

# Dynamic Layer Multicast Congestion Control

Alex Roetter

June 6, 2000

# Abstract

Multicast is a routing protocol that allows broadcasting of Internet data from one sender to multiple receivers. This requires a congestion control protocol that will determine the appropriate rate for each of the recipients of the data stream. Multicast is currently implemented in a way that requires a large latency for a client to leave a multicast session. This long latency hinders existing congestion control protocols by lengthening their reaction time in the face of congestion. This thesis reports on joint work done on a protocol which uses dynamic layers, which make the reactivity of the protocol independent of the leave latency. The original parts of this thesis are the experimental results and the analysis of the algorithm. The protocol, called Dynamic Layer Congestion Control (DLCC), is responsive to dynamically changing network conditions and scalable to a large number of receivers. It requires no receiver-to-receiver coordination, and can be made to compete fairly against TCP flows. Experimental simulations illustrate its behavior, and show how it can be tuned empirically to compete fairly against TCP sessions.

# Acknowledgments

I'd like to thank Mike Luby of ICSI and Digital Fountain for working with me throughout this past year on this project. He came up with the original idea of dynamic layers, and provided many suggestions for simulations. Michael Frumin was an invaluable research partner, providing comments and improvements to the algorithm, as well as assisting in the implementation of the simulations. John Byers, Gavin Horn, and Michael Mitzenmacher were also all members of the group that worked on DLCC. Professor David Cheriton agreed to be my advisor at Stanford for this project, and provided helpful directions for research and this writeup. Prof. Balaji Prabhakar came through at the last minute and agreed to be my second reader. Mary McDevitt of the Technical Communications Program provided helpful comments on the writeup of this thesis. The honors program exists thanks to the vision of Professor Eric Roberts, and the financial support of Ben Wegbreit. Finally, I'd like to thank my parents for providing me with many opportunities, most notably my Stanford education, as well as for endless support along the way.

# Contents

<b>Abstract</b>	<b>i</b>
<b>Acknowledgments</b>	<b>ii</b>
<b>1 Introduction</b>	<b>1</b>
1.1 Desirable Properties . . . . .	3
1.2 Multicast vs. Unicast . . . . .	6
1.3 Forward-Error Correction Codes . . . . .	7
1.4 Previous Work . . . . .	8
<b>2 Dynamic Layer Congestion Control</b>	<b>10</b>
2.1 Background . . . . .	11
2.1.1 IGMP Latencies . . . . .	11
2.1.2 AIMD . . . . .	12
2.2 Time . . . . .	13
2.3 Groups . . . . .	13
2.3.1 Logical Groups . . . . .	14
2.3.2 Physical Groups . . . . .	16

2.3.3	Control Stream . . . . .	17
2.4	Sender Behavior . . . . .	17
2.4.1	Group Shifting . . . . .	17
2.4.2	Increase Flags . . . . .	19
2.5	Receiver Behavior . . . . .	23
2.5.1	Steady State . . . . .	24
2.5.2	Loss Response . . . . .	25
2.5.3	Increasing . . . . .	26
<b>3</b>	<b>Results</b>	<b>27</b>
3.1	Experimental Setup . . . . .	27
3.2	Random vs. Deterministic $b'$ . . . . .	28
3.3	Receiver Synchronization . . . . .	30
3.4	Join Latency and Time Window . . . . .	31
3.5	Layering Factor vs. Loss Spikes and Granularity . . . . .	33
3.6	DLCC vs. DLCC . . . . .	35
3.7	DLCC vs. TCP . . . . .	39
<b>4</b>	<b>Conclusion</b>	<b>42</b>
4.1	Future Work . . . . .	44
4.1.1	Slow Start . . . . .	44
4.1.2	Simpler Aggressiveness Model . . . . .	45

# List of Tables

3.1	Variance in the randomized sender. . . . .	30
3.2	Granularity vs. loss spikes. . . . .	35
3.3	Relationship of $a$ to bandwidth sharing. . . . .	38
3.4	Relationship of $TW$ to bandwidth sharing. . . . .	38

# List of Figures

1.1	A sample multicast topology. . . . .	2
2.1	Mapping of logical to physical groups. . . . .	19
2.2	Physical group bandwidth vs. time. . . . .	20
2.3	Using exponential time delay to simulate linear increase. . . . .	22
3.1	A sample simulation topology. . . . .	28
3.2	Sample run of randomized sender. . . . .	29
3.3	Receiver synchronization. . . . .	32
3.4	Loss spikes: Drop-Tail vs RED Routing. . . . .	32
3.5	DLCC vs. DLCC and TCP vs TCP. . . . .	36
3.6	Effect of $a$ and $TW$ on bandwidth sharing. . . . .	37
3.7	Sharing ratio as TCP RTT varies. . . . .	41

# Chapter 1

## Introduction

Multicast is a routing protocol that allows one sender to communicate with many receivers. Logically, a single-source multicast session can be represented as a tree, with the sender at the root and receivers at the tree leaves. Figure 1.1 shows a symbolic representation of a multicast session. Multicast holds great promise for the Internet because other technologies, such as the increased availability of end-to-end high bandwidth links, have made new applications possible. In particular, streaming video and large scale software distribution are becoming increasingly practical applications. These applications are suitable to multicast deployment as they involve one sender sending identical data to many recipients, analogous to a television broadcast station which sends the same television show to TV sets all around the country.

Data is sent not to an IP address, but to a multicast group address. Multicast-enabled routers forward data to an outgoing link if a host (receiver or router) downstream is “subscribed” to that multicast address. Because the routers provide support, a sender can send one copy of information, with the assurance that the downstream routers will propagate that information to the appropriate downstream branches of the tree.



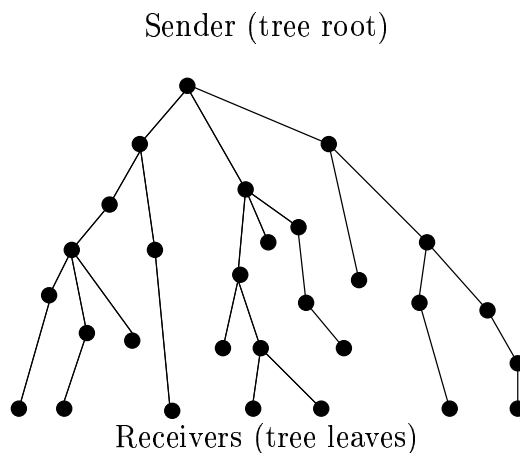


Figure 1.1: A sample multicast topology.

The Internet Group Messaging Protocol (IGMP) describes how a receiver can join and leave multicast groups. It offers facilities for the receiver to send join and leave messages to the nearest router. On a join, the router simply starts sending data from the required group to the receiver. If the router is not receiving data for that group, it recursively sends a join message for that group to the upstream router. A leave causes the router to stop sending the data, after a polling period to ensure that no other receivers are listening to the group [8].

The problem of multicast congestion control is the problem of determining how to send data to multicast receivers at the highest possible rate that the application desires, without unfairly dominating a bottleneck link and excluding traffic from other protocols such as TCP. This is particularly difficult with multicast, as the tree created can span over a variety of physical network types, each with different latencies and bandwidths. Competing traffic can vary from segment to segment across the multicast session and affect throughput on some of the links but not on others. In addition, the requirements and capabilities of individual receivers can vary dramatically.

The congestion control algorithm is responsible for calculating the rate of flow of data over every link in the network over which multicast data travels. To vary network utilization, the protocol can take action at either the sender or receiver side. On the sender side, the protocol can vary the rate at which data is sent to outgoing multicast groups. On the receiver side, the protocol can issue IGMP joins and leaves, which affect where multicast group data is routed.

This thesis reports on joint work done on the Dynamic Layer Congestion Control (DLCC) algorithm, which uses layered, time varying bandwidth multicast groups to provide congestion control capabilities to multiple receivers efficiently [4]. The original parts of this thesis are the experimental results and the analysis of the DLCC algorithm. This chapter outlines the desirable properties of a congestion control protocol, compares multicast to unicast, and describes a set of codes that make efficient multicast distribution schemes possible. Chapter 2 presents the algorithm and describes its implementation. Chapter 3 presents the results of experimental simulations of the algorithm. Chapter 4 describes how DLCC meets the requirements for a desirable protocol, and suggests directions for future work.

## 1.1 Desirable Properties

A desirable multicast congestion control (MCC) protocol is dynamic, scalable, uncoordinated, and fair to competing network traffic.

**Dynamic** The protocol should be sensitive and reactive to network conditions which vary over time, able to change the level of data sent to any portion of the multicast tree.

Network segments can fail, changing the bandwidth and latency characteristics of the

whole network. Over satellite and other wireless links, changing atmospheric conditions can affect throughput. In addition, data streams from other sessions (multicast or unicast) can start, stop, or vary their network utilization over the course of the multicast session. The MCC protocol must be responsive to changes in network capacity caused by any of these factors.

In addition, the protocol must respond quickly to these changes. A protocol which is slow to respond to a less capacitive network (due to a link failure, or other network traffic) will cause losses on the network, of its own packets as well as of packets of competing streams, until it reacts to the increased congestion by reducing the amount of data traveling over the network. A protocol which is slow to respond to a more capacitive network will not harm competing flows, but will allow bandwidth to go unused, causing the receiver to receive data at a slower rate than the network supports. Characteristics of IGMP (Section 2.1.1) make the response time difficult to minimize using current techniques.

**Scalable** Multicast has many applications when the number of receivers can be quite large.

Examples include broadcasting the superbowl live across the Internet, or providing popular content and files to millions of users simultaneously. The congestion control protocol should not require more work on the sender side as the number of receivers increases, because this would quickly outstrip sender resources. In other words, as the number of receivers  $n$  increases, the computational complexity of the sender algorithm should be  $O(1)$ .

