $\text{EBCL}_d(x)$ is a cyclic two step function. The number of sending packets during RTT $x$ seconds is $2^i$ and $2^{i+1}$ in the low step and in the high step respectively.

Figure 4.4 shows the transmission rate of each binary counting layer targeted for an RTT of 1 second. The receiver subscribes to $\text{EBCL}_d$ at $(2^{i+1} - 1)\cdot\text{RTT}$ seconds to perform
the additive increase. In Figure 4.4, the receiver subscribes to EBCL$_0$, EBCL$_1$, EBCL$_2$, and EBCL$_3$ at $1s$, $3s$, $7s$, and $15s$ respectively to get the additive increase up to 30 packets per RTT. Now consider the waiting time if the receiver misses the join time. If the receiver has to start the increase from 1 packet to $2^i - 1$ packets, a new cycle for that increase starts at $2^i \cdot \text{RTT}$ seconds after the previous cycle starts in the basic BCL. However, in the extended BCL the new cycle for that increase starts at $(2^i - 1) \cdot \text{RTT}$ seconds after the previous cycle starts. For example, for the increase from 1 to 30 packets, the new cycle starts at 32 seconds and 17 seconds in the basic BCL and in the EBCL respectively.

So far we have accommodated receivers with a specific RTT (1 sec. in our example). In practice, receivers may have widely varying RTTs, and it is desirable to simulate TCP behavior of one packet per RTT additive increase for each receiver. Extended BCLs can achieve this. In Figure 4.5, we show how BCLs can be organized to simultaneously accommodate receivers with RTTs of 1 second and 2 second. The scheme easily generalizes to support various RTTs which are powers of two. Each $BCL_i$ can be represented as two multicast sessions: $BCL_i(a)$ for 1 second RTT and $BCL_i(b)$ for 2 second RTT. The pack-
ets represented as white boxes are delivered in $BCL_i(a)$ while the packets represented as black boxes are delivered in $BCL_i(b)$. The receiver with a 2 second RTT subscribes only to $BCL_i(b)$ layers, while the receiver with a 1 second RTT subscribes to both $BCL_i(a)$ and $BCL_i(b)$, using a cumulative approach to sublayer subscription. The effect of subscribing to both $BCL_i(a)$ and $BCL_i(b)$ is the same as subscribing to $BCL_i$ in Figure 4.4.

### 4.6.3 Cost Of Additional BCLs For Join Attempt

One cost of additional layers to facilitate additive increase is that they consume additional bandwidth beyond what is used by the normal cumulative layers. To measure this cost, we use the measure of dilation, defined in Section 3.2 as definition 3.2.2.

**Lemma 4.6.1.** The worst case dilation of SMCC with a single set of BCLs is 1.75.

**Proof.** Let us suppose the highest layer subscribed to by any downstream receiver is the $j$th layer. The maximum rate induced by the join attempt of a receiver $k$ is $B_j - B_{j-2}$ when the following case holds: 1) the cumulative sending rate up through $L_j$ is the maximum rate $B_i$, and 2) the cumulative sending rate up through $L_{j-1}$ is slightly higher than the minimum rate $B_{j-2}$.

When an active receiver $k$ in $L_{j-1}$ has a calculated rate that is in the range of $L_j$ it performs a join attempt, which lasts until the total reception rate is equal to the next layer’s cumulative sending rate $B_j$. Therefore, the maximum rate induced by the join attempt is
$B_j - B_{j-2}$. The maximum reception rate of the set of receivers is $B_j$ and the bandwidth demanded in aggregate by receivers is $B_j + B_j - B_{j-2}$. Therefore,

$$\text{dilation} = \frac{B_j + B_j - B_{j-2}}{B_j} = \frac{2B_j - \frac{B_j}{4}}{B_j} = 1.75$$

Even though this worst-case dilation is not negligible, in practice it occurs only rarely (when a join attempt occurs across a bottleneck link); moreover, the average dilation during a join attempt is much smaller than this worst-case.

### 4.7 Experiments

We have tested the behavior of SMCC using the ns simulator [38]. In most of the experiments we describe here, we use RED (Random Early Detection) gateways, primarily as a source of randomness to remove simulation artifacts such as phase effects that may not be present in the real world. Use of RED vs. drop-tail gateways does not appear to materially affect the performance of our protocol. The RED gateways are set up in the following way: we set the queue size to twice the bandwidth-delay product of the link, set minthresh to 5% of the queue size and maxthresh to 50% of the queue size with the gentle setting turned on. Our TCP connections use the standard TCP Reno implementation provided with ns.
4.7.1 Preliminary Fairness Tests

Since the single rate TFMCC was well tested on the “dumbbell” topology [52], we set our initial topology to have multiple bottlenecks so that various SMCC receivers experience different network conditions. This initial topology is depicted in Figure 4.6. We set the propagation delay on each link is set to 8ms; each receiver therefore has a 64ms RTT in our simulations. Varying the delay on the links did not materially impact the performance of our protocol in the simulations we conducted. A full set of all the experiments we conducted as well as the ns source code are available online at http://www.cs.bu.edu/SMCC.
We consider a single SMCC session with two SMCC receivers and two parallel TCP flows sharing the same bottleneck link for each SMCC receiver. SMCC receiver S1 competes with 2 TCP connections on a 6Mbps link, giving a fair rate of 2 Mbps. S2 competes with 2 TCP flows on a 21Mbps link, for a fair rate of 7Mbps. We set $B_0$ to 4Mbps so that the sender’s maximum transmission rate on the base layer $L_0$ is 4Mbps. The throughput of each of the flows is plotted in Figure 4.7. S2 joins the base layer $L_0$ at 30.0 seconds, and it performs a join attempt at 47.6 seconds. After S2 subscribes to $L_1$ at 48.2 seconds, it shares fairly with the parallel TCP flows on the 21Mbps bottleneck link, while low-rate SMCC 1 shares fairly with 2 TCP flows on the 6Mbps link.

### 4.7.2 Late Join by a Low-rate Receiver

In TFMCC, a late join by a low-rate receiver results in that low-rate receiver being selected as LR, causing the sending rate of the entire session to slow down. In SMCC, the late join
of a low-rate receiver does not affect other receivers’ throughput on higher layers. Figure 4.8 (a) shows the throughput of SMCC receivers when the low-rate receiver, S1, joins late.

At the time S1 joins the session (70 seconds), the transmitted rate on the base layer is the maximum 4Mbps, while the rate on $L_1$ has been smoothly adjusting between 1 and 4Mbps to accommodate S2. The fair share for S2 behind the 6Mbps bottleneck link with two TCP competing flows is roughly 2Mbps, thus it immediately starts to experience a high loss rate. S1 is selected as LR$_0$ within a second, and its feedback subsequently controls the transmission rate of $L_0$. While the transmission rate of $L_0$ has changed from 4Mbps to S1’s feedback, the throughput of S2 is *not* adversely affected, since S2 is the LR for $L_1$, and the rate on $L_1$ instantaneously increases to compensate for the rate decrease on $L_0$. Figure 4.8 (b) demonstrates the discontinuities in the sending rates across $L_0$ and $L_1$ after time 70 seconds due to the late join of the low-rate receiver.

However, had there been other receivers subscribing only to the base layer, then the late join of a low-rate receiver clearly would affect other receivers at a same subscription level. The following rule is one of the keys to the scalability of our approach: degradation in the form of additional congestion along a path to a LR will only impose throughput degradation to *receivers at the same subscription level at that time*. Rates received at other subscription levels are generally not impacted substantially.
4.7.3 Inaccuracy of Non-LR Estimated Target Rate

Using TFMCC methods, a receiver which is not the LR may not have sufficient information to correctly estimate its targeted rate. In particular, the loss rate measured by non-LR receivers does not provide accurate information about the bottleneck bandwidth since the control equation was not modeled for this case, when the sender’s transmission rate is independent of the receiver’s packet loss events. The relevance of this point for SMCC is that a non-LR receiver may not always be able to accurately assess whether it can safely join the next layer.

