END-TO-END
MULTICAST CONGESTION CONTROL AND AVOIDANCE

By

Jiang Li

A Thesis Submitted to the Graduate

Faculty of Rensselaer Polytechnic Institute

in Partial Fulfillment of the

Requirements for the Degree of

DOCTOR OF PHILOSOPHY

Major Subject: Computer Science

Approved by the
Examining Committee:

Shivkumar Kalyanaraman, Thesis Adviser

Bulent Yener, Member

Christopher Carothers, Member

Koushik Kar, Member

Rensselaer Polytechnic Institute
Troy, New York

July 2003
(For Graduation August 2003)

END-TO-END
MULTICAST CONGESTION CONTROL AND AVOIDANCE

By

Jiang Li

An Abstract of a Thesis Submitted to the Graduate

Faculty of Rensselaer Polytechnic Institute

in Partial Fulfillment of the

Requirements for the Degree of

DOCTOR OF PHILOSOPHY

Major Subject: Computer Science

The original of the complete thesis is on file
in the Rensselaer Polytechnic Institute Library

Examining Committee:

Shivkumar Kalyanaraman, Thesis Adviser
Bulent Yener, Member
Christopher Carothers, Member
Koushik Kar, Member

Rensselaer Polytechnic Institute
Troy, New York

July 2003
(For Graduation August 2003)

CONTENTS

LIST OF TABLES

LIST OF FIGURES

ABSTRACT

IP multicast was first proposed by Steve Deering in 1989 in RFC 1112 [26]. In IP multicast, data is sent from one point to multiple points simultaneously. Routers duplicate the data when they need to forward the packets in multiple directions. Consequently, for various applications, e.g. video conference, multimedia broadcasting etc, IP multicast is more efficient than point-to-point unicast prevalent today. However, over a decade has passed, IP multicast still has not been deployed in the Internet. Congestion is one of the most important problems impeding the extensive application of multicast.

There are several major challenges in multicast congestion management. They are

- **Scalability** – A multicast group usually has multiple receivers. If the receiver population is large, whether the congestion management protocol can still work smoothly is a big concern. There are several factors that can limit the scalability of a certain protocol:
  - **Feedback Traffic** – The feedback packets sent from those nodes detecting congestion to those managing congestion.
  - **Computation Complexity** – The time needed by nodes to process congestion-related information in order to manage congestion.
  - **State Requirement** – The memory required to store operation states of congestion management algorithms on nodes.
- **Inter-session Fairness** – Different multicast sessions and unicast sessions should be fair in terms of the average throughput they get. A multicast flow in a best effort network should not use too high or too low bandwidth compared with other flows. There are two potential dangers:
  - **Drop-To-Zero** – A problem that a multicast flow gets extremely low throughput because of reacting to congestion more than necessary.
  - **TCP-Unfriendliness** – A problem that a multicast flow is more aggressive than competing TCP flows and cause them to get much lower throughput than expected.

- **Inter-receiver Fairness** – Because the network is often heterogeneous, different receivers may have different desirable throughput. It is the inter-receiver fairness requirement that the sending rate of a multicast session satisfies faster receivers without overwhelming slower ones at the same time.

Given the fact that only end-to-end congestion control protocols can be deployed in the Internet nowadays and in the future [96, 11, 28], we tackle the challenges above based on end-to-end assumptions, i.e. congestion control mechanisms are only deployed on end systems (source and destinations). Consequently, the only network nodes that we can directly control are sources and receivers. This makes the problem even more challenging because we do *not* have the *timely and exact* congestion information (such as buffer length, bottleneck bandwidth) directly from inside the network. Instead, we have to derive the needed information *approximately* from the received packet pattern. We study the following situations with different restrictions and provide better solutions than previous work.

- No support particular for congestion control is provided by receivers.

- There is support from receiver side, but only one multicast group is allowed for a multicast session.

- There is support from receiver side, and unlimited multicast groups are allowed for a multicast session.

- Congestion avoidance for the second situation above.

We first study the situation where no special support is provided at receiver side. That is, receivers have facilities of multicast transport protocols (without built-in congestion control functions) such as receiving data and monitoring their quality, but they do not have any designs specific for congestion control purposes such as measuring available bandwidth. For this scenario we propose LE-SBCC. The source leverages the ACK or NAK common in many transport protocols to derive implicit congestion information, and filter them properly before using them for rate adaptation. Since there is no special support from receivers, this is a single-rate scheme in which all receivers get data at the same rate. The computation complexity and state requirement are both $O(M)$ where M is the number of receivers in the multicast session experiencing congestion ($M \leq N$ where N is the total number of receivers).

LE-SBCC achieves good inter-session fairness (better than previous work), but does not have satisfactory scalability and inter-receiver fairness due to the scarcity of receiver support.

For the second situation, ORMCC is the solution. ORMCC uses a unique mechanism to locate the most congested receiver, which does not depend on the TCP throughput formula as other similar protocols do. The source adjusts transmit rate in accordance with the congestion information from this receiver. Since ORMCC eliminates the need to measure the RTT between the source and all receivers while having an effective feedback suppression mechanism, the feedback traffic volume remains approximately constant without regard to the total number of receivers when the network is stable. The computation complexity and state requirement of both source and receiver side are $O(1)$. As shown by simulations, ORMCC achieves more TCP-friendly throughput and incurs less feedback traffic than two best-known schemes of the same kind, PGMCC [94] and TFMCC [114]. It achieves satisfactory scalability and inter-session fairness, but lacks good inter-receiver fairness due to the one-group limitation.

For the third situation, we extend ORMCC into a multi-rate framework and have GMCC. It smoothly combines the fine-granularity control provided by ORMCC with the coarse-granularity control (layer subscription/unsubscription by receivers). While keeping the merits of ORMCC, it improves the inter-receiver fairness at less cost of IGMP traffic and router burden than previous work. As the result, GMCC addresses all three challenges above successfully.

The last situation we study is the congestion avoidance with support from receiver side given only one multicast group. We propose MCA+. While congestion control as a *reactive* measure takes effect when bottleneck queues are full and packets are beginning to be dropped, congestion avoidance as a *proactive* measure responds when backlog is being built up in bottleneck queues. We do not study other two situations. On one hand, it is impossible to do congestion avoidance without special receiver support; on the other hand, we can extend MCA+ to do multi-rate as we extend ORMCC to GMCC.

MCA+ adopts a new congestion detection mechanism based on a new concept called *accumulation*. This mechanism can detect incipient congestion and allow the source to adjust the transmit rate early. As a congestion avoidance scheme, MCA+ incurs shorter average bottleneck queue and achieves higher throughput without necessarily incurring packet loss. To our best knowledge, MCA+ is the first end-to-end multicast congestion avoidance scheme with good scalability.

Last but not least, the rate adaptation modules in all the protocols above can adopt different policies. Although we use AIMD as an example which is good for bulk data transfer, others policies can easily fit in. For instance, TFRC[33] can be used to provide smooth rate for multimedia transfer. In consequence, a large portion of each protocol above is reusable.

We tested the schemes above with simulations. For those expected to scale well (ORMCC and GMCC), we have run large scale simulations with thousands of receivers. We have also implemented LE-SBCC and ORMCC on real systems to show their practical values.

CHAPTER 1
Introduction

This dissertation studies the end-to-end multicast congestion management problem. By only assuming the packet forwarding function on routers according to the IP multicast specification RFC1112 [26] (as TCP assumes IP routers), with only source and receivers involved, we address the following challenges better than existent solutions. These challenges do not exist for unicast congestion control or are different from those in unicast context. They are the scales against which we will check the performance of our solutions.

- **Scalability** – A multicast group usually has multiple receivers. As the size of receiver population increases, the problems emerge. Several factors can degrade the performance of multicast congestion management protocols, or even stop them from working.

  - **Feedback Traffic** – The feedback packets sent from receivers to the source. If a large number of receivers detect congestion simultaneously and all send feedback packets to the source, the links close to the source will be congested by these packets joining together, and prevent the source from getting any feedback, or even crash the network. This is known as the *feedback implosion* problem.
  - **Computation Complexity** – The time needed by nodes to process congestion-related information. If it grows linearly with or faster than the receiver population size, given a very large multicast session, the CPU resource of the nodes will be exhausted and the nodes collapse. On the contrary, if it is constant and independent of the receiver population size, the algorithms can serve very large multicast sessions. Although unicast congestion control also requires low computation complexity, since there is only one receiver, the problem is not serious.
  - **State Requirement** – The memory required to store operation states of congestion management algorithms on nodes. Similar to computation complexity, a constant amount of memory used for multicast sessions of any size is desired for best performance.

- **Inter-session Fairness** – Different multicast sessions and unicast sessions should be fair in terms of the average throughput they get. The bandwidth a multicast flow gets in a best effort network (e.g. the Internet nowadays) should not be too high or too low compared with what other flows get. This requires that multicast congestion management protocols react to congestion information appropriately, no matter whether receivers experience independent or correlated congestion and send back congestion information synchronously or asynchronously. There are two potential dangers otherwise:

  - **Drop-To-Zero** [112] – A problem that a multicast flow gets extremely low throughput or its transmit rate converges to zero over time. This happens because the flow reacts to congestion too often and adjust its transmit rate lower more than necessary.

  - **TCP-Unfriendliness** – A problem that a multicast flow is more aggressive that competing TCP flows and cause them to get much lower throughput than expected. This happens usually because the multicast flow reacts to congestion too less and adjust its transmit rate higher more than necessary. TCP-friendliness [35, 33] is necessary because in today's Internet, TCP is the dominant unicast congestion control mechanism and thus crucial to the robustness of the Internet. Although in multicast research community, there has been no consensus on the exact definition of how TCP-friendly multicast congestion control mechanisms should be, it is widely accepted that the throughput of a multicast flow should be within some bounded area around the hypothetical TCP throughputs along the multicast paths (such as bounded fairness [111]).

- **Inter-receiver Fairness** [64] – In large multicast trees especially those spreading over multiple domains, there are usually many different bottlenecks of various available bandwidth. Receivers behind them thus have different achievable throughput. If each receiver has throughput matching its capability, the maximum inter-receiver fairness is fulfilled. The least requirement of inter-receiver fairness is that all the receivers have the highest possible rate without having the slower ones overwhelmed.

  There are two different approaches to attain different inter-receiver fairness policies.

The *single-rate* approach requires only uses one multicast group for a multicast session. All receivers have the same throughput rate. In consequence, the sending rate has to be adjusted in accordance with the slowest receiver. The *multi-rate* approach allows receivers to differentiate themselves and have nonuniform throughput. It comes with the cost that the designs are usually more complex and more control traffic is introduced.

1.1 Why "Bundling" Unicast Congestion Control Does not Work For Multicast

Given the multicast congestion control problem, a first thought may be that simply putting multiple copies of the same unicast congestion control mechanism together (i.e. "bundling" unicast congestion control) can work.

For example, for single-rate multicast congestion control, it might be considered enough for the source to maintain a congestion window separately for each receiver and follow the smallest window for sending packets at any moment. It is well known that congestion windows are used to estimate the number of outstanding packets (i.e. the packets that have been sent but have not been acknowledged yet). As Basu and Golestani stated in [6], *"...in large multicast groups, determining the number of outstanding packets to j ... is not feasible"* (Here j refers to a receiver), and *"This would involve sending acknowledgments from each receiver j to the sender for some or all packets, ... This procedure does not scale."* Besides, it is also difficult to decide the packet timeout thresholds (a packet is assumed to be lost if it has not been acknowledged for this threshold of time since departure), because of the heterogeneous RTTs between the source and different receivers [6].

NAK (negative acknowledgments) could be an alternative way to maintain congestion windows. Again, Basu and Golestani stated in [6] that *"The essence of a window-based scheme is to provide quick feedback from receivers in order to achieve tight control at the sender over the number of outstanding packets to each receiver."* Since NAKs are only sent when packets are lost, it is not "quick" and breaks "the essence of a window-based scheme". Also, at the losses of NAKs, those windows not getting NAKs will not have signals for adjustments and will have incorrect sizes before later NAKs arrive. Recall that the source regulates the sending rate by checking each window size, even a single erroneously maintained congestion window may incur wrong transmit rate.

In general, if we approach the multicast congestion control problem by simply "bundling" unicast solutions, the amount of feedback traffic, the computation time and the memory for operation state storage will all be multiplied by N (the number of receivers). If N is large, obviously some segments of the control loop will break sooner or later. Most likely, the network is the first to crash because network resource (bandwidth) is more scarce than end system resources (CPU power and memory).

Furthermore, to make a rate-based multicast congestion control protocol friendly to TCP, as shown by Golestani in [39], RTT has to be considered. However, since multicast is based on a tree instead of a path, there is a problem of what RTT to choose. In this dissertation, we decide to maintain TCP-friendliness on the most congested path between the source and receivers (for single-rate control) [1]. Thus the RTT of that path is used for TCP-friendliness. However, since the network conditions changes frequently and the receivers may join and leave at any time, the most congested paths in a multicast tree at different moments are probably different. Hence we need an efficient mechanism to *dynamically* locate the most congested path, which is certainly beyond the consideration of simply "bundling" a set of unicast congestion control solutions together.

1.2 Assumptions

To tackle the multicast congestion management problem, we make some realistic assumptions.

First, we assume SSM (source-specific multicast) [44] model. That is, in a multicast group, there is only one node that can send data to all others. Note however, since SSM is just a special case of the traditional model of multicast defined in RFC1112 [26], our algorithms can also be used for open group multicast (In open group multicast, any node can arbitrarily join and leave a group, and any node can send data to all others). We also assume that a group is opened and closed by its source. Any receiver can join or leave the group freely without informing any other receiver or the source.

Given the fact that the Internet core is hard to change, we make the end-to-end assumptions. That is, we only require that network routers can forward multicast packets

[1] To keep TCP-friendliness on the most congested path is also the approach adopted by many other schemes such as PGMCC [94] and TFMCC [114].

according to the IP multicast protocol RFC 1112 [26] (as they forward unicast packets according to the IP protocol RFC 791 [87]). Moreover, we do not assume any knowledge of the underlying network topology (including the interconnection, bandwidth, latency and buffer size of the links). We do not know the traffic model on any link inside the network, i.e. we do not know the volume of total traffic and the number of flows going through as well as how they change over time. We also do not have or need any routing information. Therefore, from the aspect of our algorithms, the network is a *black box* between the source and the receivers (Figure 1.1).

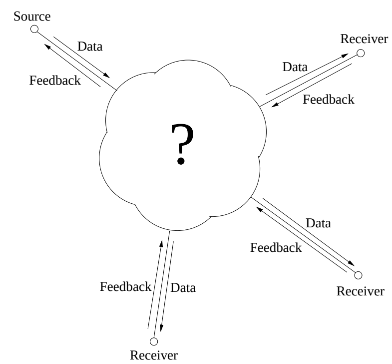


Figure 1.1: Network as A Blackbox
(The multicast congestion control algorithms in this dissertation treat the network between the source and the receivers as a blackbox.)

The only assumption we make about the traffic on the network is that, the traffic is adaptive and generated by such transport layer protocols that control the sending rate according to *additive increase and multiplicative decrease* (AIMD) policy [23]. These protocols are either different flavors of TCP (Tahoe, Reno, New Reno and SACK), or TCP-friendly. However, we do not assume any knowledge of the parameters of the background flows, such as the flavor of TCP, the maximum congestion window size and the estimated round trip time (RTT) etc. In other words, we assume that our multicast congestion control algorithms compete for bandwidth with other algorithms currently dominating the Internet. We also assume no multicast packets traverse the same link twice, and the route that a multicast packet takes is *quasi-static* [2] (i.e. it remains unchanged for at least hundreds of round-trip times).

Finally, our congestion control algorithms only regulate sending rates and do not consider whether the data need to be transmitted reliably or how to ensure reliable data trans-

[2] *quasi-static* is a term borrowed from [40] where it is used to describe congestion control categorization.

mission. If data reliability should be considered, we assume that there is another module dealing with reliability issue which direct all traffic through our congestion control design.

1.3 Abstraction of A Multicast Tree as A Unicast Path for Congestion Management Purposes

A key approach that we use to develop single-rate multicast congestion management solutions is called "most congested path" approach, with which we *dynamically* map the multicast tree in question to the unicast path between the sender and the most congested receiver, i.e. the path with the smallest available bottleneck bandwidth. By conforming the transmit rate of a multicast group to this path in a TCP-friendly manner, the work is simplified. Note that some previous work such as PGMCC [94] and TFMCC [114] also follow this method. The major difference between our work and theirs is the way to dynamically and effectively locate the most congested path as well as how to avoid redundant control traffic. The details will be discussed in Section 1.6 of related work.

We also extend this approach to be used in our multi-rate protocol, which includes an ensemble of single-rate controlled flows in a multi-rate framework. Besides the fact that each single-rate flow follows the most congested path, this approach is also used for judging whether a receiver should subscribe to or unsubscribe from any of these flows. For more details, please refer to Chapter 4.

1.4 Our Contributions

Function placement is an important aspect of networking research. For example, end-to-end arguments [96] proposed to place functions on end systems because *"functions placed at low levels of a system may be redundant or of little value when compared with the cost of providing them at the low level."* [96], and has been guiding the Internet evolvement since its proposal in early 1980's. In this dissertation, we study the multicast congestion control problem by assuming different placements of congestion control mechanisms. Considering the end-to-end principle [96], the placements are on end systems. In consequence, the solutions we develop in this dissertation are not directly comparable with non-end-to-end ones such as those in Kar and Sarkar et. al's work [52, 53, 97, 98, 99].

As the result of following end-to-end approach, the only network nodes that we can directly control are sources and receivers. This makes the problem even more challenging, since we do *not* have the *timely and exact* congestion information (such as buffer length, bottleneck bandwidth) directly from inside the network. Instead, we have to derive the needed information *approximately* from the received packet pattern. We study the following situations with different restrictions and provide better solutions than previous work.

- No support particular for congestion control is provided by receivers.
- There is support from receiver side, but only one multicast group is allowed for a multicast session.
- There is support from receiver side, and unlimited multicast groups are allowed for a multicast session.
- Congestion avoidance for the second situation above.

We first study the situation where no special support is provided at receiver side. That is, receivers have facilities of multicast transport protocols (without built-in congestion control functions) such as receiving data and monitoring their quality, but they do not have any designs specific for congestion control purposes such as measuring available bandwidth. For this scenario we propose LE-SBCC. The source leverages the ACK or NAK common in many transport protocols to derive implicit congestion information, and filter them properly before using them for rate adaptation. To do the filtering, we design a unique filter cascade. This cascade can dynamically and effectively locate the most congested path in a multicast tree, and pass just enough congestion signals for rate adaption. Due to the lack of support from receivers, LE-SBCC is a single-rate scheme. The computation complexity and state requirement are both $O(M)$ where M is the number of receivers in the multicast session experiencing congestion ($M \leq N$ where N is the total number of receivers). But we believe that is the best a single-rate solution can do under such a situation. Besides, since no special support is required from receivers, it is very easy to bind LE-SBCC with other already deployed transport protocols by upgrading only the sources.

For the second situation we propose ORMCC. It uses a novel metric called *Throughput Rate At Congestion* (TRAC) for the sake of selecting the most congested path. By using TRAC, the design eliminates the needs to measure RTT between the source and all receivers (except the slowest one), therefore suppresses this type of redundant control traffic required by other similar protocols. TRAC is also used to suppress feedback from other receivers. As the result, the feedback traffic volume of ORMCC remains approximately constant without regard to the total number of receivers when the network is stable. The computation complexity and state requirement at both source and receiver side are $O(1)$. As shown by simulations, ORMCC achieves more TCP-friendly throughput and incurs less feedback traffic than two best-known schemes of the same kind, PGMCC [94] and TFMCC [114].

For the third situation, we extend ORMCC into a multi-rate framework. By smoothly combining the fine-granularity control with the coarse-granularity control, we have a multi-rate scheme called GMCC. It provides a dynamic set of sub-sessions to receivers within an overall multicast session. In each of these sub-sessions, the sending rate is adjusted using ORMCC-like mechanism independently from other sub-sessions according to network congestion status (fine-granularity control). A receiver can join or leave these sub-sessions to get different sums of throughput rate according to the capacity of the path between the source and itself (coarse-granularity control). It reduces the IGMP join and leave operations dramatically, and avoids redundant layer settings (e.g. number of layers), therefore reduces the control traffic and the burden of intermediate routers. In this scheme, a technique called *probabilistic inter-layer bandwidth shifting* is developed to distinguish *intra-layer congestion* from *inter-layer congestion* and explore hidden available bandwidth. That problem has never been discussed before. In summary, GMCC is a fully adaptive multi-rate protocol with low complexity.

The last situation we study is the congestion avoidance [23] with support from receiver side given only one multicast group. We propose MCA+. While congestion control as a *reactive* measure takes effect when bottleneck queues are full and packets are beginning to be dropped, congestion avoidance as a *proactive* measure responds when backlog is being built up in bottleneck queues. We do not study other two situations. On one hand, it is impossible to do congestion avoidance without special receiver support; on the other hand, we can extend MCA+ to do multi-rate as we extend ORMCC to GMCC.

MCA+ adopts a new congestion detection mechanism based on a new concept called *accumulation*. This mechanism allows incipient congestion to be detected at receiver side in multicast. Once backlog is found to be building up on bottlenecks, the source can get the information from receivers and adjust the transmit rate early. To our best knowledge, MCA+ is the first end-to-end multicast congestion avoidance scheme with good scalability. It incurs shorter average bottleneck queue and achieves higher throughput without necessarily incurring packet loss.

To verify the scalability of our schemes, we have developed a multicast discrete event simulation model in ROSS [19]. With this ROSS model we can easily simulate large scale multicast sessions on personal computers. In this dissertation, simulations of up to 10,000 receivers (each behind a different bottleneck) have been successfully conducted (and we have the potential to include more receivers). By doing this, we have increased the order of magnitude of receiver node number in multicast simulation history by two.

Last but not least, the rate adaptation modules in all the protocols above can adopt any policy. We actually substitute AIMD with TFRC in LE-SBCC and achieve smoother throughput rate. For ORMCC, GMCC and MCA+, although we have not done any such kind of test, due to their modularized design, we believe they also can serve as frameworks for different rate adaptation policies as well as LE-SBCC does.

1.5 Performance Evaluation Methodology

To show the effectiveness of the multicast congestion management solutions developed in this dissertation, we need to evaluate their performance. As we assume the network to be a blackbox, the analytical study of the performance of multicast congestion control schemes is intractable. Consequently, we use simulations and experiments as the major methodology for performance evaluation. However, we do provide some theoretical analysis for several aspects of some schemes after making some simplifying assumptions, e.g. for ORMCC (Chapter 3).

The scales against which we evaluate the performance of our schemes are those challenges listed at the beginning of this chapter. The performance evaluation methods of some aspects may not seem obvious at the first thought. Hence we explain them in the following.

Feedback Traffic – For the schemes with feedback suppression (e.g. ORMCC (Chapter 3)), we count the total number (x_1) of feedback packets from all receivers. We also count the hypothetical number of feedback packets from each receiver if without suppression, and calculate the average value (x_2). If $x_2 = \alpha x_1$, where α is a small number even with a large number of receivers, the multicast feedback traffic volume is comparable to that of a unicast, which indicates the scalability of a feedback mechanism.

Drop-To-Zero – To show that the transmit rate of a multicast session is subject to drop-to-zero problem, we compare it in figures with the rates of unicast sessions that share the same bottlenecks. For example, as in LE-SBCC (Section 2.2.1.1), we use Figure 2.7 (a) (copied below as Figure 1.2) to show the drop-to-zero problem of a multicast session with inappropriate feedback filtering:

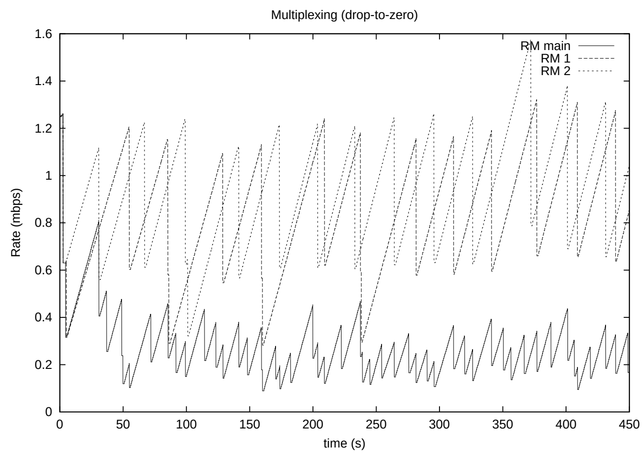


Figure 1.2: Drop-To-Zero Example
(The multicast flow ("RM main") has dramatically lower transmit rate than those of the unicast flows ("RM1" and "RM2").)

As a positive example in which a multicast session is not subject to drop-to-zero problem, the figure (Figure 3.4 (a)) used for ORMCC (Section 3.3.1) is copied below (Figure 1.3).

The topologies we used for testing drop-to-zero problem basically only contain independent bottlenecks. One example is the topology with 10000 receiver nodes for ROSS simulation (Figure 1.4 copied from Figure 3.5). We focus on independent bottlenecks and ignore dependent bottlenecks due to the following reason: Recall that we use "most congested path" approach to tackle the multicast congestion control problem (Section 1.3). To do the abstraction, we need to compare the congestion status of the paths between the source and each receiver. Let X_i denote the congestion level of the path between the source and i-th receiver. Therefore, our objective is to select $\min_{i \in R} X_i$ where R is the set of receivers. Given a subset $R' \subseteq R$, if the receivers in R' are behind a common bottleneck, we have $X_i = X_j, \ \forall i, j \in R', i \neq j$, and our objective is reduced to $\min_{i \in (R-R') \cup \{X_j\}} X_i$ where j is any receiver in R'. As the result, independent bottlenecks present more stressful network conditions.

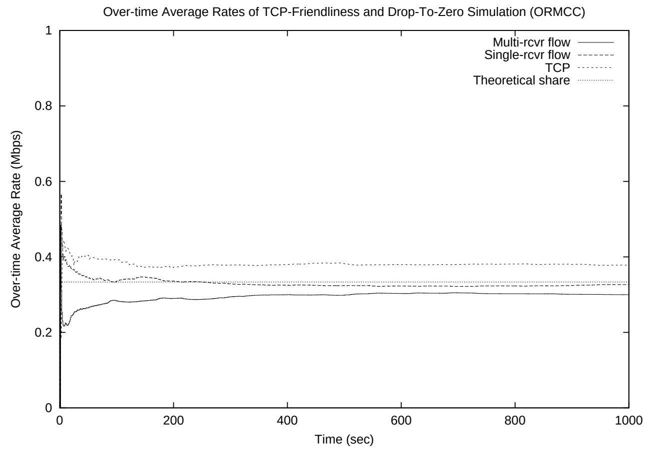


Figure 1.3: Drop-To-Zero Avoidance And TCP-Friendliness Example
(The average rate of the multicast flow ("multi-rcvr flow") is close to that of the unicast flow ("single-rcvr" flow). That is, the multicast flow avoids drop-to-zero problem. Moreover, the average rate of the multicast flow is also close to that of the TCP flow ("TCP"), showing that it is also TCP-friendly.)

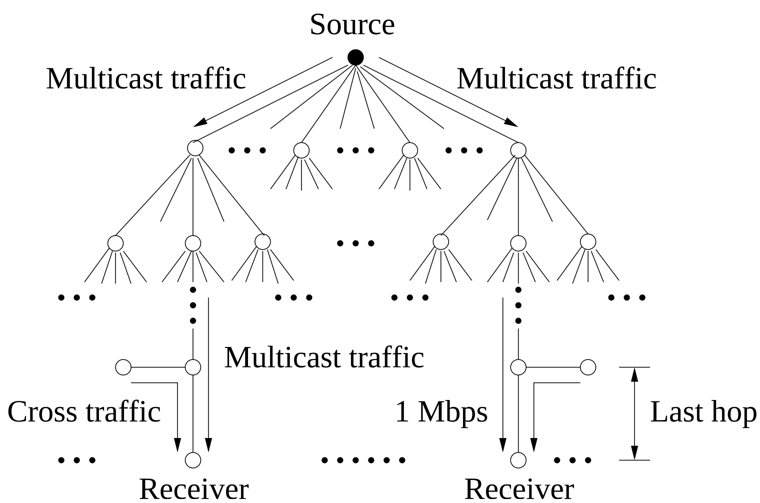


Figure 1.4: Example Topology With Large Number of Independent Bottlenecks
(For multicast congestion control, independent bottlenecks present more stressful situations than common bottlenecks do.)

TCP-Friendliness Similar to drop-to-zero problem, we describe the results in figures. In the drop-to-zero avoidance example figure (Figure 1.3), we can see that the multicast flow is TCP-friendly because it gets approximately the same throughput as TCP gets.

Since it is possible for a multicast session to have a large number of receivers, we deem it important to evaluate the performance of multicast schemes at the presence of many bottlenecks and receivers. Discrete event simulation is a credible way as we can mimic the real networks very closely with it. Therefore, we develop a multicast simulation model in ROSS [19] and run simulations of up to 10000 receivers with each receiver behind a different bottleneck and a couple of crossing flows going through the bottleneck to generate background traffic. It is worthy of mention that before us, the largest multicast discrete event simulation only has several hundred receiver nodes.

1.6 Related Work

Since the proposal of IP multicast in RFC1112 [26], researchers have proposed various solutions for multicast congestion control. In the following, we will first briefly discuss end-to-end unicast congestion control which is the foundation of our work. After that, we will discuss both end-to-end and network supported multicast congestion control solutions. Since end-to-end solutions are similar to our work in this dissertation, we discuss them in more details. For network supported solutions, we will just review them briefly since they are in a different domain.

1.6.1 End-to-End Unicast Congestion Control

Obviously, TCP [45] is the most popular and dominant end-to-end unicast congestion control protocol on the Internet nowadays. To improve its performance, people have proposed various flavors of TCP such as Reno, NewReno, SACK [49], Vegas [15] and Westwood [20]. Other people have also provided generalized end-to-end unicast congestino control algorithms such as MCFC [40] and Binomial [5]. Of them TCP is a special case.

Although the unicast work provides concepts and inspirations of how to preform network congestion control, due to the reasons we discussed in Section 1.1, we cannot easily migrate them to multicast context. Instead, there are some special considerations to be made, as listed at the beginning of this chapter, which is the focus of our study.

1.6.2 End-to-End Single-rate Multicast Congestion Control

Since LE-SBCC and ORMCC are also single-rate, we compare them with some of the well known schemes in this class.

1.6.2.1 DeLucia et. al's Scheme Using Representatives

DeLucia et. al's work in [25] is an early single-rate multicast congestion control scheme using representatives. It requires two types of feedback from receivers, *Congestion Clear (CC)* and *Congestion Indication (CI)*. CC is equivalent to ACK, sent probabilistically, and CI is equivalent to NAK. A fixed number of receiver representatives are maintained at the source. If a representative does not send CI or CC for a while, it is ultimately removed from the set. When the source receives a feedback packet from one of the representatives, it multicasts an echo packet to all the receivers. The echo packets arriving at receivers suppress new feedback packets scheduled to be sent. The source uses only the feedback from representatives to do MIMD (multiplicative increase and multiplicative decrease) rate adaptation. The scheme specifies some priority levels in the processing of CC and CI for the purposes of feedback suppression and representative selection.

In this scheme, the representative set is not guaranteed to include the slowest receiver. Furthermore, it assumes that only a few bottlenecks cause most of the congestion. Based on this assumption, suppression using echoes is the only mechanism for filtering feedback from receivers. All feedback packets arriving at the source are used for rate adjustment. In a heterogeneous network, where there may be many different bottlenecks and asynchronous congestion, the assumption may not be true. Consequently, the transmission rate may be reduced more than necessarily and stay very low or close to zero. That is, this scheme may suffer from the *Drop-To-Zero* problem.

In comparison, LE-SBCC does not maintain any representative and only requires NAK-equivalent feedback from receivers. The rate adaptation policy is AIMD (additive increase and multiplicative decrease). LE-SBCC does not suffer from Drop-to-Zero problem.

ORMCC also maintains a representative, similar to DeLucia's work. However, ORMCC only has at most one representative at any time, while DeLucia's work requires a pool of representatives. Moreover, ORMCC uses explicit rate feedback instead of single-bit feedback. Feedback suppression in ORMCC is not timer-based. That means, in ORMCC, feedback is suppressed as the result of *comparing certain metrics*, while in Delucia's scheme, feedback is suppressed by the *arrivals* of feedback echoes. Finally, ORMCC adopts AIMD instead of MIMD, and does not suffer from Drop-to-Zero.

1.6.2.2 PGMCC

Rizzo's PGMCC [94] is built on top of a reliable multicast transport protocol PGM [106]. At any time it keeps an *acker*, a representative receiver which dictates the source for rate adaptation.

PGMCC requires two types of feedback packets, ACK and NAK. NAKs are sent by all receivers when they detect packet losses, and contain packet loss rate and the highest received sequence number. The source uses the information carried in NAKs to compare the estimated throughput of different paths between the source and receivers. To compute the estimated throughput, PGMCC uses a simplified TCP average throughput formula [85, 76]. Packet loss rates are measured locally by receivers, and RTTs are measured by the source in terms of packets when it receives NAKs. The receiver with the lowest estimated throughput is chosen as the acker.

The source requires ACKs from the acker and does window-based congestion control similar to TCP, while using an extra token mechanism to regulate packet sending. PGMCC itself does not provide any feedback suppression. Instead, it relies on the intermediate nodes (network elements) of the underlying PGM protocol to aggregate feedback packets.

Since the source in PGMCC needs to calculate the estimated throughput for each receiver, its computation complexity is $O(N)$ (N is the number of receivers). Furthermore, receivers need to send feedback continuously to the source for RTT estimation purpose. We can see that PGMCC may not be suitable for very large groups. Also, according to Seada et al's work [100], feedback aggregation can degrade PGMCC performance.

LE-SBCC differs from PGMCC mainly in the following aspects: (a) LE-SBCC is a purely source-based protocol. It does not require receivers to do any measurement except the simplest packet loss detection (which is needed by any congestion control protocols), and only needs NAKs. PGMCC, instead requires more information in feedback packets, e.g. packet loss rate, and it needs two types of feedback, ACK and NAK. (b) LE-SBCC does not maintain any representative. (c) LE-SBCC does not assume any underlying transport protocols.

However, similar to PGMCC, LE-SBCC requires $O(N)$ computation complexity. Feedback aggregation can also pose performance problems to LE-SBCC, as we will show in Chapter 2.