**Uncoordinated** Despite a large number of receivers, the protocol should not rely on re-

ceivers being coordinated, or having any communication between each other. The protocol should work when receivers join and leave the session at arbitrary times, and drop out without warning. The challenging task is to have the protocol operate as efficiently as possible while remaining uncoordinated. Imagine two multicast receivers behind the same bottleneck, receiver  $A$  who is joined to multicast groups  $g$  and  $h$ , and receiver  $B$ , who is joined only to multicast group  $g$ . Since the bottleneck is already carrying the data for both groups (as both  $A$  and  $B$  are behind it), receiver  $B$  is not helping the network at all by not being subscribed to group  $h$ . In this scenario, the receivers are uncoordinated, but inefficient, as receiver  $B$  could join group  $h$  without any extra load on the network. The points downstream of the shared bottleneck, towards  $B$ , will be loaded with new data from group  $h$ , but since that is not the bottleneck link, that should not cause loss. Receivers can of course be coordinated with the sender, but must do so without any additional work for the sender, to preserve the property of scalability.

**Fair** The MCC protocol will likely be sharing a portion of the network with other instances of itself, and with TCP sessions. The session should be TCP-fair, meaning it ought to share a bottleneck link equally with a TCP session. At the very least, it should not be more aggressive than a TCP session. *TCP-fairness* or *TCP-friendliness* is defined in other congestion control literature as follows: a flow is *TCP-friendly* if its arrival rate does not exceed the arrival of a conforming TCP connection in the same circumstances [9]. Since different TCP sessions with different values for TCP's various parameters will behave differently (be more or less aggressive), it is impossible for a single MCC

session to compete fairly against all of them. However, there should be a TCP session with a set of parameters against which the MCC protocol with a set of values for its parameters competes fairly and roughly equally, that is, the long term fair share bandwidths of the two protocols are roughly the same. This work chooses an instance of a TCP session, shows how the algorithm presented here is fair to it according to this definition, and then shows how the relative long term bandwidth shares of the multicast and TCP sessions vary as TCP varies its session parameters.

## 1.2 Multicast vs. Unicast

The vast majority of traffic on the Internet today is connection based, one sender to one receiver communication known as unicast traffic. Under this model, the sender and receiver use TCP to decide the rate of data being sent to the receiver. This works well for the one-to-one case, as the sender and receiver can communicate on a per-packet basis to determine the appropriate rate of data transfer, but unfortunately this does not scale well to multicast applications.

In order to send data to multiple receivers, a unicast sender must open multiple connections, one to each receiver. The data to be sent must be sent on the outgoing link multiple times, addressed individually to each receiver. The sender must store state for managing flow control to each receiver, and must process flow control packets sent from each receiver. Using unicast to send to  $n$  receivers requires  $O(n)$  resources (in terms of both network bandwidth and sender processing power), an unacceptably large growth rate when the number of receivers becomes large. With multicast, the sender sends only one copy of data down-

stream, and routers take care of forwarding the data as appropriate. As a result, sending data requires  $O(1)$  work on the sender side for  $n$  receivers. This requirement that the server runs in constant time as the number of receivers grows means that unicast congestion control protocols cannot be used for multicast data.

### 1.3 Forward-Error Correction Codes

If a packet is lost in a unicast session, the sender resends the specific lost packet to the receiver. For multiple unicast sessions from one host serving multiple receivers (one per unicast session), packets are resent to individual receivers to fill in for the specific packets lost on specific flows. Assuming a constant loss rate, the number of re-transmissions a sender with  $n$  unicast sessions has to handle is  $O(n)$ . This re-sending scheme is unacceptable for multicast. How can we avoid the overhead required to re-send lost packets? For streaming applications, there is no problem. When broadcasting a video stream, each packet is only relevant for a brief period of time. If a packet is lost, the playback simply skips or freezes momentarily, resuming when subsequent packets are received. There is never a need to retransmit packets. For non-streaming applications, such as reliable bulk data transfer, we cannot simply ignore packets if the file is to be completely reconstructed.

A simple approach is to use a carousel technique. Assuming a packet size  $PS$  and a file of  $n$  bytes, divide the file into  $k = \frac{n}{PS}$  packets, numbered  $[0, 1, \dots, n - 1]$ . Every time a packet is sent, a counter  $i$ , which stores what packet is to be sent next, is incremented. The counter is initialized at 0, and incrementing it is done modulo  $k$ . This way the data is continually re-sent in a round-robin fashion, and the receiver can simply pick up what is lost the first

time around. Needless to say, this is time inefficient, requiring the receiver to wait a length of time equal to the total transmission time of the document to receive just one missed packet. Worst cast, the same packet could be lost the second time around, requiring another waiting period.

A class of algorithms known as Forward-Error Correcting codes (FECs) solves this problem more elegantly. Generally speaking, for an input file of  $k$  packets, these codes encode the document into  $h$  output packets, where  $h \gg k$ . A receiver needs only to receive  $a = ck$  packets (for some constant  $c \geq 1$ ) to reconstruct the original document. This allows a receiver to replace a lost packet with the next packet in the stream, and to terminate the session and decode the data once  $a$  packets have been received. Different FECs exist, which have different encoding and decoding times, as well as different values of  $c$ . Better codes have  $c \approx 1.0$  and fast encode and decode times. Examples of viable algorithms include Reed-Solomon encoding, as well as Tornado encoding [2], [5], [14].

## 1.4 Previous Work

Currently, multicast is not enabled on the Internet. One notable exception to this is the MBONE network, a multicast-enabled virtual network layered on top of the Internet [7]. If multicast were fully deployed, an adequate congestion control protocol would be essential to ensure that multicast traffic coexisted cooperatively with existing Internet traffic.

The Internet Engineering Task Force (IETF) has yet to standardize a congestion control algorithm [13]. The most prominent algorithm currently is the RLC algorithm developed by Vicisano, Rizzo and Crowcroft [20]. Though some analysis of the RLC algorithm has been

published, the IETF has not yet been convinced that enough work has been done to accept and standardize it. The algorithm also makes assumptions about the capacity of router buffering, to determine when more data can be sent. It is not clear that these assumptions are valid over the vast, diverse hardware of the Internet.



## Chapter 2

# Dynamic Layer Congestion Control

This chapter presents a congestion control algorithm called Dynamic Layer Congestion Control (DLCC) which meets the criteria outlined in Section 1.1 for a desirable multicast protocol [4]. DLCC is a layered protocol, meaning that the sender sends data to a group of multicast addresses with predefined bandwidths. Varying bandwidth levels are provided to receivers by allowing the receiver to join a subset of the layers the sender is sending data to. To receive data at a low rate, the receiver simply joins only a few relatively low bandwidth groups. To receive more data, the receiver joins additional groups. The advantage to layered approaches is that the sender can send data at whatever rate it chooses, and does not have to modify its speed to accommodate individual receivers. Other protocols have also used layering to provide multicast congestion control [3], [12], [15], [20]. If a layered approach is used, there must be a way to split the outgoing information up across multiple multicast groups. For transfer of large data sets, the use of FECs as described in Section 1.3 provides an easy way to distribute the data to be sent over multiple outgoing multicast groups. For streaming data, encoding schemes also exist to transmit the data, encoded at different bit rates [18],

[19].

## 2.1 Background

To understand the motivation behind DLCC's layering scheme, we must look at IGMP latencies to determine what receiver commands (joins and leaves) need to be optimized. In addition, we need to understand how other protocols choose to increase and decrease their bandwidths, to make sure DLCC provides similar facilities to its receivers.

### 2.1.1 IGMP Latencies

To change how many groups a receiver is joined to, a receiver must issue IGMP join or leave requests to the upstream router.

IGMP join requests are sent to the router, which, if currently receiving the requested group, simply forwards packets for that group to the host that issued the join request. If the router is not receiving this group, the router recursively issues a join to its upstream router. The *join-latency* is defined as the time between the receiver's issuing of a join request and the time the first packet from the joined group arrives at the receiver. The join-latency is close to the actual round trip time from the receiver to the point on the multicast tree that the receiver is grafting on to (i.e., to the nearest upstream router that carries the group). In the worst case, it will be on the order of the round trip time from the receiver all the way back to the sender at the tree root. Join-latency can vary from  $< 10\text{ms}$  for LAN multicast sessions, to values on the order of  $\approx 100\text{-}200\text{ms}$  for transcontinental sessions. These values depend heavily on network conditions and topologies, but these are reasonable

working values. What is important is the relationship of join-latency to leave-latency.

To stop receiving data from a group, a receiver issues an IGMP leave request. This request causes the router to poll the link on which the request was received to see if other hosts on that link still wish to be joined to the group. If no hosts want to be joined, the router stops forwarding packets to the outgoing interface. If the router has no other downstream links receiving the group, it recursively issues an IGMP leave on its upstream link. The *leave-latency* is defined as the period of time between the issuing of a leave request and the arrival at the receiver of the last packet addressed to that group. The length of the leave-latency depends on network conditions, and the type of link. Point-to-point links require no polling period, but shared mediums such as Ethernet do require one. Because there is no guarantee that the polling message is delivered, the poll is conducted several times to ensure that receivers still interested in being joined to the group have a chance to inform the router. In IGMP version two (the current standard), the leave-latency can be as long as nine seconds, orders of magnitude longer than the join-latency.

### 2.1.2 AIMD

TCP controls congestion using what is called an additive increase, multiplicative decrease (AIMD) protocol. When no loss is felt, the protocol periodically tries to increase the rate of flow by an arithmetic amount. In TCP, this is done by increasing the window size by one packet. The window size is the number of outstanding packets TCP can have on the network, packets for which no acknowledgment has been received. On detecting a loss, an AIMD protocol makes a geometric decrease in its sending rate. TCP halves its window size.

AIMD provably causes equally aggressive competing flows to eventually share bandwidth equally, regardless of their initial bandwidth shares [6]. To compete fairly against TCP and other flows, DLCC also implements AIMD.