Figures 4.9 and 4.10 and Table 4.2 depict this scenario. In Figure 4.9, TFMCC receivers TF1 and TF2 are competing with two TCP connections, T1 and T2, over a 2 Mbps bottleneck link and an 8 Mbps bottleneck link, respectively. TF1 and TF2 are not sharing the same bottleneck link, thus their losses are largely independent. Figure 4.10 shows each TCP flow’s throughput and each TFMCC receiver’s target rate calculated from the
Figure 4.10: Throughput of TF1 and TF2 over time in Fig. 4.9. Competing TCP connections plotted in the background.

<table>
<thead>
<tr>
<th>Receiver</th>
<th>Parameter</th>
<th>Time 50s</th>
<th>100s</th>
<th>150s</th>
<th>200s</th>
<th>250s</th>
</tr>
</thead>
<tbody>
<tr>
<td>TF1</td>
<td>RTT (second)</td>
<td>0.11</td>
<td>0.14</td>
<td>0.12</td>
<td>0.11</td>
<td>0.11</td>
</tr>
<tr>
<td></td>
<td>Loss rate (%)</td>
<td>1.10</td>
<td>0.36</td>
<td>0.81</td>
<td>0.81</td>
<td>0.80</td>
</tr>
<tr>
<td></td>
<td>Rate (Mbps)</td>
<td>0.76</td>
<td>1.18</td>
<td>0.88</td>
<td>0.92</td>
<td>0.90</td>
</tr>
<tr>
<td>TF2</td>
<td>RTT (second)</td>
<td>0.11</td>
<td>0.11</td>
<td>0.11</td>
<td>0.11</td>
<td>0.11</td>
</tr>
<tr>
<td></td>
<td>Loss rate (%)</td>
<td>0.08</td>
<td>0.03</td>
<td>0.01</td>
<td>0.01</td>
<td>0.02</td>
</tr>
<tr>
<td></td>
<td>Rate (Mbps)</td>
<td>3.22</td>
<td>5.58</td>
<td>7.67</td>
<td>9.37</td>
<td>7.44</td>
</tr>
</tbody>
</table>

Table 4.2: Calculated target rate of TFMCC receivers in Fig. 4.9

measured RTT and the loss rate. In the simulation, TF1 is quickly selected as $L_{R_0}$ and it fairly shares the 2 Mbps link with T1 throughout the simulation. Indeed, TF1’s target rate over time, as depicted in Table 4.2, is a reasonable approximation to its fair rate. In contrast, TF2’s target rate, also depicted in Table 4.2, is initially inaccurate (and badly underestimates the target rate) up through time 150 seconds. It then briefly overestimates its fair rate at time 200 seconds, and also overshoots its target subscription level, before converging around time 250 seconds. These estimation inaccuracies are another reason why we recommend and use conservative additive increase join attempts.
4.7.4 Responding to dynamics of competing traffic

We used a topology (Fig 4.11 (a)) to test the responsiveness to dynamic changes of local competing traffic, i.e. how increased traffic on local bottleneck links affects the receivers’ throughput on different bottleneck links. As the competing traffic increases across a bottleneck, proportional fairness ensures that an SMCC receiver sharing the same bottleneck will get less throughput. In the event such a receiver is selected as LR, the other receivers with the same subscription level also get less throughput even though they do not share the bottleneck with the LR. However, the extent of the degradation is bounded by a penalty of at most a factor of 2 on all layers except for the base layer. Moreover, we will show that in practice, the degradation is typically much smaller than this worst-case bound.

In Figure 4.11(a), receiver S1 is competing with two TCP flows for a 12Mbps bottleneck link, while both S2 and S3 are competing with two different TCP flows for a different 40Mbps bottleneck link. We now set $B_0 = 8$Mbps and all receivers have an RTT of 32ms. S2 and S3 do not share the same bottleneck link but their expected throughput is initially the same. Therefore, they have the same subscription level until new competing traffic starts.

Figure 4.11 (b) shows the throughput of each of the three SMCC receivers over time, as well as the LR (either S2 or S3) on $L_1$ over time. Initially, the simulation starts with the three SMCC receivers and TCP flows 1 through 6. At 70 seconds, 3 additional TCP flows (T7, T8, T9) sharing the 40Mbps bottleneck enter the system. Therefore, S3’s fair share drops from roughly 13Mbps to roughly 7Mbps. S3 is selected as $LR_1$ at 70.3 seconds and
the sending rate for \( L_1 \) steadily decreases, once \( L_1 \) is controlled by S3’s feedback. The receiver with the same subscription level, S2, suffers performance degradation as it gets the packets sent at the S3 feedback rate. But S2’s receiving rate is adversely affected by the increase of traffic on the path to S3 only so long as S3 is \( L R_1 \). At time 75.7 seconds, S3 drops its highest layer, \( L_1 \) when its calculated rate drops to 7.74Mbps. S2 is elected as new LR for \( L_1 \) at 76.2 seconds and its feedback controls the sending rate of \( L_1 \), which then quickly rebounds. Meanwhile, \( L_0 \) continues to be limited by S1, who continues to have a lower fair share than S3, so S3 receives at a rate of approximately 5Mbps during this time.
Although S3’s fair share is only 7Mbps, for reasons described in Section 4.4.2, it cannot make a highly accurate assessment of its expected throughput while receiving at only 5Mbps, and these inaccurate estimates induce it to make join attempts to $L_1$. S3 experiences two join attempts, both of which fail due to packet loss, between 70 seconds and 100 seconds. These two join attempts, marked by small spikes away from the S1 baseline, occur at 87.1 seconds and at 98.3 seconds. The little spikes around this time indicate these join attempt failures.

Finally, the three additional TCP flows leave at time 100 seconds. S3 performs a successful join attempt at 103.4 seconds and it reaches $L_1$ at 103.9 seconds, at which time it resumes sharing with S2.

Figure 4.11 (c) shows the identical simulation of each SMCC receiver but without the benefits of additive increase join attempts. Instead, in this simulation, the receiver naively joins an additional layer whenever the calculated rate is in the range of the sending rate of the higher layer. S3 joins the next layer at 86.8 seconds and it becomes LR for $L_1$ until 88.8 seconds. During this time, the sending rate of $L_1$ is dragged down to the rate of S3, impacting the reception rate of S2. After dropping back down, S3 joins $L_2$ at 96.1 seconds again and it is selected as LR$_2$ until 99.3 seconds. Spurious joins such as these can cause significant performance degradation; an effect which is that much more severe when multiple receivers perform spurious joins, thereby constantly dragging down the rates on higher layers.

In contrast, with additive increase joins, even when a receiver initiates joins which
are ultimately unsuccessful, it does not diminish the throughput received by other session participants during that time.

4.7.5 Fairness With Heterogeneous Receivers

We used a topology with multiple bottlenecks (Figure 4.12) to test the performance of SMCC with a set of heterogeneous receivers where the differences between the receivers’ target rates is relatively small. We consider a single SMCC session with six SMCC receivers and ten parallel TCP flows sharing the same bottleneck link for each SMCC receiver, but each SMCC receiver is not behind the same bottleneck link. S1 competes with 10 TCP connections on a 33Mbps link, giving a fair rate of 3 Mbps and the fair rates of the other SMCC receivers (S2 to S6) are 4Mbps, 5Mbps, 6Mbps, 7Mbps, and 10Mbps respectively.

We plot the throughput of each SMCC flow and the throughput of one of the competing
TCP flows in Figure 4.13. In each case, we chose the TCP connection whose mean rate was closest to the average of the ten competing flows as our representative. The throughput of each SMCC receiver fairly shares the bottleneck link with the parallel TCP flows even though lower-rate receivers are often present and drag down the rate on each level. In practice, non-LR receivers tend to periodically join the next higher layer as their estimated
throughput begins to deviate substantially from the LR’s target rate. The receiver S6 in panel (f) of Figure 4.13 is an example of relatively frequent subscription changes; note that its performance is still not as bursty as the competing TCP connection.