ORMCC only requires NAK-like feedback, making the scheme simpler, though receivers in ORMCC also have to measure the receiving rate and include it in the feedback. ORMCC also uses the concept of representative, but it does not depend on TCP throughput formula to select representative. ORMCC has other major advantages in that (1) ORMCC does not require all receivers to continuously send feedback to the source, (2) The computation complexity at both source and receiver side is $O(1)$, and (3) ORMCC provides feedback suppression by itself, without assuming any underlying transport protocol or network support.

1.6.2.3 TFMCC

Widmer's TFMCC [114] extends equation-based congestion control TFRC [33] to multicast. The source controls the sending rate by setting it to the expected throughput of the *current limiting receiver* (CLR), where CLR is the receiver found to have the lowest expected throughput of the group, similar to acker in PGMCC. Expected throughput of a receiver is calculated at receivers using the full TCP throughput formula [85, 76]. For the calculation, all receivers need to measure packet loss event rate and RTT between the source.

The RTT measurement in TFMCC has the following procedure: Each receiver needs to get an initial RTT measurement by exchanging timestamps with the source. After that, one-way delay from sender to receiver is used to adjust RTT estimation. If one-way delay indicates that RTT may have changed significantly, the receiver exchanges a packet with the source to measure real RTT. Receivers are prioritized when their packets need to be individually echoed by the source for the purpose of RTT measurement. In addition to receiver side RTT measurements, TFMCC also has sender-side RTT measurements, with which the source adjusts a reported expected rate if necessary to avoid oscillation.

TFMCC provides feedback suppression. When a receiver has a new calculation of expected rate, it starts a timer of random expiration time, calculated as a function of estimated upper bound on the number of receivers. If before the timer expires, an echo from the source arrived with the current sending rate, the receiver compares its own expected rate with the current sending rate. If less, a feedback packet is sent, otherwise, it is suppressed. On the other hand, if no echo from the source arrives before the timer expires, feedback is sent.

Still, TFMCC requires receivers to measure RTT, which has some negative effects: (a) The sender needs to send echoes to receivers individually. Let alone the relative complexity (e.g. the echo precedence), the processing overhead of the sender increases constantly with the group size. (b) Although with the one-way delay adjustment mechanism, in a dynamic network environment, receivers may need to send feedback to the source for the sake of RTT measurements relatively frequently. In consequence, traffic going toward the source may grow as the number of receivers increases.

TFMCC's feedback suppression is an enhanced version of Fuhrmann's work [37]. It needs an estimated value of maximal receiver number. The number can be either estimated by the mechanisms such as those in [36, 69], which introduces more complexity to the scheme, or has to be guessed. An estimation diverging too much from the real value may degrade the performance of the feedback suppression. Besides, because the suppression is probabilistic timer-based, additional delay is introduced into feedback.

LE-SBCC differs from TFMCC mainly because of its simplest requirement for feedback information and receivers, but TFMCC has advantage of doing feedback suppression.

ORMCC, on the other hand, distinguishes from TFMCC because receivers do not need to measure RTT. Source in ORMCC does not need to do individual packet echoing. Moreover, the feedback suppression in ORMCC is not timer-based. The feedback packets are decided to be sent or not at the moment when congestion is detected, instead of waiting for timer expiration or echo arrival.

1.6.2.4 MDP-CC

In Macker et al's MDP-CC [75], the source collects feedback with loss event rates from receivers, and predict their estimated throughputs again by TCP throughput formula [85, 76]. According to the results, the source selects a pool of slowest receivers as *congestion control representatives* (CCRs). Among the CCRs, the source chooses the slowest one as *worst path representative* (WPR). Then, the source increases/decreases the transmission rate exponentially toward the predicted rate of the WPR. The feedback suppression in MDP-CC is probabilistic timer based.

As shown in that paper [75], maintaining multiple representative candidates requires considerable complexity. The computation of the estimated throughput by the source for all receivers for representative selection restricts its scalability, as we discussed before.

1.6.2.5 LPRF

In Bhattacharyya's LPRF scheme, receivers' feedback is also filtered by the source for rate adaptation, as in LE-SBCC. However, the filter does not guarantee to catch the most congested path and therefore can suffer from performance degradation. In fact, one of LE-SBCC's filter is an improved version of LPRF. More details are covered in Chapter 2 Section 2.1.1.

1.6.2.6 Other Schemes

There are also several other schemes in this class.

Wang et al's scheme [111] proposed an ACK-based and window-based protocol. There is no feedback suppression, and all receivers send congestion signals to the source. Assume there are N receivers, the source accepts each signal with probability $1/N$. Therefore, it needs to estimate the total number of receivers.

In TCP-SMO [67], apart from the multicast session, the source maintains a separate TCP connection to each receiver, and use the aggregate congestion information of all these TCP connections to do congestion control. As mentioned in the paper, TCP-SMO can only be used for up to medium group size with 1000 receivers.

Another scheme SRM-TFRC [117] is designed for SRM [34]. It assumes the old multicast model, i.e. every node can be source in a group. The source collects feedback and calculate "rate states" for each receiver using average TCP throughput formula like TFRC [33] does. A feedback synthesizer calculates the weighted average of all receiver "rate states", and gets a target rate toward which the source adjust the sending rate.

Shi's work [104] is similar to TFMCC. Receivers measure estimated throughput using a simplified TCP throughput formula. The one with the lowest estimation communicate with the source which does AIMD instead of TFRC. RTT estimation problem is left open in this paper.

Ha's work [43] divides time into epochs. In each epoch, the source reacts to the max number of loss notifications by any receiver. However, epoch is not clearly defined here. If epoches are small, the scheme can suffer from drop-to-zero.

Morris' work [80] chooses the receiver with the lowest receiving rate in each period and adjusts the transmit rate according to the minimum reported receiving rate. It could also suffer from drop-to-zero if different receivers have minimum receiving rates alternatively in consecutive periods. Besides, the way it adjusts rate is not TCP-friendly.

Liu's work [70] actually borrows the ideas from LE-SBCC and other work. Similar to TFMCC, receivers use TCP formula or TEAR [92], then the one with lowest estimation sends NAK to the source for rate adapation. It suppresses NAK from receivers who have higher estimations.

Bouras's work [13] fits ATM network or DiffServ network better. The source collects loss rate and delay jitter from receivers, and check their congestion status with these values, calculate the rate that can satisfy receivers the best (with regard to a utility function) for the next step.

1.6.3 End-to-End Multi-rate Multicast Congestion Control

McCanne et al in [77] proposed the first end-to-end multi-rate congestion control scheme for multicast. In RLM, the source sends data over several separate multicast groups (layers). Beginning from the lowest layer, each receiver periodically joins a group of higher layer to explore available bandwidth. If packets are lost after the join, the receiver will leave the group. The bandwiths assigned to layers are static.

RLM is found to be unfriendly to TCP and improved by the RLC scheme [109]. Periodic traffic bursts are generated in RLC for bandwidth inference. At synchronization points, a receiver may join a higher layer depending on the inference. Bandwidth of layers are exponentially distributed, though still statically, to generate exponential rate decrease (like TCP) when receiver leave higher layers due to packet losses.

Still, RLM and RLC sometimes exhibit pathological behaviors, shown by [60]. The authors of that paper proposed PLM, using packet pairs to infer the bottleneck bandwidth and decide which layer to join. Some fair queueing mechanism on router is assumed for assuring the performance of this scheme. Nonetheless, bandwidths allocation of layers are still fixed.

RLS [57] is a work similar to RLC, but with more carefully designed join and leave interval.

In MSC [55], receiver monitor the effect of their joining experiments (using RLM) on all layers and base their join and leave on the observation.

RPLM [83] observed that in sparse mode multicast routing schemes such as PIM-SM [29] and CBT [3], different multicast groups with the same set of nodes may use different multicast trees and fail layered multicast congestion control protocols such as RLM, RLC and so on. It therefore proposed a layered multicast scheme based on rendezvous points (RP), improved old layered schemes by differentiating congestion on common multicast trees from per-RP congestion. Although it fetches rendezvous point information from designated routers (the multicast routers directly connected to receivers), other parts of design are end-to-end.

MLDA [105] is a little different from the schemes above in that receivers measure maximum possible receiving rate by TCP throughput formula [85] and then decide which layer to join. For RTT estimation, it uses frequent one-way measurement complemented by infrequent two-way measurement. In each round, the source collects receiver's estimated throughput and adjust the layer setting. It assumes old multicast model, in which every node in a group can be source, and uses local representatives to alleviate the scalability problem introduced by RTT measurement. Note that in each round, the sending rate is fixed, therefore some adaptability is sacrificed.

HALM [71], is a scheme with fixed number of layers but the layer bandwidth may change dynamically. Receivers estimate throughput rate using the TCP formula and send this information back to the source. The source gets samples from some receivers and calculate the optimal rate allocation for each layer periodically. The RTT measurement procedure in HALM is similar to MLDA. Feedback from receivers is suppressed by using probabilistic timers. The computational complexity of HALM increases dramatically with the number of layers.

There are also several other schemes FLID-DL [16], Fine-Grained Layered Multicast [18], STAIR [17], Wave and Equation Based Rate Control [74]. They emulate rate increase/decrease by requiring receivers to join and leave groups/layers while the source sends data to each group. That means they are closely coupled with routing and IGMP. This coupling leads to some potential problems. For example, these schemes assume that all groups in a multicast session follow the same path, but this may not be true as shown by RPLM [83]. These schemes assume that a receiver leaving a group can make the multicast tree pruned and reduce the traffic on certain paths. However, aggregated multicast trees [30] do not necessarily prune trees dynamically and hence break the assumptions of these schemes. Besides, frequent group joins and leaves can introduce significant contral-plane load at routers in the entire tree.

In Wu's work [115], receivers measure the difference between expected throughput and actual throughput (similar to Vegas) and join or leave based on the difference. Two other papers [93] and [58] proposed that receivers estimate their throughput using TCP throughput formula and decide which layers to join or leave. The sending rates in the layers in all these three schemes are fixed.

In SAMM [118], receivers estimate available bandwidth and send it toward the source. The reports are aggregated by the router (in network-supported scheme) or by source only (end-to-end scheme), then the source adjusts sending rate and layer number. Between the adjustments, the sending rates in the layer are fixed. This paper does not discuss how to dictate join and leave operations of receivers.

Jiang et al's scheme [48] uses a concept different from other end-to-end multi-rate schemes. In this scheme, receivers are split into two *disjoint* groups, V-group and B-group (note that in other schemes receivers join groups cumulatively). The source sends data to B-group at the lowest possible rate allowed by the application, and it sends data to V-group at a variable but higher rate. The source polls the whole group and gets estimated "isolated loss rate" (expected max-min share of bottleneck bandwidth) from receivers. It then calculates the rate that can lead to max "inter-receiver fairness" for V-group receivers. Receivers unable to receive data at B-group rate are pruned. Other receivers switch between the two groups according to their maximum receiving rates. This scheme cannot have many groups because otherwise there will be great redundancy of traffic.

Similarly, Cheung's work [21] also use separate groups. Receivers try different groups when they receive invitation from the source. On the other hand, the source polls receivers periodically, and adjusts sending rate in each group according to how many receivers in each group are "unloaded/loaded/congested".

Gu-In Kwon and John Byers' scheme SMCC [59] is similar to our multi-rate scheme GMCC, using single-rate multicast congestion control (in particular, TFMCC) to simplify the task of multi-rate control. However, it has certain limit on the rate adaptation within each layer, and therefore is not fully adaptive. More detailed discussion has been included in the introduction part of GMCC chapter 4.

1.6.4 Network Supported Single-rate Multicast Congestion Control

Unlike end-to-end schemes, network supported algorithms need support from routers or nodes other than source and receiver.

Siu's work [108] is an early single-rate solution for ATM network. It extended unicast congestion control functions on switches to support multicast and preserve max-min fairness characteristics.

Sedano's work [101] is based on active networks where functions can be added to router using active service [107]. It does hop-by-hop congestion control. A receiver sends calculated "proper rate" (at congestion) and ACK (at packet arrival) to upstream routers toward the source. Each router on the path adjusts its forwarding rate for this flow in accordance with the reports.

Chiu et al's work [22] is a window-based single-rate scheme. It assumes an underlying repair tree [50]. Receivers send ACKs, aggregated by intermediate nodes in the repair tree. Upon receipt of aggregated ACKs, the source adjusts its congestion window. The sending rate is smoothed to avoid the window-related burstiness. In this scheme, receivers unable to receive data at the minimum rate are pruned so that the average throughput of the group can be higher. Another single-rate scheme MTCP [91], assumes a logical tree topology. In this tree, each internal node acts as a receiver and a source at the same time. It aggregates the congestion information from its subtree and does window-based congestion control.

1.6.5 Network Supported Multi-rate Multicast Congestion Control

With network support, some schemes of this class have optimal performance as their goals.

In Sarkar's work [97], for each session/flow going through a link, routers detect its saturation status, i.e. whether this flow is using its bandwidth share fully. According to the saturation status, routers update the control packets exchanged between the source and receivers. The source then update rate according to the information in control packets. Max-min fairness [46] is achieved by this scheme. Another Sarkar's work [99] let routers drop packets if their calculated downstream rates are smaller than incoming rates. It also achieves max-min fairness.

Kar's work [52] and Sarkar's another work[98] are another two papers in which routers regulate multicast session rates. In Kar's work [52], control packets are sent back and forth with "prices" (a measure of congestion) updated by routers. Upon receipt of control packets, routers calculate optimal session rates with an optimization algorithm. In Sarkar's work [98], routers monitor buffered packets for each session in order to decide whether or not to forward a packet for the session. This scheme attains max-min fair rates. Another Kar's paper [53] is an enhanced version of [52], and allows different types of user utilization to be maximized.

MFQ [32] utilizes active queue management on routers. By dropping and queueing packets properly, it can achieve any of the several inter-session fairness such as max-min [46]. The source in MFQ is not adaptive or just changes sending rates with some fixed pattern such as that in FLID-DL [16].

LDMCC [38] works on DiffServ [10] network. It needs support from routers to separate multicast traffic from best-effort traffic, and to provide two-level priority packet marking and dropping to guide receivers to stable optimal subscription levels. Max-min fairness is achieved.

There are also other schemes in this class that do not aim for optimal performance.

SIM in [42] is more or less similar to end-to-end multi-rate congestion control, but it does require the routers to detect congestion and set congestion marks on packets as well as to aggregate receivers' feedback. Using the congestion marks and the aggregated receiving rate in the feedback packets, the source adjust layering settings (called "menu" in this paper) including the layer number and sending rate, and maintains the settings till next adjustment.

Rainbow [119] is the only window-based multi-rate scheme. Each receiver maintains a window of data to request. This window is increased when data packets arrive, and halved when loss is detected. Requests with window size are sent to the source, and are aggregated by routers before reaching the source. The source then sends packets according to the aggregated feedback which has minimum requests.

RLMP [41] requires routers to support two-level priority of packets. When congestion happens. low priority packets are dropped first. If a receiver is in multiple layers, the top layer is of low priority and other layers are of high priority. RLMP uses RLM-like methods for receivers to decide when to join or leave.

Similar to Cheung's work [21], in Ramamurthy's work [90], each receiver joins one of the separate group exclusively so that those with similar metrics of bottleneck bandwidth and RTT are grouped together. However, these groups are formed during the process of building multicast tree. That is, it is combined with multicast routing. In each group there is a single-rate control. In a dynamic best effort network, consistently changing bottlenecks may pose serious problems to this scheme.

An earlier work that also combines routing with congestion control is Shacham's work [102]. It assumes network topology knowledge and bandwidth knowledge, which are used to construct best multicast trees. Fixed rates are assigned each layer, and there is no join and leave by receivers. TopoSense [47] assumes multicast tree knowledge as well. Receivers decide whether to join or leave according to the tree topology information they have.

In a different SAMM [1] (not the one in [118]), every router watches the credit (available buffer space) and allows a packet through if all downstream routers have enough credit. Receivers send receiving rates back. These packets are merged by routers, which calculate the desired number of layers and rates in each layer. The source at the receipt of these reports adjusts the layer setting accordingly.

Legout's work [61] proposed a different routine for allocating bandwidth to multicast flows that is not TCP-friendly. Routers allocate bandwidth to multicast flows as a logarithmic functions of the number of receivers downstream. This paper argued that this can be an incentive to deploy multicast. In this scheme, receivers use protocols like RLM/PLM to join/leave layers.

LVMR [65] organizes a multicast session as a hierarchy with agents. These agents, which can be receivers themselves, collect information from the receivers in subtree and coordinate them for joining and leaving. Like other receiver-driven methods, the source does not provide rate adaptation. Some of the authors of this paper later proposed "layer-based congestion sensitivity" [66]. It makes receivers in different layers respond (join and leave) differently to congestion so as to enforce inter-layer fairness.

Some schemes do not require receiver to join and leave layers to achieve different throughput. Instead, they put functions on routers which control the layers each receiver can have. WRHMCC [95] lets each receiver maintain a window (of packets to be received) and a RTT between the source. The window is maintained in a way similar to TCP. Receivers calculate their estimated rates as the window size divided by RTT, and send them to the source and routers. The source sends at the fastest rate reported, the routers filter packets to regulate the flows to downstream receivers in accordance with the rates reported by receivers, and thus achieve multi-rate. NLM [51] uses the TTL field in IP headers and routers' TTL threshold to let router control the layers. In Nakauchi et al's work [82], there are actually not multiple multicast groups as layers. Instead, packets in different layers are distinguished by some bits in their headers. Routers filter packets belonging to higher layers and request upstream ones to do so (by signaling) in accordance with their capacity. In consequence, each receiver gets different subset of data.

1.7 Dissertation Structure

In Chapter 2, we present the filter cascade design of LE-SBCC. Some simulations and analytical discussion are provided to show the necessity of each filter. Other simulations are used to evaluate its performance such as drop-to-zero avoidance and TCP-friendliness. We also describe the positive results of replacing AIMD rate adaptation with TFRC module. An experiment using Linux implementation of LE-SBCC to show its performance in real networks is included as well.

In Chapter 3, we follow the scheme details with simulations. Along with some basic simulations to verify its performance, we compare ORMCC with two other well known single-rate multicast congestion control PGMCC [94] and TFMCC [114], showing that ORMCC outperforms them under most situations. We also provide emulation results in Emulab [113] and large scale simulation results (of 10000 receivers) in ROSS [19]. Analytical study of some aspects of ORMCC's performance are provided in Appendix C.

In Chapter 4, we compare our scheme with a recently proposed similar work SMCC [59] qualitatively. After defining the metric to measure congestion, we describe how to use it to dictate receivers' join and leave operations. The new technique of *probabilistic inter-layer bandwidth shifting* used to explore hidden available bandwidth is presented there. *ns*-2 simulation results and large scale ROSS simulation results are provided.

In Chapter 5, we begin with the theoretical description of the concept *accumulation*, then develop an end-to-end framework to measure accumulation at receiver side and associate it with congestion. Two flow charts along with detailed description are given to help readers understand the operations of the source and receiver respectively, followed by simulation results.

We conclude our work in Chapter 6 and discuss the future research briefly.

CHAPTER 2

LE-SBCC: Loss-Event Oriented Source-based Multicast Congestion Control

In this chapter, we address the multicast congestion control problem for such a situation:

No support particular for congestion control is provided by receivers.

That means, receivers have facilities of multicast transport protocols (without built-in congestion control functions) such as receiving data and monitoring their quality, but they do not have any designs specific for congestion control purposes such as measuring available bandwidth. One scenario with this situation is that, an administrator installed a multicast transport protocol commonly on all user machines (as receivers) and a multicast server (as source). Later, he/she wants to upgrade these machines for better multicast congestion control but does not have access to all user machines any more. The only choice is to upgrade the server as the source.

We present a single-rate multicast congestion scheme LE-SBCC as the solution. The scheme is *purely source-based* in the sense that it operates on a stream of loss-indications (LIs) from the receivers and does not require any other support from network elements, receivers or in the packet format of underlying multicast transport protocols. This feature distinguishes it from other single-rate schemes like PGMCC [94], Equation-based TFMCC [114, 112], MTCP [91], Kasera et al [54], Golestani et al[39], and our later scheme ORMCC in Chapter 3. These latter schemes require some form of special support at receivers (eg: acker capability, RTT/loss estimation capability), network elements (eg: scheme-specific aggregation capability, active services) or protocol packet-formats (eg: new header fields).

A *purely* source-based scheme can be implemented by a simple upgrade of the source of a multicast transport protocol without touching the receivers as long as they generate minimal congestion feedback (as in the example scenario above). The source-based nature of the scheme does not incur extra congestion-related (S,G) state in network elements and receivers, which can be directly exploited to support multi-source multicast applications (like interactive multi-player games and online military training simulations). Although at the source we maintain some per-receiver state in the worst case, efficient implementations can reduce the average case memory requirement to below 1MB for 10,000-50,000 receivers. This scheme is modular and can allow different control policies like AIMD (for data) or TFRC (for multimedia). This holds tremendous potential in the implementation of multimedia services and applications such as live media streaming, video conferencing and video-on-demand.

The LE-SBCC scheme leverage Bhattacharya et al's LPRF [7, 9]. It differs from LPRF in that it tackles the drop-to-zero problem under high degrees of multiplexing, addresses several implementation issues and has a much faster transient response. We believe that it is the first purely source-based scheme to fully address all components of single-rate source-based multicast congestion control, i.e. *drop-to-zero* issues, *TCP friendliness*, RTT estimation, robustness and scalability up to 10,000 receivers.

2.1 LE-SBCC: Scheme Description

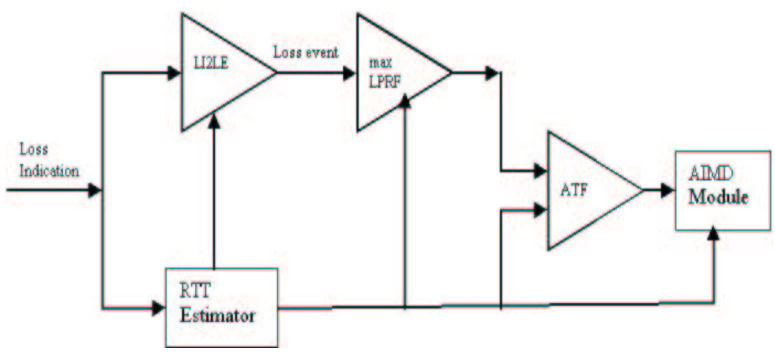


Figure 2.1: Cascaded Filter Model of LE-SBCC

The LE-SBCC scheme consists of a *purely source-based* cascaded set of filters and RTT estimation modules feeding into a rate-based additive increase/multiplicative decrease (AIMD) module (Figure 2.1). These filters, elucidated in section 2.1.1, together transform the multicast tree to appear like a unicast path for the purposes of congestion control. Also, unlike TCP, the scheme is not self-clocked but acts upon a stream of loss indications (LIs) from receivers. The filters are then designed to address the following key problems:

Drop-to-Zero: is the problem of reacting to *more loss indications (LIs) than necessary* leading to a beat-down of the multicast flow's rate[112, 9]. This occurs because the multicast flow receives LIs from multiple paths and may not filter LIs sufficiently.

TCP-unfriendliness: is the problem of reacting to *less LIs than a hypothetical TCP flow would on the worst loss path* [14, 112, 114]. TCP-friendliness is a subject of current debate [33, 94, 114]. In particular, Padhye et al's TCP throughput formula [85] requires the definition and estimation of "loss rate" and "RTT" of the paths. The unicast scheme, TFRC [33] introduced the concept of loss events (LEs) and uses the LE rate instead of the LI rate as the "loss rate." *An LE is a binary number defined every RTT per receiver which is 1 when one or more LIs are generated in that RTT by the receiver, and is 0 otherwise.* Like TFRC [33], this paper will also first derive LEs from LIs before responding to congestion.

RTT estimation: PGMCC and TFMCC [94, 114] have receivers estimate RTT and drive the control based upon paths having the maximum value of $(RTT\sqrt{p_{le}})$, where p_{le} is the *per-packet LE probability*. Our proposed scheme LE-SBCC computes $\max RTT$ and $\max p_{le}$ *concurrently* and uses both these estimates to drive the AIMD module. This decoupled use of $\max RTT$ and $\max p_{le}$ represents a sub-optimal approach in general[94, 114, 112]. However, our RTT-estimation approach tends to measure the $\max RTT$ from the *currently congested sub-tree*, which reduces this suboptimality in practice.

2.1.1 Cascaded Filter Model

Figure 2.1 illustrates the outline of the building blocks of the scheme implemented at the multicast source. The cascade of filters converts a stream of loss indications (LIs) from receivers into per-receiver LEs, filter the LEs further, and finally outputs a stream of rate-reduction indications (RRs) to the rate-based AIMD module (at most one RR per RTT). The AIMD module performs multiplicative rate decrease (MD) upon receipt of a RR and performs additive rate increase (AI) each time a RR is not received within an increase interval. Observe that this closely models the behavior of AIMD control on a *single path* where RRs are triggered by LEs [33]. The modules work as follows:

LI2LE filter: This filter converts per-receiver loss indications (LIs) into per-receiver loss events (LEs). Recall that an LE is a per-receiver binary number which is 1 when one or more LIs are generated per RTT per receiver, and 0 otherwise. The source stores a *per-receiver timestamp* ($T_{LastPassed}$) which records the time when the last LI was

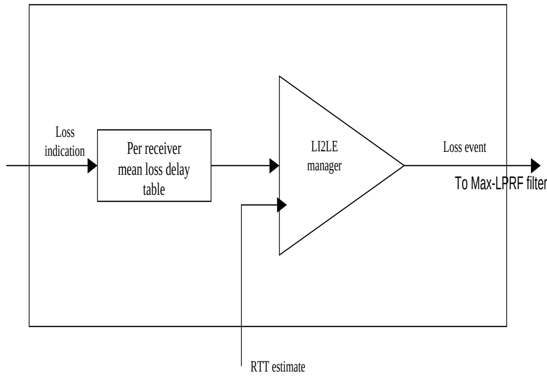


Figure 2.2: Loss Indication to Loss Event Filter (LI2LE)

converted and passed as an LE. If a new LI arrives from the receiver *after* a period $SRTT + 2\sigma$ of[3] $T_{LastPassed}$, it is converted into an LE and passed, and the timestamp $T_{LastPassed}$ is updated to the current time. Else the LIs are filtered. This filter is critical because the rate-based AIMD module is not self-clocked (no acks like TCP, PGMCC [94] or Golestani [39]); which leads to large burst losses with drop-tail queues (see section 2.2.1.3).

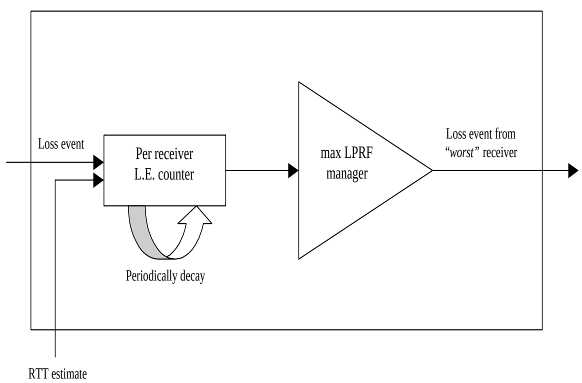


Figure 2.3: Max-LPRF: Max-Linear Proportional Rate Filter

Max-LPRF: The goal of this filter is to pass, on the average, the number of LEs corresponding to what the source would have received from the worst case receiver. This worst-case loss rate estimation problem has been studied earlier in the literature (see [7] and references therein). Our filter is an extension of Bhattacharya et al's Linear Proportional Response Filter[7] (LPRF). The *LPRF* is a filter which passes loss in-

[3] $SRTT$ = average of RTT samples similar to the variable used in TCP

dications (LIs) with a probability $\frac{L_i}{\Sigma_i L_i}$ where L_i is the number of LIs from receiver i.

We found that we could not use the LPRF filter directly because of a number of issues. Firstly, it was originally presented under Markovian model assumptions [7] which do not hold under high multiplexing conditions. Therefore we found that it was susceptible to the drop-to-zero problem. Second, *LPRF* did not quickly adapt to sudden increases in worst-case loss rates. For example, consider a steady state case with N receivers, where each receiver has a loss rate of 1%. If one of the receivers suddenly starts experiencing a loss rate of 10%, then *LPRF* converges to the "worst" receiver very slowly, shown in the following.

> Assume on average n packets are lost by each 1% loss receiver since the status changes, therefore $10n$ packets are lost by the 10% loss receiver. LPRF passes each loss indication (LI) from receiver i with probability α_i, where
>
> $$\alpha_i = \frac{X_i}{\sum\limits_{j=1}^{N} X_j}, \quad i = 1, 2, ..., N \tag{2.1}$$
>
> X_i is the total number of LIs from receiver i. Therefore, the average total number of LIs passed by LPRF under the above situation is,
>
> $$n \sum_{j=1}^{N-1} \frac{n}{(N-1)n + 10n} + 10n \frac{10n}{(N-1)n + 10n} = \frac{N + 99n}{N + 9}$$
>
> If N is very large and n is small, the value of above formula is approximately one, instead of $10n$ which is the packet number loss by the most congested receiver. Only when n is very large can the value converge to $10n$. If the packet loss history is considered, it converges even slower.

Third, the passing probability of Equation (2.1) above does not guarantee passing the number of LIs corresponding to the most congested receiver under all situations. Fourth, *LPRF* works with LIs instead of LEs. Our subsequent arguments show that using LEs is much superior to using LIs. Finally, it does not specify an RTT estimation procedure.

The *Max-LPRF* works as follows: Assuming X_i is the count of LEs from receiver i, this probabilistic filter takes as input all the LEs from receivers $(\Sigma_i X_i)$ and on an average passes the maximum number of LEs from any one receiver (i.e. $\max_i X_i$). In particular, it *passes each LE with a probability* $\frac{\max_i X_i}{\Sigma_i X_i}$. The LE counts per receiver (X_i) are decayed periodically by 10% every 100 $SRTTs$. The O(N) state requirements of both *LI2LE* and *Max-LPRF* are not a big issue because single-rate schemes are typically targeted for a small-medium scale (< 10000 receivers). *Max-LPRF tracks the worst path better than LPRF and is the crucial building block for drop-to-zero avoidance.* It operates on per-receiver LE counts since they differ dramatically from LI counts in drop-tail queueing networks with no self-clocking.

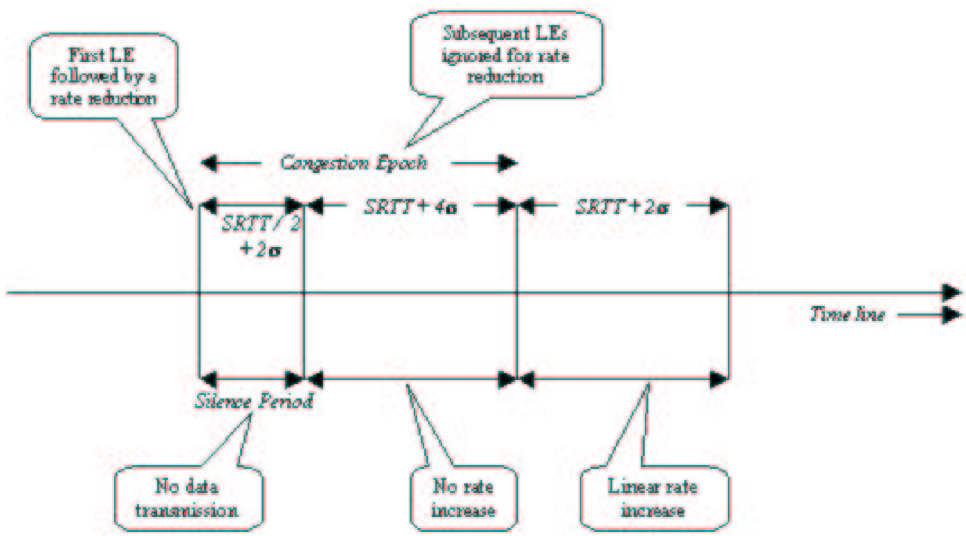


Figure 2.4: Adaptive Time Filter (ATF)

Adaptive Time Filter (ATF): This filter shown in Figure 2.4 simply drops excess LEs passed by Max-LPRF in any RTT to enforce *at most one rate reduction (RR) per* $SRTT + 4\sigma$. In addition, the filter also imposes a *"silence period"* of $\frac{1}{2}(SRTT + 4\sigma)$ when no packets are sent. The goal is to reduce the probability of losing any control traffic or retransmissions during this phase.

RTT Estimation: All filters and the AIMD module need RTT estimates which is fed by the RTT estimation module. It works similar to the TCP timeout procedure i.e. it calculates a smoothed RTT ($SRTT$) and a mean deviation which approximates the standard deviation σ. However the set of samples is pruned to exclude a large fraction of samples which are smaller than $SRTT/2$ (i.e. smaller by an order of magnitude) to bias the average RTT higher. The rate increase uses intervals of length $SRTT + 2\sigma$ while other functions differ in their use of σ as described earlier. Observe that this LI-driven RTT estimation procedure will tend to opportunistically measure the worst RTT of paths which are generating more LIs. That is, the procedure tends to estimate the worst RTT from the *currently congested sub-tree*. Therefore, the decoupling of RTT estimation from worst-case loss rate estimation does not lead to significant suboptimality in practice.

In summary, for ideal operation, the scheme expects: (a) at least one LI per receiver seeing loss per RTT if packets are lost in that RTT (i.e. at least one LE per RTT) and (b) timely generation/forwarding of feedback by receivers/network elements to allow reliable RTT sampling [4]. In general, if expectations (a) and (b) are not satisfied completely or satisfied in an unreliable manner, the scheme performance will degrade. In particular, the scheme is sensitive to arbitrary delays introduced by receivers/network-elements in generating an LI corresponding to a lost packet. Further *LI aggregation* like in PGM [106] itself is a form of filtering which suppresses receiver IDs, timing information and reduces LI or LE counts, thus conflicting with (a) and (b) above[5]. We examine such performance effects in the following sections. The pseudo code for the scheme is presented in Appendix A.

2.1.2 Scalability Discussion

As readers have noticed, in LI2LE filter we maintain a state for each receiver reporting LI (loss indication). Therefore, the number of states grows linearly with the number of loss reporting receivers. However, notice that the set of receivers that experience and thus report packet losses is different from the set of *all* receivers in a multicast session. Let M be the receiver amount of the former kind and N be that of the latter kind. Clearly, $M \leq N$ because all receiver do not necessarily experience packet losses at any moment. Consequently, the number of states we maintain at the source for LI2LE filter (i.e. M) does not necessarily grow linearly with the total number of receivers.

[4]Within these constraints the scheme may be successfully applied to unreliable multicast transport protocols

[5]LIs carry RTT information which affects the RTT estimation, and LEs carry receiver IDs which affects the max-LPRF filter.