## 2.2 Time

To provide synchronized behavior to both the sender and receiver, as well as to responsively deal with loss, all nodes (both sender and receiver) have a global notion of time. Time is divided up into segments called *time windows*, where a time window is very short relative to the total length of a multicast transmission. The length of this time window also determines the algorithm's responsiveness because decisions to join or leave groups are only made on time window edges, the transition from one time window to its successor. In DLCC, the length of the time window is given in seconds by the parameter  $TW$ . Typical values range from several hundred milliseconds to a couple of seconds.

A running counter  $t$  of the current time window is kept, incremented every  $TW$  seconds. Incrementing  $t$  is done modulo  $P$ , where  $P$  is the total number of physical groups as defined in Section 2.3.2. The current time period  $t$  is put in the packet header of every outgoing packet, so the receiver can remain synchronized with the sender.

## 2.3 Groups

The major problem that this protocol tackles is the problem of long leave latencies that plagues IGMP as it is currently formulated. It does this by using dynamic layers, meaning

that the rate at which data is transmitted on a given multicast group varies over time. Older schemes have used static layers (layers on which the bandwidth is constant) as their layering system. An advantage to a static layering scheme is that it requires fewer actual multicast group addresses to transmit data. Depending on an ISP's charging scheme (some plan on charging on a per-group basis), this could be a drawback of DLCC. The problem with static layers is that lowering one's bandwidth requires issuing a IGMP leave request. This can unfortunately take a long time just when the network is congested and needs most to lower the rate of data being sent on it to avoid loss.

The RLC algorithm tries to avoid this problem by “testing” the network, briefly increasing the rate being sent on all the groups, to see if that induces loss [20]. If it does not, that is taken as a sign that the network can support a join of more groups, and the receiver then issues joins. Though this helps ameliorate the problem of having to leave, it is not clear that these brief bursts are always sufficiently long to induce loss. Router buffers could simply absorb a brief burst, whereas they could not sustain continued transmission at that rate. Also, other traffic not present during the burst test could ramp up and cause the multicast session to need to drop a group, even though the burst test did not induce loss.

To implement these dynamic layers, DLCC has two concepts of groups, *logical groups* and *physical groups*.

### 2.3.1 Logical Groups

Logical groups are constant bandwidth flows that the receiver can join to in a cumulative fashion. The groups are ordered  $[1, \dots, N]$  where  $N$  is the number of logical groups in the

session. Cumulative joining ensures that if a receiver is currently joined to the  $i$ th group, the receiver is also joined to groups numbered  $[1, \dots, i - 1]$ .

Each logical group is some non-negative multiple times the base bandwidth of the session. The bandwidth of the  $i$ th group  $B_i$  is given in terms of the multiple  $b_i$  and the base bandwidth  $BW$  as follows:

$$B_i = b_i * BW \quad (2.1)$$

To provide different available levels for the receivers, different values for  $b_i$  can be chosen. The cumulative nature of the algorithm ensures that the total bandwidth  $\alpha_n$  for a receiver joined to  $n$  groups ( $0 \leq n \leq N$ ) is:

$$\alpha_n = \sum_{i=0}^n b_i * BW \quad (2.2)$$

Though groups are numbered from 1,  $b_0$  is set to 1, and corresponds to the control stream bandwidth (Section 2.3.3). A scheme for choosing each  $b_i$  can be chosen on a per application level, depending on the requirements for fine grain control, range of bandwidths, and number of groups used. The following scheme is proposed and used in the simulations of Chapter 3. Receivers are provided with a range of bandwidth levels  $[\alpha_0, \dots, \alpha_n]$  at which they can receive data, where each  $\alpha_i$  is a constant factor greater than its predecessor  $\alpha_{i-1}$ . This is accomplished by defining the bandwidth received when joined to  $n$  groups as follows:

$$\alpha_n = x^n \quad (2.3)$$

Where  $x$  is the layering base factor, and is set to a constant for a particular session. From

Equations 2.3 and 2.2, it follows that each individual  $b_i$  can be set as follows:

$$b_i = x^i - x^{i-1} \quad (2.4)$$

This scheme allows a broad range of granularities depending on the value of  $x$ . Joining another logical group always increases the total bandwidth sent to the receiver by a factor of  $x$ .

### 2.3.2 Physical Groups

Physical groups are actual multicast addresses. For a given physical address the bandwidth is a function of  $t$ , the current time window the session is in, which changes with frequency  $\frac{1}{TW}$ .

The number of physical groups  $P$  is not equal to  $N$ , the number of logical groups. In addition to having  $N$  physical groups, which correspond to logical groups, there are also  $C$  additional groups, who all transmit at the rate of zero. The value  $C$  is computed as follows:

$$C = \text{ceil} \left( \frac{LL_{max}}{TW} \right) \quad (2.5)$$

Where  $LL_{max}$  is a conservative estimate of the maximum leave latency in seconds.  $LL_{max}$  depends not only on the inherent leave latency, as specified by IGMP, but also on the time the leave request takes to get from the node to the upstream router and the time to empty the buffer on the downstream link. In the worst case, the upstream router could be right by the sender itself, so  $LL_{max}$  should incorporate a conservative upper bound on the time to

get from a distant receiver to the sender, in addition to the IGMP latency. The reason for these  $C$  extra groups will be explained in Section 2.5.1. After  $C$  is calculated, the number of physical groups  $P$  is given by  $P = C + N$ . Both the sender and receiver know the multicast group addresses of every physical group in use by the session.

### 2.3.3 Control Stream

In addition to the physical groups, a control stream exists whose bandwidth is always set to  $BW$  and does not change over time. The control stream also includes in its packet header the value of  $t$  for the current time period. This stream is joined to by every receiver in the session, and stays joined to for the entire duration of the receiver's session. Each receiver must be able to receive data at a rate of at least  $BW$ , because  $BW$  is the minimum bandwidth the sender can output.

## 2.4 Sender Behavior

### 2.4.1 Group Shifting

If the bandwidths of each physical group are to be determined, a mapping from logical groups to physical groups must be provided. With that mapping, we simply send at a rate  $B_i$  to the physical group mapped to the  $i$ th logical group. Since  $P > N$ , some physical groups will not have corresponding logical groups. Those groups do not send any data.

For logical groups  $[l_1, \dots, l_N]$  and physical groups  $[p_1, \dots, p_P]$ , at the beginning of the simulation when  $t = 0$ , we simply assign the first  $N$  logical groups to the first  $N$  physical

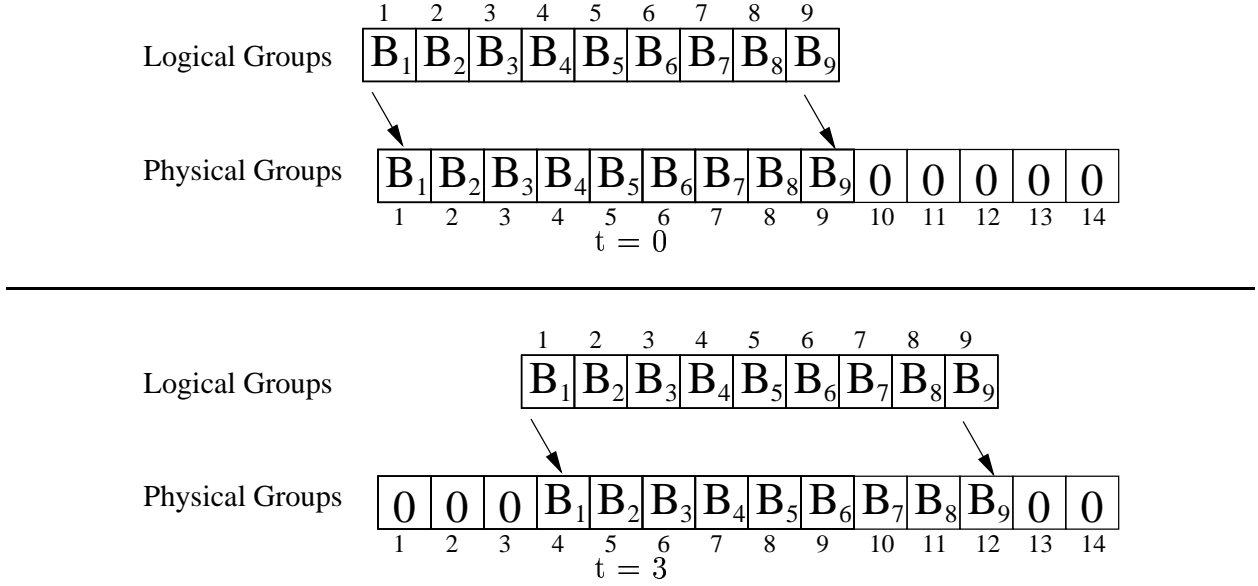


groups. The remainder of the physical groups  $[p_{N+1}, \dots, p_P]$  are set to zero.

In general, the following mapping assigns logical groups to physical groups during a time window  $t$ :

$$l_i \rightarrow p_{i+t \bmod P} \quad (2.6)$$

In this mapping, the modulo operator ensures the subscript is in the range  $[1, \dots, P]$ . The remaining physical groups are assigned a value of zero. On every time window edge, the bandwidth of a physical group  $p_i$  drops by a factor of  $x$  to the bandwidth of the logical group one less than the previous logical group assigned to  $p_i$ . For example, using a layering factor of  $x = 2$  (Section 2.3.1), a group's bandwidth will halve every time period then drop to zero ( $16 \rightarrow 8 \rightarrow 4 \rightarrow 2 \rightarrow 1 \rightarrow 0$ ). Figure 2.1 Shows how logical groups are assigned to physical groups over time. Figure 2.2 shows how an individual group's bandwidth changes over time. Each line at the bottom of the graph is an individual physical group, and the control stream bandwidth can be seen as the constant, relatively low bandwidth flow. The high bandwidth line at the top of the graph is the aggregate amount of data the receiver receives from all its groups. Notice the small gaps in time between when one group drops from the top bandwidth to the next lowest level and when another group ramps up from zero to the highest bandwidth. This small gap accounts for the variance in the aggregate bandwidth line, and is caused by the join latency. This effect is discussed further in Section 3.4.