Next, consider Figure 4.14 (a) which plots the reception rate of S1 and S2. Like S6, S2 has relatively high subscription changes since its fair rate of 4Mbps is equal to $B_0$. S1, with a fair rate of 3Mbps, is typically selected as the LR on $L_0$. Whenever S2 joins $L_1$, it quickly becomes the LR and may impact the throughput of receivers on that layer. The plot depicted in Figure 4.14 (b) shows this impact. For example, at time 67.1 seconds, S2 becomes $LR_1$ and drags the cumulative rate $T_1$ down from 6.7Mbps to 5.0Mbps. At 99.6 seconds and 176.4 seconds, the sending rate is set from 7.1Mbps to 4.1Mbps and from 6.8Mbps to 4.6Mbps, respectively. There are other cases where the newly joined S2 becomes LR on $L_1$, but its degradation of rate from the previous sending rate of $L_1$ to the target rate of S2 is below 10%.

Next, we measured the dilation as a function of time of link (A-B) in Figure 4.12 and...
plot this dilation in Figure 4.15. Recall that the dilation of a link was defined and bounds on the worst case dilation of SMCC were proven in Section 4.6.3. Figure 4.15 shows that in this example, the measured dilation is much smaller than the worst case bound of 1.75.

### 4.7.6 Comparison of SMCC and STAIR with competing TCP flows

Finally, we use the topology in Figure 4.12 to compare the performance of SMCC and STAIR. We add six STAIR receivers in this topology so that each STAIR receiver is sharing the same bottleneck link with a SMCC receiver and ten parallel TCP flows. We increase the link capacity of each bottleneck link to maintain the same fair rates of all receivers as in the experiment of Figure 4.13. The measured throughput of each SMCC and STAIR flow with the throughput of one of the competing TCP flows are depicted in Figure 4.16. We only plot the throughput of SMCC and STAIR to avoid excess clutter. The average throughput of competing TCP flows is labeled in the plot. Both STAIR and SMCC are sharing fairly the same bottleneck link with competing TCP flows. In Figure 4.16, it
Figure 4.16: Throughput of STAIR and SMCC receivers, $B_0 = 4$ Mbps

It is easy to see the difference in the rate changes between two schemes. STAIR follows TCP’s AIMD rate change algorithm, but SMCC provides much smoother rate changes. In addition to this difference, SMCC uses small number of layers and provides less frequent subscription changes comparing to STAIR.
Chapter 5

Reliable Overlay Multicast with Loosely Coupled TCP Connections

5.1 Motivation

Although IP multicast delivery provides excellent scalability in terms of bandwidth consumption and server load, IP multicast has serious deployment limitations due to lack of consensus on protocols which include multiple rate congestion control and reliable content delivery mechanism. In addition to this reason, multicast across Autonomous System (AS) presents another set of issues such as inter-domain routing protocol, billing, and address allocation issues. A recent research trend has proposed to achieve multicast-based delivery using end-system, or application-level, multicast [4, 8, 17, 26, 21, 40] since an end-system approach avoids the considerable deployment hurdles associated with providing multicast
Figure 5.1: Buffering is inadequate for handling rate mismatches.

functionality at the network layer and additional transport-level functionality such as congestion control and reliability can be realized by employing standard unicast transport protocols. This methodology has been successfully applied to develop best-effort, UDP-based methods for streaming applications, augmented with congestion control. At first glance, it seems that a similar approach can be applied to high-bandwidth applications requiring reliable delivery, merely by employing separate TCP connections at each application-level hop. Use of TCP is clearly desirable, as it is universally implemented, provides built-in congestion control and reliability, and does not raise any questions of fairness. However, as we demonstrate next, naively architecting the overlay in this fashion leads to substantial performance degradation.

Consider a high-bandwidth upstream TCP flow relaying content through an end-system to a low-bandwidth downstream TCP flow (as depicted in Figure 5.1). As the transfer progresses, the intermediate end-system is forced to buffer a growing number of packets delivered by the upstream flow, but not yet sent to the downstream flow. This unwieldy set of in-flight packets will soon exceed the finite application level buffers available for relaying data at the intermediate end-system, and then there is a problem to solve. One solution, as
proposed in [48], is to use push-back flow control to rate-limit the TCP connection of the upstream sender. But it is easy to see that push-back flow control will recursively propagate all the way back to the source, and thus this devolves into a scenario in which all TCP connections in the delivery tree must slow to a rate comparable to that of the slowest connection in the tree. Using this method, even if there is no bottleneck on a given source-to-receiver path, that receiver will nevertheless be forced to slow to the rate of the slowest receiver. In this sense, this method has performance which closely resembles TCP-friendly single-rate multicast congestion control [52, 43]. On the other hand, it is not clear how to devise a TCP-based solution which provides an effective, multiple-rate remedy.

Our main contribution is the design and evaluation of ROMA\(^1\) (Reliable Overlay Multicast Architecture), a TCP-based content delivery architecture. The primary set of target applications are applications requiring reliability and high bandwidth, such as delivery of large files. ROMA enables multiple-rate reception, with individual rates that match the end-to-end available bandwidth along the path, while using small buffers at application-level relays, and the standard TCP protocol. The key to our methods is to make a departure from the straightforward approach in which each intermediate host forwards all received packets to the downstream hosts to achieve reliability. Instead of using this store-and-forward approach, we apply a forward-when-feasible approach, whereby each intermediary forwards only those received packets to downstream hosts that can immediately be written into the downstream TCP socket. We then handle reliability at the application layer.

\(^1\)This work appeared in the proceedings of IEEE INFOCOM'04 [30].
using erasure resilient codes, also known as FEC codes, applying well-known techniques developed for reliable (IP) multicast. The central component that enables our methods is the use of the digital fountain approach [35], a paradigm which is ideally capable of encoding \( n \) packets of original content into an unbounded set of encoding packets; and where receiving \( \text{any} n \) distinct encoding packets allows the complete, efficient reconstruction of the source data. Using the best codes currently available [33], a very close approximation to an idealized digital fountain can now be realized. This method has been widely used to enable receivers to recover from packet losses in the network; we apply it here to enable us to drop packets at TCP socket buffers which are full.

Our second contribution is performance evaluation of the chains of TCP connections that arise using our approach. We refer to these chains of TCP connections from the sender to end-hosts on a ROMA overlay as \textit{loosely coupled}, since an upstream TCP connection may or may not affect the performance of downstream TCP connections, but a downstream connection \textit{never} affects the performance of upstream connections. Applying standard equation-based methods [39], we examine the expected throughput across a chain of TCPs given per-hop RTTs and per-hop loss rates, where hop refers to a hop in the overlay. Conventional wisdom indicates that overlay multicast typically incurs a performance penalty over IP multicast, due to factors such as link stress, suboptimal routes, increased latency, and end-host packet processing. However, TCP chains offer us an opportunity to \textit{increase} performance by finding an alternative overlay path whose narrowest hop in the chain gives better expected TCP throughput than the default IP path. This performance
improvement is much in the spirit of alternative detour routes described in [45, 1]; these papers observe that IP does not provide the “best” path, measured in terms of delay or loss rates. We find that the best ROMA path is often a multi-hop path in which the minimum expected TCP throughput along any overlay hop is maximized.

Our third contribution is extensive PlanetLab [41] experimentation and insights gained from preliminary deployment of our system. We use a prototype Internet implementation that we built to validate our analysis for chains of TCP connections and to deploy our reliable multiple rate content delivery scheme. One interesting finding is that for many pairs of PlanetLab end-hosts, we can often optimize the ROMA layout to provide considerably better end-to-end measured throughput using a chain of loosely coupled TCPs than we could using a single, direct TCP connection.

The remainder of this chapter is organized as follows. In Section 5.2, we discuss other overlay multicast protocols and related work on constructing alternatives to the end-to-end path that IP provides. In Section 5.3, we further motivate our work by describing some candidate architectures and the limitations of those proposed solutions. Then, in Section 5.4, we present the details of the ROMA architecture, followed by an analysis of chains of TCP connections in Section 5.5. Extensive experimental results conducted on PlanetLab validate our analytical findings in Section 5.6.
5.2 Related Work in Overlay Design

A large body of work has recently been proposed to support multicast functionality at the application layer, including [4, 5, 6, 15, 16, 21, 26, 40, 47]. The design of overlay network layout has also been impacted by work initiated in the measurement community. We review and critique work in these two areas that are relevant to our proposed methods.