Another potential problem is the risk of feedback implosion. Since LE-SBCC is a purely source-based scheme, it does not have control over receiver facility and thus leaves the task of feedback suppression to other parts of multicast transport protocols.

In summary, in terms of scalability, we believe that LE-SBCC has done the best a purely source-based multicast congestion control protocol can do.

2.2 Performance Evaluation

We have evaluated the performance of LE-SBCC to test for Drop-to-Zero avoidance, TCP-friendliness and LI aggregation effects. We use the following methods to test our scheme:

1. Simulations to observe the detailed scheme dynamics and background TCP dynamics for tens of receivers, and to explore the scheme performance for up to 10000 receivers.

2. Simple Markov chain based modeling to obtain a better understanding the choice of LEs as opposed to LIs in our scheme.

3. Linux-based implementation/experimentation (high multiplexing degrees and up to tens of receivers) to understand implementation issues and test performance on a real network.

2.2.1 Evaluation: Drop-to-Zero Avoidance

Recall that Drop-to-Zero Avoidance is the problem of reacting to more loss indications (LIs) than necessary leading to a beat-down of the multicast flow's rate. Drop-to-Zero problem occurs in the following scenarios :

1. High multiplexing, where many flows share a common bottleneck; and the multicast flow receives independent feedback from several such paths. In such a case, the loss rates observed at the source are independent of sending rate, and the multicast flow receives feedback from several paths which requires filtering.

2. Large receiver sets with heterogeneous loss probabilities for different receivers result in a huge number of LIs. It becomes important that the scheme does not react to more LIs than that generated by the worst-loss receiver.

3. LI aggregation leads to degradation of loss and timing information as some LIs are suppressed by the intermediate aggregator. Aggregation effects are discussed in a later section.

We have analyzed the drop-to-zero issue using simulations and Markov Chain based theoretical analysis.

2.2.1.1 Simulations Illustrating Drop-to-Zero Avoidance

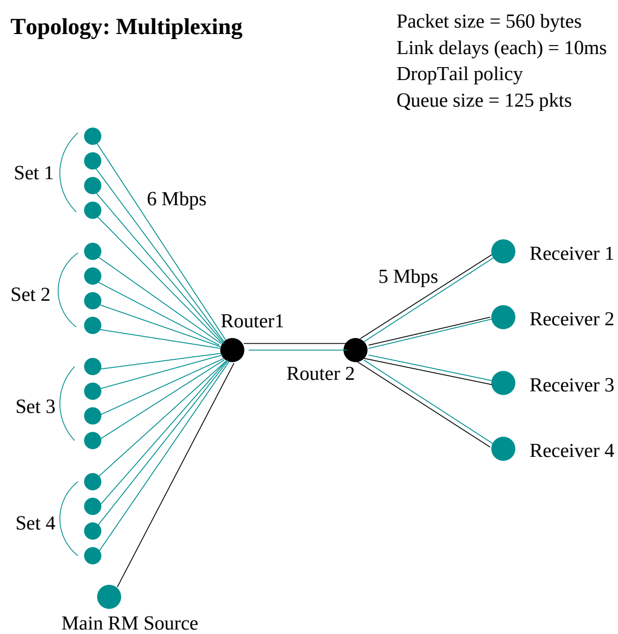


Figure 2.5: Topology to Test Drop-to-Zero Avoidance and TCP-Friendliness

Consider a set of reliable multicast (RM) flows that goes to receiver 1 through receiver 4 as shown in Figure 2.5. The capacity of the links connecting the sources (on the left) to "Router 1" are 6 Mbps each. The link from "Router 1" to "Router 2" has 6 Mbps for each flow (totally 102Mbps). There is a *Main RM* source which sends data to receivers 1 through 4. The links from "Router 2" to each of the receivers are bottlenecks at 5 Mbps each. A set of 4 background unicast flows implementing the LE-SBCC algorithm compete on each of the 4 bottlenecks links. The *unicast* flows in "Set 1" through "Set 4" compete on the bottleneck links to "Router 2-Receiver 1" through "Router 2-Receiver 4" respectively. The fair rate for all flows is therefore 1 Mbps. The buffer size (125 pkts) is roughly twice the bandwidth times fixed delays. The packet size is 560 bytes. This topology tests the performance of the

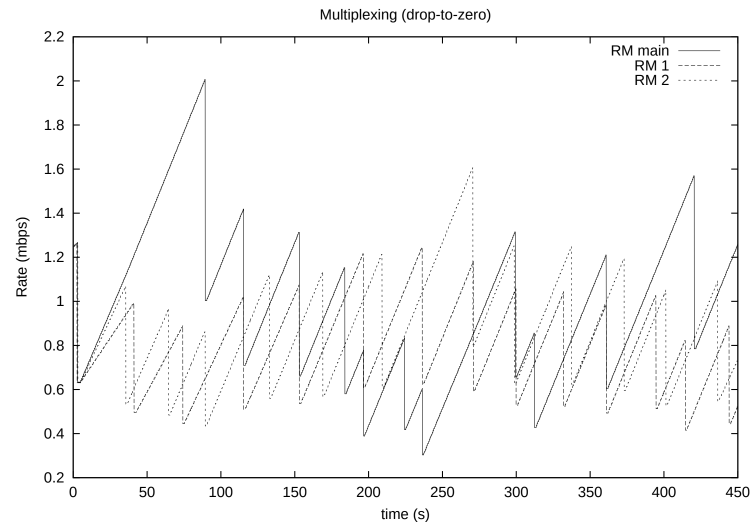


Figure 2.6: Drop-to-Zero Avoidance Results
(LE-SBCC competes fairly with unicast background flows and does not suffer from drop-to-zero problem.)

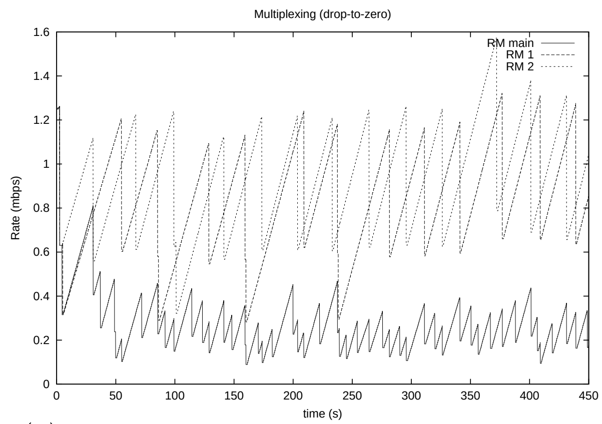
multicast congestion control scheme under independent loss rates on paths, and reasonable degrees of multiplexing on each path. In effect, the loss rate on such congested paths is less dependent on the rate change of any single flow on the path (especially the multicast flow).

Figure 2.6 (rate graph) shows that the multicast flow rate (solid line) competes fairly with a selected subset of unicast background flows (dotted lines) and the average rate is close to 1 Mbps, the fair share. More importantly, the multicast flow does not get beaten down because of the independent loss indications received from each path. This is a non-trivial illustration of drop-to-zero avoidance. Observe that the same topology also illustrates the basic applicability of LE-SBCC to multi-sender multicast, because the competing unicast flows use the same congestion control scheme. Each sender's multicast traffic is controlled as if it were the only sender.

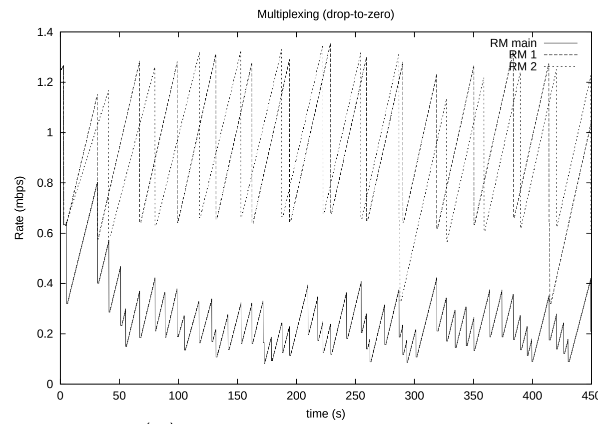
The analysis of the number of packets transmitted, the number of LIs and LEs and rate reductions (RRs) also illustrates the impact of the cascaded set of filters. In particular, the main RM flow transmitted a total of 167762 pkts in 450s and received 949 LIs. After LI2LE filtering, there were only 32 LEs, a dramatic reduction! The Max-LPRF and ATF further filter LEs to result in a total of 12 rate-reductions. When we go back to the graph in Figure 2.6 and count the number of rate-reductions in multicast as well as unicast flows, we observe that all flows have 12 rate-reductions. This is another measure of the fairness of the scheme to AIMD-based background traffic. The issue of TCP fairness is further explored in section 2.2.2.

As discussed in the next section, the dramatic difference between the number of LIs and LEs is because of bursty losses caused by the lack of self-clocking (unlike TCP), rate-based nature of control and the use of sizable drop-tail buffers.

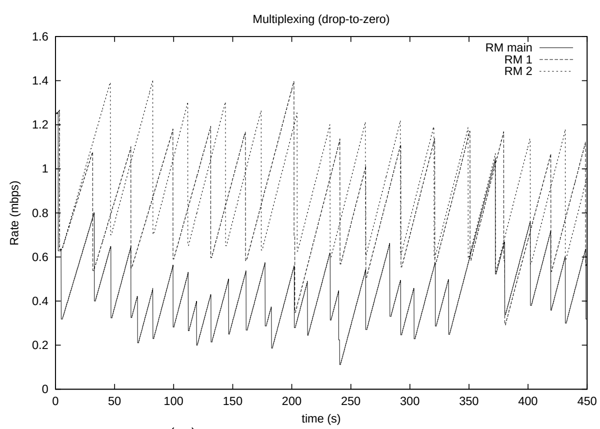
2.2.1.2 Effects of Removing a Filter From the Cascade



(a) Removing both LI2LE and maxLPRF



(b) Removing maxLPRF



(c) Removing LI2LE

Figure 2.7: Effects of Removing A Subset of Filters from The LE-SBCC Cascade
(Removing any of the filters results in performance degradation.)

A natural question is the relative importance of each filter in the LE-SBCC's cascade of filters. To further demonstrate the important role that each filter plays in our scheme, we repeat the simulations for the configuration shown in Figure 2.5 for three different cases:

(a) Removing both *LI2LE* and *maxLPRF* filters

(b) Removing only the *maxLPRF* filter, and

(c) Removing only the *LI2LE* filter.

In all these experiments all other parameters are kept constant as before. The results are illustrated in Figure 2.7. Clearly, we observe the drop-to-zero problem to some degree in all cases (solid curves always below dotted curves). The results are also affected by the fact that in some cases the max-LPRF operates on LIs. In these cases, the max-LPRF processes a larger absolute population of samples, and deals with a different relative distribution of counts from receivers (which affects the critical ratio: $\frac{\max_i X_i}{\Sigma_i X_i}$). Thus each filter in the cascade has a key role in filtering of LIs and avoiding the drop-to-zero problem.

2.2.1.3 Theoretical Analysis of LE vs LI Probability

Recall that our scheme uses LEs instead of LIs for the max-LPRF filter and ultimately in rate reduction decisions. Therefore, our scheme is critically dependent on the number of LEs passed by filters at the source. Thus it becomes important to study the relationship between LIs and LEs to get an understanding of when it is really important to use LEs instead of LIs.

This section presents a simple theoretical analysis to show that using LEs in general leads to a reduced probability of drop-to-zero because the number of LEs is always less than the number of LIs. Moreover, under bursty loss conditions, the number of LEs is significantly less than the number of LIs. This justifies the presence of LI to LE filter used in the cascade. Also, observe that the ratio $\frac{\max_i X_i}{\Sigma_i X_i}$ (used in MaxLPRF) could be very different if X_i represented the count of LEs vs if it represented the count of LIs. These two reasons justify the use of LEs instead of LIs.

Consider the following Markov Chain model for unicast transmission. Assume that the RTT is constant and that the system moves in cycles of RTT. In each RTT, the source transmits with a constant rate, and may experience one or more packet losses. At the end of each RTT, it changes its rate based upon the AIMD policy and depending upon the receipt of LEs. Therefore the system may be modeled as a Markov chain with N discrete states (from 1 to N), where the state x specifies the number of packets sent during that RTT. Therefore, $x = Rate(x) \times RTT$. The process starts in state 1, and may go up to the maximum allowable rate (which corresponds to state N). When a loss event (LE) is encountered within an RTT, the system transitions to state $\lfloor x/2 \rfloor$. When there is no loss event in an RTT, the system moves to state $x+1$. This Markov chain model is illustrated in Figure 2.8. The goal of this Markov chain is to estimate the ratio of per-packet LI probability (p_{li}) to per-packet LE probability (p_{le}), i.e., $E(p_{li})/E(p_{le})$. We analyze this chain for two cases: the non-bursty and bursty packet loss case.

Non-bursty Loss Case

We assume that the packet loss probability is uniform. Define p: the probability that a packet is lost, and a: 1 - p. In this case $p_{li} = p$. In state x, the probability of having at least one loss is $1 - a^x$. The LE probability (p_{le}) is thus $\sum s(x)(1-a^x)$ where $s(x)$ is the probability of being in state x. Let $m = \lfloor N/2 \rfloor$. We have five cases to consider :

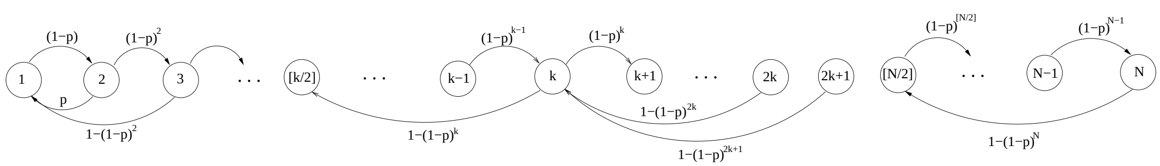


Figure 2.8: State Transition of Packets Sent Per RTT

1. We are in state 1. We could increase the rate and move to 2. We could also get losses at states 2 or 3, and move to 1. We could also get a loss, and stay in 1. In steady state,
$$s(1) \times a = s(2) \times (1-a^2) + s(3) \times (1-a^3)$$

2. We are in any state x between 2 to $m-1$. This is the general case mentioned above. In steady state,
$$s(k) = s(k-1) \times a^{k-1} + s(2k) \times (1-a^{2k}) + s(2k+1)(1-a^{2k+1})$$

3. We are in state m. We could increase the rate and move to $m+1$. We could also get loss at state N only, and move to m. In steady state,
$$s(m) = s(m-1) \times (a^{m-1}) + s(N) \times (1-a^N)$$

4. We are in any state x between $m+1$ to $N-1$. The only difference between this state and 3 is that we cannot enter this state by a rate reduction, as the $N = \max(rate \times RTT)$. In steady state,

$$s(k) = s(k-1) \times a^{k-1}$$

5. We are in state N. We cannot increase the rate further. The only way to leave this state is by cutting its rate, and moving to state m. In steady state,

$$s(N) \times (1 - a^N) = s(N-1) \times a^{N-1}$$

We did a Matlab analysis to numerically solve the chain for different large N's. The solutions yield the steady state probabilities $s(x)$ and hence the probability p_{le}. We then plot the ratio of LI probability p_{li} and LE probability p_{le} ratio in Figure 2.10 (a). The plot shows that there is not much difference between LI and LE probability. In particular, the maximum ratio of LI to LE probability is less than 1.25. In other words, the number of LEs is at most 25% smaller than the number of LIs (which happens when the LI rate is 30-40%, a large absolute loss rate).

The uniform loss probability assumption made above is not valid in bursty packet loss scenarios (though it could apply to RED gateways). The combination of our scheme with a drop-tail buffer leads to a very bursty packet loss pattern, which yields a very different picture of LI vs LE probability.

Bursty Loss Case

Our scheme uses a rate-based AIMD policy, which is not self-clocked (unlike TCP). This lack of self-clocking is because the scheme depends only upon LIs and not a stream of acks for its basic operation. A simple control-theoretic analysis [89] shows that a rate-based scheme with linear rate-increase, and lack of self-clocking leads to a bottleneck queue which grows quadratically. Therefore, if the bottleneck has a larger buffer, the queue increases quadratically for a longer period of time. Moreover, when the buffer is full and packets are about to be dropped, the sending rate differs significantly from the optimal transmission rate. So, when the queue overflows, the number of packets dropped corresponds to $(R - C)T$ where R is the aggregate transmission rate (which is much larger than C) and C is the bottleneck capacity, and T is the round trip time (RTT) including the queueing delay. This leads to a large burst of packets dropped during the RTT before the receivers send LIs and the sources detect congestion. Therefore it is important to understand how the number of LEs differ from the number of LIs under such bursty loss conditions. We show that under such conditions, the choice of LEs is even more justified.

We use the Markov chain model as used earlier, albeit with some changes. To approximate the bursty loss behavior, we assume that if a packet is lost during any instant within the RTT, all remaining packets transmitted during that RTT are also lost. This assumption is also made by Padhye et al [85] for TCP throughput modeling. Under this assumption, the probability p assumes different semantics. p here is defined as the probability that a packet is lost given that either it is the first packet in the RTT or the preceding packet in the RTT is not lost. Assume like before that $a = 1 - p$.

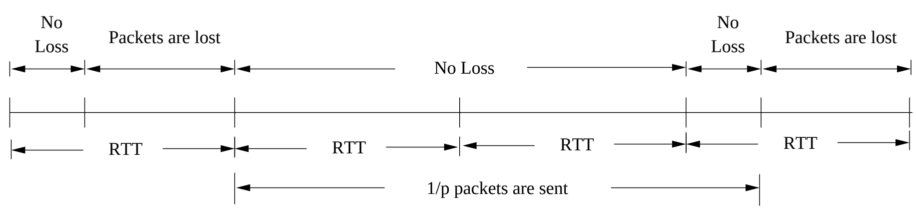


Figure 2.9: p's Semantics in Bursty Loss Case

In this case, $p \neq p_{li}$ (see Figure 2.9) because p_{li} is affected by every packet loss unlike p. However, the overall real LI probability p_{li} can be derived from p. First, recall that the state x specifies the number of packets transmitted during a certain RTT. Assuming that the first i packets are not lost, $x - i$ is the number of lost packets in that RTT. Under our assumption, during a RTT,

$$P(\text{The first i packets are not lost and the subsequent packets in the round are lost}) = (1-p)^i p$$

Therefore, at state x, the expected number of lost packets is $\sum_{i=0}^{x}(x - i)(1 - p)^i p$. The LI probability p_{li} is the expected number of lost packets divided by the expected total number of packets transmitted:

$$p_{li} = \frac{\sum_{x=0}^{\infty} s(x) \sum_{i=0}^{x} (x-i)(1-p)^i p}{\sum_{x=0}^{\infty} x \cdot s(x)}$$

At state x, the expected number of loss events (LE) is the probability that there is at least one loss, that is $1-(1-p)^x$. Therefore,

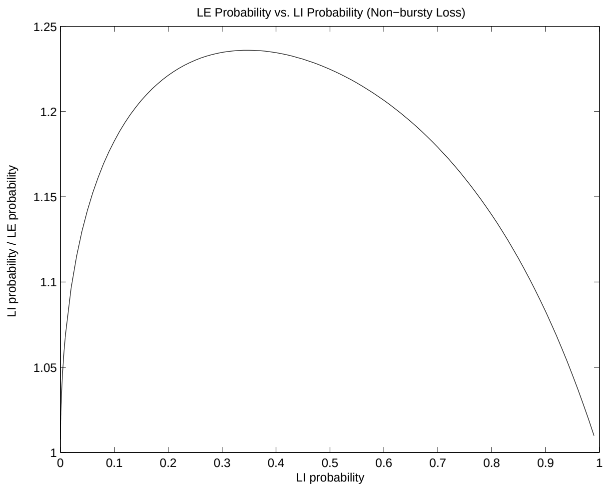
$$p_{le} = \sum_{x=0}^{\infty} s(x) \cdot (1-(1-p)^x)$$

As a result, ratio of LI and LE probabilities ($\frac{p_{li}}{p_{le}}$) is substantially altered. This is illustrated in Figure 2.10 (b). The figure plots the ratio as p_{li} ranges from 0 to 1. The ratio $\frac{p_{li}}{p_{le}}$ is huge when p_{li} is small (which is the common case of network operations). This is because the small overall LI probability leads to concentrated burst loss in few RTTs leading to a small number of LEs in comparison to LIs.

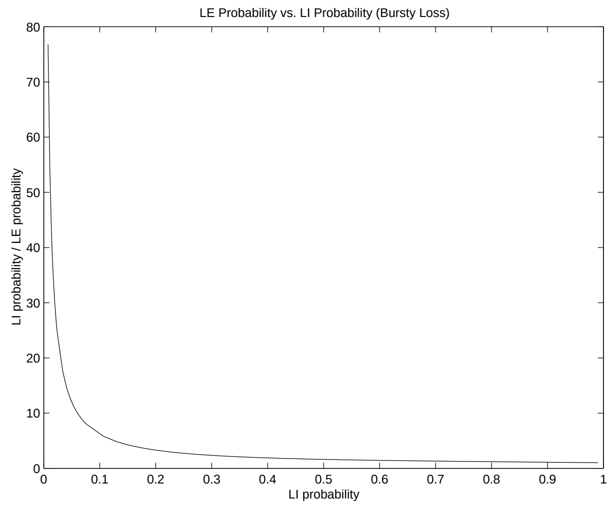
For lower loss probabilities, the p_{li}/p_{le} values are higher. To see the relation between LI and LE more clearly in this region, we have plot another graph for the low LI probability region (Figure 2.10 (c)). We observe high p_{li}/p_{le} ratios (9-90). Comparing these numbers to the numbers of LIs and LEs observed in simulation (see section 2.2.1.1), we observe that the model gives the same order of magnitude of results as the simulation results. In that case $p_{li}/p_{le} = 949/32 = 29.7$. Therefore, we can conclude that the LI2LE filtering is critical in our cascade filter design.

2.2.1.4 Drop-to-Zero Avoidance in Medium-Large Scale Trees

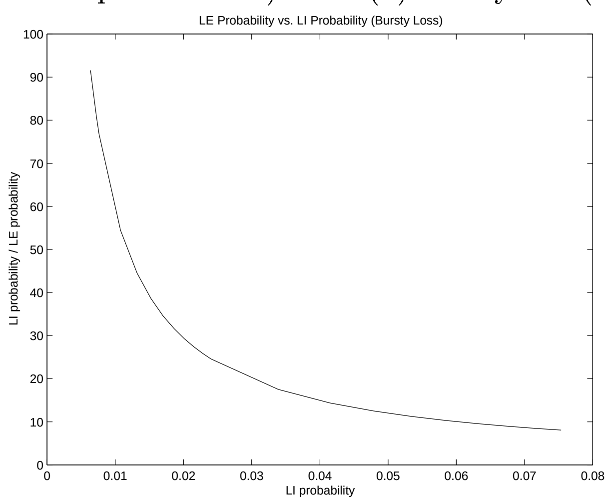
Recall that large receiver sets result in increased number of LIs being generated. To test the performance of our scheme in such scenarios we gradually increase the number of receivers in the multicast session, and run simulation for each addition with the assumption of constant RTTs and that each path experiences a uniform loss probability. We use a topology shown in Figure 2.11 where all the receivers and the single source are connected to the same router. One of the receivers in the session has the worst loss rate, and the rest have lower loss probabilities. We expect that the number of LEs passed by source for rate



(a) Non-bursty loss (All loss probabilities)



(b) Bursty loss (All loss probabilities)



(c) Bursty loss (Low loss probabilities)

Figure 2.10: Ratio of LI and LE Probabilities
(LE is superior to LI as congestion signals.)

reduction should remain fairly constant and equal to the number of LEs generated by the worst loss receiver.

The number of receivers in the multicast tree is increased from 1 to 10001 in a gradual manner. We assume one receiver has 10% loss, all others have 2% loss and study the effect of increasing the number of 2% receivers. The above numbers and topology were chosen as they represent a general case where one of the receivers (the 10% one) clearly has the worst loss rate. The expected number of LEs passed by the filters should be almost equal to the number of LEs generated by the worst loss receiver. The number of LEs passed for rate reduction by the source is plotted against the number of receivers(Figure 2.12(a)). The experiments

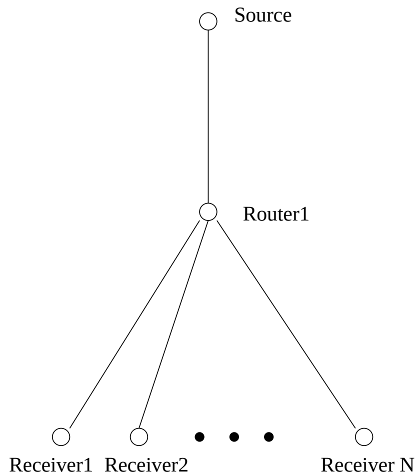
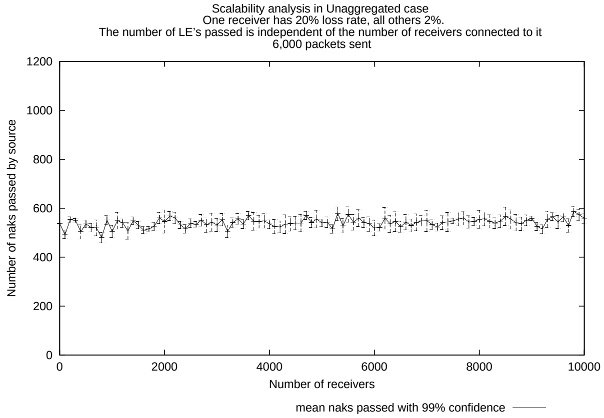


Figure 2.11: Topology : Large Heterogeneous Receiver Sets

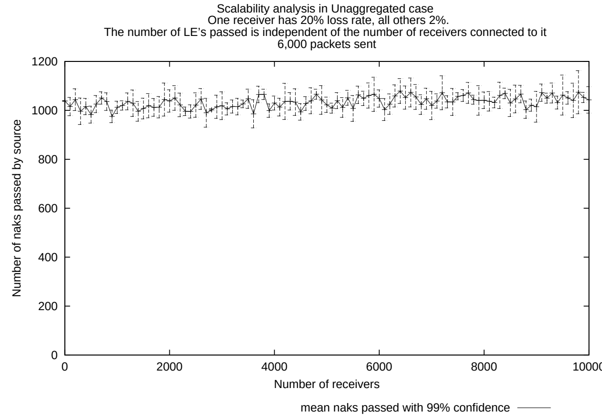
were repeated for different random seeds and 99% confidence intervals are plotted. Under no aggregation, the number of LEs passed by source remains fairly constant(500-520). The number of LEs passed are consistent with the non-bursty theoretical analysis performed above. The expected number of LIs by the 10% receiver would be $0.1 \times 6000 = 600$. By referring to the LI probability(p_{li}) vs LE probability(p_{le}) graph(Figure 2.10 (a)) for the non bursty model, we see that for 10% loss p_{li}/p_{le} ratio is around 1.2. Therefore the expected number of LEs would be $600/1.2 = 500$. This number agrees very closely with the numbers obtained by simulation. We also repeated the same experiment with the max loss receiver having a loss rate of 20%. Again, the number of LEs passed (Figure 2.12(b)) remains fairly constant (around 1000). This indicates that under same RTT, fixed uniform loss probabilities, and no aggregation conditions our scheme passes a fairly constant number of LEs even when large number of receivers are present. This number is equal to the LEs generated by the worst loss receiver. Thus we achieve Drop-to-Zero avoidance in medium-large scale trees.

2.2.2 Evaluation: TCP friendliness

TCP-friendliness is a subject of current debate [33, 94, 114]. A good measure of TCP-friendliness would be the average throughputs of the competing TCP and RM flows. To illustrate TCP-friendliness, we use two topologies, a) where a common bottleneck is shared by many RM flows and a single TCP flow and, b) many TCP flows and a single RM flow.



(a) 10% Max loss receiver



(b) 20% Max loss receiver

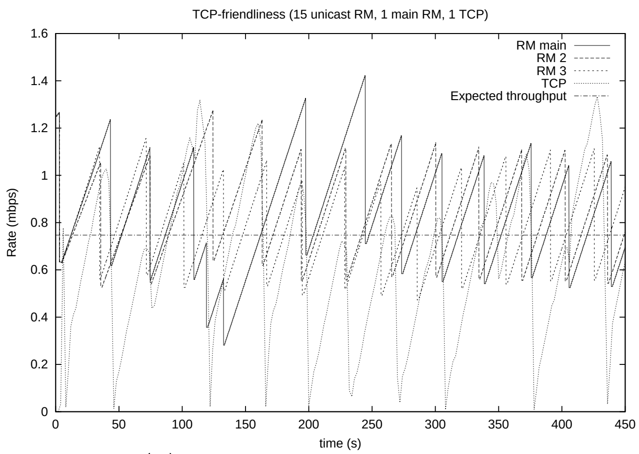
Figure 2.12: Scalability Analysis
(LE-SBCC avoids drop-to-zero in medium-large scale trees.)

LE-SBCC would be TCP-unfriendly if either of the TCP or RM flow is beaten down. In Figure 2.5, consider two cases: case (a) when *one* of the flows of "Set 4" is replaced by a TCP Reno flow while the rest are LE-SBCC flows; and case (b) when *all* the flows in "Set 1" through "Set 4") are replaced by TCP flows.

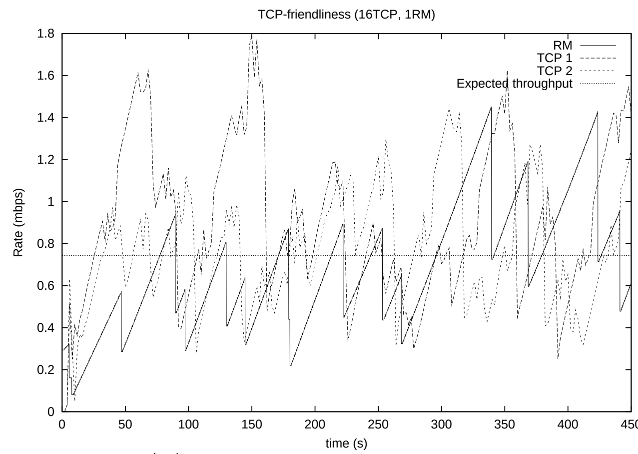
Simulation results: The simulation results for these two cases is shown in Figures 2.13(a) and 2.13(b) [6]. The dips by the TCP flow close to zero represent timeouts. Again observe that the main RM flow (solid line in both graphs) shares the bottleneck(s) fairly with the competing TCP flow(s). This becomes clearer by observing that the rate graphs for the RM and TCP flows oscillate about roughly the same mean and with the same variance. The horizontal dotted line is the expected TCP throughput plotted post-simulation using the simplified TCP equation $(1.22MSS/RTT\sqrt{p_{le}})$, where p_{le} is the LE probability of the worst-loss receiver. Recall that for the purpose of congestion control, we aim to reduce the multicast tree to a unicast path with the worst-loss receiver being the only one with LE probability p_{le} and a worst RTT. Clearly from the Figures 2.13(a) and 2.13(b), the rate values oscillate about the theoretical mean shown by horizontal dotted line. This fact is a good measure of the TCP friendliness of our scheme, further demonstrates the validity of choosing LE rather than LI probabilities in our scheme.

[6] To avoid cluttering, the TCP throughput is sampled every 2s

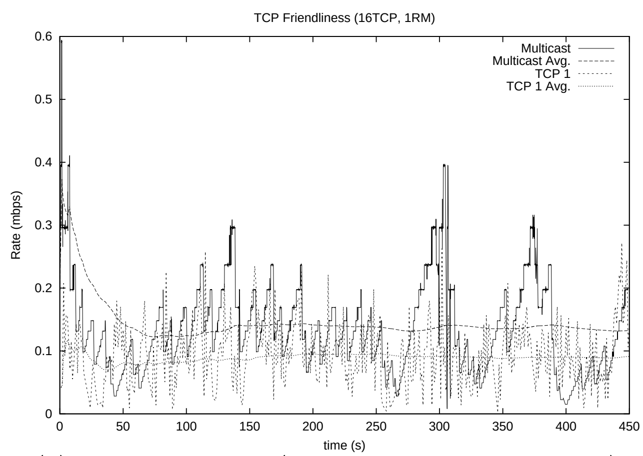
Experimentation results: The scheme is implemented in a Linux 7.0 testbed. The topology used is identical to the one used for the simulations as described above. Again the RM flows are fair to the competing TCP flow (Figure 2.13(c)). More details on this are presented in the section on implementation issues.



(a) 1 TCP Reno vs 16 RM



(b) 1 RM vs 16 TCP Reno

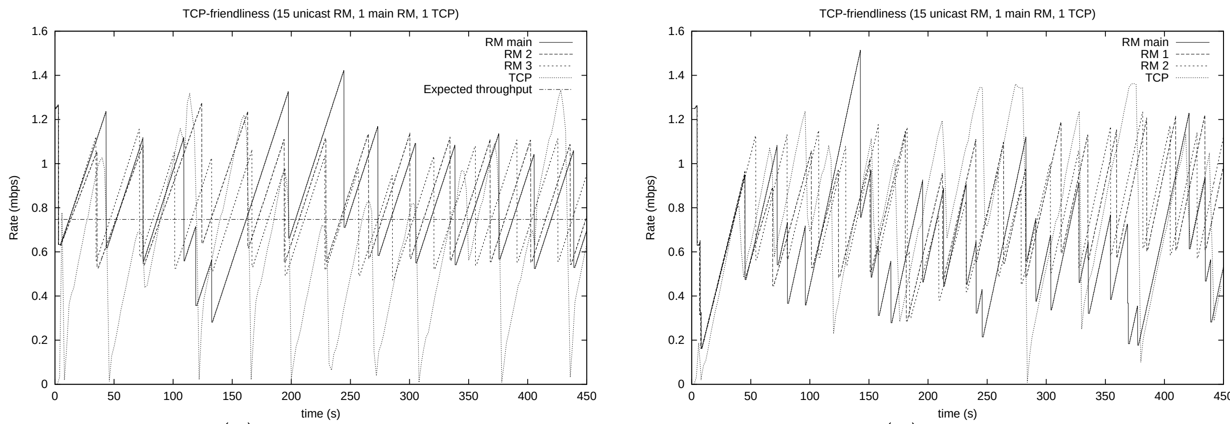


(c) Experimental (1RM vs 16 TCP SACK)

Figure 2.13: TCP-Friendliness Results (Simulation and Experiment)
(LE-SBCC shares the bottleneck(s) fairly with the competing TCP flow(s).)