Figure 2.1: Mapping of logical to physical groups for  $N = 9$ ,  $P = 14$ .

### 2.4.2 Increase Flags

In order to ensure synchronized receivers, the sender must dictate to receivers when they are allowed to increase their subscription level. *Subscription level* is defined as the number of logical groups the receiver is currently joined to. A receiver joined only to the control stream has a subscription level of zero. Every packet sent on every logical group has an increase flag bit, which is set to true if a receiver who has that logical group as their highest subscribed layer may increase its subscription level in the current time window. A receiver whose subscription level is  $i$  is joined to  $[l_1, \dots, l_i]$  and may raise its subscription level to  $i + 1$  only during a time window when the increase flag for logical group  $i$  is set.

We do not want receivers deciding individually whether or not they can increase. Unsynchronized increases will induce network losses when the new level creates a total bandwidth ( $\alpha_{i+1}$ ) which is above the capacity of a bottleneck in the network. This will cause other receivers to sense a loss and lower their subscription levels. As a result, the sender must

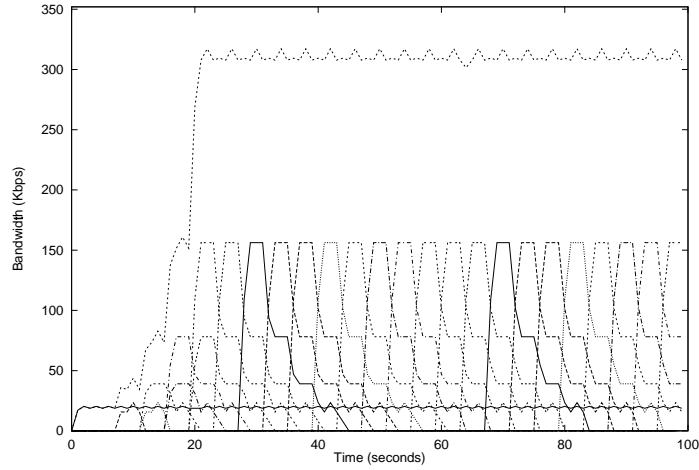


Figure 2.2: Physical group bandwidth vs. time.

dictate increase flags to the receivers, so receivers all increase in a synchronized fashion.

Increase flags must be cumulative, meaning that if in a given time window an increase flag for layer  $i$  is set, then the increase flags for all lower layers  $[0, \dots, i-1]$  must also be set. This restriction exists also for the sake of synchronizing receivers. Imagine two receivers  $A$  and  $B$ , where  $A$ 's subscription level  $SL_A > SL_B$ . Each receiver responds to loss as described in Section 2.5.2, by lowering its subscription level by one. If  $A$  increases  $SL_A$  while  $SL_B$  remains constant, a loss could occur which bumps  $SL_A$  back down to its previous level, and  $SL_B$  to *lower* than it was originally. In this case,  $A$  and  $B$  are both behind a common bottleneck, as  $A$ 's traffic is inducing losses on  $B$ 's traffic. Rather than their synchronizing,  $A$ , who is at a higher subscription level, is causing  $B$  to *lower* its  $SL_B$ . Assuming that increase flags are cumulative, this situation could never occur, as both  $A$  and  $B$  have increase flags at the same time, and a loss incurred would cause both receivers to drop down to their previous levels. Similarly, it could be the case that  $B$  has an increase flag when  $A$  does not, and a successful join in this case would bring the two receivers one step closer to equal subscription levels.

Cumulative grouping also makes loss detection easier. If subscriptions were not cumulative, a loss could occur on a logical group  $g$  in a router upstream of the receiver. By the time that loss could be felt by the receiver, it could have increased its subscription level and left  $g$ . The receiver would never notice the loss on  $g$ , and would continue to increase its subscription level even though a packet had been dropped. Cumulative layering ensures that increasing a subscription level never involves leaving a logical group, and therefore guarantees that a client will never accidentally ignore loss events.

Choosing an algorithm for setting increase flags must take into account the property of additive increase. Averaged over time, DLCC must try to increase linearly (additively), or else it will not compete fairly against AIMD protocols, most notably TCP. The increase flag algorithm must then be chosen with respect to the individual bandwidths of the logical groups. For example, using a layering factor  $x$  of 2 as described in Section 2.3.1, increasing the subscription level by one doubles the total amount of bandwidth subscribed to. As a result, increase flags for layer  $i$  must occur twice as frequently as increase flags for layer  $i + 1$ , because the bandwidth gained going from  $SL_{i+1}$  to  $SL_{i+2}$  is twice the bandwidth gained going from  $SL_i$  to  $SL_{i+1}$ . Though the groups are spaced exponentially, the increases are spaced out over time exponentially, causing the time averaged increase to be linear. Figure 2.3 illustrates this for  $x = 2$ .

Two different approaches were used for determining increase flags, a deterministic and a randomized scheme. Both schemes are calculated once at the beginning of each time window to determine what flags to set for the duration of the current window.

**Deterministic** This algorithm uses a reverse binary counter to decide on increase flags.

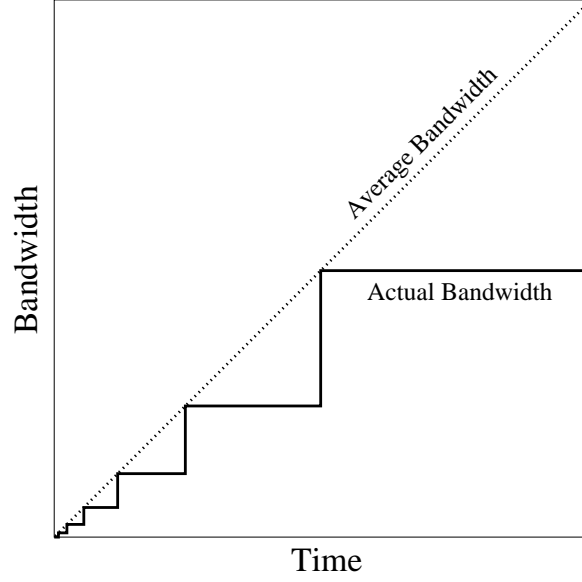


Figure 2.3: Using exponential time delay to simulate linear increase.

Let  $b$  be a binary number with  $s$  bits, written as  $b_0b_1\dots b_{s-1}$ . Start the simulation with  $b$  equal to zero, and increment it every time window. Let  $b'$  be the number written backwards, interpreted as a value in the range of zero to one.

$$b' = .b_{s-1}b_{s-2}\dots b_0 \quad (2.7)$$

For example, the decimal number 23 is written in binary as 10111, written backwards as a real number between 0 and 1 is .11101, or .90625 in decimal.

**Randomized** Rather than the above mentioned deterministic scheme, for every time window a pseudo-random number  $b'$  is chosen in the range  $[0\dots 1]$ .

Given  $b'$ , we must assign a likelihood to each group that its increase flag will be set, and the likelihood must decrease as the logical group gets higher. This likelihood is the sum of one over the bandwidths of the group in question and of all lower groups. To determine

what groups in a given time window have their increase flag set, we find the highest  $i$  that satisfies the following equation:

$$\frac{a}{\sum_{j=1}^i b_j} \geq b' \geq \frac{a}{\sum_{j=1}^{i+1} b_j} \quad (2.8)$$

The  $i$  calculated is then interpreted as the highest subscription level that a receiver can be at and still be allowed to increase in the current time window. The increase flag for all logical groups numbered  $i$  and lower is set, and the increase flag for all groups numbered *greater* than  $i$  is cleared. This inequality might not have a solution, in which case no increase flags are set. This tells the receiver it can increase only if  $SL = 0$ . Because these flags tell the receiver that they are allowed to increase only if their highest group is  $i$  or lower, all new joins will occur on groups numbered  $i+1$  or lower. In this equation,  $a$  is the *aggressiveness factor*, a constant parameter to the session, that affects how frequently the sender sets increase flags. It can be used to fine tune the aggressiveness of a particular session against another one. Note that this calculation of bandwidths considers only the coefficients  $b_i$  of the logical groups, and not the actual bandwidths that are determined from Equation 2.2.

## 2.5 Receiver Behavior

The receiver behaves by receiving packets from the sender, which inform it of the current time period, and issuing joins and leaves as appropriate to respond to loss and attempt to increase its bandwidth.

Upon startup, the receiver subscribes to the control stream. It does this because it is the smallest available bandwidth, and because the control stream tells the receiver the current

time window.

### 2.5.1 Steady State

In the steady state, the receiver is neither experiencing loss nor trying to increase its overall subscription rate. All it has to do is stay subscribed to the same number of logical groups. If the sender were using all static groups, this would require no work on the receiver side. Due to the dynamic scheme, the receiver must issue IGMP joins and leaves on physical groups to keep the number of logical groups constant.

Assume that the receiver is subscribed to  $n$  logical groups,  $[l_1, \dots, l_n]$ , which are mapped at the current time window  $t$  to physical groups  $[p_{t+1}, \dots, p_{t+n}]$ . It is important to note that the receiver is joined only to physical groups  $t + 1$  through  $t + n$ . In these examples, a physical group subscript is computed modulo  $P$ , and shifted by one if necessary so that it is in the range  $[1, \dots, P]$ . On the transition from time window  $t$  to time window  $t + 1$ , two things of consequence happen due to the shifting of logical groups. First, group  $l_1$ , which used to be mapped to  $p_{t+1}$ , is reassigned to  $p_{t+2}$ . This means that  $p_{t+1}$ , which used to carry data, now has no data on it. Second, the highest logical group  $l_n$ , which used to be assigned to  $p_{t+n}$ , is now reassigned to physical group  $p_{t+n+1}$ , which the receiver is not subscribed to.