PRM [6], ALMI [40], Overcast [26], and RMX [15] all address the issue of reliability in distributing content to end hosts. PRM was designed for applications which do not require perfect reliability and focuses on improving the rate of data delivery while maintaining low end-to-end latencies. ALMI and Overcast employ TCP to provide reliable file transfers between any set of hosts. However, like the methods of [48], ALMI uses a back-pressure mechanism to rate-limit the sender, resulting in a single rate control. Overcast was explicitly designed with the goal of building distribution trees that maximize each node’s throughput from the source. However, the technical focus of Overcast was exclusively on topology optimization, and they did not consider issues associated with the transport protocol. Other works have also focused on the problem of efficient tree construction and on the challenges of optimizing the tree layout so as to minimize network costs such as average latency; or to minimize overlay costs, such as link stress; or to perform load balancing, such as by bounding the maximum fanout [4, 5, 6, 16, 21, 47].

Results from the measurement community have also been used in designing and optimizing overlay layouts. Savage et al. [45] showed that the default IP path between two hosts often is quantitatively inferior to a “detour” route taken through an intermediate end-
system. Using a large set of Internet path measurements taken between geographically diverse hosts, they identified detour paths which have superior round-trip time, loss rate, or available bandwidth compared to the default path with a surprising degree of regularity (at least 30 percent of measured paths had a detour path with shorter round-trip time, and over 75 percent had a detour path with lower aggregate loss rate). These results enabled the authors to identify detour paths over which the expected TCP throughput was higher than the default path (validated with actual TCP transfers). The designers of RON [1] employed the idea of alternative paths in an overlay context, and used paths similar to detour paths both to improve performance and to route around faults in their overlay. In our work, we leverage a similar measurement-driven strategy to identify the best routes in our overlay so as to optimize the layout of the set of TCP connections in our delivery tree. Our analysis goes beyond the conservative model used to estimate TCP performance common to both [45] and [1] — we find that their methods underestimate the actual throughput that a chain of TCPS is likely to see in practice.

### 5.3 Candidate Architectures

We first develop a basic model for an overlay network and motivate our approach by describing the challenges that reliable content delivery imposes and the limitations of current TCP-based solutions. Figure 5.2 depicts two intermediate systems using a TCP-based overlay architecture. We refer to the node’s *incoming buffer* as its TCP receive buffer for
Incoming TCP Buffer

TCP Flow

Outgoing TCP Buffer

Overlay Node

Overlay Node

TCP Flow

Figure 5.2: Overview of TCP-based Content Delivery in an Overlay.

its upstream link. Similarly, the outgoing buffers of a node refer to its TCP send buffers for its downstream links. In Section 5.1, we mentioned a simple store-and-forward method which tightly couples the TCP connections in the delivery tree:

- **Store-and-Forward**: For every packet arriving on an incoming buffer, buffer the packet, then forward it to all outgoing buffers.

As we saw in Figure 5.1, when a downstream link is slower than an upstream link, as the transfer progresses, the intermediate host is forced to buffer a growing number of packets using the store-and-forward approach. Working within the store-and-forward paradigm, there are two solutions, but both lead to performance problems of their own. We describe these alternatives next, then move beyond the store-and-forward paradigm in the next section.
5.3.1 Limited Buffer Space Solution

If the host has finite buffer space in application layer, the push-back flow control or back-pressure mechanism [48, 40] can be used to avoid buffer overflow. The basic operation of this approach is to dequeue the packet from the incoming buffer only after it has been relayed in all of the outgoing buffers. In addition, coupling the flow control and congestion control avoids any buffer overflow in the face of different speed of downstream link. The intermediate host sends back an acknowledgment to its parent only if the arriving packet can be copied into all outgoing buffers. If there is insufficient space on any outgoing buffer, the host stalls. This results in queue buildup at the incoming buffer and subsequent decrease of the advertised window. The effects of this decreased advertised window will ultimately propagate all the way back to the source. This approach therefore results in performance which translates to single rate congestion control, where all nodes in the tree are sending packets to downstream links at approximately the speed of the slowest link.

5.3.2 Unlimited Buffer Space Solution

Another alternative is to generalize the notion of what constitutes an application layer buffer for each downstream node. Since each intermediate node is also participating in downloading the content, it must store all received packets for its own use. When the content is large, this storage will take place on disk, instead of in a system buffer. Therefore, the application can implement store-and-forward by dequeuing each reliably received packet from the incoming TCP buffer and storing that packet on disk. Concurrently and
Figure 5.3: Adaptive reconfiguration of the overlay.

independently, each outgoing buffer can be filled from disk, using appropriate prefetching methods to hide the substantial costs of I/O where possible. This approach enables multiple rate transmission, but with the following limitations:

- A separate application buffer for each downstream node is required.

- Complexity to support I/O accesses to fill each outgoing buffer is needed.

- The overlay cannot be adaptively reconfigured.

The first two limitations are clear, but the third (and arguably the most serious), requires more careful discussion.

A robust overlay network should have the ability to adaptively reconfigure itself when congestion or failures of intermediate nodes occur. Therefore, a host must be able to switch its parent to maximize its performance. But in many situations, this design does not facilitate such a transition. Consider the case of host D in the example in Figure 5.3, in which B, C and D are performing a reliable download from A. The average reception rate for host D is 10Mbps, that of host C is 5Mbps. Due to the different transfer rates, the data
received by D will be a prefix of the content that is twice the length of the prefix received by C at any point in time. Now suppose that an hour into the transfer, the B to D link becomes congested, degrading performance to 1Mbps. Host D would now prefer to use the route through C, but since C is thirty minutes behind (in terms of received data), this alternative route is useless to D. (Note that this problem is specific to multi-rate reliable transfers; it does not apply to the single rate back-pressure solution or to live streaming applications). A similar synchronization problem also arises in asynchronous transfers when hosts initiate the downloads at different times.

This significant limitation seems to be difficult to find a workaround for, but in fact, the use of erasure-resilient codes in the ROMA architecture that we describe next provides a very satisfactory solution that does not encounter any of the limitations presented in this section.

5.4 Reliable Overlay Multicast Architecture

We now describe ROMA, a simple reliable multi-rate overlay multicast architecture for reliable content delivery. The two central novelties leveraged in our design are the use of erasure resilient codes, as we describe in more detail in Section 5.4.1, together with the use of a forward-when-feasible paradigm, rather than the standard store-and-forward paradigm:
• Forward-when-feasible: For every packet arriving on an incoming socket, for each outgoing buffer, determine whether it can immediately accept the newly arrived packet. Copy the packet to those buffers which can accept it, then deliver the packet to the application. (Those outgoing buffers which are full will never receive or transmit a copy of this particular packet).

Together with the encoding methods we employ, an intermediate host using the forward-when-feasible approach does not have to store all received data in an application-level buffer and as a result, managing buffer overflow is not a problem. In practice, we use one additional level of indirection to implement the forward-when-feasible paradigm, a point we touch upon in the following more detailed overview of the ROMA architecture:

• Each node runs TCP between the upstream and downstream link nodes and itself.

• While there are interested participants, the sender transmits a continuous erasure-resilient encoding of the content of size $n$ along its downstream links.

• Each host dequeues the arrival packet from the incoming TCP buffer and copies the packet to a small application layer buffer managed as a circular queue. If the buffer is full, then the host overwrites the buffer in a circular fashion.

• Each intermediate host copies data to all outgoing buffers that have available space.

• Each host completes its reception after receiving a set of encoding packets of size approximately $1.03n$ (explanation of this small 3% overhead to follow).
Upon completing the reception of the original content, the node may leave the
ROMA group by closing its TCP connections. In the event it elects to continue
servicing downstream connections, it may do so either by continuing to relay en-
coded content generated by the source, or by generating encoding symbols of its
own from the full content, and closing its upstream connection.

In the next section, we provide more details of erasure-resilient codes, and the node
architecture in the ROMA system. We also describe how to transmit the encoded data on
a byte-stream transport protocol like TCP.