2.2.2.1 DropTail vs. RED Queues

We evaluated the TCP-friendliness performance of our scheme with the two queue management policies - *Tail Drop* and *Random Early Detection (RED)*. These are the two widely used queue management policies and it is important to test the performance of LE-SBCC with both the policies. The topology used for this purpose is the same as in the above experiment with the following key parameters: TCP version: *Reno*, Queue size = *125 packets*, Packet size = *560 bytes*.



(a) DropTail (b) RED

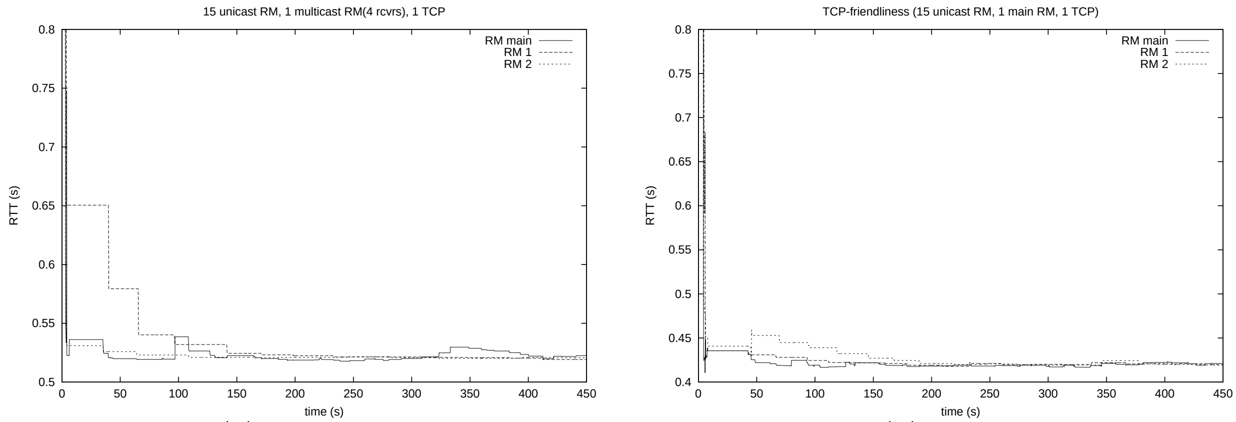
Figure 2.14: Droptail vs. RED: Rate Graphs
(LE-SBCC remains fair towards the competing TCP flows with either droptail or RED queues.)

As we can see from Figure 2.14, the throughput is marginally lower with RED than with DropTail and so is the measured RTT (Figure 2.15). (The absolute RTT values are smaller in RED, because when losses occur, RED has lower average queue length.) Also observe that the RTT converges much faster with RED than DropTail. This is because the RED policy drops packets before the queue is full resulting in more frequent and timely loss indication from the receivers. The scheme remains fair towards the competing TCP flows in both cases and we do not observe any drop-to-zero with RM flows.

2.2.2.2 Different Loss Paths

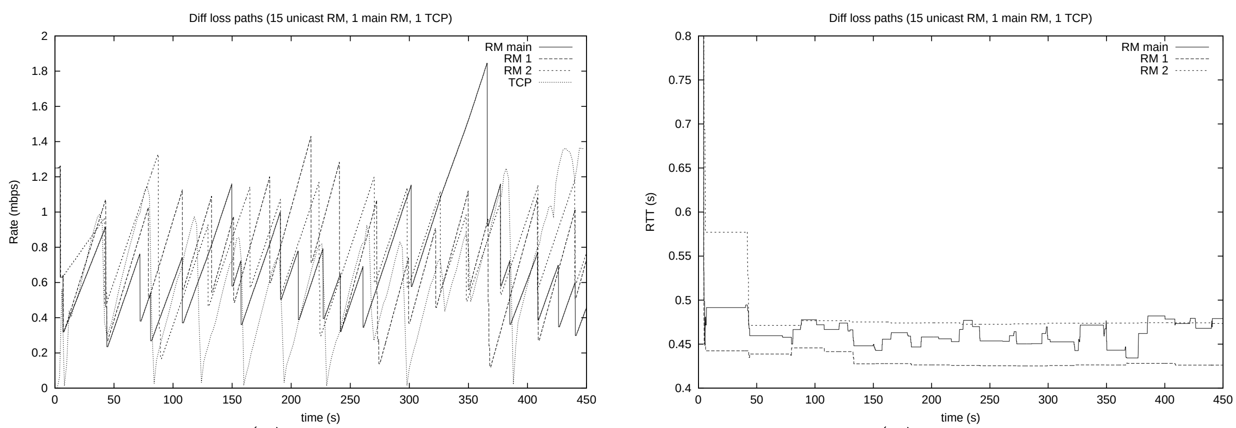
It was speculated that the scheme would not perform as well with the multicast tree having paths with varying degree of loss probabilities and RTTs. We perfomed the following simulation to test the robustness of our scheme under these heterogeneous conditions. Again the topology remains the same as the above experiment with modifications in following key parameters: The buffer is 125 packets on all links except that it increases on the four links from the router to receiver 1, receiver 2, receiver 3 and receiver 4 with the values 25, 50, 75 and 100 respectively.

Having different buffer sizes is an indirect way to affect RTT and loss probabilities on a link. The link from router to receiver 1 has the smallest buffer (25 packets). Therefore,



(a) DropTail (b) RED

Figure 2.15: RTT Convergence - Droptail vs. RED
(The RTT measured by LE-SBCC converges much faster with RED queues than with droptail queues.)



(a) Rate (b) RTTs

Figure 2.16: Rate and RTT Graphs for Topology with Different Buffers (Loss Rate and RTTs)
(LE-SBCC works well under heterogeneous loss probability and RTT conditions.)

it has the smallest RTT and the largest loss rate. On the other hand, the link from router to receiver 4 has the largest buffer (100 packets). So it has the largest RTT and smallest loss rate. For the *Main Source*, MaxLPRF will respond to LEs from receiver 1, whereas its RTT would be biased higher (average of the 4 different RTTs measures received from the 4 receivers). The graph (Figure 2.16) clearly demonstrate that the algorithm works well under heterogeneous loss probability and RTT conditions.

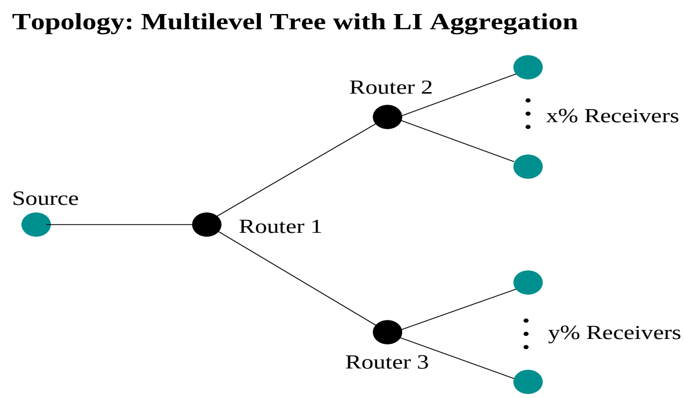


Figure 2.17: Topology to Illustrate LI Aggregation Effects

2.2.3 Evaluation: LI Aggregation Effects on Performance

Aggregation of loss indications (LIs) has an effect on the number of LEs and timing information reaching the source, and therefore affects the performance of LE-SBCC. To evaluate these effects, consider the simple multi-level topology in Figure 2.17. As the number of receivers and their loss rates are varied, consider three cases which we simulated:

1. **No LI Aggregation** performed at the routers.

2. **Partial LI Aggregation:** LI aggregation performed at router 2 and router 3 only (and not in router 1).

3. **Complete LI Aggregation:** LI aggregation performed at all routers.

Consider cases 2 and 3. Assume one receiver X has a fairly high loss rate, and all others have very low loss rates. Ideally, the number of LEs reaching through the source (after filtering) should be equal to the number of LEs generated by worst-loss receiver X. If a larger fraction of the LIs sent by X are suppressed by those sent from other low-loss receivers, then fewer and fewer LIs generated by X might reach the source. We call this the partial aggregation case (case 2). This could occur because X has a longer RTT, and as the number of receivers increase, we expect more overlaps between sequence numbers lost by X and other receivers. In the worst case, no LIs generated by X would reach the source. Then even after LI2LE filtering, the maximum number of LEs generated by *any single receiver* would now become much lower, and thus affects the critical ratio $\frac{\max_i X_i}{\Sigma_i X_i}$ used by Max-LPRF. Hence MaxLPRF would pass through lesser number of LEs. The source hence would make fewer rate reductions, becoming *more aggressive compared to TCP flows on the worst loss* *path.* Observe that fewer rate reductions also means that the multicast flow even under cases of full (worst-case) aggregation *does not experience the drop-to-zero problem.*

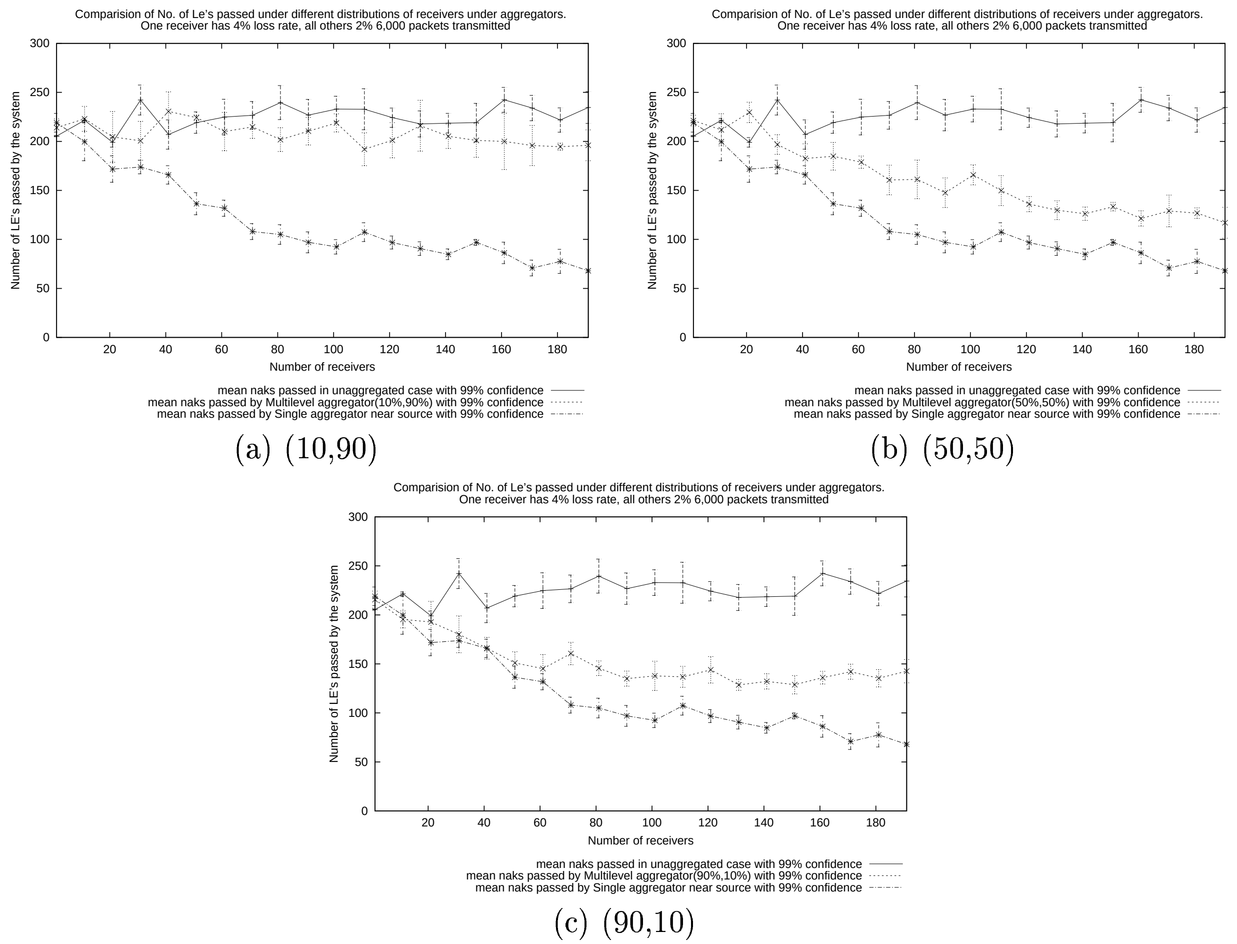


Figure 2.18: Partial Aggregation Effects
(Aggregation poses a TCP-friendliness performance problem for LE-SBCC in the cases where the worst-loss receiver's feedback is significantly suppressed.)

The graphs shown in Figures 2.18 (a), (b), (c) illustrate the effects of no aggregation, partial and worst-case aggregation. The graphs plot the number of LEs passed by the filter cascade against the total number of receivers. The topology is same as in Figure 2.17. One receiver has loss probability of 4% (and has a longer RTT) while all others have 2%. This would bias the aggregation process against the worst-loss receiver's LIs. We vary the distribution of receivers under the two aggregators in the multicast tree. The notation (x,y) in Figure 2.18 indicates that x% of all the receivers (including the receiver with 4% loss probability) are attached to Router2 and remaining y% are attached to Router3. The experiments were repeated for different random seeds and confidence intervals plotted. In each experiment, 6000 packets were sent which leads to an expectation of 240 LEs (given a worst-loss receiver having 4% loss rate) after the cascade of filters is applied. If fewer than 240 LEs are passed, the multicast rate would be higher than competing TCP on the worst path.

In all graphs of Figures 2.18, the no-aggregation case passes just under 240 LEs (99% confidence intervals), which implies that the scheme works very well without aggregation. This is also similar to the results seen in Figure 2.12 where we used different loss probabilities and examined a larger number of receivers (up to 10,000 receivers). Now, the performance of the intermediate curve (partial aggregation) in Figures 2.18 varies depending upon the pattern of aggregation. Observe that if more LIs from the max loss receiver get through without begin aggregated, the intermediate (partial aggregation) curve is almost comparable to the scenario with no aggregation (eg: Figure 2.18 (a)). However, in the case of Figures 2.18 (b) & (c) the max loss receiver is present in the subtree with a larger number of receivers. Combined with the effect of longer RTT, these configurations bias the aggregation process against the worst-loss receiver because of the increased probability of overlap between the ranges of lost packets seen by the worst-loss receiver and other receivers connected to Router2. Hence Figures 2.18 (b) & (c) shows the partial aggregation curve closer to the worst-case curve. The worst-case curve in all the three graphs depicts the effect of complete aggregation done at Router1, which suppresses all the LIs from the worst-loss receiver.

In summary, aggregation poses a performance problem for LE-SBCC in the cases where the worst-loss receiver's feedback is significantly suppressed. However the effects lead to TCP unfriendliness only. Stability and drop-to-zero avoidance are solved. It should be noted that aggregation issues pose a problem for PGMCC too (see [100]). Any randomization or technique to break the bias against such worst-loss receivers in the aggregation process (eg: see [100]) would solve this performance degradation issue both for LE-SBCC and PGMCC.

2.3 TFRC module

TFRC [33] is a congestion control policy which provides smooth rate changes, which makes it suitable for audio/video applications. TFRC was originally designed for unicast. But our approach allows the use of the TFRC policy instead of the AIMD policy for rate adaptation. However, we need to modify TFRC to fit it into our framework.

2.3.1 TFRC Module Design

For AIMD, we used loss events (LEs) to trigger rate reductions. However, for TFRC we need to convert them into a LE rate to update the data transfer rate. TFRC [33] requires the *receiver* to calculate the LE rate and send it back to the sender. In contrast, our scheme is purely source based. There is no such kind of feedback from the receivers. We modify this mechanism so that the LE rate can be calculated at the source. Recall that the output of cascaded filters (Section 2.1.1) are loss events (LEs). The source can count the loss intervals (the number of packets between consecutive LEs) and calculate the LE rate and therefore the data transfer rate. The details follow.

With the following definitions,

s_0: the interval since the most recent LE.
s_i: the i-th most recent loss interval. $(i \geq 1)$
N: the total number of loss intervals from the beginning of the data transfer.
n: an integer greater than 0.
α: $0 \leq \alpha \leq 1$.

the average loss interval $\hat{s}_i$ is calculated as following.

$$\hat{s}_i = \frac{\sum_{k=1}^{n} s_{i+k-1} w_k + \sum_{k=i+n}^{N} s_k \alpha^{k-i-n+1}}{\sum_{k=1}^{n} w_k + \sum_{k=i+n}^{N} \alpha^{k-i-n+1}}$$

In [33], $n = 8$, $\alpha = 0$, w_1 to w_8 have values of 1, 1, 1, 1, 0.8, 0.6, 0.4, 0.2 respectively. In our scheme, the randomness of the MaxLPRF filter can cause larger variance of the average loss interval and therefore more oscillation of the LE rate. To counter this effect, we chose different values for these parameters. In our experiments, $n = 8$, w_1 to w_8 are all 1, and $\alpha = 0.95$. (When there are not enough loss intervals, (1) $N < n$, set w_{N+1}, ..., w_n and α to 0, (2) $N = n$, set α to 0.) They yield satisfying results. Note that this method and the parameter values need not be the only viable set for calculating average loss interval. Other methods and parameter values could be explored.

Whenever a LE is detected, $\hat{s}_1$ is calculated, the reported LE rate p is $1/\hat{s}_1$. Then the data transfer rate T (bytes/sec) is calculated with the following formula [33] and enforced.

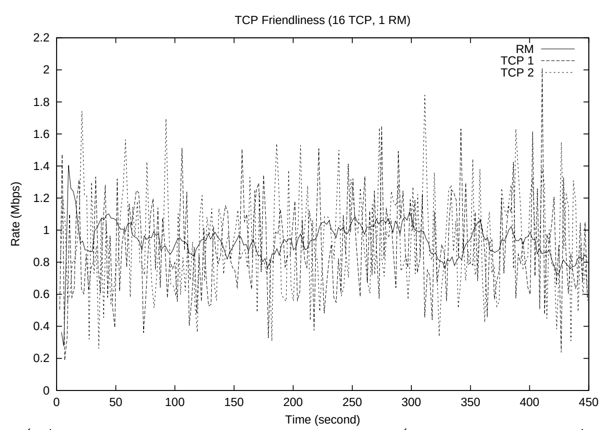
$$T = \frac{s}{R\sqrt{\frac{2p}{3}} + t_{RTO}(3\sqrt{\frac{3p}{8}})p(1+32p^2)}$$

where s is the packet size, R is round-trip time, p is LE rate, $t_{RTO} = SRTT + 4 * \sigma$.

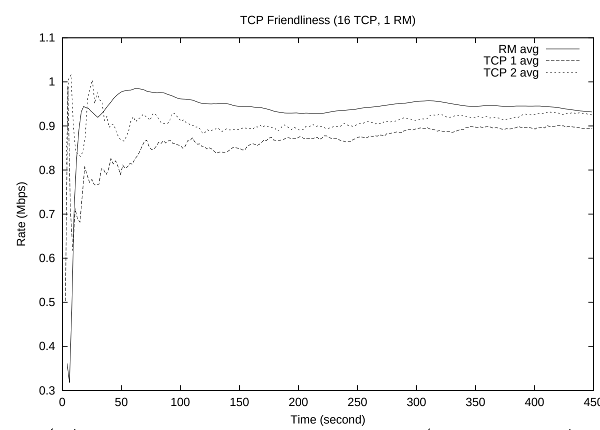
During absence of congestion, after each RTT, $\hat{s}_0$ is re-calculated. The final average loss interval $\hat{s}$ is $\max(\hat{s}_0, \hat{s}_1)$. If it is different from the previous value of $\hat{s}$, the rate T will be updated with the reported LE rate $p = 1/\hat{s}$. If congestion is absent for a long time, $\hat{s}_0$ will become larger and larger, so is $\hat{s}$, and the reported LE rate will drop. Consequently, the rate T will increase, given that the RTT does not vary much. Note that our adaptation of TFRC for multicast differs from the recent work in TFMCC [114] where each receiver measures the RTT and loss event (LE) rate and the source has a filter to decide which feedback to use.

2.3.2 Simulation Results

The results presented here show that the LE-SBCC scheme with the TFRC module works relatively well.



(a) 1 RM vs 16 TCP Reno (instant rates)



(b) 1 RM vs 16 TCP Reno (avg. rates)

Figure 2.19: TFRC Module Simulation Results
(LE-SBCC with TFRC module has smoother rate changes and is still fair with TCP flows.)

We tested the scheme with the TFRC module on the topology shown in Figure 2.5. The instant rates of multicast flow and two randomly selected TCP flows are shown in Figure 2.19 (a). Compared to Figure 2.13 (b), the rate changes are much smoother. Furthermore, the multicast flow is still fair to other TCP flows, as shown in Figure 2.19 (b) [7].

[7] avg. rate at time t = amount of data sent in period $[0, t]$ / t

In a summary, TFRC as a module can be installed into our framework and work well, demonstrating the modularity of the scheme and its applicability to multimedia.

2.4 Linux Implementation and Experimentation

For any scheme to be deployable it needs to be implemented and tested in real-world networks. We implemented LE-SBCC on a real network to study implementation issues and performance under real-world conditions. Specifically, we tested TCP friendliness and drop-to-zero problems. Lot of real world issues and limitations not found in simulation had to be tackled. Such issues are discussed in detail in the following sections.

2.4.1 Implementation

The scheme is implemented on a simplified version of PGM. However, it is not limited to PGM or just reliable multicast.It has been implemented on top of UDP using RedHat Linux 7.0. The following aspects of PGM have been implemented:

- ODATA (original data packet), RDATA (retransmitted data packet), NAK (negative acknowledgement packet), NCF (NAK confirmation packet) packet formats specified in PGM.
- Transmit window at source.
- Receive window at receiver.

By implementing the features mentioned above, we achieved a reliable transport layer. On the other hand, for the sake of simplicity, we omitted certain aspects of PGM:

- SPM (source path message) packets.
- Network elements support.

SPM packets are sent by source periodically to receivers. They carry information about transmit window which affects the generation of NAK's. To compensate this, each packet from source carries the information about transmit window. Whenever a receiver gets a packet (ODATA, RDATA or NCF), it checks the sequence number continuity and sends NAK's if needed.

The program runs as a user process without requiring any root privileges. As a result, the granularity of the timer used is 10 ms which is quite coarse. Coarse timers can introduce unnecessary delays. Assume that an event is supposed to occur at time $t + \delta t$. The timer for it is started at t. If the δt is less than 10 ms, due to the coarseness of the timer, the event will actually occur at $t + 10$ ms. Then extra delay of $10ms - \delta t$ is introduced. Notice that the extra delay can accumulate. That is, the extra delay introduced by the first event can be added to that introduced by the second event. To reduce such negative effects, the following mechanisms are used:

- After processing any timeout event, the next event is checked against the current time obtained with the API *gettimeofday()*. If the timeout should actually happen, it will be processed. This process continues until no more timeout events are available. Since times returned by *gettimeofday()* have microsecond resolution, the interval between two successive events can be less than 10 ms, provided events are processed sufficiently fast.

- When a timeout event is scheduled, its timeout value is checked. If its timeout is within 5 ms (half of the timer granularity) from now, it is immediately triggered. This introduces a minor random factor ($< 5ms$) into event scheduling, but reduces the unnecessary accumulative delays mentioned above. In fact, together with the first mechanism, this results in possible bursty events. When the events are for packet transmission, bursty packet transmission occur. The optimal solution for this problem would be to get a system timer of finer granularity.

- We introduce a variant of TBF to control the packet transmission rate at source. Upon arrival of a token, if there are enough packets, the source will transmit as many packets as needed to maintain the required transmission rate. If there are not enough packets, all of them would be transmitted.

By inspecting the time out information, we have observed that the mechanisms described above worked as expected, although the approximation leads to somewhat bursty transmissions.

2.4.2 Experimentation

The experiments were conducted on our testbed consisting of RedHat Linux 7.0 systems. During the experiments, the diff-serv package *tc* in RedHat Linux 7.0 were used to restrict bandwidth and link latency. Also, a multicast router daemon *mrouted* [8] was used to route multicast packets from one LAN to others.

The LE-SBCC program runs in user mode subject to OS priority and timer constraints. It cannot send packets at high rates (> 1Mbps). Furthermore, in real experiments, it is very difficult to strictly control certain parameters such as link latency. Thus it is not possible to match the simulations exactly. However, we show that our scheme is TCP friendly and avoids drop-to-zero in real networks.

2.4.2.1 TCP Friendliness Experiment

During the TCP friendliness experiment, TCP SACK was enabled. We used TBF queue discipline of a software package *tc* to restrict the bottleneck bandwidth. Details of the configuration are listed as follows:

- Round trip link latency (set by *tc*): 500 *ms*
- Bottleneck bandwidth (set by *tc*): 1Mbps
- Source initial transmission rate: 200Kbps
- Source initial RTT estimate: 200 *ms*
- Receiver maximum random backoff before sending NAK's: 100 *ms*

The fixed part of RTT is the round trip link latency of 500 *ms*. Therefore, any RTT sample is at least of that value. Other parts of delay include hardware I/O delay, router buffer delay, OS processing time and so on. It is very hard to find out the exact delay.

Upon detecting a packet loss, a receiver waits for a certain amount of time before sending a NAK. This time is called backoff time in PGM. A backoff time is chosen randomly from the interval [0, max random backoff] according to uniform distribution. This backoff increases the RTT. The maximum random backoff time is a configurable parameter. When it is large in relation to RTT, it will affect RTT estimation. We chose this time to be 100*ms*

[8] Source code and binaries of *mrouted* can be found online at *http://www.vcas.video.ja.net/mice*

so that the receivers have sufficient time to listen for NCF after a loss detection. This helps to avoid duplicate NAK's for the same sequence number from different receivers. At the same time, this does not affect RTT significantly.

For this experiment, we used the topology in simulation (Figure 2.5) with the bandwidths and delays changed. All TCP sources and the multicast source ran on one machine. Behind the bottleneck, there were four receiver machines. To achieve load balancing, each of the receiver machine had four TCP receivers and one multicast receiver. The result, shown in Figure 2.20, roughly agrees with that of simulation.

In the graph, the average rate of multicast is higher than that of TCP. The reasons can be:

1. The approximation we introduced in the source rate-controller leads to somewhat bursty transmissions for multicast sources.

2. The implementation of our scheme is in user space, thus has lower priority and is less reactive to the network situation than TCP which runs in kernel and has higher priority.

3. The timer of our program is coarse, which could make our program less reactive.

4. TCP timeouts are triggered due to small window size, while the multicast scheme does not have such rate-cutting mechanisms.

The TCP flow in the graph was randomly chosen from 16 TCP flows. The results indicate TCP friendliness and drop-to-zero avoidance of our scheme.

2.5 Summary

In this chapter, we have presented an *LE-oriented, purely source-based multi-sender multicast congestion control scheme (LE-SBCC)*. It can be used for both reliable and unreliable multicast transport. Since the scheme is completely contained at the source, it is simple to implement and deploy.

The model is a simple cascade of filters followed by an AIMD module. The filters together transform the multicast tree to appear like a unicast path for the purposes of congestion control. In the section of theoretical analysis of loss events (LEs) probability

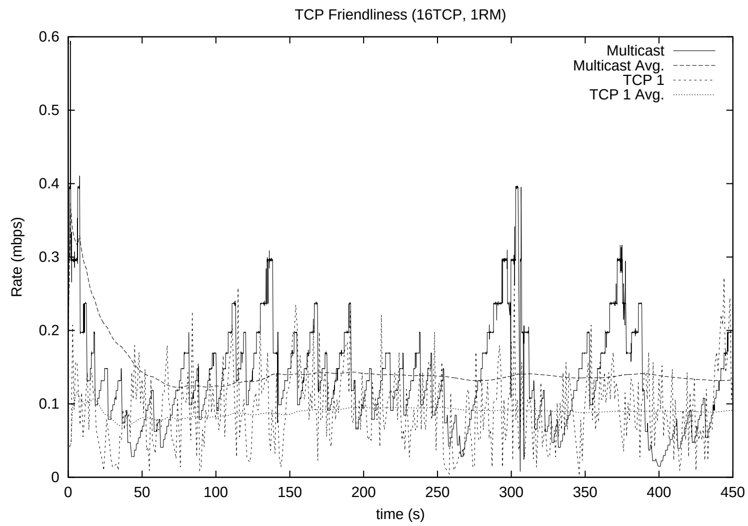


Figure 2.20: TCP-Friendliness Experiment Result (Linux Implementation)
(LE-SBCC avoids drop-to-zero and is roughly TCP-friendly in real networks.)

versus loss indications (LIs) probability, we justified our choice of LEs for rate reduction consideration. By studying the effects of removing a filter from the cascade, we showed that all filters are important. Several simulation and Linux-based implementation experiments confirm the effectiveness of our scheme.

Without LI aggregation, the scheme correctly filters the LEs (converted from LIs); avoids the drop-to-zero problem while being TCP-friendly. With aggregation, the scheme is likely to be more aggressive towards TCP, but never incurs the drop-to-zero problem. Like all single-rate schemes, its scalability is limited by the heterogeneity of the tree which may depress the source rate. Besides, LE-SBCC maintains a number of states which grows with the number of receivers experiencing congestion (*not* the total number of receivers), and there may be heavy feedback traffic as LE-SBCC does not control the receiver side. However, we believe LE-SBCC has the best scalability given the purely source-based restriction.

In general, the AIMD module in LE-SBCC may also be replaced by a source-based TFRC scheme [33] or rate-based binomial scheme [5]. With some modifications, we adapted the TFRC for unicast [33] to our framework. The performance shown by simulation experiments is satisfying. Since TFRC is suitable for audio/video transportation, our scheme can be applied to multimedia.

CHAPTER 3
ORMCC

In this chapter, we consider the multicast congestion control problem for the following situation:

> *Support is allowed on receiver side, but only one multicast group is allowed for each multicast session.*

For example, if a company needs to multicast some data over the public network (assuming multicast is supported) using the open group model [26], but it only has one class-D address from IANA (Internet Assigned Numbers Authority) because there is not enough multicast address resource, it will encounter such a situation.

Here we present ORMCC. It assumes a new metric *TRAC (Throughput Rate At Congestion)* in feedback and other support from receivers, so that some processing can be distributed to receivers and most feedback can be avoided, making ORMCC truly scalable. Features of ORMCC include $O(1)$ state, *statistics-based* feedback suppression etc. Comparison with PGMCC [94] and TFMCC [114] in simulation shows that ORMCC achieves better performance under most situations. ORMCC has also been implemented and tested on real systems in Emulab [113].

The general concept of ORMCC is as follows: The source dynamically selects one of the slowest receivers as *Congestion Representative (CR)*, and only considers its feedback for rate adaptation. The slowest receivers are those with the lowest average TRACs. When there is no CR, all receivers may send feedbacks to the source. Once a CR is selected, only the CR and those receivers with average TRAC lower than that of the CR can send feedbacks so that feedbacks are efficiently suppressed. Also notice that the scheme is not concerned with reliability issue and only considers congestion control. Therefore, like LE-SBCC, it is applicable to both reliable and unreliable multicast.

An example operation can illustrate how our scheme works more clearly. In Figure 3.1 (a), the source has chosen a receiver behind the most congested path as CR by comparing average TRACs of receivers. Only the CR will send feedback while other receivers suppress their feedback. The feedback is CI(μ) in Figure 3.1 where CI means *congestion indication*

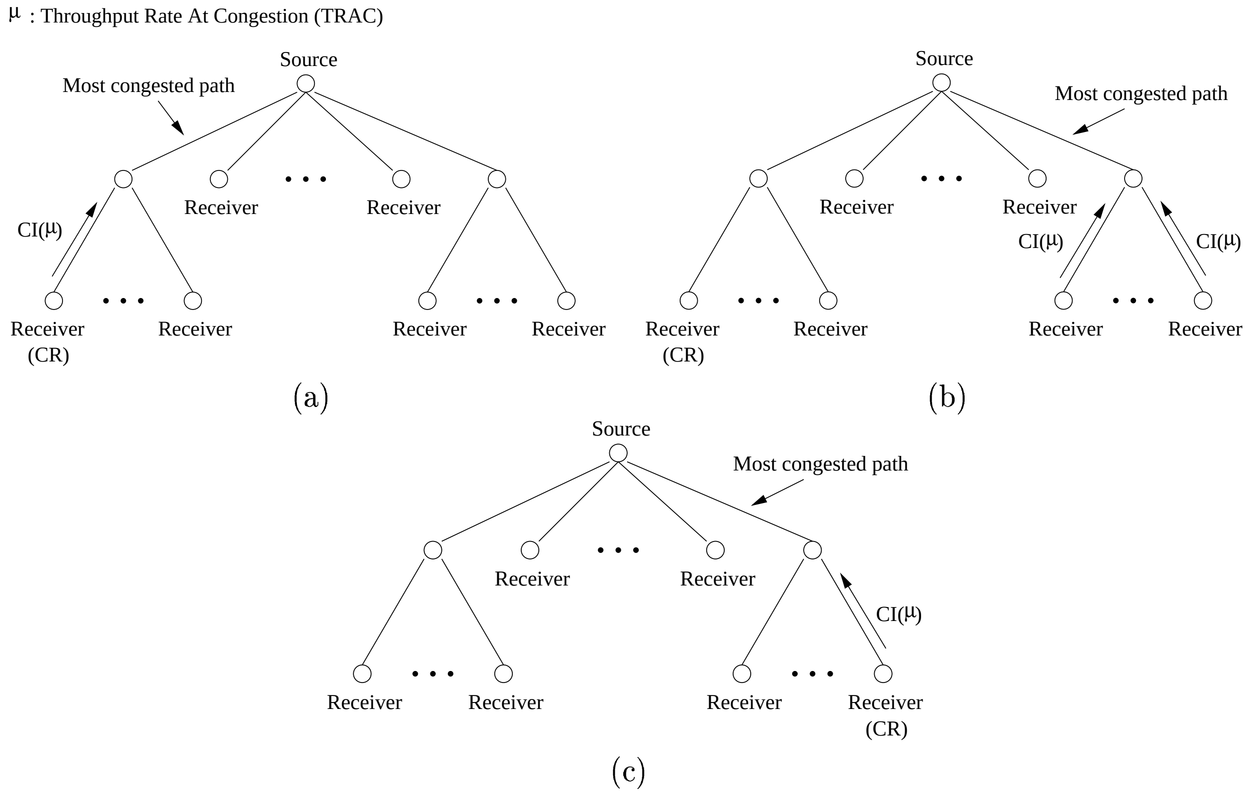


Figure 3.1: Example of ORMCC Operation

and μ is TRAC measured by receiver. After a while, another path becomes the new most congested path. Those receivers behind that path will see average TRACs lower than that of the current CR, and will send feedbacks (Figure 3.1 (b)). As a result, one of them will be chosen as the new CR. After that, again, other receivers suppress their feedback (Figure 3.1 (c)).