The first condition does not require any immediate action, but in  $C$  time windows, the group  $p_{t+1}$  will go from carrying no bandwidth to carrying the bandwidth of the highest logical group. To ensure that the receiver is not joined to that group when it surges in bandwidth, it issues a leave on the window edge, when its bandwidth drops to zero. Note that  $C$  time windows is exactly  $C * TW$  seconds away, equal to the maximum leave latency

$LL_{max}$  by Equation 2.5. This allows the receiver ample time to leave the group before the group is again mapped to a logical group carrying data.

The second condition requires a join of  $p_{t+n+1}$  in order to maintain the same subscription level for the new time window. The receiver issues a join for that group as soon as the receiver enters the new time window. The behavior of leaving one group and joining one group takes place on every time window edge. This is not a significant amount of bandwidth, as an IGMP join or leave message is quite small compared to a typical data packet (approximately 80 bytes), and multiple upstream messages are grouped intelligently by routers into one upstream message. All this traffic is upstream, and will not contribute to the downstream congestion of sender data.

### 2.5.2 Loss Response

The receiver continually monitors loss throughout the session. Loss can be calculated based on sequence numbers, with out-of-order arrival or missing packets constituting a loss event. On each time window edge, the receiver looks to see if a loss event occurred during the time window that just finished. If there was such an event, it decreases the number of logical groups to which it is subscribed by one. It does this by issuing only the leave it would have issued in the steady state, and does not issue the join. This multiplicatively decreases the total bandwidth received by a factor of  $x$ .



### 2.5.3 Increasing

On each time window edge, the receiver checks the new time window's increase flag for the highest logical layer to which the receiver is subscribed. If the flag is set, the receiver increases the number of logical groups to which it is subscribed. If the receiver is not joined to any groups other than the base layer, it can attempt an increase on every time window edge. It does this by not only joining  $p_{t+n+1}$ , the group it joins in the steady state, but also by joining  $p_{t+n+2}$ , the physical group corresponding to the new highest logical group. The receiver also issues the leave that the steady state receiver issues.

# Chapter 3

## Results

This chapter describes and presents the results of DLCC simulations. An analysis of the two increase flag algorithms (randomized and deterministic), and an analysis of different layering factors are presented. Join latency's effect on server efficiency is explained. Receiver synchronization is verified experimentally, and the algorithm's ability to compete against itself and against TCP is explored.

### 3.1 Experimental Setup

These simulations were implemented in C++ and TCL using a modified version of the network simulator *ns*, version 2.1b5 [1]. Simulations were conducted on a 550MHz Pentium III with 512MB of RAM and 19GB of disk available to Linux, kernel version 2.2.13. Experiments were run on a network in a double star configuration. All senders are on their own segments of the network, which feed into a bottleneck link. The bottleneck link ranges in capacity from 500Kbps to 10Mbps. On the other side of the bottleneck, a link exists for each

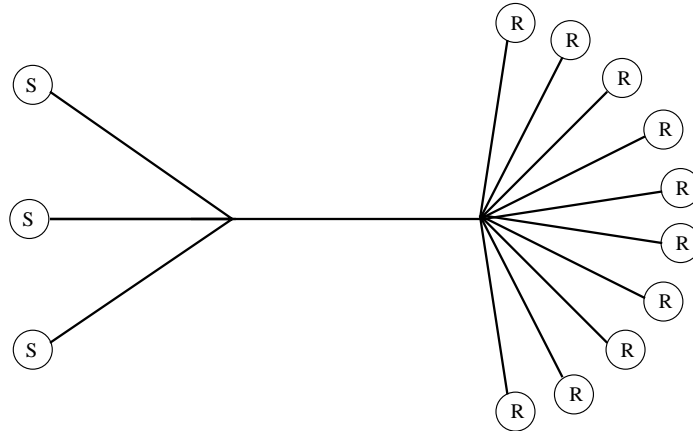


Figure 3.1: A sample simulation topology. The nodes on the left are senders, the nodes on the right are receivers, and the horizontal link in the middle is the bottleneck link.

receiver. The non-bottleneck links have a higher capacity relative to the bottleneck link, and never experience loss. Figure 3.1 shows a sample network topology for 3 senders sending across the bottleneck to 10 receivers. The senders and receivers here might be communicating via DLCC or TCP. This varies depending on the simulation being run. Routers contain outgoing buffers able to queue 50 packets, and experiments with both Drop-Tail and RED routers were run. The latency of every link is 10ms, and the multicast leave latency is nine seconds. Running any experiments with different values of the leave latency did not affect the results.

## 3.2 Random vs. Deterministic $b'$

In Section 2.4.2, a deterministic and random method for calculating  $b'$  were proposed. Simulation results show that the deterministic method is a more advantageous scheme. This is true for two reasons. First, the behavior of the algorithm is highly dependent on the particular pseudo-random generator used at a particular sender host. This can certainly

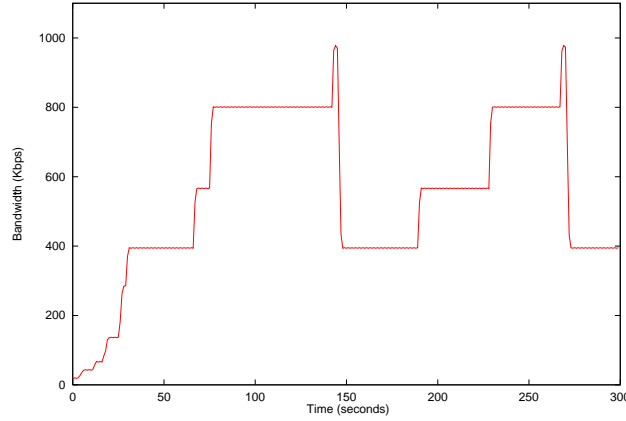


Figure 3.2: Sample run of randomized sender. The receiver reaches the bottleneck 2 times, first at time 143.1 seconds.

vary from machine to machine, and it is best to isolate that from the sender performance if possible. Even with the same generator, numerous runs of the protocol cause drastically different results.

So that the variance over multiple runs of the pseudo-randomly chosen  $b'$  could be seen, an identical experiment involving a single sender to a single receiver was run multiple times. The number of times that the receiver reached the bottleneck, thereby experiencing loss and dropping its subscription level, was measured. In addition, the time it took until the receiver reached the bottleneck the first time, as well as the variance of those times, was recorded.<sup>1</sup> The bandwidth graph of a single run of the randomized sender is shown in Figure 3.2, and the results of the multiple runs are shown in Table 3.1. In contrast, the deterministic algorithm has no variance between runs.

Second, since for small amounts of time the pseudo-randomly chosen  $b'$  is not uniformly distributed over  $[0 \dots 1]$ , the additive increase can vary from time to time. While at times

---

<sup>1</sup>Variance was calculated according to the following equation [17]. For a variable  $X$  and its expected (mean) value  $\mu$ :

$$\text{Var}(X) = E[X^2] - \mu^2$$

Number of Runs	30
Number of bottleneck reaches range	0-4
Number that reached the bottleneck	28
Number of bottleneck reaches mean	1.64
Bottleneck reach time range	69.14-256.1
Bottleneck reach time mean	172.1
Bottleneck reach time variance	2034.4

Table 3.1: Variance in the randomized sender. The 2 runs that never reached the bottleneck in the 300 seconds of the experiment were not factored into the calculation of the bottleneck reach time statistics.

the receiver will increase quite frequently, creating a steep rate of climb, other sequences of pseudo-random numbers will cause the receiver to increase more slowly. The uniform rate of climb provided by the deterministic receiver, as well as its property of zero variance across multiple runs, make it much easier to make the algorithm compete fairly against itself or TCP. The remainder of the results presented in this chapter use the reverse binary (deterministic) method for calculating  $b'$ .

### 3.3 Receiver Synchronization

As stated in Section 2.4.2, increase flags must be cumulative so that receivers can synchronize. A group of receivers are synchronized if they are behind a common bottleneck and all at the same subscription level. Synchronization should happen as soon as possible, as unsynchronized receivers needlessly under-utilize network resources.

Figure 3.3 shows receiver synchronization of 100 receivers with one sender, where all receivers synchronize behind a common bottleneck. Simulations with different values of the parameters from the graph shown were run, but they all synchronized eventually. Even with multiple senders, all receivers associated with a particular sender synchronize. The

only requirement to guarantee synchronization is that the sender increase signals must be cumulative.

As seen from the bandwidth graph, RED routers respond more quickly to overshooting a bottleneck and causing loss than do Drop-Tail routers. RED routers randomly discard an incoming packet when the queue size in the router buffer exceeds the average value by some amount [11]. The advantage to this is that it keeps the average router buffer size quite small, allowing for shorter packet arrival times from the sender to the receiver. This makes the receiver notice loss more quickly, which is why increases over the bottleneck are followed quickly by a decrease in bandwidth. In Drop-Tail routing, loss is only felt after the router buffer is completely full. The receiver does not notice loss until all the packets in the buffer are delivered. This accounts for the longer amount of time it spends at the bottleneck capacity, inducing much higher losses. Figure 3.4 shows the number of packets lost when the receiver exceeds the bottleneck under both queuing models.