5.4.1 Erasure-resilient Codes

We now review the basics of erasure-resilient codes\(^2\), a close relative of error-correcting
codes: While error-correcting codes provide resilience to bit errors, erasure-resilient codes
provide resilience to packet-level losses. We use the following terminology. The content
being sent by the encoder is a sequence of symbols \(\{x_1, \ldots, x_i\}\), where each \(x_i\) is called
an \textit{input} symbol. An encoder produces a sequence of \textit{encoding symbols} \(y_1, y_2, \ldots\) from
the set of input symbols. For our application, we will set the input and encoding symbol
size both to be equal to a packet payload. For the family of erasure-resilient codes we use,
parity-check codes, each encoding symbol is simply the bitwise XOR of a specific subset
of the input symbols. A decoder attempts to recover the original content from the encoding
symbols. For a given symbol, we refer to the number of input symbols used to produce the

\(^2\)Often referred to as forward error-correcting (FEC) codes.
symbol as its degree, i.e. $y_3 = x_3 \oplus x_4$ has degree 2. Using the methods described in [35], the time to produce an encoding symbol from a set of input symbols is proportional to the degree of the encoding symbol, while decoding from a sequence of symbols takes time proportional to the total degree of the symbols in the sequence. Encoding and decoding times are a function of the average degree; when the average degree is constant, we say the code is sparse. Well-designed sparse parity check codes require recovery of a few percent (less than 5%) of symbols beyond $\ell$, the minimum needed for decoding. The decoding overhead of a code is defined to be $\epsilon_d$ if $(1 + \epsilon_d)\ell$ encoding symbols are needed on average to recover the original content. (There is also a small amount of overhead for the space needed in each packet to identify which input symbols were combined, which is typically represented by a 64-bit random seed.)

Provably good degree distributions for sparse parity check codes were first developed and analyzed in [35]. However, these codes are fixed-rate, meaning that only a pre-determined number of encoding symbols are generated, for example only $c\ell$, where $c$ is a small constant $> 1$. In our application, this can lead to inefficiencies as the origin server will eventually be forced to retransmit symbols. Newer codes, called rateless codes, avoid this limitation and allow unbounded numbers of encoding symbols to be generated on demand. Two examples of rateless codes, along with further discussion of the merits of ratelessness, may be found in [33, 36]. Both of these codes also have have strong probabilistic decoding guarantees, along with low decoding overheads and average degrees. In our experiments, we simulate use of LT codes [33], and assume a fixed decoding overhead
of 3%.

The main benefit of erasure codes in our architecture is that it makes it possible to design the control mechanisms independently of reliability. Intuitively, using an erasure-resilient encoding, packets can flow through ROMA intermediaries (all with small buffers) toward a set of destinations, and can be dropped whenever they reach a bottleneck (in the form of a full buffer). With this intuition, one can see that this provides for a multiple rate solution. The use of codes also enables a number of additional benefits, including the ability to tolerate asynchronous joins, the ability to adaptively reconfigure the topology, and the ability to speed up downloads with collaborative peer-to-peer transfers as described in [8].

In the next section, we provide the node architecture in the ROMA system. We also describe how to transmit the encoded data on a byte-stream transport protocol like TCP.

### 5.4.2 Transmitting Encoding Symbols with TCP

One nuance of using codes is that the encoding symbols must be treated atomically, i.e. receipt of a fraction of an encoding symbol is not useful. For this reason, some care must be taken to send encoded symbols as logical segments, or datagrams, across TCP. The main difficulty is to ensure that each encoding symbol is written in its entirety into the TCP socket. But, using only application-level, system-independent calls, it is not simple to determine whether a given packet will fit into the TCP send buffer without performing the write explicitly. Our solution (depicted in Figure 5.4) is to maintain a one-packet
overflow buffer per socket to store those bytes which could not be successfully written into the socket. Before performing a subsequent write to the socket, the contents of the overflow buffer are written first. Using this strategy, encoding symbols are always written contiguously and in their entirety to the buffer.

5.4.3 Intermediate and Sender node Architecture

In our overlay multicast architecture, we assume that each host is also participating in downloading the content and therefore must read data from the upstream socket into an application layer buffer before writing into disk. In our implementation, we use an application buffer of 1MB to overcome the limitation of small default socket buffer sizes on many systems. Most implementations have an upper limit for the sizes of the socket send buffer and the upper limit is only 256 KB in many systems. Use of the application buffer for additional buffering at intermediate hosts avoids known pitfalls associated with bursty
packet arrivals when high bandwidth connections with large window sizes use small socket buffers [46].

As described earlier, each intermediate host dequeues arriving packets and copies them to an application buffer. If the buffer is full, then the host writes the packets in the buffer into disk and overwrites the buffer in a circular fashion. The downstream socket buffer is filled from this application buffer, with each downstream socket making sure not to wrap around the tail end of the circular queue.

The sender architecture is virtually identical to that of the intermediate node except that the application buffer is filled with fresh encoding symbols (typically precomputed and stored on disk) at a speed that is sufficient to satisfy the fastest downstream connection. As with intermediate nodes, the sender also maintains a one-packet overflow buffer for each downstream node to avoid sending a fraction of an encoded packet. The functionality of the sender is as follows:

- Files are encoded into encoding symbols that are stored on disk prior to their delivery.

- A single, fixed-length memory buffer is used for all receivers.

- If the fastest receiver exhausts all data in the buffer, the buffer is filled with new data from the disk.

The sender’s functionality is similar to the Cyclone webserver architecture [44], which is optimized for delivery of content in situations in which a group of clients is concurrently
downloading a small set of large, popular files. In particular, the sender can be optimized to employ the sliding cache buffer mechanism in the Cyclone design to minimize the waiting time to fill the buffer from the disk.

5.5 Chains of TCP Connections

We now provide a simple analysis of the chains of TCP connections that arise in the design of our system.

5.5.1 Modeling Chains of TCP Connections

For simplicity, we begin with the simple case of an overlay host with just one upstream and one downstream TCP connection, depicted as host B in Figure 5.5. In this example, B is just relaying received packets to its one downstream host, i.e. it sends TCP ACKs for received packets back to host A and transmits data segments to host C. We assume that the overlay host has sufficient memory space in the application layer to store received packets in the case of a slower downstream link. (In the ROMA design, this assumption is realized provided the application layer buffer is sufficiently large that it never drains).
In this simple model, we assume that the intermediate host B dequeues the packet from its TCP receive buffer fast enough to prevent flow control algorithms from impacting its upstream transmission rate. Finally, we assume that the relevant network conditions (loss rate, RTT) along the chain of connections are stationary over time. These assumptions of unlimited buffer and fast dequeuing make this chain of TCP connections loosely coupled, which we define as follows:

**Definition 5.5.1.** A chain of TCP connections is loosely coupled if an upstream TCP connection may or may not affect the performance of a downstream TCP connection, but a downstream connection never affects the performance of an upstream connection.

If the downstream transfer rate is slower than the upstream transfer rate, then the application layer buffer will grow without bound. In this case, the downstream TCP will behave like a TCP driven by an application that always has data to send, and thus the performance of the downstream TCP is independent of the upstream TCP.

Alternatively, consider the case in which the downstream transfer rate is larger than the upstream transfer rate. In this case, host B will periodically drain the application level buffer filled by the upstream connection when sending packets to C, and thus the downstream TCP connection has to wait for incoming packets to send. Therefore, in this case, the downstream throughput to C is limited to that of the upstream rate into B.