In the following sections, we will describe the details of the scheme, followed by simulation and experiment results. Theoretical analysis is in Appendix C.

3.1 Scheme Details

As we have mentioned in the introduction, in ORMCC, receivers send TRACs back to the sender whenever necessary, and the sender dynamically chooses a representative (CR) out of them and use only its TRACs to adjust the sending rate. In this section, we will present the details of how the whole scheme works, followed by a list of the features of ORMCC.

3.1.1 Feedback Required from Receivers – CI(μ)

When receivers detect congestion by packet losses, they need to inform the source so that the source can adjust the transmission rate accordingly. We denote a feedback packet as CI(μ) (CI stands for congestion indication, μ stands for rate information). Like NAKs, CI(μ)s may be sent only when a receiver detects packet losses (though they may also be suppressed). Assume that at the arrival of packet A, a receiver detects that some packets have been lost. It will then send a CI(μ) to the source. Such a CI(μ) contains (1) The sequence number of A, for the sake of RTT measurement, and (2) The output rate measured at the arrival of A, for the sake of CR allocation. Note than when multiple packet losses are detected by the arrival of one packet, only one CI(μ) will be sent (if without suppression), while multiple NAKs are generated for the same situation. With suppression, CI(μ)s may be sent at a less frequency.

To avoid oscillation, we average the output rate over a short period of time.[9] Note that the output rates used here are different from those in normal sense, because they are measured only when packet losses are detected due to congestion. To distinguish them, we give the notation *Throughput Rate At Congestion (TRAC).*

TRAC is different from packet loss rate since it captures the congestion status of bottlenecks better. Consider two flows going throughput different bottlenecks at different rates, for example, 1Kbps and 100Kbps respectively. If both flows experience 0.5% packet loss rate on average, we cannot say that the bottlenecks they go through are equally congested. TRAC is also superior than average receiving rate because most packets not contributing to congestion are excluded and the noise of measurement is reduced.

3.1.2 Allocation of The Slowest Receiver

ORMCC compares average TRAC of all receivers to locate the slowest ones, and choose one of them as the *Congestion Representative (CR)*. By using TRAC, it avoids computing TCP throughput formula [76] [85] which requires RTT and packet loss rate.

Since TRACs are measured at receivers upon packet losses, they indicate how much bandwidth a flow can get out of the fully loaded bottleneck, assuming congestion is the only reason for packet losses. The less it can get, the more congested the bottleneck is. Therefore, we choose one receiver with the lowest average TRAC as the CR, and let the source only

[9] In our simulations and implementation, we use one second.

consider the CI(μ)s from that receiver for rate adaptation. Average TRAC is calculated by means of *EWMA (Exponentially Weighted Moving Average)*. Denote TRAC as U, average TRAC as $E(U)$. With a sample U, $E(U)$ is updated as $E(U) \leftarrow (1-\alpha)E(U)+\alpha U$. Deviation U_σ is also updated as $U_\sigma \leftarrow (1-\alpha)U_\sigma+\alpha|E(U)-U|$ for the sake of CR updating and feedback suppression (see Section 3.1.3 and 3.1.4).

The receivers help the source to select a receiver with the lowest average TRAC by sending in CI(μ)s only if their average TRACs are low enough to qualify them as CR. More details of how receivers help will be covered in Section 3.1.4 of feedback suppression.

3.1.3 Update of CR under Dynamic Conditions

Network conditions always keep changing, and we need to continuously keep our choice of CR up-to-date. There are mainly two situations under which CR needs to be updated: (1) The situations of some non-CR receivers change so that one of them sees more severe congestion than the current CR does. (2) While the situations of all non-CR receivers remain unchanged, the previously most congested path is improved so that the current CR sees less congestion than other receivers, or it leaves the multicast session.

Tracking the slowest receiver by examining average TRACs can deal with situation (1), but to cope with situation (2) needs more effort. Under this situation, there can be no CI(μ)s from the current CR. Recall that the source only consider the CI(μ)s from the CR for rate adaptation and ignores all other CI(μ)s. If the source does not change CR in time, the transmission rate will be out of control. To detect that, we estimate an upper bound (denoted as T^{cr}_{max}) of the idle time (denoted as T^{cr}) before the source receives a first CI(μ) from the CR when the bottleneck is fully loaded. To use T^{cr}_{max}, suppose we somehow detect that the bottleneck is fully loaded at time t. If there has been no CI(μ) from the current CR for T^{cr}_{max} since t, we can say that the current CR is now inactive and needs to be changed.

Let's look at Figure 3.2. When the CR is still active, we measure samples of T^{cr} at the source, using feedback packets only from CR. When the transmission rate reaches $E(U^{cr}) + 4U^{cr}_\sigma$[10], where $E(U^{cr})$ and U^{cr}_σ are the average and deviation of the current CR's TRAC respectively, we assume that bottleneck becomes fully loaded and start to count. Let the current time be t_0. At a later time t_1, the first CI(μ) since t_0 arrives at the source from the CR. $t_1 - t_0$ is then a sample of T^{cr} and we update the average and deviation of T^{cr} with

[10]According to Chebychev Inequality, about 94% of the random sample are less than this value.

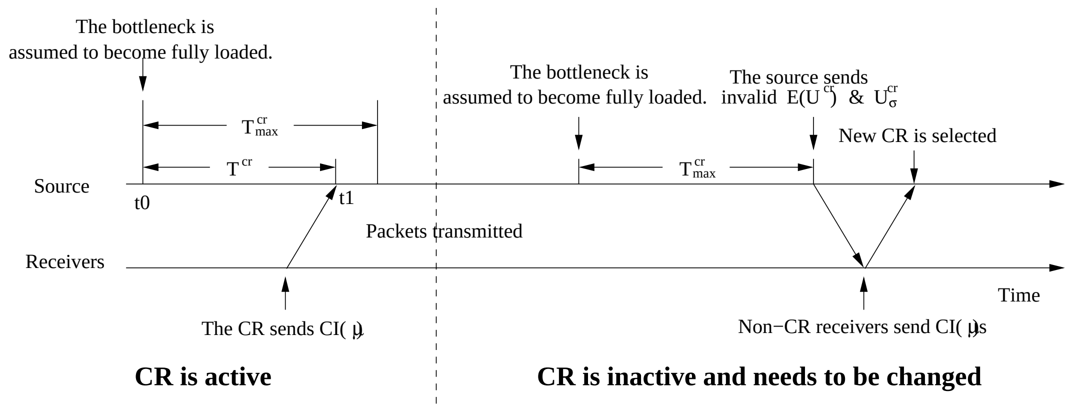


Figure 3.2: Congestion Representative (CR) Update Procedure

EWMA. T^{cr}_{max} is the average value of T^{cr} plus eight times its deviation[11]. When the CR is not active, for the duration of T^{cr}_{max} since we start to count, no CI(μ) will be received by the source. The source then requests feedback from other receivers for new CR selection, as described in Section 3.1.4.

3.1.4 Feedback Suppression by Receivers

Effective feedback suppression can reduce the risk of feedback implosion, and allow a multicast congestion control scheme to be used for large groups. In ORMCC, the source conveys the average $E(U^{cr})$ and the deviation U^{cr}_{σ} of the CR's TRAC to receivers whenever the CR is updated or $E(U^{cr})$ and U^{cr}_{σ} are changed. Only if a receiver's own average TRAC $E(U)$ is less than $E(U^{cr}) - U^{cr}_{\sigma}$[12], will it send CI($\mu$)s. Note that a receiver does not maintain TRAC deviation U_{σ} because it is computed by the source with the help of TRAC in CI(μ)s.

$E(U^{cr})$ and U^{cr}_{σ} conveyed by the source can be set to infinitely large values so that all receivers can send CI(μ)s. This is needed when the current CR is inactive and the source needs to trigger feedbacks from all receivers for new CR selection (Figure 3.2).

To avoid the pathology that many non-CR receivers send CIs at the same time, each feedback packet from non-CR receivers is delayed for a random period t, where t is in the interval of [0, T] and conforms to a truncated exponential distribution [84],

$$f_X(x) = \begin{cases} \frac{1}{e^{\lambda}-1} \cdot \frac{\lambda}{T} e^{\frac{\lambda}{T}x}, & 0 \le x \le T \\ 0, & \text{otherwise} \end{cases}$$

[11]We choose the value of 8 to be conservative.

[12]We don't use $E(U) \ge E(U^{cr})$ because we want to be conservative and keep CR stable.

T is set to $2RTT_{max}$ in our simulations, where RTT_{max} is the maximum RTT the source has ever seen. By the delayed sending time of a CI at a receiver, if an incoming packet indicates that a new CR has been chosen and the CR's statistics ($E(U^{cr})$ and U_σ^{cr}) suppresses CIs from this receiver, the scheduled CI will be canceled. In this way, synchronous bursts of CIs are avoided.

It should be noticed that the feedback suppression in our scheme is based on statistics (i.e. TRAC) comparison, and no knowledge of the whole group is needed. The major difference from previous probabilistic timer-based feedback suppression schemes is that in those schemes, scheduled feedback packets are canceled by the arrivals of echo packets from the source. Our feedback suppression mechanism is effective since the amount of CI(μ)s sent to the source is independent of the total receiver number. More insight will be given in the theoretical analysis at Appendix C.4.

3.1.5 Rate Adaptation

ORMCC is a rate-based scheme, using the policy of additive increase and multiplicative decrease (AIMD). That is, if there are no CI(μ)s from the CR, the transmission rate is increased by s/RTT per RTT, where s is the packet size, RTT is that between the source and the CR. If a CI is received from the CR, let the TRAC in this CI be μ, we adjust the transmission rate to the minimum of $\beta\mu$ and the current rate. CI(μ)s from other non-CR receivers will be ignored, and at most one rate cut is allowed per RTT.

The rate reduction factor β is an important parameter of ORMCC. The larger the β, the more aggressive is ORMCC. To keep ORMCC TCP-friendly, from a later discussion in Appendix C.2, we will see that β must be at least 0.5. Moreover, the exact value of β depends on how ORMCC is implemented. According to the simulation and experiment results, we suggest $\beta = 0.65$ for implementation on user level, and $\beta = 0.75$ for implementation in system kernel. The reason is that, if ORMCC is implemented on user level, due to the coarseness of timers, its traffic is more bursty than that of TCP running in kernel. To cancel that effect, β should be set lower.

3.1.6 RTT Estimation

Unlike a NAK, which includes the sequence number of a lost packet, a CI(μ) of ORMCC includes the sequence number of a packet upon the arrival of which packet losses are detected.

The source calculates the difference between the sending time of this packet and the arriving time of this CI(μ) to get a sample of RTT. By doing this, we avoid the unnecessary delay between the supposed arriving time of a lost packet and the time of its loss being detected. Nevertheless, since CI(μ)s are sent only when packet losses occur, RTT estimated by CI(μ)s includes the maximum bottleneck queueing delay and thus is still the upper bound. On the other hand, ACKs as those in TCP may or may not include bottleneck queueing delay. Therefore, on average, RTT estimated by CI(μ)s is larger than that by ACKs under the same situation. In fact, this is the reason why we set β to some value higher than 0.5.

As we can see from the details above, ORMCC has the following features:

- $O(1)$ **States** The states maintained by source and receivers are $O(1)$. That is, the number of states is constant and independent of the number of receivers in a multicast session.
- **Simple Operations** Operations of source and receivers are all simple, without requiring intense computation. In particular, there is no need to do per-receiver RTT estimation.
- **Effective Feedback Suppression** With our statistics-based feedback suppression mechanism in place, the amount of feedbacks is independent of the total number of receivers.

The pseudo code of ORMCC's algorithm is provided in Appendix B for reference. The code for *ns*-2 and Unix can also be found at [120].

3.2 Properties about ORMCC Performance

It is desirable to check the performance of a multicast congestion control scheme by theoretical analysis. We have done that for ORMCC to show the following properties:

Property 1 *ORMCC is capable of tracking the slowest receiver and select it as* CR (Congestion Representative) *to direct rate adaptation.*

proof (See Appendix C.1.)

Property 2 *ORMCC is TCP-friendly on the representative path, i.e. the path between the source and the CR.*

proof (See Appendix C.2.)

Property 3 *ORMCC is immune to drop-to-zero problem, i.e. the sending rate won't be reduced more than enough and converge toward zero upon asynchronous congestion.*

proof (See Appendix C.3.)

Property 4 *Feedback suppression in ORMCC is very effective.*

proof (See Appendix C.4.)

3.3 Simulations and Experiments

We have run simulations on *ns*-2 [2] (for small topologies) and ROSS [19] (for large topologies) as well as experiments in Emulab [113] to validate the performance of ORMCC. The simulations checked the following aspects:

(1) TCP-Friendliness

(2) Drop-to-zero avoidance

(3) Performance at CI loss

(4) Multiple bottleneck fairness

(5) Slowest receiver tracking

(6) Feedback suppression

We also compared the performance of ORMCC with PGMCC[94] and TFMCC[114] on both *ns*-2 [13] and ROSS [14]. In all simulations, the data packet size is 1000 bytes, the bottleneck buffer size is 50K bytes, the initial RTT is 100 milliseconds.

For experiments on real systems in Emulab[113], we implemented ORMCC on top of UDP as a user level program. TCP-friendliness and drop-to-zero behavior are tested. The result is presented at the end of this section.

[13] we used ns2.1b7a for ORMCC and TFMCC, but used ns2.1b5 for PGMCC, due to the restriction of its source code.

[14] The ROSS implementations of PGMCC and TFMCC are based on their *ns*-2 source code.

3.3.1 TCP-Friendliness and Drop-To-Zero Avoidance

We used a star topology (Figure 3.3) to generate asynchronous and independent congestion on different paths. There are 129 ends nodes in the topology. Between each pair of source i and receiver i ($i = 1 \ldots 64$), there are one TCP Reno flow and one single-receiver ORMCC flow. Furthermore, there is a multi-receiver ORMCC flow from source 65 to all 64 receivers. Therefore, on a path between the router and any receiver, the multi-receiver ORMCC flow competes with a TCP flow and a single-receiver ORMCC flow.

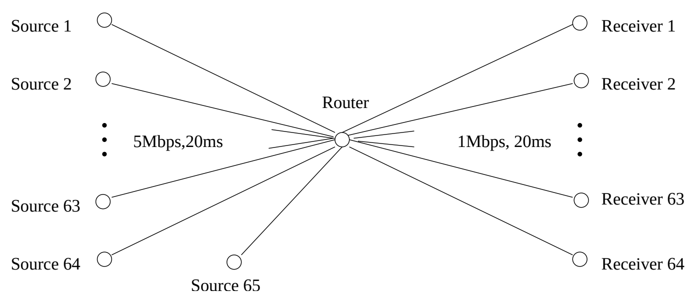


Figure 3.3: 64-Receiver Star Topology with TCP Background Traffic

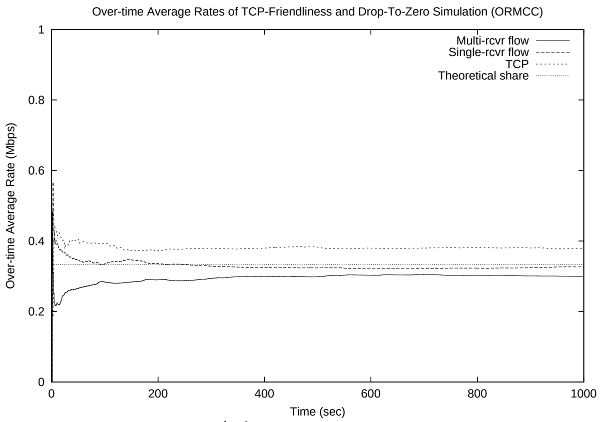
We randomly chose a receiver node and plot in Figure 3.4(a) the over-time average rates [15] of all three flows going to it. The fact that the average rates of the TCP flow, the single-receiver ORMCC flow and the multi-receiver ORMCC flow are close to each other indicates that *(1) ORMCC is TCP-friendly, and (2) ORMCC does not suffer from drop-to-zero problem.*

We also conducted experiments on the same configuration for PGMCC[16] and TFMCC. Results in Figure 3.4 (b) and (c) show that but the average rates of their multicast flows deviate more from corresponding unicast flows.

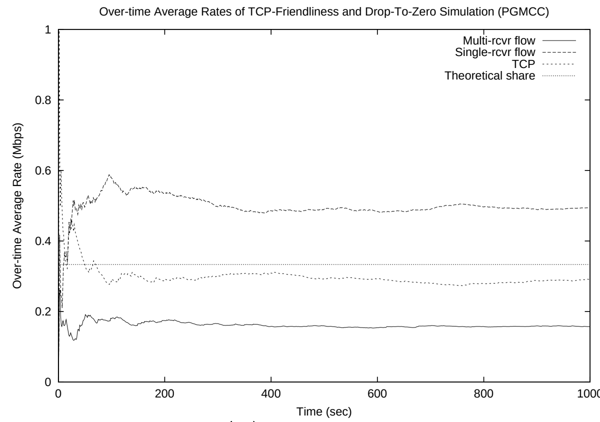
For ROSS simulations, we used a tree topology (Figure 3.5). Since there has not been TCP implementation on ROSS yet, we did not test TCP-friendliness on ROSS but focused on the drop-to-zero problem. The ORMCC multicast traffic is from the source at the root to all the receivers at the bottom of the tree. The last hop of every path from the source to the receiver is the bottleneck (1Mbps), and has some unicast flows on it as background

[15] Over-time average rate at time t is defined as the traffic volume between $[0, t]$ divided by t.

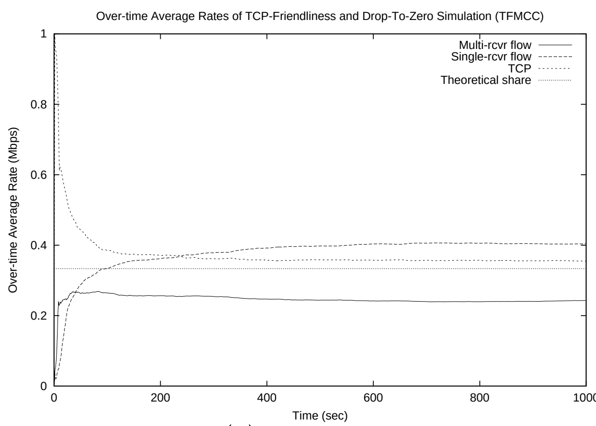
[16] For PGMCC simulations, since ns2.1b5 can only accommodate up to 128 end nodes, we can only have 63 pairs of unicast source and receiver instead of 64 pairs. Moreover, we only measure the sending rate of original data packets, because repair packets for PGMCC are routed by net elements to individual receivers whoever need them instead of all receivers. Nevertheless, the proportion of repair packets is less than 1/10 and is thus negligible. This has the same effect as measuring sequence number increment in [94].



(a) ORMCC



(b) PGMCC



(c) TFMCC

Figure 3.4: TCP-Friendliness and Drop-to-Zero Avoidance
(ORMCC is more TCP-friendly and avoids drop-to-zero better than PGMCC and TFMCC.)

traffic. The end-to-end propagation delay of both multicast and these unicast flows are set to be the same.

The first ROSS simulation has *ten thousand* receivers involved in an ORMCC multicast session. The background traffic consists of single-receiver ORMCC flows. Figure 3.6 shows that the average throughput over the simulation stays at a reasonable level. Four randomly chosen background traffic flows are plotted for comparison. Although the multicast throughput is slightly lower than those of the unicast flows, it is understandable because the multicast session always tracks the slowest receiver which changes over time.

The second ROSS simulation is used to compare the performance of ORMCC, PGMCC and TFMCC when they run along with each other. The topology in Figure 3.5 is still used but with different number of receivers. Three simulations with the receiver number of 10,

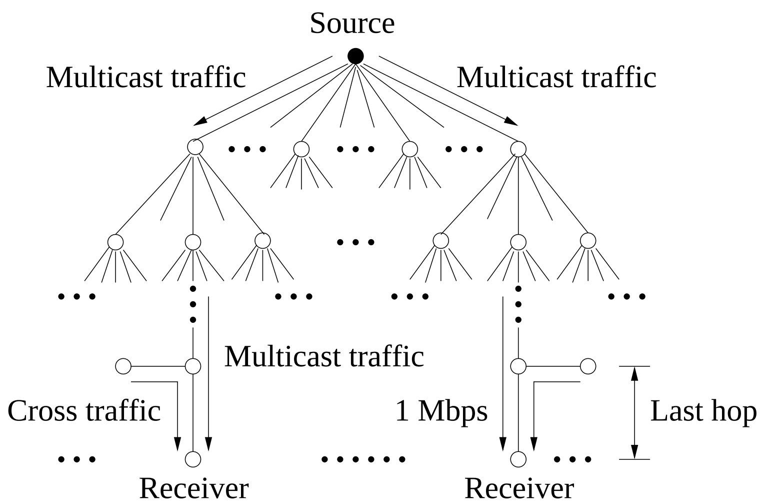


Figure 3.5: Tree Topology for Large-Scale Simulations in ROSS

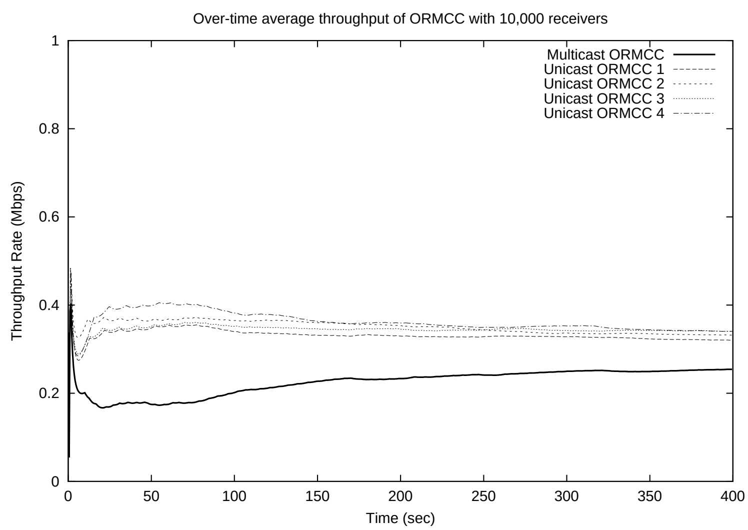
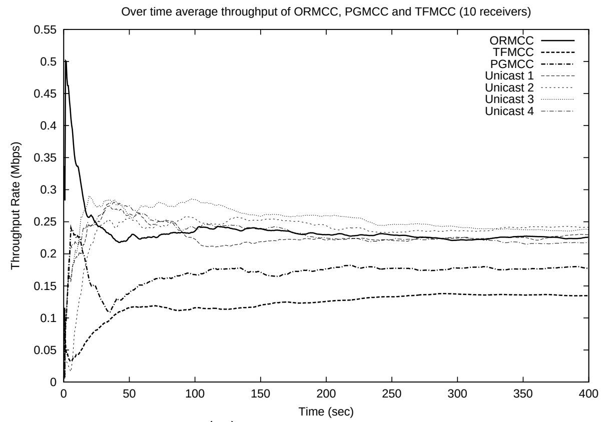
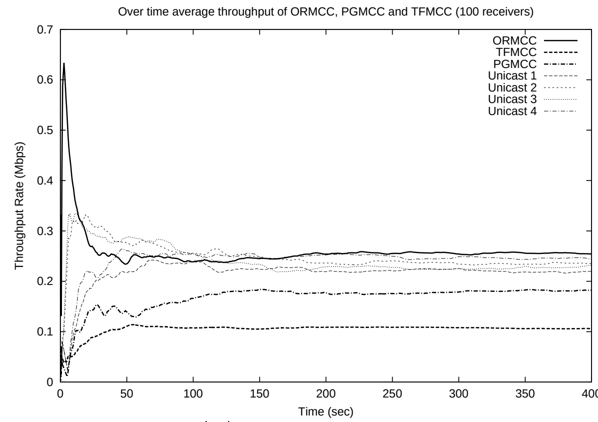


Figure 3.6: Drop-to-zero Avoidance in 10,000-Receiver Simulation
(ORMCC still avoids drop-to-zero at the presence of large number of receivers and independent bottlenecks.)

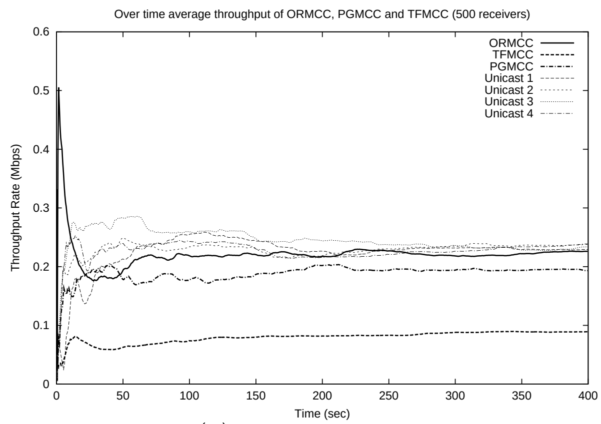
100 and 500 have been conducted. Single-receiver PGMCC flows instead of ORMCC flows are used for background traffic due to the close-to-TCP design of PGMCC. The series of figures (Figure 3.7) show that ORMCC always gets the approximately the same throughput as the unicast PGMCC flows do. The multicast PGMCC gets slightly lower but consistent throughput, while the throughput of TFMCC decreases as the number of receivers increases. As the result, we can see that ORMCC outperforms PGMCC and TFMCC in terms of drop-to-zero problem.



(a) 10 receivers



(b) 100 receivers



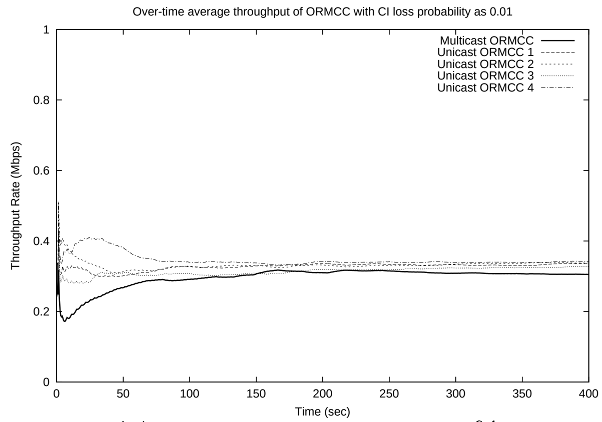
(c) 500 receivers

Figure 3.7: Over-Time Average Throughput of ORMCC, PGMCC and TFMCC Running Together

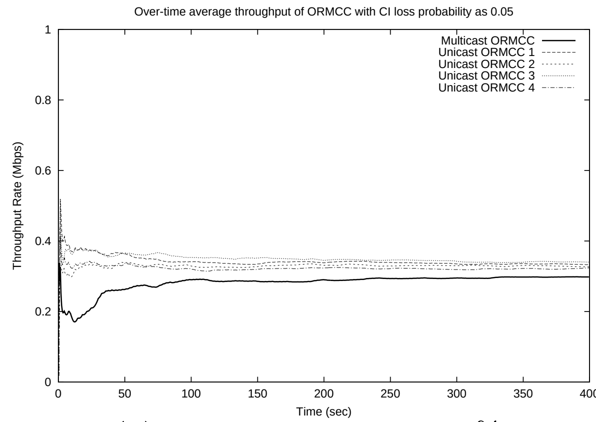
(When competing with each other, ORMCC gets best average throughput and is fairest with background unicast flows.)

3.3.2 Performance at CI loss

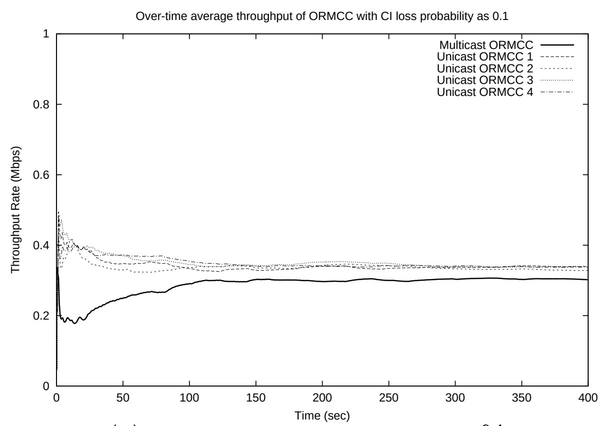
Since ORMCC heavily relies on CIs from the receivers, it is a concern whether CI losses can negatively affect the performance of ORMCC. To check this aspect, we ran a set of 500-receiver simulations on ROSS with different CI loss probabilities for the multicast flow, again using the topology of Figure 3.5 and single-receiver ORMCC flows for background traffic on the last hops. The results in Figure 3.8 show that ORMCC does not suffer from performance downgrade when up to 10% CIs are dropped. The reason is that, if a CI from CR is lost and the source does not reduce the sending rate, the bottleneck keeps being congested, packets keep being dropped and therefore, more CIs are sent from the CR. Some CIs finally arrive at the source and trigger rate reduction.



(a) CI loss proportion = 1%



(b) CI loss proportion = 5%



(c) CI loss proportion = 10%

Figure 3.8: Over-Time Average Throughput of ORMCC When There Are CI Losses
(Feedback packet loss does not affect ORMCC's performance.)

3.3.3 Multiple Bottleneck Fairness

In real world, there are usually more than one bottleneck on a path. It is desirable to check how long ORMCC flows compete with short ones and what kind of fairness ORMCC can achieve. we ran a simulation on Figure 3.9. There is a long multi-receiver multicast flow, going through two bottlenecks, from Src 1 to receivers in Group 1. There are also two short multicast flows going through only one bottleneck from Src 2 to Group 2 and from Src 3 to Group 3 respectively. Each group has 16 receivers. RED queues are used on the routers to reduce the effect of RTT estimation.

According to proportional fairness, the long ORMCC flow should get one-third of the bottleneck bandwidth, 0.33Mbps. The result in Figure 3.10 (a) shows that ORMCC achieves approximately proportional fairness. Similar fairness achieved by PGMCC and TFMCC in

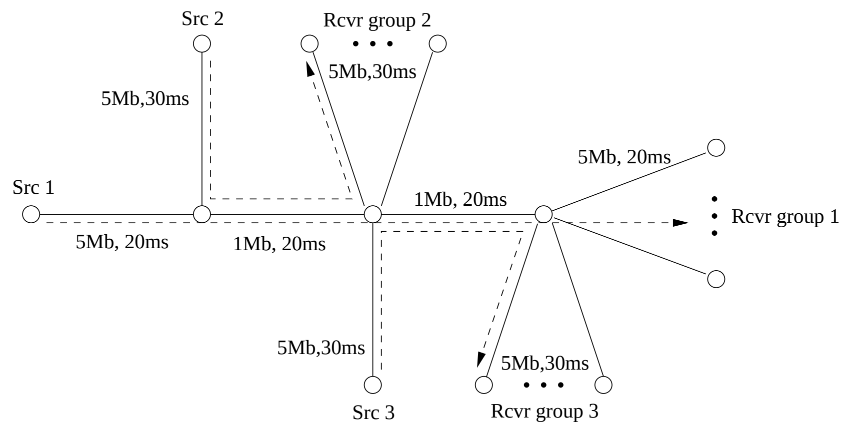
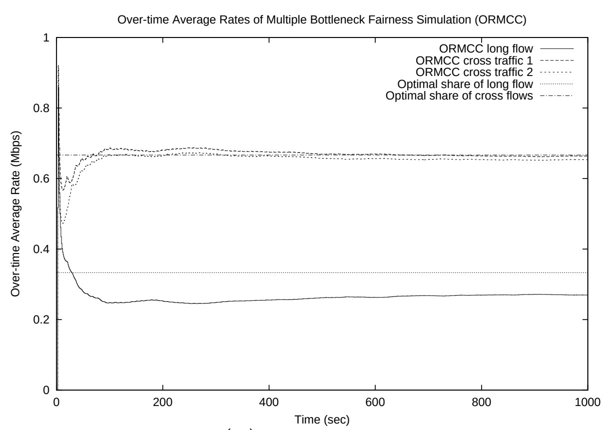
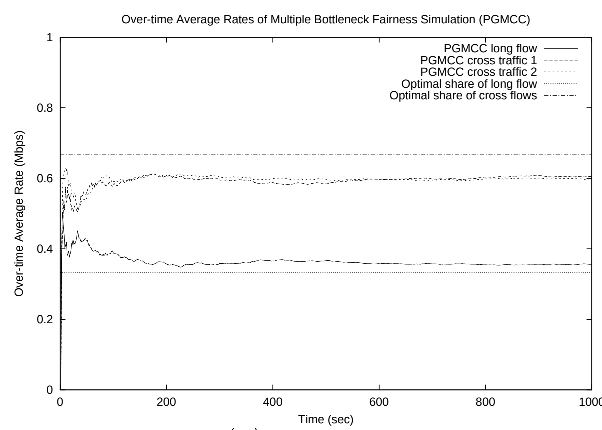


Figure 3.9: Linear Network with Multiple Bottlenecks (Totally 48 Receivers)

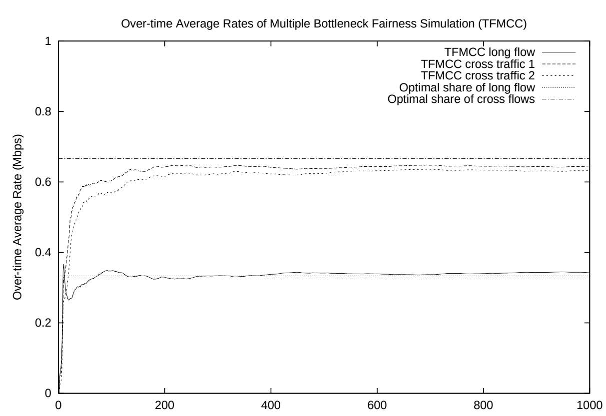
the same configuration is also shown in the figure.



(a) ORMCC



(b) PGMCC



(c) TFMCC

Figure 3.10: Fairness of Sharing Bottleneck Bandwidth
(ORMCC, PGMCC and TFMCC all achieve approximately proportional fairness.)