### 3.4 Join Latency and Time Window

When establishing a DLCC session, a couple of factors should influence the time window value. As we will see in Section 3.6, the setting for  $TW$  affects the overall aggressiveness of the algorithm. In addition, the expected join latency, if known, can be used to influence the choice of time window. Figure 2.2 shows that the bandwidth for a client in the steady state is not constant over time, but rather oscillates slightly around its total bandwidth level. This has to do with the effect of join latency on the protocol. On every time window edge, the bandwidth (even in the steady state) drops by a factor of  $x$ . As a result, the client issues a

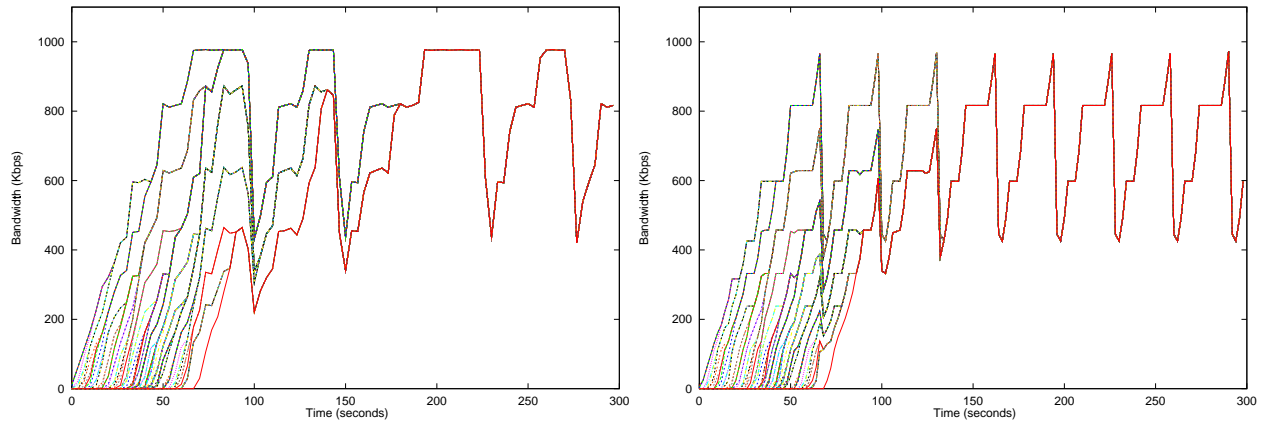


Figure 3.3: Receiver synchronization. The graph on the left uses Drop-Tail routing, and the graph on the right uses RED routing. Each line is the bandwidth of an individual receiver. This simulation was run over a 1Mb bottleneck, with receiver starting times interleaved throughout the first 70 seconds of the simulation. The sender had a 500ms time window, and an aggressiveness factor  $a$  of 2. The base bandwidth was 20Kbps, and 1.4 was the base layering factor  $x$ .

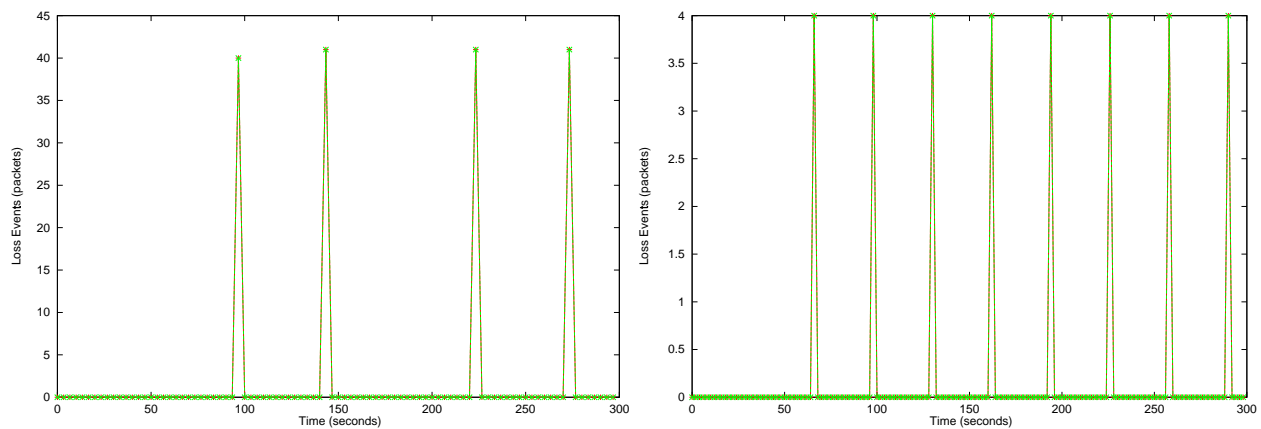


Figure 3.4: Loss spikes: Drop-Tail vs RED Routing. Drop-Tail is on the left, RED is on the right. Drop-Tail routing caused 4 loss spikes, with average size of 40.75 packets. RED routing caused 8 losses, each consisting of 4 packets. Though loss performance is more favorable with RED routers, all receivers synchronize regardless of which queueing method is used.

join to compensate for the loss of bandwidth. The bandwidth does not then get back to its previous level for a period of one join latency. Assume that the join latency  $j$  is less than the time window (as is typical), and is expressed as a fraction of the  $TW$  such that  $[0 \leq j \leq 1]$ . The actual bandwidth  $\alpha_{act}$  is related to the theoretical bandwidth  $\alpha_n$ , as shown in Equation 3.1.

$$\alpha_{act} = j \left( \frac{\alpha_n}{x} \right) + (1 - j)\alpha_n \quad (3.1)$$

For a client whose join latency is equal to the  $TW$ , the maximum bandwidth received is a factor of  $x$  smaller than the bandwidth being sent by the server to the groups to which the client is subscribed. The reason for this is that by the time the client joins to all of the groups, the groups have already fallen down by a factor of  $x$ . For a join latency greater than a time window, the actual bandwidth is even less. Though a small source of inefficiency, this knowledge of join latency can be used when setting up the server to try to make the time window large relative to the join latency.

### 3.5 Layering Factor vs. Loss Spikes and Granularity

When choosing a value for the constant layering factor  $x$ , the tradeoff between fine grain control (granularity), number of groups, and loss response must be considered. For a large  $x$ , few groups are required to reach high bottlenecks, but the difference in bandwidth between groups is large. The total bandwidth increases by a factor of  $x$  every time the subscription level is raised by one. For a smaller  $x$ , more groups are required to reach the same bandwidth. This offers more fine grain control to the receiver. A prime consideration when setting



bandwidth levels is determining the loss spikes that are generated. A loss spike is defined as a time window in which losses are felt by the receiver. Its value is equal to the total number of packets dropped for the duration of the spike. A single increase can cause multiple spikes over consecutive time windows. Appropriate settings for  $x$  were determined by running a single session on different bottlenecks, such that the total number of groups required to reach the bottleneck was either 5, 10, 15, or 20. As the number of groups increased, the more fine grain control created smaller loss spikes, as increasing over the bottleneck does not increase by as large a percentage of the total bottleneck bandwidth. For applications where an approximate bottleneck bandwidth is known,  $x$  should be set to the smallest value that generates acceptably small loss spikes. In some applications, large losses might not be a concern, and  $x$  can be smaller to reduce the number of groups in use.

Table 3.2 shows the results of these experiments. RED routers, a 500 millisecond time window, and 20Kbps base bandwidth were used. Different aggressiveness factors were set for the different bottlenecks, to ensure that the receivers reached the bottleneck during the course of the 400 second experiment. For the 0.5Mb bottleneck  $a = 2$ , for the 1Mb bottleneck  $a = 4$ , for the 5Mb bottleneck  $a = 20$ , and for the 10Mb bottleneck  $a = 40$ . These values caused the receivers to reach the bottleneck quickly, always within the first 75 seconds of the experiment. For larger bandwidths, as the number of layers rose, the loss spike size dropped. For smaller layers, this effect was not as consistent, because the loss spikes were so much smaller to begin with, and other factors could affect them and disrupt the continuous decrease that we see in higher layers as the number of groups rises.

	Number of Layers			
Bandwidth	5	10	15	20
500Kbps	3.9	1.75	6.8	4.3
1Mbps	24.3	46.4	4.1	6.1
5Mbps	285.1	203.7	128.2	96.4
10Mbps	556.3	475.2	285.6	207.5

Table 3.2: Granularity vs. loss spikes. As the number of layers increases, the receiver has more fine grain control and the number of packets dropped when the bottleneck is overshot diminishes. This experiment used RED routers with 50 packet buffers. The values in the table are the average size (in packets) of all loss spikes that occurred during the experiment.

### 3.6 DLCC vs. DLCC

Experiments were run to see how DLCC competes against other instances of itself. As a control, 4 identical instances of the protocol were run against each other, all starting at different times throughout the beginning of the experiment. Flows which start later ramp up to the rate of the other flows and end up sharing the bottleneck link equally with other instances. As multiple receivers on one sender synchronize, the number of receivers does not affect the fair share bandwidth between different senders across the bottleneck. Figure 3.5 shows the bandwidths of a representative group of DLCC sessions, involving four senders transmitting to four receivers over a 1Mb bottleneck. The four receivers started at times 0.1, 20, 40 and 70 seconds. The specific start times do not affect the overall fair share of bandwidth, which is approximately equal. To provide a point of comparison, 4 TCP sessions were also run against each other over an identical bottleneck to compare their sharing, as well as their total link utilization. The 4 starting times of the TCP sessions were identical to the 4 times of the DLCC sessions. TCP is more effective at utilizing the entire bandwidth because it is much more reactive. The round trip time was approximately 60ms, meaning that every 60ms a TCP session could attempt an increase, utilizing on average 97.6% of the