To develop formulas for the expected TCP throughput as a function of the per-hop loss rates and RTTs, we employ the following equation derived in [39] and repeated from Section 2.1:
This provides an estimate of the expected throughput $T$ of a TCP connection as a function of the packet size $s$, the measured round-trip time $rtt$, and the steady state loss event rate $p$. For simplicity in the remainder of the exposition, we use the following simpler formula as a shorthand for the equation above:

$$T \approx \frac{\sqrt{1.5}}{rtt \sqrt{p}}.$$ (5.2)

To extend this result to a chain of loosely coupled TCP connections, our observations above demonstrate that a given hop in the chain either has local network conditions that limit its rate to a value below that of the upstream connections or is already limited by the rate of the upstream connections. Also recall that by the definition of loosely coupled connections, events downstream have no bearing on upstream throughput. Letting $rtt_i$ and $p_i$ respectively denote the round-trip time and loss rate experienced by a TCP connection traversing overlay hop $i$, the expected throughput to a ROMA host below hop $j$ is:

$$T \approx \min_{i<j} \left( \frac{\sqrt{1.5}}{rtt_i \sqrt{p_i}}, \frac{\sqrt{1.5}}{rtt_j \sqrt{p_j}} \right).$$ (5.2)

In an overlay setting, one factor which is not captured by this simple equation is the
impact of link stress, which occurs when distinct overlay hops $j$ and $k$ share underlying physical links. Link stress further implies that measured values of $p_j$ and $p_k$ are not independent. We show the effect of link stress in our experimental results, but do not incorporate this effect into our simple model.

5.5.2 Examples and Comparison with Other Models

To develop some intuition, consider the example in Figure 5.6, in which the propagation delay and the loss rate on each link are labeled. Using the direct route from A to C, and using the simple version of the throughput equation, the expected throughput of direct unicast from A is about 9.0Mbps. In contrast, the throughput from A to C via B using a chain of two TCP connections is about 22.2Mbps, which is also the expected throughput along the direct B to C connection. In other words, the loss and delay on the hop from A to B have no measurable impact on the performance along the detour path from A to B to C.

It is worth noting that in previous work, a different, and more conservative formula was used to estimate the throughput of a chain of TCP connections. Following the methodology
used in [1, 45], the aggregate RTT is defined as the sum of $\text{rtt}_i$ along the path and the aggregate loss rate is defined as $1 - \prod (1 - p_i)$ (assuming uncorrelated losses). Instantiating these values into the simple version of the throughput equation gives:

$$T \approx \frac{\sqrt{1.5}}{\sum \text{rtt}_i \sqrt{1 - \prod (1 - p_i)}}. \quad (5.3)$$

Plugging the values from the example in Figure 5.6 into this equation gives expected throughput across the detour route of 9.0 Mbps, or no different than the direct route. Indeed, it is easy to see that in general, this aggregation model treats a “split” TCP connection no differently than its aggregate. In practice, our experimental results demonstrate that this method of aggregation underestimates throughput, while the model embodied by Equation 5.2 provides a much more accurate estimate.

5.5.3 Discussion

Conventional wisdom indicates that overlay multicast incurs a performance penalty over IP multicast, due to factors such as link stress, stretch factor, and end host packet precessing. However as we have seen in the example in Figure 5.6, TCP chains also offer us an opportunity to increase performance compared to direct unicast. This performance improvement comes from finding an alternative overlay path whose narrowest hop in the chain (as perceived by TCP) is wider than the default path used by IP. In general, an improvement in throughput can be realized whenever one identifies a decomposition of a
long TCP control loop into several smaller loops in which each member of the chain has expected throughput greater than that of the original loop. As we have argued, this gain applies even when the aggregate loss rate and the aggregate RTT across this chain are larger than the values of the original long loop. Breaking long TCP control loops in the context of overlay networks has a similar effect as *split TCP* [3], which shortens the TCP feedback loop and separates lossy components. Ideas from split TCP are commonly used in satellite communication and in various terrestrial wireless contexts to improve TCP performance.

At this point, we feel compelled to note that TCP is not actually a mandatory component of the ROMA architecture. In principle, any TCP-friendly protocol with forward error correcting codes can be used to achieve the same performance benefits and our analysis still applies as long as it follows Equation 5.1. In practice, however, we prefer to use TCP because it is already ubiquitous and well understood, and because we feel that alternative protocols would merely be imitating the TCP behavior. (The potential benefits of UDP over TCP, e.g. for streaming or other real-time applications, do not seem to apply to reliable transfer applications.)

In the next section, we show that there exist ample opportunities to exploit this advantage in constructing overlay topologies so as to maximize the total throughput to participant hosts across the Internet.
Table 5.1: End-to-End Measured Throughput

<table>
<thead>
<tr>
<th>Receiver</th>
<th>BU</th>
<th>UCLA</th>
<th>UTK</th>
<th>Arizona</th>
<th>GT</th>
<th>Duke</th>
<th>Cornell</th>
<th>Berkeley</th>
<th>UW</th>
</tr>
</thead>
<tbody>
<tr>
<td>BU</td>
<td>64.1Mbps</td>
<td>12.6Mbps</td>
<td>19.0Mbps</td>
<td>16.0Mbps</td>
<td>27.0Mbps</td>
<td>39.0Mbps</td>
<td>21.3Mbps</td>
<td>6.9Mbps</td>
<td>8.7Mbps</td>
</tr>
<tr>
<td>UCLA</td>
<td>17.9Mbps</td>
<td>88.0Mbps</td>
<td>18.8Mbps</td>
<td>20.3Mbps</td>
<td>20.5Mbps</td>
<td>14.5Mbps</td>
<td>14.0Mbps</td>
<td>38.6Mbps</td>
<td>39Mbps</td>
</tr>
<tr>
<td>UTK</td>
<td>47.00Mbps</td>
<td>18.8Mbps</td>
<td>98.00Mbps</td>
<td>21.0Mbps</td>
<td>46.3Mbps</td>
<td>39Mbps</td>
<td>29.1Mbps</td>
<td>18.7Mbps</td>
<td>12.8Mbps</td>
</tr>
<tr>
<td>Arizona</td>
<td>21.3Mbps</td>
<td>21.7Mbps</td>
<td>21.0Mbps</td>
<td>92.0Mbps</td>
<td>22.2Mbps</td>
<td>15.4Mbps</td>
<td>19.0Mbps</td>
<td>21.7Mbps</td>
<td>19.3Mbps</td>
</tr>
<tr>
<td>GT</td>
<td>53.0Mbps</td>
<td>18.8Mbps</td>
<td>74.0Mbps</td>
<td>22.8Mbps</td>
<td>92.0Mbps</td>
<td>45.0Mbps</td>
<td>34.5Mbps</td>
<td>18.8Mbps</td>
<td>17.1Mbps</td>
</tr>
<tr>
<td>Duke</td>
<td>40.9Mbps</td>
<td>10.2Mbps</td>
<td>26.8Mbps</td>
<td>10.3Mbps</td>
<td>31.8Mbps</td>
<td>92.0Mbps</td>
<td>15.4Mbps</td>
<td>9.5Mbps</td>
<td>9.2Mbps</td>
</tr>
<tr>
<td>Cornell</td>
<td>33.1Mbps</td>
<td>14.5Mbps</td>
<td>30.7Mbps</td>
<td>19Mbps</td>
<td>28.7Mbps</td>
<td>15.4Mbps</td>
<td>98.0Mbps</td>
<td>16.3Mbps</td>
<td>16.5Mbps</td>
</tr>
<tr>
<td>Berkeley</td>
<td>10.1Mbps</td>
<td>30.3Mbps</td>
<td>12.6Mbps</td>
<td>17.1Mbps</td>
<td>13.4Mbps</td>
<td>9.4Mbps</td>
<td>11.4Mbps</td>
<td>98.1Mbps</td>
<td>38.4Mbps</td>
</tr>
</tbody>
</table>

5.6 Experiments

We have implemented ROMA and conducted experiments on the PlanetLab distributed testbed [41]. PlanetLab consisted of 160 machines hosted by 65 sites in June 2003, when we ran experiments on a subset of roughly 30 sites. All PlanetLab machines run a Linux-based operating system and they all meet certain hardware requirements (see details in [41]). Most of the hosts in PlanetLab in 2003 were university hosts and those hosts in the U.S. are connected through Abilene, which had high capacity and was highly available. Therefore, while the experiments we conducted on PlanetLab are not intended to be representative of typical Internet performance, they nevertheless enable us to validate our models and performance of our architecture across a substantial set of Internet paths.