Table 3.1: Dynamics in Slowest Receiver Tracking Simulation

	0 - 200 sec	200 - 400 sec	400 - 600 sec	600 - 800 sec	800 - 1000 sec
Most congested bottleneck	Link 1	Link 2	Link 3	Link 2	Link 1
Supposed CR	Receiver 1	Receiver 2	Receiver 3	Receiver 2	Receiver 1

3.3.4 Slowest Receiver Tracking

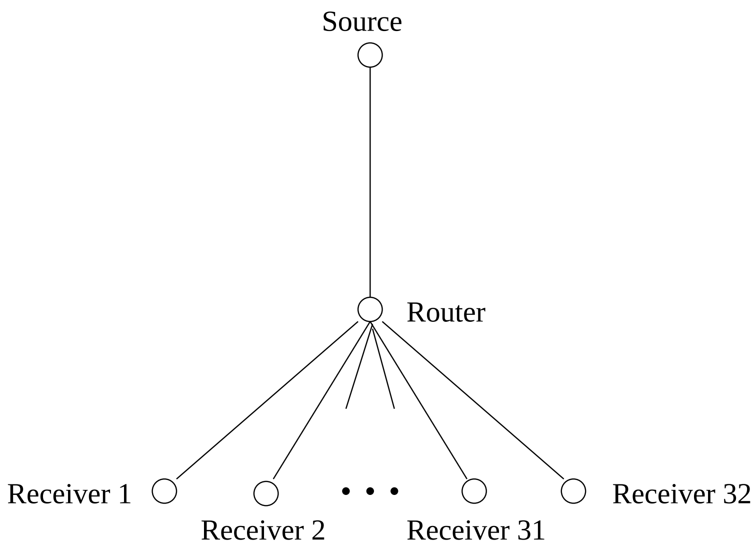


Figure 3.11: One-Level Tree with 32 Receiver Nodes

This simulation is used to test ORMCC's capability to quickly track the slowest receiver and select it as CR. In the tree topology of Figure 3.11, there is an ORMCC flow between the source and all the 32 receivers. There are three dynamically generated bottlenecks using TCP Reno flows. Denote link i as the link between the router and receiver i ($i = 1 \ldots 32$), each link has 2Mb bandwidth and 20 ms delay. The simulation time is 1000 seconds. During the whole simulation, one TCP flow runs on link 1; between 200th and 800th seconds, three TCP flows run on link 2; between 400th and 600th seconds, seven TCP flows run on link 3. The most congested bottlenecks and the supposed CRs at different time are shown in Table 3.1.

The dynamics include both conditions causing CR switches, i.e. (1) A slower receiver appears, (2) The current slowest receiver is absent. RED and drop-tail queue management policies are used separately in our simulations. Simulation results are shown in Figure 3.12 (a) and (b). Vertical dash lines show when the ORMCC source switched CR. We can see that ORMCC updates CR and adapts its transmission rate in a timely manner. Note that when using RED queues, the ORMCC source sometimes switched CR a little slower than drop-tail situation. The reason is because RED queue drops packets in a random manner, it takes longer for the slowest receiver to have a lower average TRAC measurement.

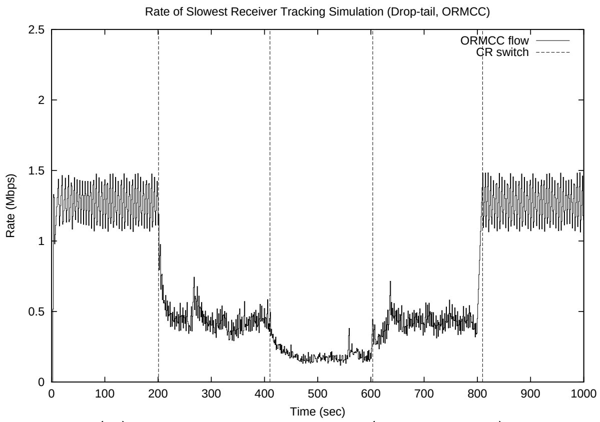
Under the same situation, as shown in Figure 3.12 (c),(d), (e) and (f), PGMCC and TFMCC also track the slowest receiver, though sometimes with more representative switches. We also noticed that there is much oscillation of PGMCC's rates due to its design of mimicking TCP, while the rates of ORMCC and TFMCC have similar smoothness.

Moreover, to further verify the mechanism of CR absence detection, we tested a pathological situation in ROSS, where any receiver stops sending CIs after being CR for 20 seconds. If the source can detect the in-activeness of CR and switch CR quickly, the sending rate will be adjusted lower in time and the bottlenecks won't be kept congested. The simulation used the configuration in Figure 3.5 and there are 500 receivers. By comparing the result in Figure 3.13 and Figure 3.6, we can see that the throughput patterns of both simulations are the same, which indicates that the mechanism we designed to detect CR absence is effective.

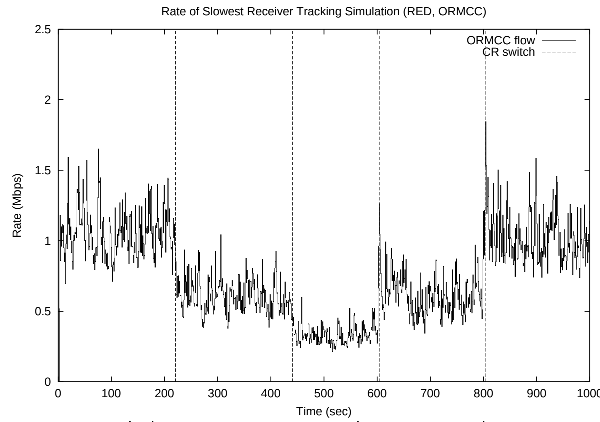
3.3.5 Feedback Suppression

To check the effectiveness of the feedback suppression mechanism in ORMCC, we refer back to the *ns*-2 simulation of TCP-friendliness and drop-to-zero avoidance. In totally ten simulations, the average total number of CI(μ)s sent by all receivers is 816 (standard deviation is 14.8), the average total number of suppressed CI(μ)s is 34601 (standard deviation is 422.0). The average number of CI(μ)s would have been sent by a receiver if without suppression, is $(34601 + 816)/64 \approx 553$. As we discussed in the analysis (Appendix C.4), realistic measurement error can lead to a little bit more CIs. Since $816 < 2 \times 553$, we can still say that the overall feedback volume with suppression is approximately equal to that from a single receiver if without suppression. The high ratio of CI(μ)s suppressed, $34601/(34601 + 816) \times 100\% \approx \mathbf{97.7}\%$, shows that *our feedback suppression is very effective.*

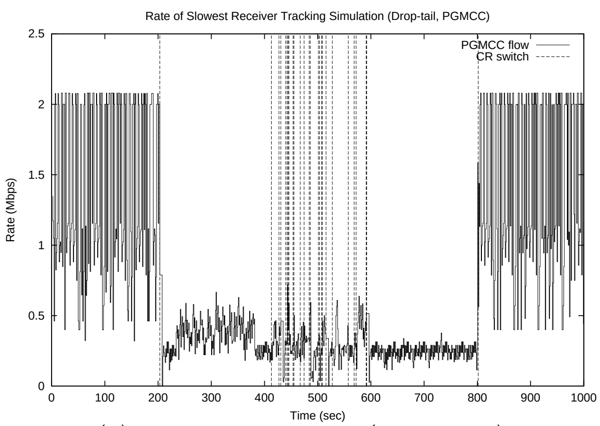
For comparison, in a typical PGMCC simulation with the same configuration, the total number of feedback packets received by the source is 74830 (NAK 55222, ACK 19608); for TFMCC, it is 5344. Their feedback volume is much larger.



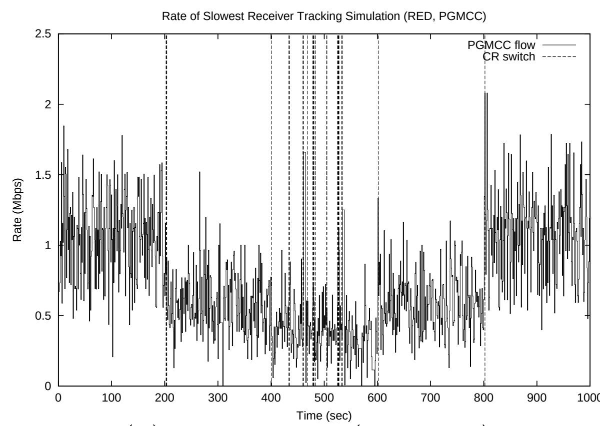
(a) Droptail Queue (ORMCC)



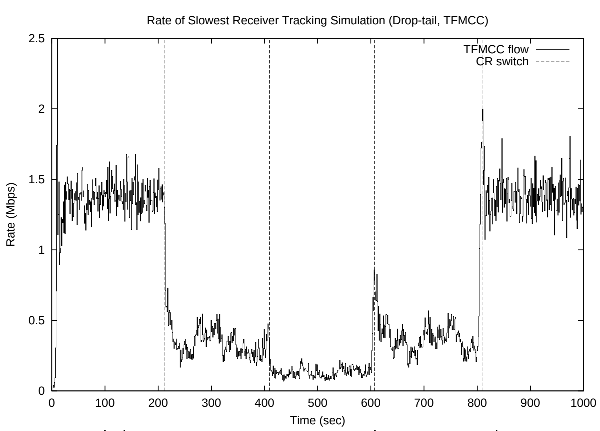
(b) RED Queue (ORMCC)



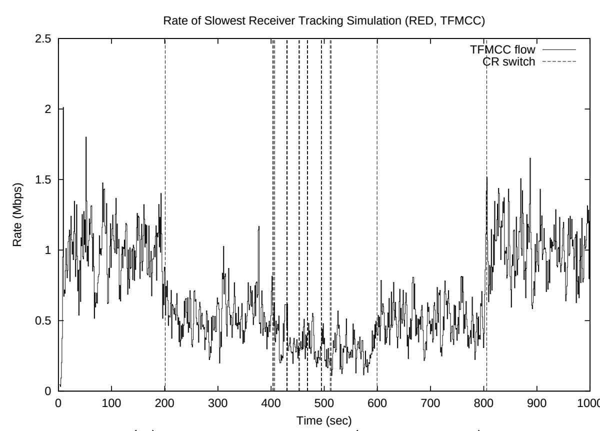
(c) Droptail Queue (PGMCC)



(d) RED Queue (PGMCC)



(e) Droptail Queue (TFMCC)



(f) RED Queue (TFMCC)

Figure 3.12: Capability of Tracking The Slowest Receiver
(ORMCC tracks the slowest receiver in time with fewer representative switches than PGMCC and TFMCC.)

3.3.6 Comparision with PGMCC and TFMCC in Heterogeneous Dynamic Network

As the last simulation, we constructed a dynamic network to test the stability and adaptability of ORMCC, again compared with PGMCC and TFMCC. In Figure 3.14, each

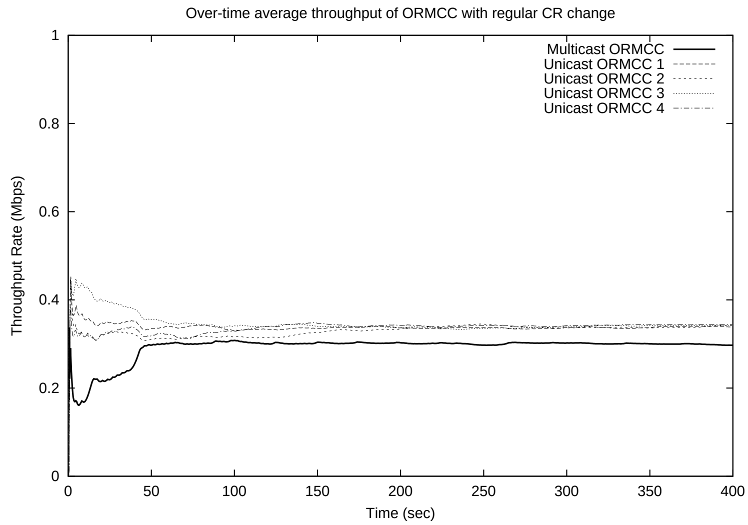


Figure 3.13: Average Throughput With Frequent CR Quitting
(ORMCC switches CR in time if the CR stops reaction.)

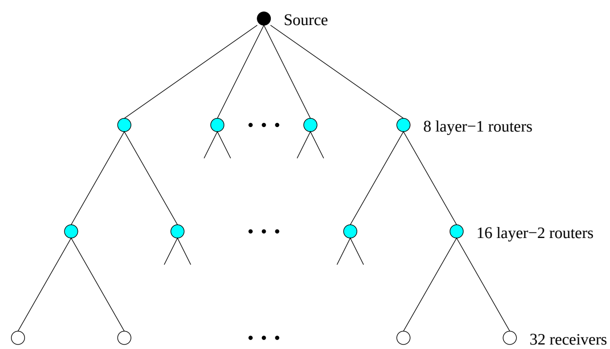


Figure 3.14: Heterogeneous Dynamic Network

link has 2Mbps bandwidth. 2 links at the first level, 4 links at the second level, and 8 links at the third level has 200ms delay. All other links have 20ms delay, while on any path between the source and a receiver, there is at most one link of 200ms delay. On each link, two TCP Reno flows are randomly turned on and off according to Pareto distribution with average value of 60 seconds, and two UDP flows of 200Kbps on and off with average value of 1 second. These flows dynamically generate bottlenecks and make the network heterogeneous. At last, there is a multicast flow from the source to all the receivers. The multicast flow can use either ORMCC, PGMCC or TFMCC. Therefore, at any moment, there are at most five flows on any link: one multicast flow, two TCP flows and two UDP flows, and the multicast flow is expected to get an average throughput rate of 500Kbps or so.

We ran 10 simulations for each of the three schemes. In Table 3.2 we can see that with smaller amount of feedbacks, ORMCC can achieve higher throughput. That means, *ORMCC*

Table 3.2: Comparison of Average Throughput and Feedback Volume in Heterogeneous Dynamic Network

	ORMCC	PGMCC	TFMCC
Average Throughput	**415.4** Kbps	126.6Kbps	226.7Kbps
Average Feedback Number	**866.9**	5009.6 (NAK only)	3312.9

(ORMCC has higher throughput and needs less feedback in heterogeneous and dynamic environment.)

has better stability and adaptability in heterogeneous and dynamically changing networks.

3.3.7 TCP-Friendliness and Drop-To-Zero Avoidance Test in Emulab

To do a preliminary check of ORMCC's performance in real world and understand the issues of implementation, we implemented ORMCC in C++ on top of UDP as a user level program and ran it in Emulab [113] [17]. The operating system we used is RedHat 7.1, and mrouted[81] is used for multicast routing. On the topology shown in Figure 3.15, the links between the peripheral nodes and their parent nodes have 50 ms propagation delay and 1.0Mbps bandwidth. All other links have 100Mbps bandwidth and 0 ms propagation delay.[18] From the center node to any peripheral node, there is a single-receiver ORMCC flow and a TCP flow. Also, there is a multi-receiver ORMCC flow from the center node to all 36 peripheral nodes. The experiment time is 1000 seconds.

Since we implemented ORMCC on user level, its traffic is more bursty than that of TCP running in kernel. As described in Section 3.1.5, the rate cut factor (β) should be adjusted accordingly. We tried three different values: 0.5, 0.65 and 0.75. According to Figure 3.16 [19], when $\beta = 0.5$, the TCP flows got more bandwidth; when $\beta = 0.75$, the ORMCC flows are more aggressive. $\beta = 0.65$ works the best, where the multi-receiver ORMCC flow got almost the same bandwidth as TCP flows did, and thus TCP-friendly. Moreover, among all the values tested for β, the average rates of the multi-receivers ORMCC flow and single-receiver ones are always close, showing that ORMCC is immune to drop-to-zero problem.

3.4 Summary

Another single-rate multicast congestion control scheme ORMCC is presented in this chapter. It uses an conventional concept of representative named *Congestion Representative (CR)* here. However, by leveraging a new metric *TRAC*, the whole scheme is simple while

[17] Emulab is accessible at http://www.emulab.net.
[18] The propagation delay here means the delay artificially introduced by some particular software.
[19] The mean and confidence interval are calculated out of all the flows of the same category.

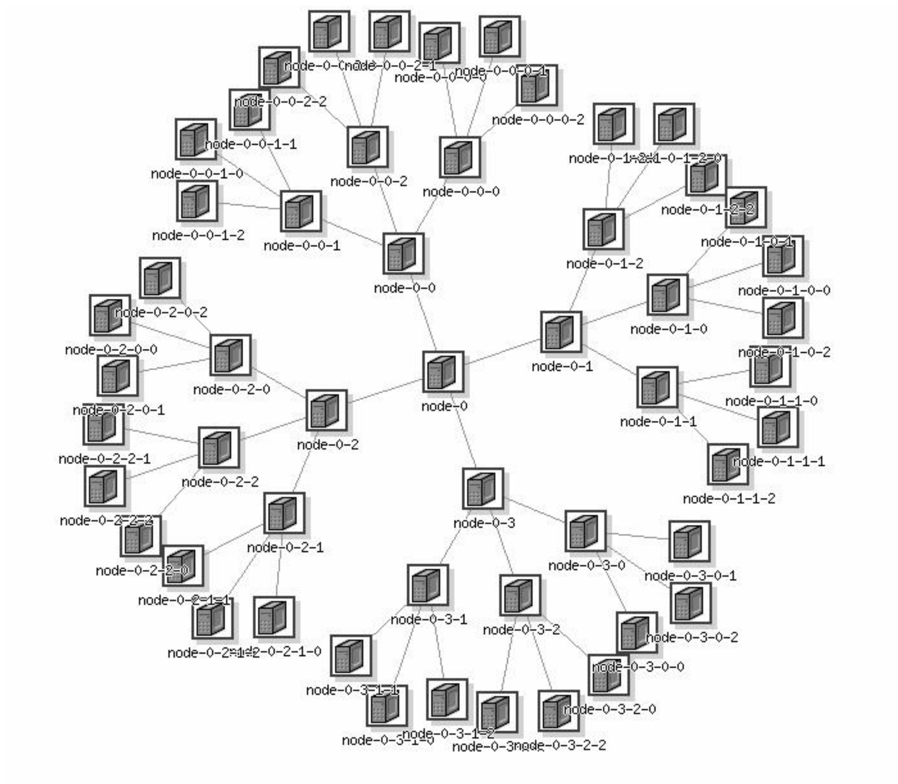
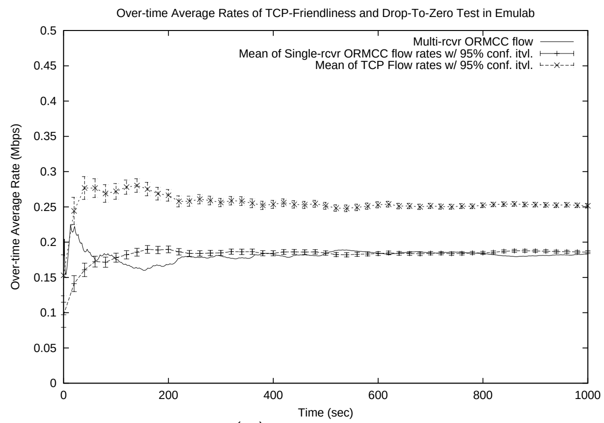
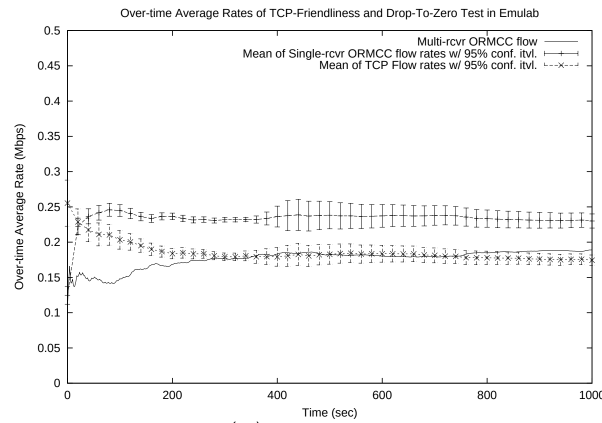


Figure 3.15: Topology Used in Emulab for TCP-Friendliness and Drop-to-Zero Test (36 Receiver Nodes)

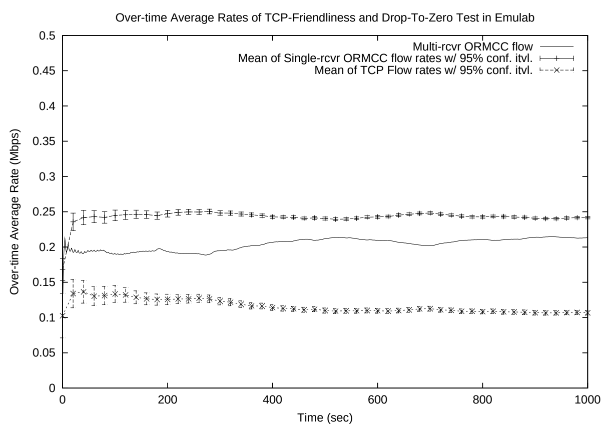
still capable of effectively addressing the problems of TCP-friendliness, drop-to-zero, slowest receiver tracking and feedback suppression. The states maintained by source and receivers are $O(1)$; operations of source and receivers are all simple without requiring intense computation, in particular there is no need to measure RTTs between all receivers and the source; statistics-based feedback suppression is highly effective. To confirm the performance of ORMCC, we have not only provided theoretical analysis, but also run simulations to compare our scheme with PGMCC[94] and TFMCC[114]. Furthermore, we have implemented ORMCC on top of UDP and run it on real systems in EMulab[113]. The results are promising. As an emphasis, we summarize the comparison with PGMCC and TFMCC in simulations in Table 3.3. We can see that ORMCC achieves better performance than PGMCC and TFMCC under most situations. Code for ns-2 and Unix is available at [120] for public test.



(a) $\beta = 0.5$
ORMCC is less aggressive than TCP



(b) $\beta = 0.65$
ORMCC is TCP-friendly



(c) $\beta = 0.75$ ORMCC is more aggressive than TCP

Figure 3.16: TCP-Friendliness and Drop-to-Zero Test Result in Emulab
(ORMCC is TCP-friendly and avoids drop-to-zero on real systems with proper parameter setting.)

Table 3.3: Summary of ORMCC Performance Compared with PGMCC and TFMCC in Simulations

Simulation Name	*Performance Comparison*	*Figure/Table*
TCP Friendliness and Immunity to Drop-to-Zero	Better than PGMCC and TFMCC	Fig. 3.4
Multiple Bottleneck Fairness	Comparable with PGMCC and TFMCC	Fig. 3.10
Slowest Receiver Tracking	Better than PGMCC and TFMCC	Fig. 3.12
Feedback Suppression	Better than PGMCC and TFMCC	Table 3.2
Performance in Heterogeneous Dynamic Network	Better than PGMCC and TFMCC	Table 3.2

(In general, ORMCC outperforms PGMCC and TFMCC.)

CHAPTER 4
GMCC: Generalized Multicast Congestion Control

In this chapter we counter the multicast congestion control problem in the situation as follows:

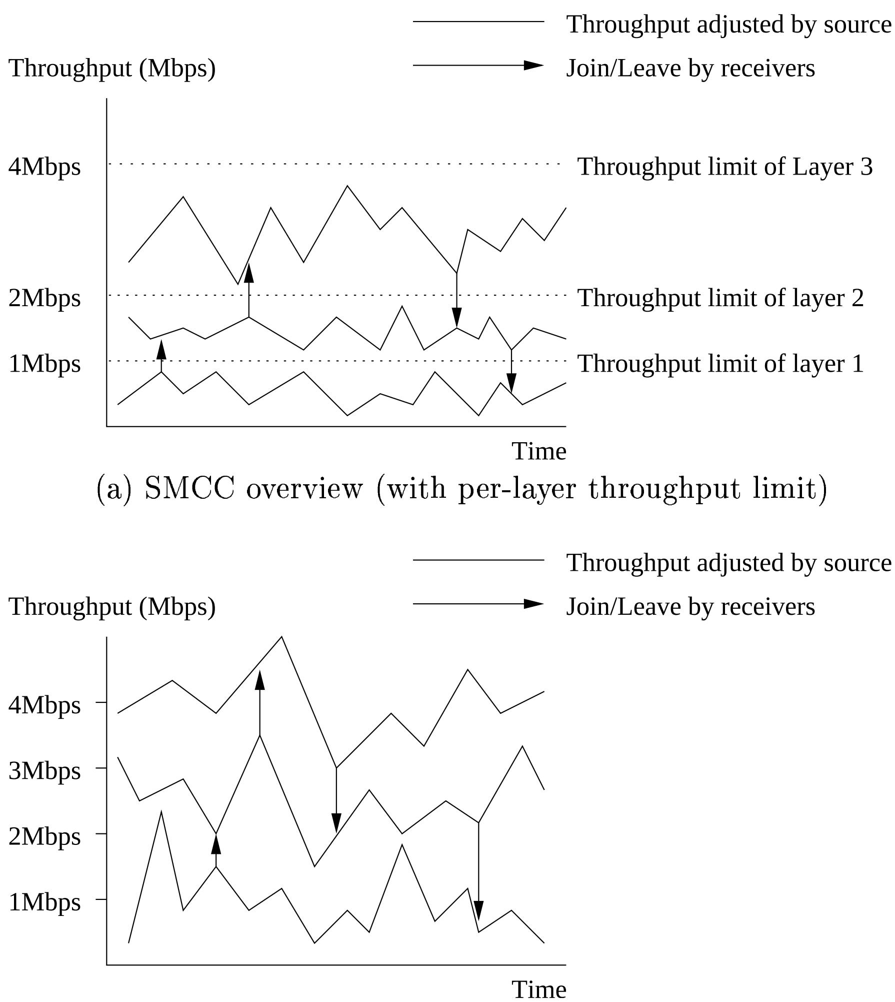
> *Support is allowed at receiver side, and unlimited multicast groups are allowed for a multicast session.*

This situation will usually be seen when we are using the SSM (source specific multicast) model [44]. SSM allows each source to have almost infinitely many multicast groups. Therefore multicast group address resource won't pose a serious problem.

By developing a multi-rate framework and incorporating an enhanced version of ORMCC into it, we have a new solution for this kind of situations. The new scheme provides the multirate feature at low complexity by encompassing a set of independent single-rate sub-sessions (a.k.a layers). Various receivers can subscribe to different subsets of these layers to achieve different throughput. Since single-rate congestion control is just one of the special cases of this new scheme, we named it *Generalized Multicast Congestion Control*, in short, GMCC.

GMCC is similar to a recently proposed scheme SMCC [59] but with certain advantages. SMCC is also a hybrid of single-rate and multi-rate multicast congestion control. It combines a single-rate scheme TFMCC[114] with the receiver-driven idea. In each layer, the source adjusts sending rate *within a certain limit* based on TFMCC, and receivers join or leave layers cumulatively according to their estimated maximum receiving rates using TCP throughput formula [85]. Since the flows in each layer are adaptive to network status, the number of join and leave operations are greatly reduced. The congestion control is more effective.

SMCC requires static configuration of the maximum sending rates for each layer. For example, the lowest layer is set to 1Mbps maximum, the second layer is set to 2Mbps maximum, and the i-th layer is set to 2^{i-1}Mbps. A receiver with an estimate of maximum throughput rate as 3Mbps needs to join the lowest two layers. This static setting can negatively affect the performance of SMCC. Consider the following scenario: with the settings above, all receivers have their estimated throughput rate below 1Mbps. Some receivers are behind 100Kbps bottlenecks, some behind 300Kbps, and others behind 800Kbps. Since there is only one layer for bandwidth less and equal to 1Mbps, all these receivers have to receive at a single rate. That means SMCC is degraded to a single-rate scheme under this situation. Consider another scenario: still with the above layer settings, all receivers have 100Mbps bandwidth. To fully utilize their bandwidths, they will need to join seven layers. Obviously, if the source is configured properly, only one layer is need. Therefore, six layers are redundant due to the mis-configuration.



(a) SMCC overview (with per-layer throughput limit)

(b) GMCC overview (no per-layer throughput limit)

Figure 4.1: Qualitative Comparison of SMCC and GMCC
(GMCC without per-layer restriction is more flexible than SMCC.)

GMCC solves these problems while having the merits of SMCC. Figure 4.1 shows the difference between SMCC and GMCC visually. With GMCC, in the first example scenario above, receivers will be able to receive data at 100Kbps, 300Kbps and 800Kbps respectively, and in the second example scenario, only one layer will be used. All these are done without changing any setting of the GMCC source.

In brief, GMCC has the following advantages:

(1) It is *fully* adaptive. The sending rate in each layer can be adjusted without rigid limits. Together with the automatically adjusted number of layers, it always allows heterogeneous receivers to receive at different rates.

(2) The number of layers used is just enough to accommodate the differences among the throughput desired by receivers. No redundant layers are used.

(3) It is not coupled with equation-based rate control mechanism such as TFMCC. The rate control mechanism at source can be replaced by others based on representative (the most congested receiver).

GMCC also addresses the issue of starting and stopping traffic within layers depending on whether there are receivers in the layers.

In the remainder of this chapter, we will describe the details of GMCC, and show some simulation results to demonstrate the performance of GMCC.

4.1 Scheme Details

The goal of multi-rate multicast congestion control is to fully the available bandwidth on different paths between the source and receivers. One key issue is then how and when a receiver joins or leaves a layer to increase or decrease its total throughput rate. The second issue is how the source controls the throughput in each layer. The basic ideas of GMCC solutions to these issues are the following:

- In each layer, the source chooses a most congested receiver as *congestion representative* (CR) and adjusts the sending rate of this layer according to the CR's feedback (Section 4.1.2).

- The source starts traffic in a layer when the first receiver joins and stops traffic in a layer when the last receiver leaves (Section 4.1.3).

- Each receiver joins layers cumulatively, and is allowed to be the CR of at most one layer.

- When a receiver detects that it is much less congested than the most congested receiver (i.e. the CR) in the highest layer it has joined, meaning it can potentially receive at a higher rate, it joins an additional layer *successively* (Section 4.1.4).
- When a receiver detects that it is the most congested receiver in more than one layer, which means it confines or can potentially confine the sending rates of more than one layer, it leaves the highest joined layer (Section 4.1.5).
- Receivers make decisions of join and leave based on statistics. Statistics can be used only if (1) At least a certain number of samples have been collected, and (2) Every layer has a CR.

As shown in the above ideas, it is important for a receiver to detect whether it is more congested than another. We propose to use *Throughput Attenuation Factor* (TAF) for this purpose described in next section. After that, we describe various aspects of the GMCC scheme.

4.1.1 Throughput Attenuation Factor

Throughput Attenuation Factor (TAF) [20] is a metric *measured at the receiver side* to indicate how congested the receiver is. It comprises two parts, *Individual Throughput Attenuation Factor* (ITAF) and *Congestion Occurrence Rate* (COR), each describing a different aspect of congestion.

4.1.1.1 Individual Throughput Attenuation Factor

ITAF is defined as

$$1 - \frac{\mu}{\lambda}$$

measured only in congestion epochs (A congestion epoch is an event when one or more consecutive packets are lost.[21]) μ is the instantaneous output rate and λ is the rate of input generating this portion of output. It shows how much proportion of input is lost during an instance of congestion, and therefore indicates *how serious* this instance of congestion is.

ITAF may be measured in the following way in implementation: Each data packet carries the instantaneous sending rate information, assumed to be λ_n for the packet of

[20] This section is a self-contained overview of our technical report [63] which covers more details.

[21] We assume that packet loss is due to congestion only.

sequence number n. When a packet of sequence number n arrives, the receiver divides this packet size by the latest packet arriving interval and gets the instantaneous receiving rate μ_n. If the receipt of sequence number n indicates a packet loss, a ITAF is obtained as

$$1 - \frac{\mu_m}{\lambda_m}$$

where m is the received sequence number immediate prior to n.

4.1.1.2 Congestion Occurrence Rate

COR is defined as the reciprocal of the interval between two consecutive congestion epochs. For instance, if the loss of packet n and $n+i$ ($i > 1$) is detected at time t_1 and t_2 respectively (with the packets from $n+1$ to $n+i-1$ received), then a sample of COR would be

$$\frac{1}{t_2 - t_1}$$

COR shows how frequently congestion happens.

With ITAF and COR defined, TAF is the product of these two factors, i.e.

$$TAF = ITAF \times COR$$

The larger TAF, the more congested is a receiver. Usually the samples of ITAF and COR change abruptly (e.g. due to bursty loss). Therefore, we collect a certain number[22] of ITAF and COR samples, average the samples and use the mean value for TAF calculation. That means, we compare congestion on an average sense. For more detailed analysis of TAF, please refer to our technical report [63]. In GMCC, each receiver measures its own TAF and maintains the mean Θ and standard deviation Θ^σ of the latest N TAF samples for the purpose of TAF comparison.

4.1.2 Sending Rate Control Within A Layer

Given a layer with active receivers, the source chooses a most congested receiver (e.g. Receiver 2 in Figure 4.2) in this layer as congestion representative (CR) and uses its feedback

[22] A number of 30 is used in our simulations.

Table 4.1: Some Key Symbols in Section 4.1

Symbol	Meaning
Θ_i	Average TAF of receiver i
Θ_i^σ	Standard deviation of receiver i's TAF
θ	Average ITAF of a receiver's highest joined layer measured during periods without bandwidth shifting
θ'	Average ITAF of a receiver's highest joined layer measured during bandwidth shifting periods
N	Number of TAF/ITAF samples kept for calculation
J	Number of positive TAF/ITAF comparison results required to join an additional layer

for rate adaptation. When the CR detects packet loss, it sends feedback packets called *congestion indications* (CIs) back to the source that decreases the sending rate by half. To avoid reducing rate too much, the source decreases the sending rate at most once per SRTT (smoothed RTT). The samples of RTT are collected by the source at the receipt of CIs. The value of a sample is the time difference between the CI arrival and the departure of the data packet triggering the CI. SRTT is calculated by exponential weighted moving average formula: SRTT = (1 - ε) SRTT + ε RTT ($0 < \varepsilon < 1$, we use 0.125). At the absence of CIs, the sending rate is increased by $s/SRTT$ each SRTT, where s is the packet size.

To choose or update a CR, the source needs to compare the TAF statistics from receivers sent in by CIs. Given receiver i and j, and j is the CR, assume their average TAFs are Θ_i and Θ_j, their TAF deviations are Θ_i^σ and Θ_j^σ respectively. If the following condition is satisfied, receiver i is chosen as the CR (When there is no CR yet, Θ_j and Θ_j^σ can be adjusted to make the condition always true.) This condition and those to appear later are all based on statistical inference [78].

$$\Theta_i > \alpha_1 \Theta_j + \alpha_2 \sqrt{\frac{{\Theta_i^\sigma}^2 + (\alpha_1 \Theta_j^\sigma)^2}{N}} \tag{4.1}$$

α_1, α_2 are configurable parameters. In our simulations, α_1 is set to 1.25 since we want to bias toward the current choice of CR to avoid unnecessary oscillation, α_2 is set to 1.64 for a 90% confidence level.

Although the source needs to perform TAF comparison, it is not necessary for all receivers to send CIs to the source. In GMCC, receivers check the condition of (4.1) in advance. Only if the condition is true do they send CIs. The information of CR is broadcast to all receivers for the checking beforehand.