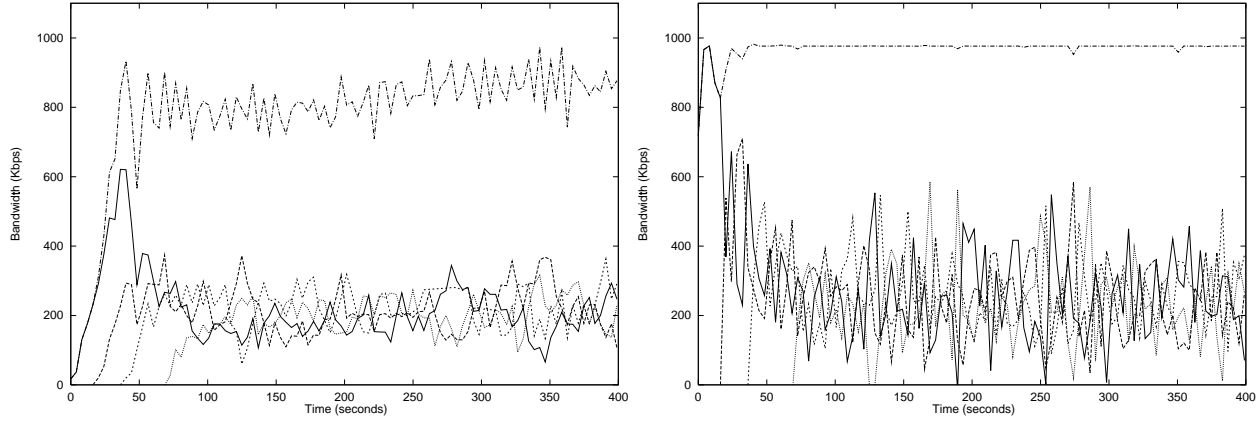
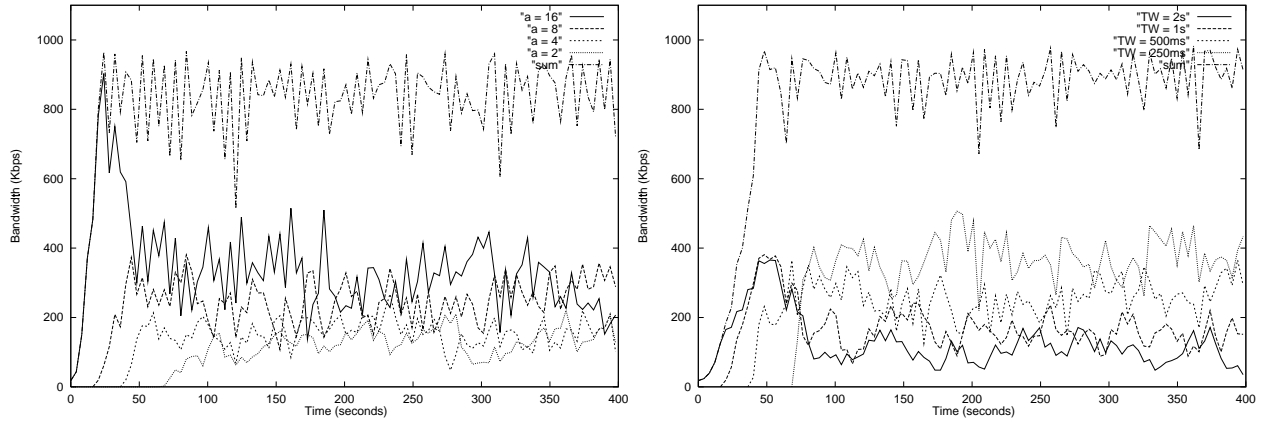


Figure 3.5: DLCC vs. DLCC and TCP vs. TCP. The bandwidth of four identically parameterized DLCC sessions is shown on the left, with the aggregate bandwidth also graphed on top. These protocol instances had  $a = 4$ ,  $BW = 20\text{Kbps}$ ,  $x = 1.298$  (this corresponds to 15 groups required to reach the bottleneck), and  $TW = 1$  second. For comparison purposes, four TCP sessions compete in an identical network in the graph on the right. The TCP RTT was approximately 60ms.

bottleneck. DLCC's  $TW$  was 1 second, and it used 84.4% of the bottleneck. Bottleneck utilization was measured over the last 200 seconds of the 400 second experiment to allow time for the receivers to join and stabilize in their steady state behavior. All bandwidth measurements in all simulations are taken in this manner, averaged over the second half of the experiment.

In terms of sharing, both TCP and DLCC share the bottleneck equally with similar instances of themselves. TCP's four sessions transmitted at an average rate of 244Kbps, with a range from 234.5 to 254.5Kbps. The DLCC sessions transmitted at an average rate of 211.0Kbps, with a range from 193.9 to 238.3Kbps. Changing the bottleneck bandwidth or other DLCC parameters did not affect the sharing characteristics of the sessions. Identical DLCC sessions share a common bottleneck approximately equally.

Two main factors contribute to DLCC's overall aggressiveness. This is largely determined by how often the receiver attempts an increase. The two parameters that affect this are the

Figure 3.6: Effect of  $a$  and  $TW$  on bandwidth sharing.

aggressiveness factor  $a$ , as well as the time window  $TW$ . Experiments were conducted to look empirically at the affect that these two parameters have on overall bandwidth sharing. Fixing all parameters at their values from the previous experiment, the aggressiveness factor was varied from 16 to 1 to see how the protocol's fair share of bandwidth responded. In another experiment, all values were kept fixed at the previous experiment's values, except the time window, which was varied from 2 seconds to 250 ms. These experiments are displayed graphically in Figure 3.6.

At a high level, we see a less than linear relationship between  $a$  and bandwidth, and a less than linear relationship between  $1/TW$  and bandwidth. Tables 3.3 and 3.4 show the relationship of both  $a$  and  $TW$  to the protocol's share of bandwidth. In order to be exactly linear, we'd expect the flow with twice the  $a$  value to increase twice as frequently (on average) as the control. At identical subscription levels, the expected time until an increase is half as long for a session with twice the  $a$ . As that flow increases its subscription level  $i$ , its expected time until an increase grows, as the inverse sum of  $b_j$  (Inequality 2.8) drops as  $i$  increases. When the higher bandwidth flow receives twice the bandwidth, its expected time to increase

Aggressiveness	Bandwidth (Kbps)	Ratio
2	135.8	NA
4	155.5	1.15
8	258.7	1.66
16	297.1	1.15

Table 3.3: Relationship of  $a$  to bandwidth sharing. The ratio column is the ratio of bandwidth to the bandwidth at the preceding aggressiveness level.

Time Window	Bandwidth (Kbps)	Ratio
2s	106.5	NA
1s	152.5	1.43
500ms	258.0	1.69
250ms	367.5	1.42

Table 3.4: Relationship of  $TW$  to bandwidth sharing. The ratio column is the ratio of bandwidth to the bandwidth at the next longer time window.

is the same as the lower bandwidth, despite its  $a$  being twice as large. This oscillation between half the expected time and the same expected time accounts for bandwidth sharing values being proportional to  $a$ , but less than linear. The same analysis can be applied to two flows whose  $TW$  varies by a factor of two. At the same subscription level, the flow with the smaller  $TW$  receives a new increase flag twice as often, and is expected to increase twice as frequently. The frequency of increase flags drops off as that flow gets to higher bandwidths. This dependency of increase flag frequency on bandwidth accounts for the less than linear relationship.

The base bandwidth  $BW$  does not have a significant effect on the total bandwidth. It affects the absolute amount of bandwidth at a given subscription level, but does not alter the frequency at which the receiver attempts an increase. It is for this reason that  $BW$  does not affect the session's share of bandwidth. When multiple receivers are competing against each other, one of them is constantly bumping up against the bottleneck, causing loss in some or all of the other receivers. This happens frequently, especially as the number of

sessions grows. This means that the current level of a session is not relevant in determining its bandwidth, as it is constantly being knocked down. What is important is the frequency at which it attempts to increase, determined by  $a$  and  $TW$ , not the absolute bandwidth levels. Multiple experiments across a range of session parameters and bottlenecks did not discover a correlation between base bandwidth and share of the bottleneck. In fact, shares that rose and then fell again as the base bandwidth increased were observed in some simulations. Similarly, the layering factor  $x$  didn't have a consistent effect on the share of bandwidth for a particular session. This is likely due to increases causing only brief bursts, which overflow the bottleneck and crash down to their previous levels. As such, the amount by which the receiver increases is not as relevant to the bandwidth share, as it will inevitably drop back down. In some cases, raising  $x$  does raise the share of bandwidth, because increasing by a large factor fills the router buffers with packets from the large  $x$  flow, and that increases the flow's bandwidth until it drops back down. This effect is not consistent, as the steady state bandwidth is much more influential, and depends on the relationship of the specific layers' bandwidth to the bottleneck bandwidth, and not on  $x$ . Most commonly, a bandwidth share that both increases and decreases as  $x$  becomes larger was observed, showing no consistent relationship between the two values.

### 3.7 DLCC vs. TCP

DLCC must also compete fairly against TCP. Unfortunately, not all TCP sessions are equally aggressive. TCP's bandwidth share is given by the TCP response function [16]. Making a few simplifying assumptions, such as a constant packet size (for both DLCC and TCP), as

well as a TCP retransmit timeout  $t_{RTO} = 4 * RTT$ , we can write the total bandwidth  $T$  as proportional to the round-trip time  $RTT$  and packet loss rate  $p$ . The preceding definition of  $t_{RTO}$  is a heuristic which approximates TCP's behavior well [10]. These assumptions allow us to write  $T$  as follows:

$$T \propto \frac{1}{RTT \left( \sqrt{\frac{2p}{3}} + (12\sqrt{\frac{3p}{8}})p(1 + 32p^2) \right)} \quad (3.2)$$

From the TCP response function, we see that TCP is inversely proportional to the round trip time. This is true because every round trip time the sending rate is increased by one packet per round trip time. A longer round trip time means more infrequent rate increases, and the bandwidth by which TCP increases (1 packet /  $RTT$ ) is smaller.