For our experiments, we considered 1 GB transfers using a packet size of 1 KB. As a baseline, we conducted end-to-end transfers of this size between pairs of hosts using TCP. We report on a representative subset of these baseline measurements across Abilene in Table 5.1, where each entry represents the average measured throughput of ten independent
measurements from source nodes to destination nodes conducted in June 2003. In addition, entries on the diagonal report measurements between two PlanetLab nodes at the same university. We will use the name of university as the host name throughout this section for simplicity. One important observation is the substantial bandwidth asymmetry we see in our measurements. In some cases, there are significant constant factor differences: for example, between BU and UTK, the path asymmetry is 47 Mbps vs. 19 Mbps.

This table is intended primarily to give a flavor of the data rates we are working with, and does not capture the variability of throughput measurements over time (which we found to be relatively small on the lightly loaded Abilene backbone), nor does it provide a highly accurate measure of available bandwidth. One could certainly adapt our methods to address dynamic changes in available bandwidth, for example, by using non-intrusive methods to monitor available bandwidth such as pathload [25] or PTR [23]. In the following sections, we use values from the table as input to our algorithms to construct overlay multicast trees.

Next, we describe additional details involving experimentation with ROMA. First, in some cases, we identified PlanetLab hosts whose throughput appeared to be constrained by their local network configuration (perhaps due to router capability, link capacity, or rate limiting). For example, several hosts were unable to concurrently send and receive at sustained rates above 10Mbps. We did not include measurements from these rate-limited hosts in Table 5.1. Also, we disallowed these hosts from being intermediate hosts in our experiments, but did allow them to be leaf nodes.
At each of the hosts where we deployed ROMA, we established a 1 MB application buffer (as in Figure 5.4), primarily to facilitate copying between upstream and downstream sockets. Finally, as described in section 5.4.1, the erasure resilient codes we propose in ROMA induce decoding overhead; the codes we simulate have decoding overhead of approximately 3%. We include this decoding overhead into the ROMA throughput calculations whenever we compare to direct unicast throughput (which would not use codes).

### 5.6.1 Multiple Rate Reliable Multicast

Our first experiment uses the topology depicted in Figure 5.7 to validate that a single slow link does not impact the performance either at upstream nodes, or at nodes in other regions of the tree. Figure 5.8 compares the average throughput at each host when a 1GB transfer is performed using two different methods defined as follows:

- **ROMA**: An overlay multicast tree is established to all hosts, and the throughput is measured at each point in the multicast tree.
Figure 5.8: Host throughput with B.U. as the sender

- **Unicast To Parent**: For each host, we measure the throughput of a TCP transfer directly from its parent to the host itself. (This corresponds to a single entry in Table 5.1.)

  The values reported are the average measurements across ten trials.

  In this topology, every upstream link offers better unicast throughput than all of its downstream links, thus the throughput on any path from the sender to a receiver decreases monotonically. Here, the set of downstream links fanning out from every intermediate host also have different characteristics. Figure 5.8 shows that the ROMA throughputs measured by each host are diverse, but all are similar to the throughput of a single unicast connection to its parent node, as we desire. Clearly, slow links do not degrade the performance of unrelated peer hosts or ancestors in the tree.

  In Figure 5.8, note also that some hosts have slightly decreased throughput using ROMA as compared to a direct unicast connection to their parents (and beyond that of 3% decoding overhead). For example, consider the intermediaries at GT and Duke. The
unicast throughput of connections to GT and Duke from BU were 53.9Mbps and 40.9Mbps respectively while the multicast throughput of GT and Duke while running in the ROMA experiment were 47.8Mbps and 35.6Mbps respectively. This is primarily because the ROMA experiment is running under the disadvantage of delivering data across all tree edges simultaneously. The throughput degradation comes from the effect of link stress, which is defined as the number of identical copies of a packet carried by a physical link in an overlay [17]. In our example, downstream and upstream links from a single node often share some physical links. When these shared links are a bottleneck, the contention at these resources negatively impact the performance of those overlay connections crossing the link.

These measurement results also provide agreement with our analytical argument that the expected throughput is the minimum of the throughputs along the path (and no worse). Baccelli et al. provide a mathematical analysis for chains of TCP, called TCP connections in tandem, with an unlimited buffer space assumption [2]. Their work also shows that the throughput of each host in this topology is determined only by the network conditions along the overlay hop to its parent host.

In Figure 5.10, we report on a similar experiment, showing the throughput at each host in the topology depicted in Figure 5.9. As before, we see the effect of stress on links at Univ. of Arizona and GT. Another interesting case is the throughput at Duke. Although Duke’s upstream link from Georgia Tech has high throughput (31.8 Mbps), the measured throughput at Georgia Tech using ROMA is much lower (17.8 Mbps) and therefore the
measured ROMA throughput to Duke is limited to this lower value. This experiment (and many similar experiments not presented due to space limitations) provide confirmation that the TCP throughput of the overlay host is bounded above by the minimum throughput across the upstream links.

### 5.6.2 Throughput Improvement from Chains of TCP

In the following two sections, we report on experiments in which use of ROMA can actually improve the throughput as compared to direct unicast (and even when the aggregate RTT and loss rate increase). Consider the simple example in Figure 5.11 derived from our
Internet experiments. The throughput and RTT from the pairwise unicast measurements between the three hosts are as labeled. Since we were unable to directly derive the loss rate of TCP connections without root access on all hosts, we used equation (1) to compute the approximate loss rate based on the measured throughput and the measured RTT. To doublecheck our measurements, we also concurrently ran TFRC [20] on the same path and measured the average loss rate from TFRC.

Even though both the aggregate loss rate (0.0344%) and the aggregate round-trip time (42.2 ms) increase, the throughput to UIUC via GT along the detour path is consistently larger than that achieved by direct unicast from BU. Using ROMA, the measured throughput to UIUC was 37.2 Mbps, which is the minimum of the throughput across the overlay links, as our model predicts. This throughput improvement comes from the benefit of employing chains of TCP connections.

### 5.6.3 Maximizing Overall Throughput

The earlier analysis and the experiments in the previous sections point to a natural method for optimizing the layout of an application level multicast tree using ROMA: construct
the single-source “widest path” tree, i.e. the tree that maximizes the minimum per-hop available bandwidth to every destination. In this section, we sketch a simple algorithm for building this widest path tree and construct the tree for a PlanetLab overlay rooted at the University of Washington (depicted in Figure 5.12) using end-to-end measurements from an extended version of Table 5.1.

The algorithm to construct the widest path tree is a simple variant of Dijkstra’s algorithm [18], which is typically used to construct single-source shortest path trees. In a standard invocation of Dijkstra’s, links have associated weights representing propagation delay, and the algorithm repeatedly and greedily selects the unvisited node closest to the source, where proximity is measured by the sum of the weights on the path. To construct the widest path tree, links have associated weights representing available bandwidth (as per the entries in Table 5.1), and the algorithm repeatedly and greedily selects the unvisited node with the widest path from the source, where path width is measured by the
minimum of the weights on the path. The short proof that this greedy algorithm constructs the widest path tree follows the same reasoning as the shortest path tree argument.

The multicast tree depicted in Figure 5.12 is a widest path tree rooted at UW that we constructed using this algorithm from a set of measurements extending Table 5.1. We note that the widest path tree is not typically unique, since decisions below an unavoidable bottleneck link are immaterial. To build the first level of the tree, we used the fact that the maximum available bandwidth from UW to other hosts is about 39 Mbps (to UCLA is 39 Mbps, to Berkeley is 38.6 Mbps). At the second level of the tree, we used the fact that the maximum available bandwidth from UCLA or Berkeley to other nodes is about 21 Mbps, which is higher than any other available bandwidth from UW. Below these upper levels, we broke ties arbitrarily, since the available bandwidth between all pairs of hosts not on the west coast were mostly higher than 21 Mbps on Abilene.

Using this same topology, we compare the throughput of each host in each of the following three scenarios.

- **ROMA**: An overlay multicast tree is established to all hosts, and the throughput is measured at each point in the multicast tree.
- **Direct Unicast**: The throughput is measured when the content is transferred across a single unicast connection to the individual host.
- **N * Unicast**: The throughput is measured when all N hosts simultaneously download the content via separate unicast connections from the sender.
Comparing ROMA against direct unicast jointly demonstrates the performance advantages derived by split TCP connections, and the disadvantages of using an overlay infrastructure. The comparison of ROMA against “N * unicast” demonstrates the benefit of multicast by reducing the transmissions of many copies of the same data on outgoing links from UW.