Notice that there is no limit to the maximum or minimum sending rate of each layer as in SMCC. The sending rate in each layer can be increased or decreased to any level required for adaptation. Besides, other rate control mechanisms such as those in PGMCC [94] and TFMCC [114] can be used in place of the current one, as long as the transmission rate is controlled by the source based on the feedback packets from the most congested receiver.

4.1.3 ON-and-OFF Control of Layers (by Source)

In any GMCC session, there is always a basic layer in which the source keeps sending packets subject to rate control. All other layers must be turned on (i.e. start traffic) or shut down (i.e. stop traffic) at right time to avoid bandwidth waste. Each GMCC session can limit the number of layers to be used. This number is configured at the source and broadcast to receivers periodically. Receiver subscriptions must not exceed this limit. Therefore, if only one layer is allowed, GMCC works the same as a single-rate scheme. The source can potentially control the number of layers to limit the throughput of the whole session.

ON: When a receiver joins a layer which did not have any receiver yet, the source needs to start sending packets in this layer, i.e. turn on this layer. Since this receiver can infer that there is no CR in this layer yet from the CR statistics conveyed by the source, it will send CIs. Upon the receipt of these CIs, the source realizes there is at least one receiver in this layer and therefore begins transmitting data. The receiver will be immediately chosen as the CR for rate adaptation need.

OFF: If all receivers have left a layer, the source has to stop sending data in this layer, i.e. shut down this layer. Each CR (one per layer) needs to send heartbeat packets once per RTT (known from the source) to the source to maintain its validity. If the source has not received any heartbeat packets from a CR for 8 RTTs, it will request CIs from receivers to choose a new CR. If after 4 RTTs, there is still no CR chosen, the source will set the sending rate to a very low level (e.g. one packet per RTT) and wait for another 20 RTTs.[23] The layer will be shut down if no response comes in during all these periods. In the above procedure, the second period is needed to avoid sudden rate decrease in case there are still

[23] All the numbers used here are heuristic.

other receivers in this layer. On the receiver side, to cooperate with the source, the receivers need to send back CIs to the CR once they know the previous CR is invalid. To reduce the total number of feedback packets, receivers may randomize their feedback according to their TAF value (e.g. the larger the TAF, the sooner CIs are sent). Once a new CR is chosen, its TAF statistics can be used by other receivers to suppress their feedback packets scheduled to send (Section 4.1.2).

4.1.4 Joining An Additional Layer (by Receiver)

Whenever a receiver enters a GMCC session, it subscribes to the basic layer of GMCC and stays there till it quits the session. Beyond this basic layer, the receiver must perform join operations to increase its total throughput rate at right time. A receiver joins an additional layer *successively* when it detects that its throughput rate can be potentially increased. There are three situations, and we describe how the join decision is taken in each case.

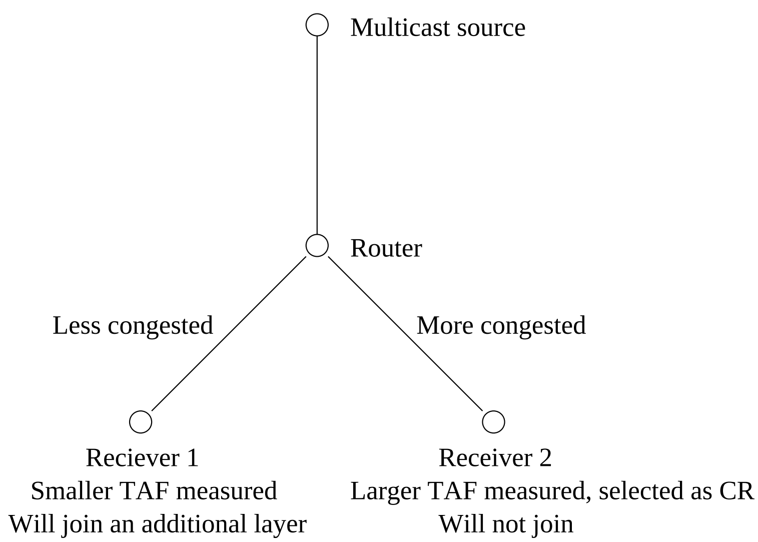


Figure 4.2: A Topology Example for Join Operations under Situation 1 and 2

4.1.4.1 Situation 1: Frequent congestion epochs

This case is suitable for those receives that frequently detect congestion and thus gather enough samples for TAF measurement quickly. Assume we observer receiver i, and the CR is receiver j. When there is congestion *in the highest layer* that receiver i is in, it measures TAF. Once there are at least N TAF samples, it check the following condition,

$$\Theta_j > \beta_1 \Theta_i + \beta_2 \sqrt{\frac{(\beta_1 \Theta_i^{\sigma})^2 + \Theta_j^{\sigma 2}}{N}} \tag{4.2}$$

β_1 and β_2 are parameters. We are conservative about join, therefore we heuristically choose $\beta_1 = 2$, and $\beta_2 = 2.58$ for a 99% confidence level. If the condition in (4.2) is true for J consecutive times, the receiver will join an additional layer. $J \geq 1$ is another parameter controlling conservativeness of join operations and we use $J = 30$ in our simulations. The reason to use relatively small N for samples and J for TAF comparison results, instead of to use a single large N for samples, is that calculating the mean and deviation of a large set of samples is expensive. Meanwhile, this method can catch the dynamics of networks.

For example in Figure 4.2, Receiver 1 is behind a less congested bottleneck and measures smaller TAF on average. Receiver 2 is behind a more congested link and have larger TAF on average. At some point, Receiver 1 will have detected that the condition in (4.2) has been true for J times, and decide to join an additional layer.

Although the TAF comparison in other layers can also stimulate the receiver to join more layers, restricting it in the highest joined layer has equivalent effect and simplifies the design.

4.1.4.2 Situation 2: Infrequent congestion epochs

If the congestion detected by a receiver is light, it may take a long time for this receiver to collect enough samples to make a join decision under situation 1. The solution is to let receivers join under another situation.

When a CR gets a new TAF sample and updates its TAF statistics, it sends a CI to the source with new TAF information. The information is then broadcast to all receivers. When a non-CR receiver notices that the CR of its highest joined layer has updated TAF statistics, this receiver assumes that there is packet loss at this moment and calculates a test version of TAF using its current average ITAF value and the *hypothetical* packet loss interval. For example, a receiver has joined up to layer L. At time t_1 the receiver detects packet loss at layer L and calculates average ITAF as x, COR and then TAF. At a later time t_2 (no packet loss between t_1 and t_2) the receiver notices that the CR in layer L has updated TAF statistics. It then calculates a test version of average COR y using the sample $1/(t_1 - t_2)$, and computes a test version of TAF as xy together with the mean and deviation of TAF. Using this mean and deviation, it checks the condition in (4.2). Once there are J consecutive positive results, it joins layer $L+1$ from layer L.

Note that the test version of COR and TAF are not accepted as permanent samples since they are not true samples. Once used, they are discarded. Consequently, the judging in situation 1 won't be affected.

4.1.4.3 Situation 3: Multiple layers on a shared bottleneck

Still, there is a special case which cannot be dealt with by the solutions for Situation 1 and 2. Consider a topology in Figure 4.3 containing two bottlenecks. The links L_x and L_y have ample bandwidth to avoid any congestion. At the beginning, R1 and R2 are both in only one layer. Therefore, only Bottleneck 2 can be fully utilized, and R2 will join a second layer. After that, Bottleneck 1 is also full. At a later moment, R3 enters the session. The congestion it detects will be approximately the same as that detected by R1. In consequence, R3 stays in only one layer, without knowing it can actually join an additional layer without increasing the congestion on Bottleneck 1. The reason is that the congestion generated by *intra-session flows* of other layers is not distinguished from that by *inter-session flows*, whereas the congestion of the former kind can actually be ignored in the context of deciding whether to join. In consequence, some available bandwidth that can actually be exploited is hidden. This problem also occurs in SMCC, but the paper [59] did not consider it.

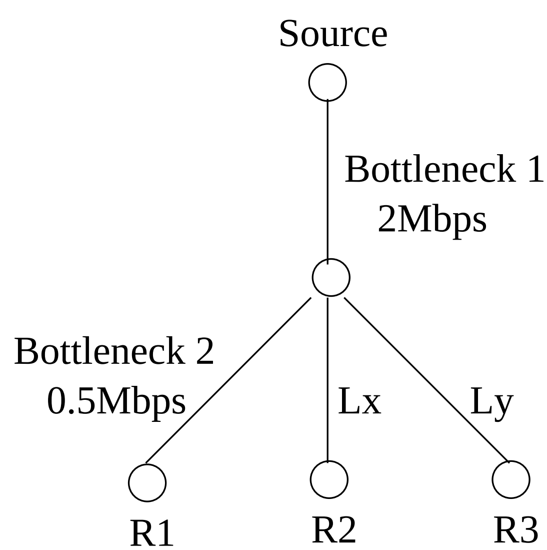


Figure 4.3: A Topology Example Where Probabilistic Inter-Layer Bandwidth Shifting Is Needed (Situation 3)

A solution can be that, for the above example, sometimes we try to send more (e.g. 0.55Mbps on average) in the first layer, while sending less in the second layer (e.g. 0.45Mbps on average). If R3 does not see any increased congestion, it will know that a portion of the congestion is incurred by intra-session flows, therefore can join the second layer.

Certainly the above method should be carefully managed because sending more in a layer might cause more severe congestion on some paths. We developed the following technique called *probabilistic inter-layer bandwidth shifting* (PIBS). Assume multiple layers (layer 1 to n, $n > 1$) are used in a multicast session. Let the period between two consecutive rate reductions (in the same layer) be a *rate control period* (RCP). At the beginning of each RCP at layer i ($1 \leq i < n$), with probably ρ, the source decides that it will send data at the $(1 + \delta)$ level (otherwise send normally). That is, at any moment during this whole RCP, if the calculated sending rate is λ_i, the source will actually send packets at the rate of $\lambda_i + \min(\delta\lambda_i, \lambda_{i+1})$. At the same time, at layer $i + 1$, the actual sending rate is adjusted to $\max(0, \lambda_{i+1} - \delta\lambda_i)$. Briefly, the source "shifts" some bandwidth from layer $i + 1$ to layer i. To avoid significant unfairness to non-GMCC flows, ρ and δ must be small (both are 0.1 in our simulations). Also, at any moment, no two layers are allowed to perform bandwidth shifting simultaneously.

Given a receiver R and a multicast session the receiver is in, assume ℓ layers go through the bottleneck on the path between the source and R. If $\ell > 1$, for any layer $i < \ell$, according to the definition of ITAF (Section 4.1.1), the average ITAF measured by R at layer i during bandwidth shifting periods (θ') should be approximately the same as that measured during periods without bandwidth shifting (θ). On the contrary, if θ' is larger than θ, it means shifting bandwidth to layer i cause more congestion, indicating that no layer above i goes through the same bottleneck.

Assume R's highest joined layer is k, and the highest layer with traffic for the whole multicast session is L. If $k < L$, R will check the following condition at layer k once it has at least N samples for both θ and θ'.

$$\theta - \gamma\sqrt{\frac{\sigma^2 + \sigma'^2}{N}} \leq \theta' \leq \theta + \gamma\sqrt{\frac{\sigma^2 + \sigma'^2}{N}} \tag{4.3}$$

σ and σ' are the standard deviations corresponding to θ and θ' respectively. If condition (4.3) is true, the receiver R will join layer $k + 1$. It should be noticed that although

ITAF samples are distinguished under this situation, they are treated as the same for TAF calculation under situation 1 and 2.

4.1.4.4 Two Exceptional Cases

Even if a receiver decides to join under the three situations above, to prevent spurious join, there are two more cases to be checked before the join operation really occurs.

(1) If any layer in the whole session does not have a CR yet, the join attempt should be canceled.

(2) If a receiver is already a CR for some layer, or detects that it may become a CR in any of its joined layers, it also refrains itself from join. The detection is done by checking the following condition, assuming this receiver is i and the CR is j,

$$\Theta_i > \Theta_j + \omega\sqrt{\frac{\Theta_i^{\sigma 2} + \Theta_j^{\sigma 2}}{N}} \tag{4.4}$$

ω decides confidence level, and we used 3.5 for 99.99%.

The first case above means that if in a layer there is no CR yet, the sending rate may not have stabilized. Either the sending rate has not been increased enough to fully utilized the available bandwidth, or the rate is still in the process of decreasing to adapt to the network situation. Under this vague situation, we cannot draw a conclusion whether it is appropriate or not for a receiver to join, and therefore have to wait. The second case shows that a receiver has the potential to become a CR in a layer. The reason of a receiver being CR is because that the total throughput rate of this receiver has matched its share of the bottleneck bandwidth. As a result, this receiver has to restrain the source from increasing the sending rate too much. Obviously, as long as the receiver is a CR or may become a CR, there is no more room for the its throughput rate to increase.

It is worth mentioning that we do not have "join attempt" as SMCC does. We believe that in GMCC, since both the sending rates in each layer and the number of layers can be dynamically adjusted, as a multicast session goes on, the combination of sending rate settings and the choice of layer number will evolve to the extent that will accommodate the heterogeneity among the receivers, so that a join won't cause abrupt severe congestion. Moreover, omitting join attempts significantly simplifies the design.

4.1.5 Leaving a Layer

A receiver always unsubscribes from the highest joined layer. After a receiver joins a layer, it needs to wait for some time to allow the network stabilize. This is achieved by collecting N more samples for TAF statistics in *all* joined layers before it checks whether to leave. Then, if the receiver is the CR or satisfies the condition in (4.4) in more than one layer, it leaves the highest layer it is in. The reason is the same as explained in the second exceptional case of join at the end of Section 4.1.4.

4.2 Simulations

We have run several *ns*-2 [2] simulations and a large scale ROSS [19] simulation to test the performance of GMCC. Drop-tail routers are used, router buffer size is set to 20K bytes. Reno TCP is used for background traffic. The following aspects of GMCC performance have been tested:

(1) Effectiveness of the adaptive layering, to show that GMCC does not use redundant layers to satisfy heterogeneous receivers.

(2) Responsiveness to traffic dynamics, to show how GMCC responds to dynamically changing competing traffic.

(3) Effectiveness of probabilistic inter-layer bandwidth shifting (PIBS), to show that the technique of PIBS is valid.

(4) Throughput improvement, to show that GMCC can achieve good throughput for heterogeneous receivers.

In the third simulation, we will also show that feedback packets of non-CR receivers can be suppressed efficiently, as described in Section 4.1.2.

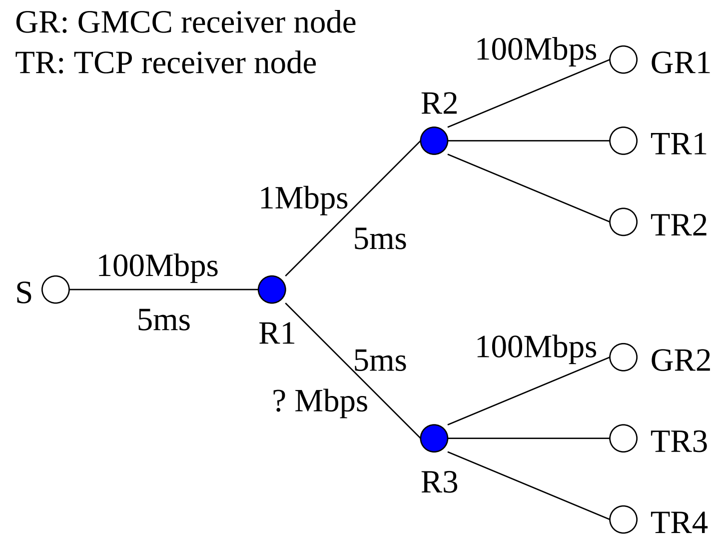


Figure 4.4: Topology for Layering Effectiveness Test (Sec. 4.2.1)

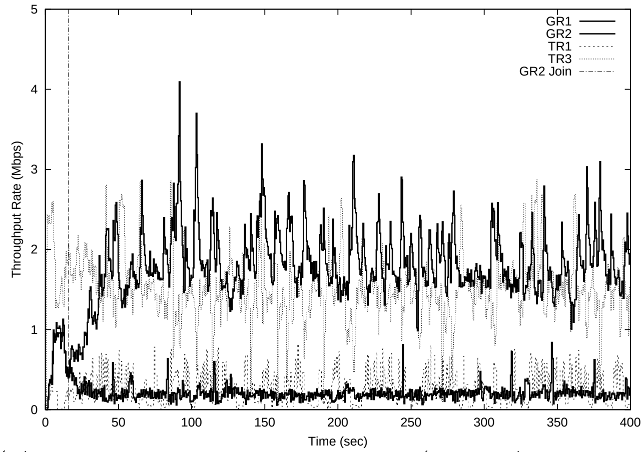
4.2.1 Effectiveness of the adaptive layering

GMCC uses barely enough layers to satisfy heterogeneous receivers, as shown in the following simulations. In the topology of Figure 4.4, four TCP flows go from node S to TR1,TR2,TR3,TR4 respectively. A GMCC session has S as the source and GR1,GR2 as the receivers. In the first simulation, the bandwidth of the link between R1 and R3 is set to 5Mbps. In the second simulation, it is set to 10Mbps. Obviously, in both simulations, if with efficient layer settings, only two layers are needed, where GR1 subscribes to only one layer, and GR2 subscribes to both.

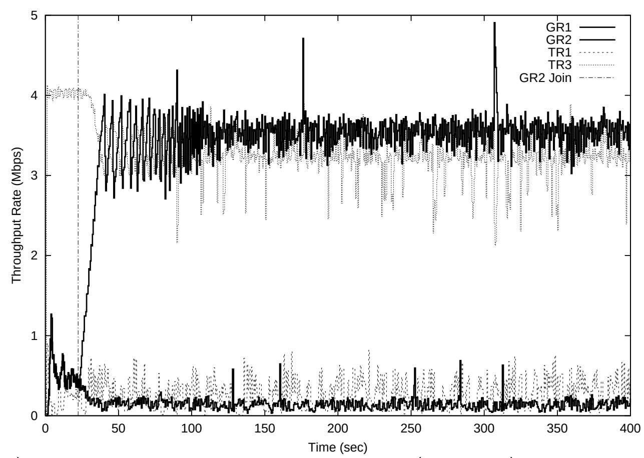
The throughput of the flows in these two simulations are shown in Figure 4.5. GR2 joined an additional layer at 15.8-th second and at 22.4-th second in the first and second simulations respectively, and stayed in two layers till the end of simulations. In contrast, GR1 only subscribed to the basic layer. This conforms to the expectation above and shows that the GMCC does not use more layers than necessary. For comparison, consider SMCC with 1Mbps,2Mbps,4Mbps limits for the lowest three layers. In the second simulation, since GR2's average throughput rate is above 3Mbps, it will have to subscribe to at least three layers with some redundancy.

4.2.2 Responsiveness to traffic dynamics

There are two types of response to traffic dynamics. The first type of response is by the source that adjusts sending rates within layers. GMCC's rate adaption by source is almost



(a) Throughput when the link (R1,R3) is 5Mbps



(b) Throughput when the link (R1,R3) is 10Mbps

Figure 4.5: Effective Layering Test Result (sec. 4.2.1): Instantaneous Throughput in The Topology of fig. 4.4
(GMCC does not use redundant layers to differentiate receiver throughput.)

the same as that in the previous chapter of ORMCC. Therefore, we omit the examination of source response to traffic dynamics here. The second type of response is by receivers by means of joining and leaving layers. It can be considered as a complementary measure of the first type response, since the latter is limited by CRs.

We used the star topology in Figure 4.6 to test the receivers' responsiveness to the dynamics of crossing traffic on the bottleneck. A GMCC session has GS1 as the source node and R1, R2 as the receiver nodes. On each of the links of (R,R1) and (R,R2), there are six TCP competing flows at the beginning of the simulation. During the period between 100-th and 200-th second, five TCP flows on the link (R,R2) pause, leaving one TCP flow as the

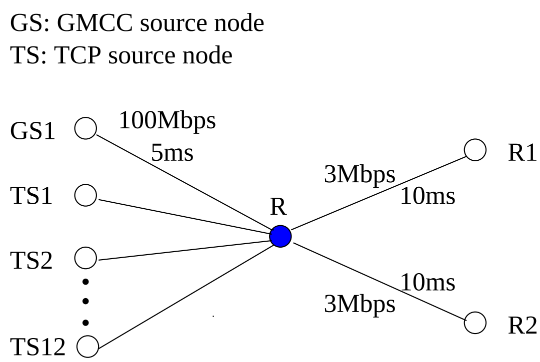


Figure 4.6: Star Topology for Testing Responsiveness to Traffic Dynamics (Sec. 4.2.2)

only competing flow.

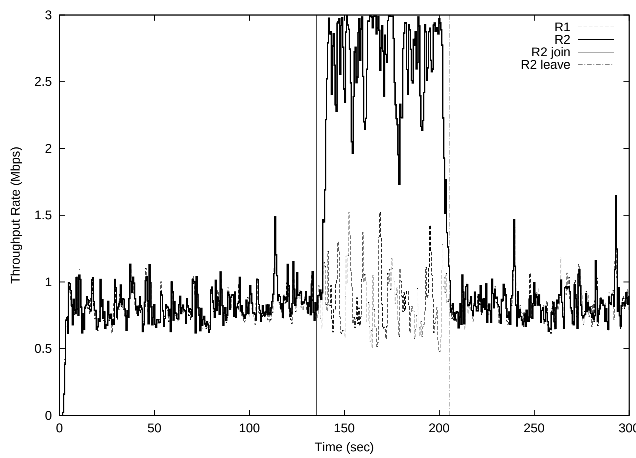


Figure 4.7: Responsiveness to Traffic Dynamics (sec. 4.2.2): Throughput of The Two GMCC Receivers

(GMCC receivers join more layers when there is more available bandwidth and leave when there is less.)

As shown in Figure 4.7, receiver R2 joined an additional layer at 135.412-th second. After those five TCP flows pause, the link (R,R2) became much less congested than (R,R1). Therefore, this join operation is appropriate. There is 35-second gap between the pause and the join operation, though. That is relatively long because GMCC is conservative about join and therefore requires enough number of samples and positive TAF comparison results (see Section 4.1.4). However, GMCC is quicker when making decisions about unsubscription. In this simulation, R2 left the layer at 205.178-th second. On the other hand, since there is no traffic dynamics on the link (R,R1), receiver R1 remains in one single layer.

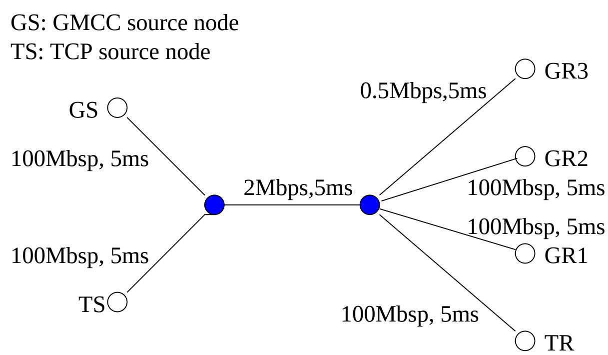


Figure 4.8: Topology for Testing Probabilistic Inter-Layer Bandwidth Shifting (Sec. 4.2.3)

4.2.3 Effectiveness of probabilistic inter-layer bandwidth shifting

Recall that probabilistic inter-layer bandwidth shifting (PIBS) is a technique we developed in Situation 3 of Section 4.1.4 to distinguish the congestion incurred by intra-session flows from that by inter-session flows. This technique enables the receivers to join under some situations with shared bottlenecks. To verify that PIBS is a valid technique, we ran a simulation on the topology in Figure 4.8. A TCP flow originates at TS and ends at TR as background traffic. The GMCC flows in a multicast session go from GS to GR1,GR2 and GR3. The 2Mbps bottleneck is shared by all three GMCC receivers, and the 0.5Mbps bottleneck only affects GR3. At the beginning of the simulation, only GR1 and GR3 are in the session. At 100th second, GR2 enters the session. Figure 4.9 shows that in one simulation instance, GR2 subscribed to an additional layer at 170.146-th second based on bandwidth shifting. Again, there is long delay because GMCC receivers need to collect enough samples before making decisions.

We noticed that in some other instances of this simulation, a join operation for another reason (in particular, under situation 2 in Section 4.1.4) happened before the results of bandwidth shifting took effect, and the joini operations triggered by bandwidth shifting were suppressed. This is not unexpected because the flows are dynamic and the comparisons in GMCC are all probabilistic. It is possible that during some random periods the condition in situation 2 becomes true and triggers a join operation.

We can also see how feedback suppression works in this simulation. As the CR in layer 1, GR3 sent 4424 feedback packets; as the CR in layer 2, GR1 sent 5448 feedback packets. Most of these packets are heartbeat packets, sent once per RTT of around 110 ms. GR2, since it is not CR at any time, only sent 2 CIs. Therefore, feedback from non-CR receivers

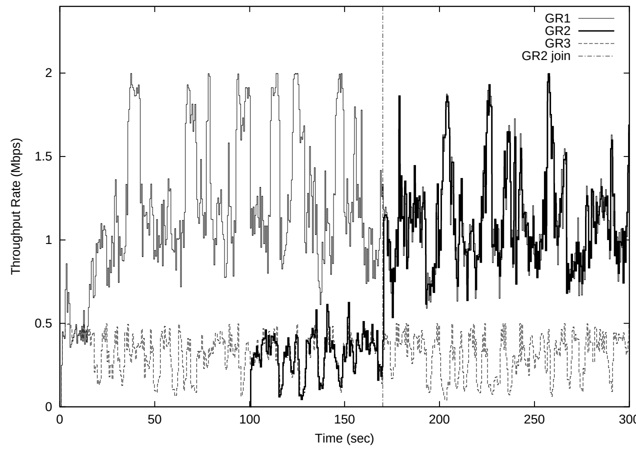


Figure 4.9: PIBS Result (sec. 4.2.3): Throughput of All GMCC Receivers
(The technique of ***probabilistic inter-layer bandwidth shifting*** (PIBS) can exploit hidden available bandwidth.)

is efficiently suppressed.

4.2.4 Throughput Improvement

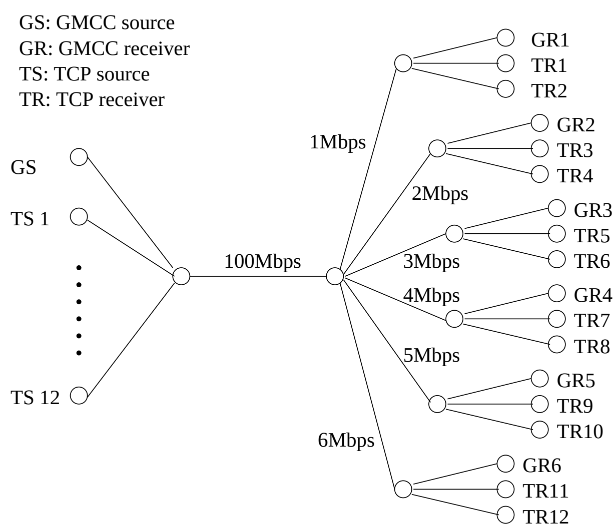
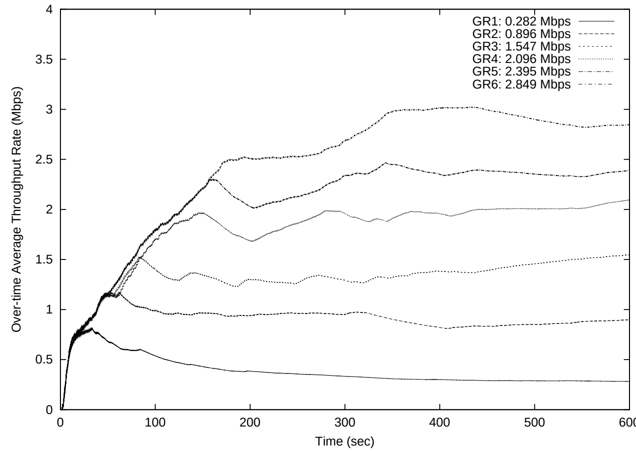
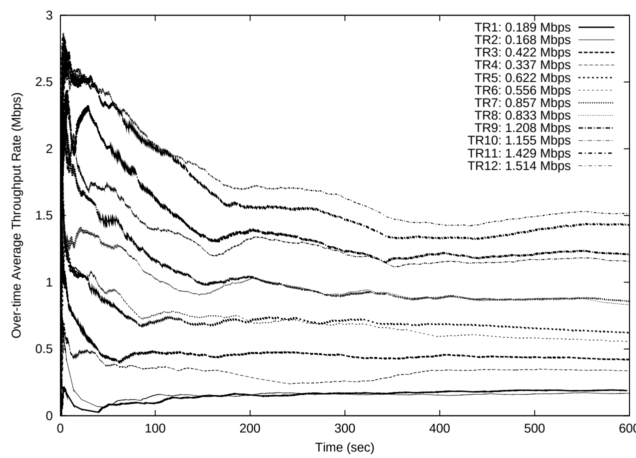


Figure 4.10: Topology for Testing Throughput Improvement (Sec. 4.2.4)

The topology in Figure 4.10 contains six bottlenecks and is used to test how GMCC improves the throughput of heterogeneous receivers with relatively slight difference of expected throughput. All the links are of 5ms delay. The bandwidths of the bottlenecks are from 1Mbps to 6Mbps. On each of them, there are two TCP flows as competing traffic. A GMCC session is held between the source GS and six receivers (GR1 to GR6). Simulation time is 600 seconds.



(a) Over-time Average Throughput Rate of GMCC receivers



(b) Over-time Average Throughput Rate of TCP receivers

Figure 4.11: Throughput Improvement (Sec. 4.2.4): Receiver Throughput in The Topology of Fig. 4.10
(GMCC receivers behind different bottlenecks get different throughput matching the bottleneck capacity.)

Figure 4.11 shows the over time average throughput rate of all receivers. Over time average throughput rate at time t is defined as the total throughput through time t divided by the total run time. We can see that the six GMCC receivers do achieve different throughput rates, with GR6 being the highest and GR1 being the lowest.

Besides, there were only a few join and leave operations in this simulation. Compared to the previous multi-rate schemes where join and leave happen every RTT or so, GMCC clearly provides a great improvement. The number of join and leave operations of each receiver is listed in Table 4.2. Note that since join and leave are triggered by statistical comparisons, there were several oscillations that increased the operation numbers (e.g for

Table 4.2: Number of Join and Leave Operations
(The number of IGMP operations is small and incur very light control traffic.)

	GR1	GR2	GR3	GR4	GR5	G6
Join	4	12	11	10	9	7
Leave	4	11	8	6	4	2

GR2).

We need to mention that in this simulation, GMCC receivers achieve higher throughput than TCP correspondents. The reason is that each flow in a GMCC layer is a single-rate congestion control flow independent of other flows. It competes for bandwidth like any other flow does. For example, when GR2 subscribes to two layers, there are then two TCP flows and two GMCC flows on the 2Mbps bottleneck. The throughput of GR2 is the sum of both GMCC flows, and therefore can be approximately twice as much as each of the TCP flows. However, due to the limit by CRs in lower layers, assuming there are n TCP flows and m GMCC flows on a bottleneck, a receiver may not get the share of $m/(m+n)$. GR6 here is an example. Although what we observed for GMCC in this simulation is different from traditional TCP-friendliness concept, we don't consider it as a serious problem, because each GMCC flow within a layer still competes in a TCP-friendly manner. This is more or less the same as people open multiple TCP connections to transmit a single object over the Internet. More important, using independent GMCC flows of this kind greatly simplifies the task to achieve multi-rate for multicast.

We have also run a large simulation in ROSS on the topology of Figure 4.12, the same one used for ORMCC (3) large scale simulation. The background traffic on the last hops is generated by two single-receiver PGMCC flows (since its behavior is close to TCP), and the last hop is the only bottleneck on the path from the source to a receiver. There are 1200 receivers, each behind a different bottleneck. All the bottlenecks are divided into ten even groups, their bandwidths being from 0.2Mbps to 1.2Mbp with difference as 0.2Mbps.

The simulation ran for 2000 seconds. The average throughput and the deviation of each group of receivers is shown in Figure 4.13. The average throughput grows linearly with the bottleneck bandwidth, again showing that the multi-rate feature of GMCC is effective. The numbers of join and leave oeprations are in Table 4.3. (Group i is the group of receivers

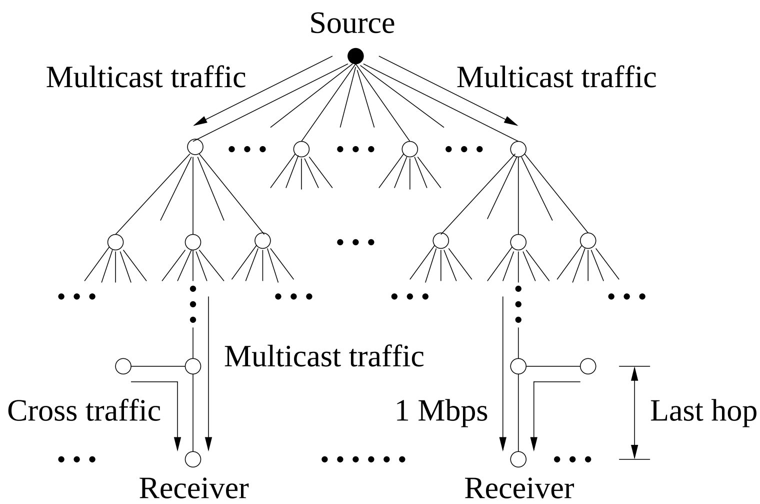


Figure 4.12: Tree Topology for Large-Scale Simulations in ROSS

Table 4.3: Number of Join and Leave Operations in Large Scale Simulations
(The number of average per-receiver IGMP operations is still very small at the presence of many receivers behind different bottlenecks.)

	Group 1	Group 2	Group 3	Group 4	Group 5	Group 6
Join	1913	2757	2986	2988	2896	2947
Leave	0	246	155	35	0	0

behind the bottlenecks of $i \times 200$Kbps bandwidth) Even in the most active groups, on average, each receiver has less than 15 join operations and much fewer leave operations (around 1) within 2000 seconds. Obviously, the volume is very light.

4.3 Summary

We have presented a multi-rate multicast congestion scheme called GMCC. By combining single-rate congestion control and traditional multi-rate techniques (mostly joining and leaving layers by receivers) in a novel way, it provides a simple design for a perplexing problem of which most previous solutions are complicated. While having the merits of a similar previous scheme SMCC [59], it is *fully* adaptive and surmounts the limits posed by SMCC's required static configurations. A new technique called *probabilistic inter-layer bandwidth shifting* is proposed as the solution to explore hidden available bandwidth, which is a problem not mentioned in SMCC. Besides, the rate control mechanism at source can be replaced by other representative-based mechanisms.