With DLCC, one can pick a TCP RTT and fine tune DLCC's parameters to compete fairly with it. Since TCP RTTs are never known in advance, it is important to see how the sharing properties change as the actual RTT deviates from what we pre-programmed the DLCC session to compete fairly against. Simulations were run to test the effect of a varying RTT on TCP vs. DLCC sharing. A DLCC instance which competes fairly against TCP with a 100ms RTT was hand picked. The DLCC session we choose happens to have  $TW = 1s$ ,  $x = 1.298$ ,  $BW = 20Kbps$ , and  $a = 26$ . The aggressiveness factor was hand tuned after the other parameters were set in order to compete fairly against TCP. In these experiments, the *sharing ratio* is defined as the bandwidth of DLCC divided by the bandwidth of TCP. The sharing ratio of this DLCC session with a TCP session with an RTT of 100ms is 1.05. A plot of the sharing ratio between this DLCC session and TCP's with varying RTTs is shown in Figure 3.7.

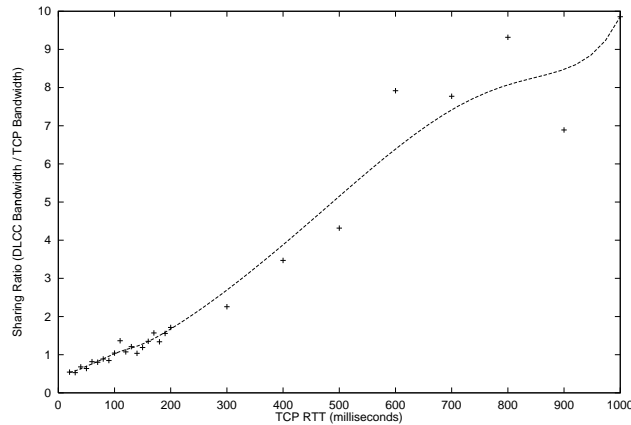


Figure 3.7: Sharing ratio as TCP RTT varies.

As we can see, the sharing ratio versus RTT relationship is roughly linear, as predicted by Equation 3.2. A simpler DLCC model, with only one variable affecting DLCC performance, would be much easier to analyze, and easier to tune against TCP for fairness purposes. In the absence of such a model, it is still useful to analyze the relative performance of DLCC and TCP as TCP's defining characteristic, the RTT, varies. This allows us to pick a DLCC session to compete against a known RTT and roughly predict its performance as the actual RTT differs from the prediction.



# Chapter 4

## Conclusion

This thesis has presented a multicast congestion control protocol we call the Dynamic Layer Congestion Control (DLCC) protocol. This uses a layered approach to multicast, having multiple groups, each transmitting data at varying rates, to provide different subscription levels to the receivers. By joining and leaving groups, the receiver can fine tune its exact subscription level. In particular, this protocol meets the criteria outlined in Section 1.1 for a viable congestion control protocol.

**Scalable** A receiver requires no communication back to the sender to make subscription level decisions. As a result, the sender maintains no state for any of its receivers, and the amount of work required on the sender side does not increase as the number of receivers increases. This makes the protocol usable even when there are many receivers.

**Dynamic** The protocol, by both continually monitoring loss and attempting to increase its fair share of bandwidth, is aggressive, yet responsive to network congestion. On each time window edge (when the receiver receives a packet from the next time window),

it checks for loss in the previous time window, and reacts accordingly, performing a multiplicative decrease in received bandwidth if a loss is detected. Assuming a random, uniformly distributed stream of loss events over time over the course of a DLCC session, the expected average response time between a loss and a receiver's reaction to it is  $\frac{TW}{2}$ , where  $TW$  is the length of the time window in seconds. As this is a parameter to the protocol instance, the sessions can be made more or less responsive as desired. In addition, in the absence of a detected loss, receivers will attempt to join the lowest bandwidth unjoined group upon receipt of an increase signal from the sender. This behavior continues indefinitely, as long as no loss is felt. The receiver efficiently uses available bandwidth, greedily increasing its rate as fast as possible, pursuant to the fairness rules of the protocol. The only limit on the maximum rate receivable is the limit the sender sets by the number and bandwidths of the groups it is sending data to. This is presumably dependent on the specific application, or available sender resources, and is set sufficiently high to fulfill the requirements of the specific application. Most importantly, the responsiveness does not depend on the leave latency.

**Uncoordinated** Receivers coordinate with the sender only by receiving messages from it, without any feedback to the sender. In addition, receivers do not communicate with each other, and receivers can drop out of the session, or join at any time. In fact, the sender has no knowledge of who, if anyone, is joined to any of its multicast groups. Despite being uncoordinated, the receivers behind a common bottleneck will all synchronize behind that bottleneck, that is, they will all be at the same subscription level.

**Fair** With respect to protocol fairness, we have shown how the protocol behaves when it competes against other instances of itself, and how it competes against instances of TCP. DLCC can be made to share a link equally with a TCP session with a given round trip time. In addition, the sharing ratio of DLCC to TCP is roughly linearly proportional to the TCP RTT.

## 4.1 Future Work

### 4.1.1 Slow Start

DLCC is currently quite sensitive to the bottleneck bandwidth. In the experiments from Section 3.5, bottleneck bandwidths were varied from 500Kbps to 10Mbps. This required changing the aggressiveness factor from 2 for the low bandwidths up to 40 for the 10Mb link, in order to ensure that the receiver reached the bottleneck in approximately the same amount of time. In practice, the bottleneck is not often known, and the receiver needs to ramp up quickly to the bottleneck. TCP accomplishes this by its slow start mechanism, where it doubles its sender window every RTT until loss is first felt. After that time, it increases linearly above that bandwidth rate, and doubles below it. DLCC needs a similar slow start mechanism. This is more complicated for multicast because the mechanism must allow some receivers to be in slow start doubling mode, while others are in steady state additive increase mode. This must all happen while the receivers remain synchronized.

### 4.1.2 Simpler Aggressiveness Model

DLCC currently has many parameters which affect its overall aggressiveness, most notably the time window and aggressiveness factor. A problem with DLCC and the current method of calculating  $b_i$  is that it leaves the algorithm's aggressiveness dependent on a variety of variables ( $TW$ ,  $a$ ,  $BW$ ,  $x$ ). This makes analyzing its performance relative to TCP complicated, as many factors affect DLCC's aggressiveness. The exact mathematical model behind the behavior of the algorithm with respect to its parameters is not known, and can be estimated roughly from looking at simulation results. A model with only one parameter affecting overall fair share bandwidth would be much simpler, and would allow us to easily pick a TCP RTT against which DLCC would share a bottleneck link equally. This model would make changes to DLCC parameters, such as  $a$  and  $TW$ , not affect the overall fair share of bandwidth. After such a model is finalized, simulations on more complicated topologies and experiments in real networks could be run and analyzed theoretically.

# Bibliography

- [1] UCB/LBNL/VINT Network Simulator ns (Version 2). Available at <http://www-mash.cs.berkeley.edu/ns/ns.html>.
- [2] J. Blomer, M. Kalfane, M. Karpinski, R. Karp, M. Luby, and D. Zuckerman. An XOR-Based Erasure-Resilient Coding Scheme. *ICSI Technical Report No. TR-95-048*, August 1995.
- [3] T. Brown, S. Sazzad, C. Schroeder, P. Cantrell, and J. Gibson. Packet video for heterogeneous networks using CU-SeeMe. In *Proceedings of IEEE International Conference on Image Processing*, September 1996.
- [4] J. Byers, M. Frumin, G. Horn, M. Luby, M. Mitzenmacher, and A. Roetter. Improved Congestion Control for IP Multicast Using Dynamic Layers. Submitted to *the Second International Workshop on Networked Group Communication*, November 2000.
- [5] J.W. Byers, M. Luby, M. Mitzenmacher, and A. Rege. A Digital Fountain Approach to Reliable Distribution of Bulk Data. In *Proceedings of ACM SIGCOMM*, September 1998.

- [6] D. Chiu and R. Jain. An Analysis of the Increase and Decrease Algorithms for Congestion Avoidance in Computer Networks. *Computer Networks and ISDN Systems*, June 1989.
- [7] H. Eriksson. MBONE: The multicast backbone. *Communications of the ACM*, August 1994.
- [8] W. Fenner. Internet Group Management Protocol, Version 2. RFC2236, Available at <http://www.ietf.org/rfc/rfc2236.txt>.
- [9] S. Floyd and K. Fall. Promoting the Use of End-to-End Congestion Control in the Internet. *IEEE/ACM Transactions on Networking*, August 1999.
- [10] S. Floyd, M. Handley, J. Padhye, and J. Widmer. Equation-Based Congestion Control for Unicast Applications. Accepted to SIGCOMM 2000.
- [11] S. Floyd and V. Jacobson. Random Early Detection gateways for Congestion Avoidance. *IEEE/ACM Transactions on Networking*, August 1993.
- [12] D. Hoffman and M. Speer. Hierarchical video distribution over internet-style networks. In *Proceedings of IEEE International Conference on Image Processing*, September 1996.
- [13] Internet Engineering Task Force. <http://www.ietf.org>.
- [14] M. Luby, M. Mitzenmacher, A. Shokrollahi, D. Spielman, and V. Stemann. Practical Loss-Resilient Codes. In *Proceedings of the 29<sup>th</sup> ACM Symposium on Theory of Computing*, 1997.

- [15] S. McCanne, V. Jacobson, and M. Vetterli. Receiver-driven Layered Multicast. In *Proceedings of ACM SIGCOMM*, August 1996.
- [16] J. Padhye, V. Firoiu, D. Towsley, and J. Kurose. Modeling TCP Throughput: A Simple Model and its Emperical Validation. *SIGCOMM Symposium on Communications Architectures and Protocols*, August 1998.
- [17] Sheldon M. Ross. *Introductory Statistics*. McGraw Hill, 1996.
- [18] N. Shacham. Multipoint communication by hierarchically encoded data. In *Proceedings of IEEE INFOCOMM*, May 1992.
- [19] D. Taubman and A. Zakhor. Multi-rate 3-D subband coding of video. In *IEEE Transactions of Image Processing*, September 1994.
- [20] L. Vicisano, L. Rizzo, and J. Crowcroft. TCP-like congestion control for layered multi-cast data transfer. In *Proceedings of IEEE INFOCOMM*, March 1998.