In Figure 5.13 we depict the head-to-head comparison of our three methods. The figure shows that even though the overlay multicast generates some link stress, it is still far superior to “N * unicast” at all nodes. We also see that in many cases, the throughput of ROMA is better than direct unicast case and that this throughput advantage of ROMA comes from finding the widest path to destinations. We also see the effect of link stress, especially at nodes with considerable fanout, which results in ROMA having slightly worse performance than direct unicast.

Figure 5.14 depict the relative performance of pairs of these three methods as ratios. A ratio of 1.0 indicates no difference in throughput, while a ratio of 2.0 indicates a two-fold speedup. The results show that ROMA provides excellent performance compared with
the other unicast methods, and provides significantly improved performance over a single end-to-end TCP connection with surprising regularity.

### 5.6.4 ns Experiments

Although the experiments on PlanetLab enable us to test ROMA over the Internet, we are not able to tune network conditions to validate some parts of our model. One interesting network condition to validate the model of TCP chains is that all connections between end systems have the same network conditions, where the expected throughput of each receiver in our model is the minimum per-hop available bandwidth (Equation 5.2). We use Figure 5.15 to validate our model under this condition. Each connection between end
hosts has 44Mbps link capacity and is competing with 10 TCP flows, resulting in almost the same background traffic on each link. The average throughput of host B, host C, and host D over 100 trials are 4.27Mbps, 3.92Mbps and 3.62Mbps respectively. We have observed that even though the network conditions are almost identical, the performance degradation on adding each additional connection is about 9%.

In Figure 5.16, we pick one of our 100 trials whose mean rate is closest to the average throughput, and plot the number of packets in the buffer of host B to see why the performance degradation occurs. Time elapses on the x-axis, while the value of y-axis indicates the number of available packet in the application buffer, and it is measured at each time the host tries to send a packet to the downstream host. If this value is zero, then the intermediate host cannot send any packets even though the congestion window is larger than zero and has to wait for incoming packets from the upstream connection to send. If the intermediate host experiences this case, the downstream connection is not only affected by
its network conditions, but also its upstream connection. This case happens in the following conditions. If the upstream connection experiences TCP timeout, then the congestion window of upstream TCP connection is dropped to 1. Note that the congestion window for the downstream connection is independent from that of the upstream connection. Thus the timeout of the upstream TCP connection will make a big difference between the incoming rate and the sending rate. The downstream connection will consume packets in the application level buffer and the intermediate host may have no packet to send at a given time. This can cause the performance degradation of downstream connection. Even though the work in [2] gives more accurate analysis on chains of TCP, the performance degradation in the situation described above was not considered and presented. Our future work is to investigate the performance of TCP connections in more detail, and give the mathematical analysis.
Chapter 6

Conclusion

In dissertation, we have proposed scalable solutions for reliable multicast congestion control which have the following advantages: true AIMD multiple rate scheme (STAIR) and scalability with simplicity by leveraging a single rate scheme to build multiple rate congestion control (SMCC). Our approaches not only address the heterogeneous available bandwidth among the receivers, but also provide a fair share of network resources with competing TCP flows. We also proposed a new architecture (ROMA) for reliable content delivery in overlay network while providing different throughout among receivers and using TCP as a transport protocol for each end-to-end connection.

First, we have presented STAIR: a hybrid of cumulative, non-cumulative and stair layers to facilitate receiver-driven multicast AIMD congestion control. Our approach has the appealing scalability advantage that it allows receivers to operate asynchronously with no need for coordination; moreover, receivers with widely differing RTTs may simulate
different, TCP-friendly rates of additive increase. While asynchronous joining and leaving of groups at first appears to run the risk of consuming excessive bandwidth through a shared bottleneck, in fact judicious layering can limit the harmful impact of this issue.

Our approach does have several limitations. First, while our congestion control scheme tolerates heterogeneous audiences, it is primarily designed for users with high end-to-end bandwidth rates in the hundreds of Kbps range or higher. We expect that slower users would wish to employ a different congestion control strategy than the one we advocate here. Second, congestion control approaches which use non-cumulative layering and dynamic layers cannot be considered general purpose (just as TCP’s congestion control mechanism is not general-purpose) since not all applications can take full advantage of highly layer-adaptive congestion control techniques. For now, the only application which integrates cleanly with our congestion control methods is reliable multicast of encoded content.

We have also presented a multiple rate multicast congestion control design which leverages proven single rate congestion control methods by orchestrating an ensemble of independently controlled single rate sessions. A compelling argument for this new methodology is its evident simplicity: unlike all other viable multiple rate congestion control protocols, ours requires only a small amount of carefully crafted new functionality. By maintaining appropriate invariants on the session rates of individual TFMCC flows, specifying a clean mapping from reception rates to subscription levels and providing a non-disruptive method for additive increase join attempts, we build a sound multiple rate multicast con-
gestion control scheme called SMCC. A final advantage of our approach is its modular design; TFMCC or pgmcc could easily be replaced by an improved equation-based rate or window-based control mechanism.

Finally we presented ROMA, a new architecture for reliable distribution of large content across an overlay network using TCP. ROMA enables multiple-rate reception, with individual rates that match the end-to-end available bandwidth along the path, while using a minimal amount of resources at the application layer. A key component that our method employs is the use of erasure-resilient codes to provide reliability. The degree of freedom that the use of codes provides enables us to loosen the tight coupling of TCP connections that is needed in other designs to provide reliability, but also limits performance. The use of a digital fountain approach in our architecture also provides us with many additional benefits: small buffers, the ability to adaptively reconfigure the topology, and the ability to speed up downloads with collaborative peer-to-peer transfers.

Another contribution of our work is the analysis of chains of loosely coupled TCP connections that are established using our approach. We provide a simple model for the expected throughput across a chain of TCPs given per-hop RTTs and per-hop loss rates, along with validation using Internet experimentation. Our analysis and experimental results show that TCP chains offer an opportunity to increase performance by finding an alternative overlay path that is “wider” (as far as TCP is concerned) than the default path provided by IP. This observation also guides the construction of multicast trees that ROMA uses.
Bibliography


[41] PlanetLab. [https://www.planet-lab.org/](https://www.planet-lab.org/).


Curriculum Vitae

GU-IN KWON

Boston University
Department of Computer Science
111 Cummington Street
Boston, MA 02215
http://www.cs.bu.edu/

Office: (617) 353-9077
Department: (617) 353-8919
Fax: (617) 353-6457
guin@cs.bu.edu
http://cs-people.bu.edu/guin/

Education

Ph. D. 2005, Computer Science
"Scalable Architectures For Multicast Content Distribution"
Boston University
Major Advisor: John Byers

M.A., 1998, Computer Science
Queens College (City University of New York)

B.A., 1995, Computer Science
Inha University

Research Experience

1999-2004 - Research Assistant
Department of Computer Science, Boston University, Boston, MA
Work focused upon the design, verification, and implementation of Internet protocols. Published results in Multicast congestion control and protocol verification.

Teaching Experience

Fall 2003 - Teaching Assistant, CAS CS555 Computer Networks, Boston University

Spring 2001 - Teaching Assistant, CAS CS591 Advanced Computer Network, Boston University
Fall 2000 - Teaching Assistant, CAS CS101 Introduction to Computer Science, Boston University

Spring 2000 - Teaching Assistant, CAS CS101 Introduction to Computer Science, Boston University

Fall 1999 - Teaching Assistant, CAS CS101 Introduction to Computer Science, Boston University

Summer 1999 - Lecturer, CAS CS101 Introduction to Computer Science, Boston University

Journal Publications


Refereed Conference Publications


Community Activities

- Reviewed Papers For
  - ACM SIGCOMM
  - IEEE Global Internet (GI)
  - IEEE J-SAC
  - IEEE ICC
  - Computer Networks (Elsevier)