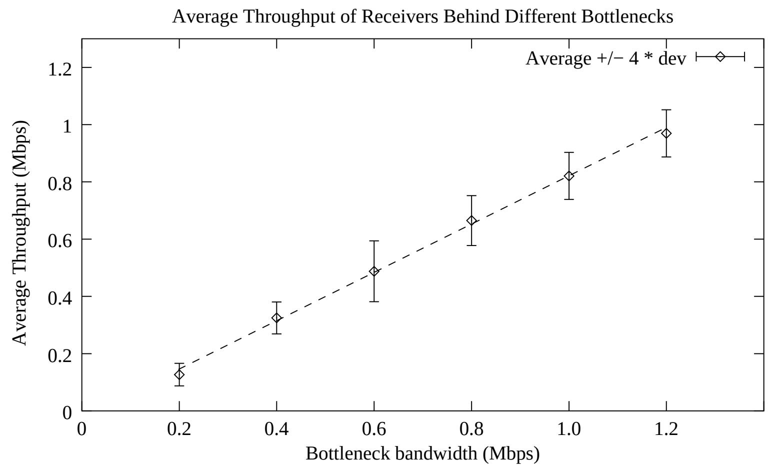


Figure 4.13: Average Throughput and Deviation of Differnt Groups of Receivers
(Different groups of GMCC receivers get different throughput proportional to their bottlenecks.)

There is still another potential problem of SMCC. Assume receiver R has joined a set of layers $\mathcal{L}$. In SMCC, R calculates its estimated throughput using TCP throughput formula [85] with the overall "loss event rate" [59, 114] of all the layers in $\mathcal{L}$ as one of the parameters. If the layers in $\mathcal{L}$ have different underlying multicast trees, the overall loss event rate will be higher than the individual value of any single layer, thus the throughput will be underestimated. Since in SMCC, receivers rely on estimated throughput to decide how many layers to join, and the source uses estimated throughput from receivers to control sending rates, the underestimation might degrade the performance. On the contrary, GMCC treats each layer independently instead of considering all of them as a whole. Therefore, GMCC does not have this problem. However, it needs more careful exploration.

CHAPTER 5

MCA+: An End-to-end Multicast Congestion Avoidance Scheme with Feedback Suppression

In this chapter, we consider the multicast *congestion avoidance* [23] problem for the situation as that considered in ORMCC (Chapter 3). Again, the situation is:

> *Support is allowed on receiver side, but only one multicast group is allowed for each multicast session.*

We do not study the situation as that in LE-SBCC (Chapter 2) since it is impossible to do full congestion avoidance without special receiver support. On the other hand, we can extend MCA+ to do multi-rate as we extend ORMCC to GMCC. Therefore, we save the work and do not study the situation as that in GMCC (Chapter 4) either.

Congestion avoidance [23] is different from congestion control in the sense that our scheme detects and responds to network congestion without necessarily inducing packet loss. We propose MCA+. It is a single-rate scheme, i.e., it goes at the rate allowed by the worst congested receiver, since only one group is allowed for each multicast session under the situation we consider. One previous single-rate multicast congestion control work by DeLucia et al [25] uses the congestion detection mechanism proposed in TCP Vegas [15], a unicast congestion avoidance scheme. However, in DeLucia's scheme, incipient congestion is only measured at source side for the paths between the source and representative receivers, while other receivers still detect congestion by packet losses. Therefore, it is not a fully congestion avoidance scheme. For comparison, in our scheme, incipient congestion is detected at receiver side for all paths. Other examples of single-rate multicast congestion control schemes include PGMCC [94], TFMCC [114] and references within. These schemes are "congestion control" schemes in the sense that receivers wait for a packet loss which is signalled back to the source as loss indications. If bottlenecks provide packet marking support (similar to TCP-ECN [88]), packet loss may be avoided in the above schemes, and they could also be classified as "multicast congestion avoidance"schemes.

MCA+ is a revision of our previous scheme MCA (Appendix D) and accommodates some contributions from the congestion control scheme ORMCC (Chapter 3). It responds to incipient congestion in order to avoid packet loss. The schemes uses a new concept of *"accumulation"* (defined as the number of buffered bits of a flow inside the network) and simple thresholding techniques proposed in our recent unicast work [110] to achieve congestion avoidance on a *purely* end-to-end basis, i.e., with no marking support from interior bottlenecks. In this sense, our work is comparable to the unicast work of TCP Vegas [15] which assumes a similar model, albeit in a unicast, window-based TCP context. Just like TCP Vegas [79, 73], our scheme's congestion model detects congestion end-to-end as real queues are *being* built up. This model is inherently *incompatible* with the TCP model of *waiting for packet losses to occur* before detecting congestion. The only way to ensure compatibility is to have a packet marking scheme at bottlenecks which indicates congestion as the queues build up [72]. Therefore, in the absence of packet-marking support, our MCA scheme (like Vegas [79, 73]) *cannot directly compete* with TCP in the same queue and will be beaten down if it does. So, like Vegas, our scheme focusses on the conceptual rather than deployment issues. We discuss possible deployment scenarios briefly in the conclusion of this paper.

In this scheme, we use representative for rate control purpose at source[24] side. That is, at any time, the source keeps record of the slowest receiver (we call it *Congestion Representative (CR)*), and adjusts the sending rate according to that receiver's feedback. At the same time, receivers themselves also suppress their feedback if necessary so that not to overwhelm the source. Both mechanisms make use of a new metric *Good Throughput Rate At Congestion (G-TRAC)*, defined as *the receiving rate upon congestion times one minus congestion rate.* Briefly speaking, the receiver with the lowest average G-TRAC is chosen as the slowest receiver, i.e. CR, and other receivers suppress their feedback if their average G-TRACs are higher than that of CR. Using G-TRAC is a contribution in that our scheme then does not require receivers to *continuously* (either densely or sparsely) exchange packets with the sender (e.g. to measure RTT).

MCA+ consists of four key building blocks (Figure 5.1), two at receivers and two at the source. There is a *congestion detection* block at receiver side, which detects congestion by accumulation and possible (though rare) packet loss. A *congestion response* block resides at the source, which implements a rate-increase/decrease policy based on the CR's feedback. Moreover, a *filtering block* at receiver side blocks *Congestion Indications (CIs)* generated by

[24] In this paper, we use the terms *source* and *sender* interchangeably.

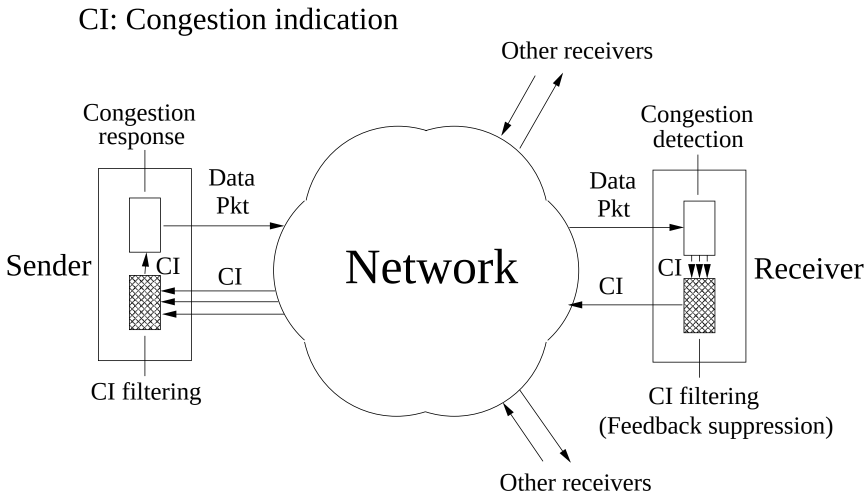


Figure 5.1: MCA+ Model

congestion detection block if necessary , i.e. does feedback suppression, and another *filtering block* at sender side only allows CIs from CR through. All blocks only require small constant number of states and light computation.

In brief, MCA+ has the following features:

(1) It is an end-to-end scheme, without requiring special support from network, such as packet marking.

(2) It uses *accumulation* instead of packet loss to detect congestion, and thus can react to incipient congestion.

(3) It provides efficient non-timer-based feedback suppression.

(4) State and computation complexity at both source and receiver side are $O(1)$.

Simulations show that MCA+ achieves good bottleneck utilization, while avoiding Drop-to-Zero problem [114, 94, 9]. Drop-to-Zero is the problem of reacting to *more feedback* than necessary leading to a beat-down of the multicast flow's rate[114, 94, 9]. This occurs because the multicast flow receives feedback from multiple paths and may not filter them sufficiently. TCP-unfriendliness is the problem of reacting to *less feedback* than a hypothetical TCP flow would on the worst loss path [14, 94, 114]. Though the congestion detection model is incompatible with that of TCP (and we cannot directly demonstrate fairness with TCP), we demonstrate the fairness between multi-receiver and single-receiver MCA+ flows.

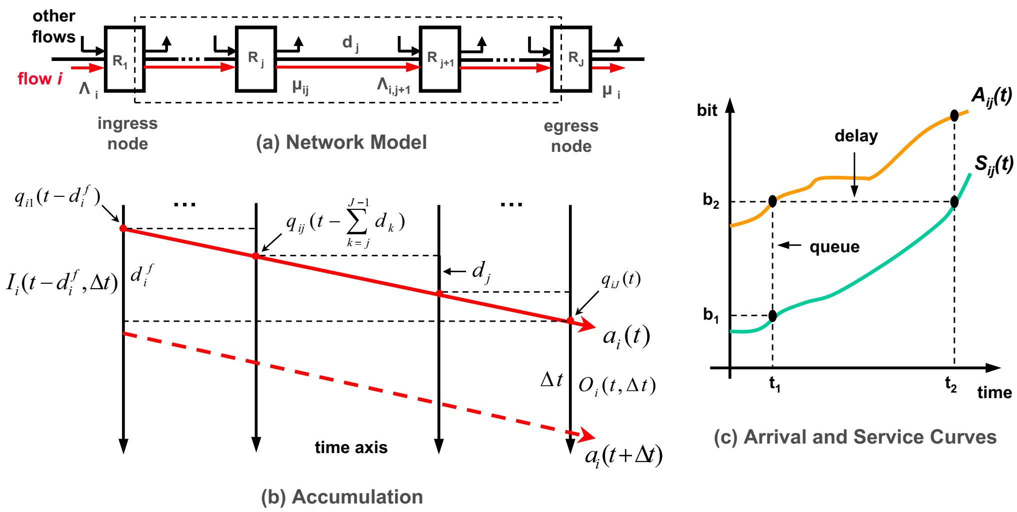


Figure 5.2: Network Fluid Model: Accumulation Concept

5.1 Concepts And Model

In MCA+, congestion detection is based on the accumulation concept. In this section we define the accumulation concept using a bit-by-bit fluid model [24] [86], and develop an algorithm for measuring it in a multicast context at receivers.

5.1.1 Accumulation

The concept of *accumulation* was first developed in our earlier unicast work [110]. We summarize the core ideas here. The discussion below assumes unicast fluid flows, but we extend it to multicast in a later section.

Consider an ordered sequence of FIFO nodes (routers) $\{R_1, \ldots, R_j, R_{j+1}, \ldots, R_J\}$ along the path of a *unidirectional* flow i in Figure 5.2(a). The flow comes into the ingress node R_1 and, after passing some intermediate nodes $R_2, \ldots, R_{J-1}$, goes out from the egress node R_J. At time t in any node R_j $(1 \leq j \leq J)$, flow i's input rate is $\lambda_{ij}(t)$, output rate $\mu_{ij}(t)$. The propagation delay from node R_j to node R_{j+1} is d_j.

We define the arrival curve $A_{ij}(t)$ of a flow i at a node R_j as the number of bits from that flow which have cumulatively arrived at the node up to time t, and similarly the service curve $S_{ij}(t)$ as flow i's bits cumulatively serviced at node R_j [24] [86], drawn in Figure 5.2(c). For any FIFO node R_j, both $A_{ij}(t)$ and $S_{ij}(t)$ are continuous[25] and non-decreasing functions. If there is no packet loss, then at any time t, by definition, flow i's buffered bits $q_{ij}(t)$ in

[25] This is strictly true if we accept that a bit is infinitely small.

node R_j is the difference between $A_{ij}(t)$ and $S_{ij}(t)$, as shown in Figure 5.2(c):

$$q_{ij}(t) = A_{ij}(t) - S_{ij}(t). \tag{5.1}$$

The change of flow i's queued bits at node R_j is,

$$\Delta q_{ij}(t) = q_{ij}(t+\Delta t) - q_{ij}(t) = (\overline{\lambda}_{ij}(t,\Delta t) - \overline{\mu}_{ij}(t,\Delta t)) \times \Delta t = I_{ij}(t,\Delta t) - O_{ij}(t,\Delta t) \tag{5.2}$$

where $I_{ij}(t,\Delta t)$ and $O_{ij}(t,\Delta t)$ are incoming and outgoing bits of flow i at node R_j during the time interval $[t, t+\Delta t]$; $\overline{\lambda}_{ij}(t,\Delta t)$ and $\overline{\mu}_{ij}(t,\Delta t)$ are the correspondent average input and output rates, respectively.

Now consider the flow's queueing behavior at a *sequence* of FIFO nodes. Define flow i's *accumulation as a time-shifted, distributed sum of the queued bits* in all nodes along its path from the ingress node R_1 to the egress node R_J, i.e.,

$$a_i(t) = \sum_{j=1}^{J} q_{ij}(t - \sum_{k=j}^{J-1} d_k) \tag{5.3}$$

which is shown as the solid slant line in Figure 5.2(b). Note this definition includes only those bits backlogged inside the node buffers, not those stored on transmission links. With the definitions of

$$\lambda_i(t) = \lambda_{i1}(t),\ \mu_i(t) = \mu_{iJ}(t) \tag{5.4}$$

we calculate flow i's accumulation change as follows:

$$\begin{aligned}
\Delta a_i(t) &= a_i(t+\Delta t) - a_i(t) \\
&= \sum_{j=1}^{J} \Delta q_{ij}(t - \sum_{k=j}^{J-1} d_k) \\
&= [\overline{\lambda}_i(t - d_i^f, \Delta t) - \overline{\mu}_i(t,\Delta t)] \times \Delta t \\
&= I_i(t - d_i^f, \Delta t) - O_i(t,\Delta t)
\end{aligned} \tag{5.5}$$

where $d_i^f = \sum_{j=1}^{J-1} d_j$ is the forward direction propagation delay of flow i from node R_1 all the way down to node R_J. Similar to Equation (5.2), $I_i(t - d_i^f, \Delta t)$ and $O_i(t,\Delta t)$ are flow i's bits coming into and going out of network during two *different* time intervals but both of length Δt; while $\overline{\lambda}_i(t-d_i^f, \Delta t)$ and $\overline{\mu}_i(t, \Delta t)$ are the correspondent average ingress and egress rates. The result, illustrated in Figure 5.2(b), shows the change of a flow's accumulation on its path is only related to its input and output at the ingress and egress nodes. Further, this means it is possible to control accumulation at only the ingress and egress nodes.

Given a time sequence $\{t_1,\ t_2,\ \ldots,\ t_k,\ \ldots\}$, denote $a_i(t_k)$ as $a_i(k)$, $\overline{\lambda}_i(t_k - d_i^f, \Delta t)$ as $\overline{\lambda}_i(k)$, and $\overline{\mu}_i(t_k, \Delta t)$ as $\overline{\mu}_i(k)$, according to Equation (5.5), we have,

$$a_i(k+1) = a_i(k) + (\overline{\lambda}_i(k) - \overline{\mu}_i(k))(t_{k+1} - t_k)$$

It shows that accumulation can be measured by using correlated periods, e.g. $[t_k, t_{k+1}]$ at egress and $[t_k - d_i^f, t_{k+1} - d_i^f]$ at ingress. This can be done by sending synchronization data "*out-of-band*," i.e., the synchronization data experiences only the fixed one-way delays and not the queueing delays.

Given the fluid flow assumption, denoting the queue length of node j at time $t_k - \sum_{(x=j+1)}^{J} d_x$ as $q_j(k)$, accumulation also satisfies the following property:

$$a_i(k) > 0 \Leftrightarrow \exists j \;\; q_j(k) > 0 \text{ and } a_i(k) = 0 \Leftrightarrow \forall j \;\; q_j(k) = 0 \tag{5.6}$$

In other words, for a network of fluid flows *a zero-threshold for the per-loop accumulation measure is equivalent to a zero-threshold on the real queue at some bottleneck.* The properties are rigorously developed in reference[110].

In summary, accumulation and output rate are quantities which can be *measured with only per-flow information.* It allows us to build a *fully* distributed, transparent closed-loop congestion avoidance mechanism using them. In particular, a simple congestion avoidance approach would be:

a) Use simple thresholding techniques on the accumulation measure to detect epochs of congestion.

b) Use feedback to guide the congestion response policy to achieve fine-grained control over input rate dynamics.

While the above discussion referred to unicast, the same approach can be applied to multicast if the machinery for accumulation measurement can be instrumented, which is the focus of the following sub-sections.

5.1.2 Accumulation Measurement

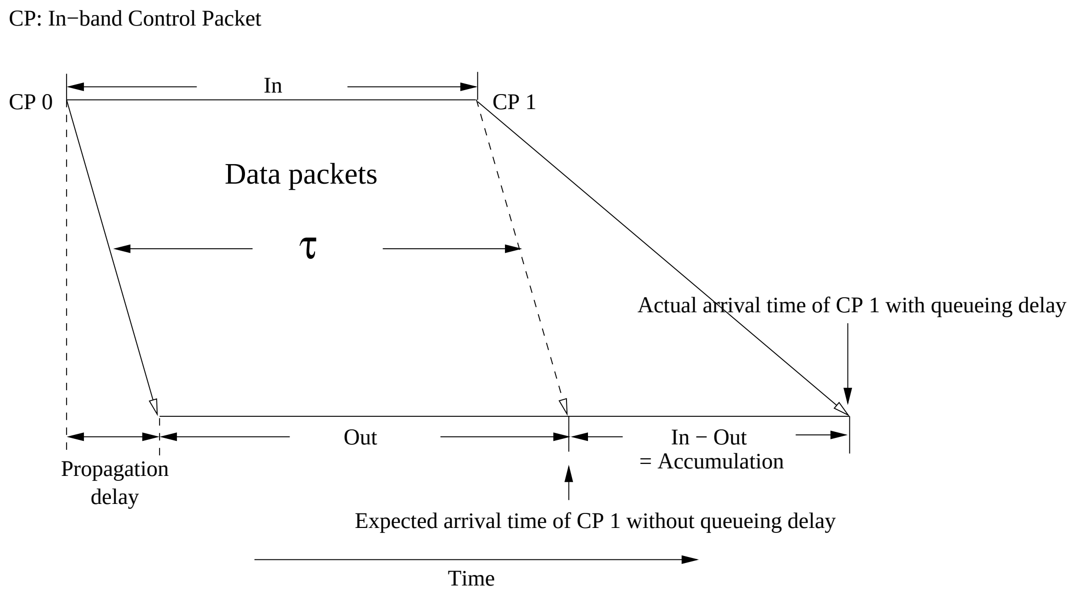


Figure 5.3: Accumulation Measurement with In-band Control Packets

To perform end-to-end accumulation measurement in real world, we relax three key assumptions made in the previous section. First, we send synchronization (or control) data "*in-band*" instead of "*out-of-band*", i.e., it sees both fixed propagation delays and queueing delays. Second, we send packets instead of bit-by-bit fluid. Therefore, to account for the randomness introduced, the thresholding procedure has to be amended and a new re-synchronization procedure is performed at the end of congestion epochs. Third, we develop the scheme for multicast, i.e., divide functionality between the source and receivers such that the reverse control traffic is minimized. Forward control traffic is multicast to all receivers. Note that first release of assumption is the major reason of approximate accumulation measurement. Again, as we describe in the introduction, our aim is to react to congestion as early as possible so as to avoid unnecessary packet loss and increase bottleneck utilization as much as possible.

Figure 5.3 illustrates the measurement of accumulation using *in-band* control packets. [26] Assume that the first control packet (CP0) sees no queue and hence arrives at receivers after exact propagation delay. The second control packet (CP1) is multicast after the mea-

[26] By control packet, we actually mean a data packet with some one-bit flag turned on and with its sending time in the optional field. If this bit is turned off and no sending time is carried, the data packet is then simply a normal data packet.

surement interval τ. A receiver measures *"out"*, the number of bytes received since the receipt of CP0 for a period of τ. After CP1 arrives, the receiver knows how much the sender has sent during the period of τ, i.e. *"in"*. $in - out$ is then the accumulation. Observe that this measure works correctly in a rate-based system where packets are sent uniformly and the input burstiness is also control by a rate-shaper.

Figure 5.3 still assumed a fluid model. Packetization introduces randomness and burstiness in the system. In particular, even for a perfectly smoothed packetized transmission, when underloaded, bottlenecks can have a *average* steady state queue of half packet and *maximal* queue of one packet for *each flow* going through it. Therefore, the fluid flow formula (5.6) which implies a zero-threshold on accumulation no longer holds. In our simulation, we use the following hysteresis technique: declare congestion if accumulation becomes larger than two packets, and subsequently declare end of congestion when accumulation falls below 1 packet. If there are other sources of noise which affect accumulation (eg: scheduling noise at operating systems or at bottlenecks), the thresholds should be set higher. This technique is hence conservative in detecting congestion, i.e., a receiver may unilaterally detect congestion even if there is no network congestion (eg: in multi-bottleneck cases). Higher thresholds reduce the probability of such errors, at the price of larger worst-case queues. We find through simulation that the settings above work very well.

Again because bottlenecks may have steady state queue even when underloaded, the initial control packet (CP0 in Figure 5.3) may not see zero queueing delay. In other words, our assumption of synchronization at the first control packet may be erroneous. Also, as a side effect of the hysteresis scheme described above, the receiver could end a congestion epoch with non-zero accumulation. To counter these issues, we introduce the notion of *"re-synchronization"* as illustrated in Figure 5.4. We begin with the default assumption that we have synchronized correctly at CP0, and then measure accumulation during successive intervals based upon this assumption. If a control packet arrives at the receiver before its expected arrival time, we re-synchronize (not shown in the figure) and set accumulation to zero. Also, if we detect the end of a congestion epoch, we re-synchronize.[27] Figure 5.4 shows a case when the re-synchronization happens perfectly, i.e., accumulation is zero, and the re-synchronization point overlaps the expected arrival time of the control packet without

[27] Believing that the intervals between synchronizations won't be long, we assume that clock skew can be ignored.

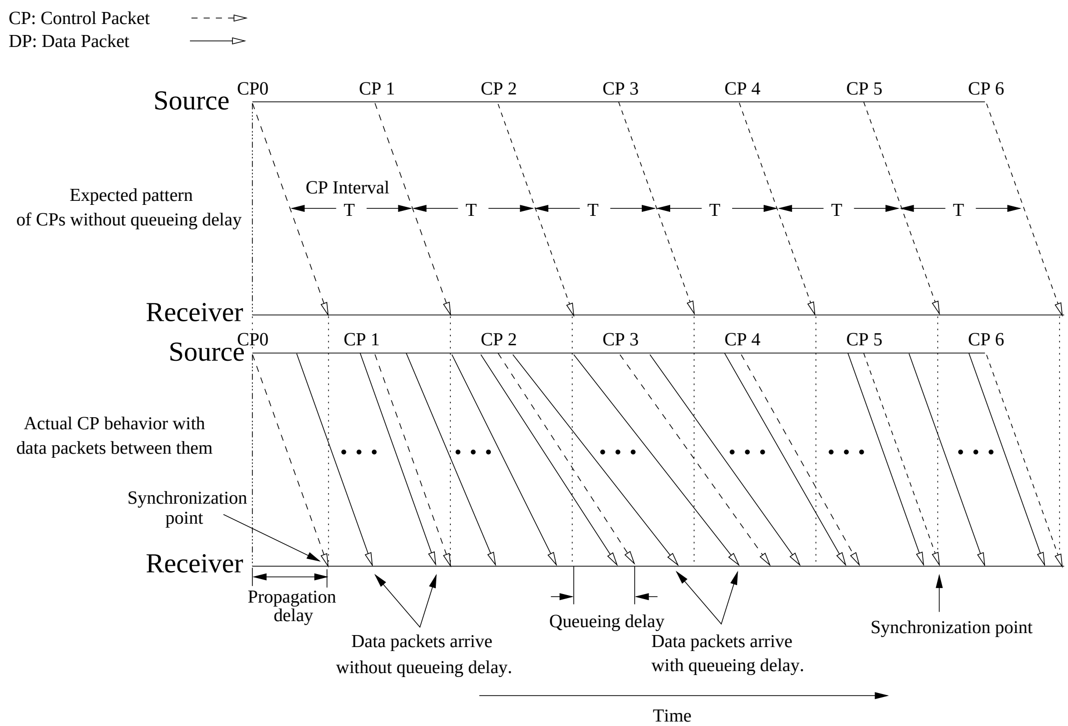


Figure 5.4: Congestion Epochs: Synchronization Points and Accumulation

queueing delay. In practice, any residual positive accumulation is carried over to the next epoch.

5.2 MCA+: Scheme Description

First, for clarity, we list the acronyms in the following:

CI Congestion indication. A packet conveyed by receiver to inform the source of congestion.

CR Congestion representative. The slowest receiver based on whose CIs the source makes rate adjustment decisions.

G-TRAC Good throughput rate at congestion, defined as the product of receiving rate during congestion and one minus congestion rate. It is explained in more details at Section 5.2.1.3.

Briefly speaking, in MCA+, the sender keeps multicasting data and control packets to receivers, and the receivers detect congestion upon receipt of control packets by measuring accumulation and checking packet loss. When there is not congestion, the sender does not receive any CI, and keeps increasing sending rate periodically. If there is congestion, receivers convey the information to the source by sending CIs if they pass the suppression filter. When CIs arrive at the sender, another filter is applied to choose CIs from the CR. The sender then reduces the transfer rate based on chosen CIs.

Therefore, there are two operation parts in MCA+, one is at source side, the other at receiver side. We present the details in the following.

5.2.1 Source Operations

The major task of the source is to adjust sending rate according to congestion information from receivers. The key of source operations is to choose the slowest receiver as the CR. If there are CIs from the CR, the sending rate is reduced, otherwise the rate is increased per estimated RTT. Consequently, the source needs to consider the following problems:

(1) RTT estimation

(2) Rate adaptation

(3) CR switching

(4) CI filtering

The solutions to the first two problems are straight-forward, while the solutions to the last two use a new metric called G-TRAC, which will be explained momentarily. Key operations are also shown in the flow chart of Figure 5.5.

5.2.1.1 RTT estimation

RTT is important since the source needs it for the sake of rate adaptation and CR switching. A sample of RTT is obtained whenever a CI arrives at the source, with the value as the time difference between the CI arrival time and the sending time of the packet which triggered the CI. Given a sample value s, the RTT is smoothed using EWMA (exponential weighted moving average), i.e. $SRTT = 7/8 \cdot SRTT + 1/8 \cdot s$. The deviation is calculated as $\sigma = 7/8 \cdot \sigma + 1/8 \cdot (|SRTT - s| - \sigma)$.

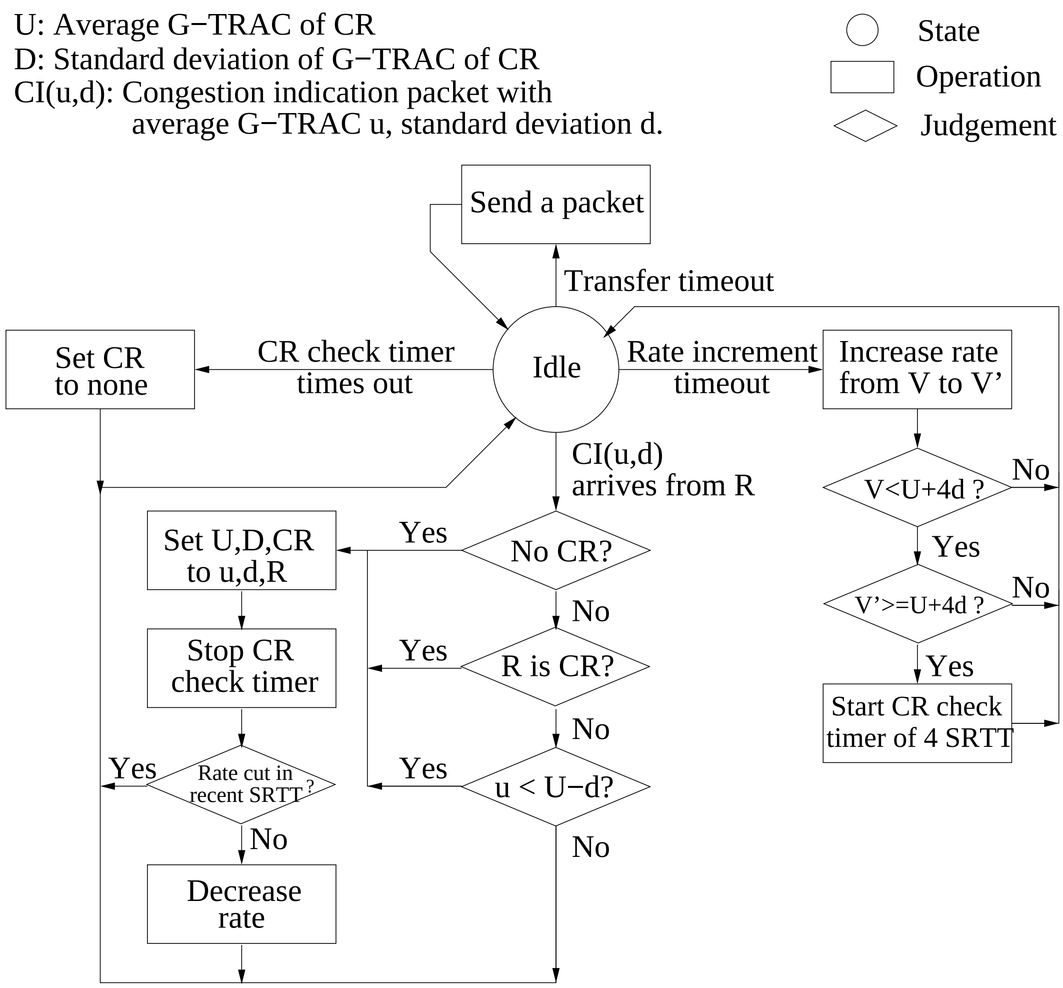


Figure 5.5: Source Operations

5.2.1.2 Rate adaptation

If no CI comes in, in every period of $SRTT + 4\sigma$, the source increases the transfer rate and send one more packet per period, similar to what TCP does. If a CI passes all the filters, i.e. it is from the CR, the source needs to check whether the rate has been reduced during the most recent $SRTT + 4\sigma$. If not, the rate is then cut. This is to guarantee that at most one rate reduction is performed per RTT.

In each CI, there is one bit indicating whether the CI is triggered by accumulation over threshold or by packet loss. If it is the former, the rate is reduced by 10%; if the latter, the rate is reduced by 25%, since packet loss means more severe congestion. The percentages of cutting rate are subject to choice. However, generally, if they are set higher, congestion will be cleared more quickly while bottleneck utilization may decrease; if set lower, there is more risk of persistent congestion while bottleneck utilization may increase. In our simulations, the two values mentioned above worked satisfactorily.

5.2.1.3 CR Switching

Since network condition keeps changing, the choice of CR must be updated accordingly so that the sending rate can always be adjusted in favor of the most congested bottleneck. There are mainly two reasons to change the CR: (1) The current CR is still active but another receiver becomes the new slowest one, (2) The current CR is absent. We have two different techniques to cope with these two situations respectively.

Before we describe those two techniques, we first present an important concept of *G-TRAC (Good Throughput Rate At Congestion)*, which is required by both CR switching techniques (and feedback suppression conducted by receivers). TRAC itself is the receiving rate a receiver measures during congestion. It is used by the congestion control work ORMCC (Chapter 3). To avoid the random error due to burstiness, TRAC can be averaged over a short period, for example, the most recent RTT. A receiver also measures congestion rate p (*not* loss rate), which is the number of CIs divided by total number of packets sent by the sender over a certain period. G-TRAC is then the TRAC weighted by $(1-p)$. Every receiver in a multicast session measures its own G-TRAC and maintains the average and deviation, which are sent along with CIs.

For CR initialization (when there is no CR yet), the source simply chooses the receiver whose CIs arrive at the source first. It will then refine the choice using the following two techniques.

As shown in Figure 5.5, when the source receives a CI, it checks the G-TRAC average in the CI. If it is lower than $U - D$ (where U and D are the G-TRAC average and deviation of the current CR respectively), the receiver sending this CI will be chosen as the new CR, and U and D are updated with the values in the CI. We use $U - D$ as the lower bound because we want to be conservative and bias toward the current CR to avoid unnecessary oscillation. Generically, $U - kD$ can be used as the lower bound, where k decides on how quickly CR is updated and thus the level of oscillation. In our simulations, $k = 1$ showed good results.

While the source has a choice of CR, it needs to continuously check whether the CR is still alive. The method is, at the moment when the sending rate becomes greater than or equal to $U + 4D$, the source starts to count. If there is no CI coming from the CR within $n(SRTT + 4\sigma)$ after that, the source deems the CR absent, thus resets the choice of CR, and requests CIs from other receivers. We used a heuristic and conservative $n = 4$ for our simulations. Setting n too large will result in delay of detecting CR absence, while setting n too small can result in erroneous judgments of CR absence.

5.2.1.4 CI filtering

Although CIs have already been suppressed a lot by receivers themselves (to be discussed in Section 5.2.2.2), sometimes the source can still receive CIs from multiple receivers simultaneously. Under this situation, the source checks whether CR should be updated, as discussed above in Section 5.2.1.3. Then, it accepts a CI only if it is from the CR. Note that a CI leading to CR switching is also accepted.

5.2.2 Receiver Operations

Receivers need to detect congestion and convey the information to the source. At the same time they should suppress their feedback (CIs) whenever necessary, so that the source won't suffer from feedback implosion. Therefore, two major tasks are performed by receivers,

(1) Congestion detection

(2) Feedback suppression

The operations are explained by Figure 5.6 and the following specifications.

5.2.2.1 Congestion detection

Detecting congestion by means of accumulation have been well explained in Section 5.1.2, so we skip it here. In addition to accumulation, packet loss is also considered in case of some extraordinary situations. In fact, whenever packet loss is detected, CI is also sent. There is an one-bit field in CI indicating whether the congestion is detected by accumulation or by packet loss. The source adjusts the sending rate differently according to this bit (Section 5.2.1.2).

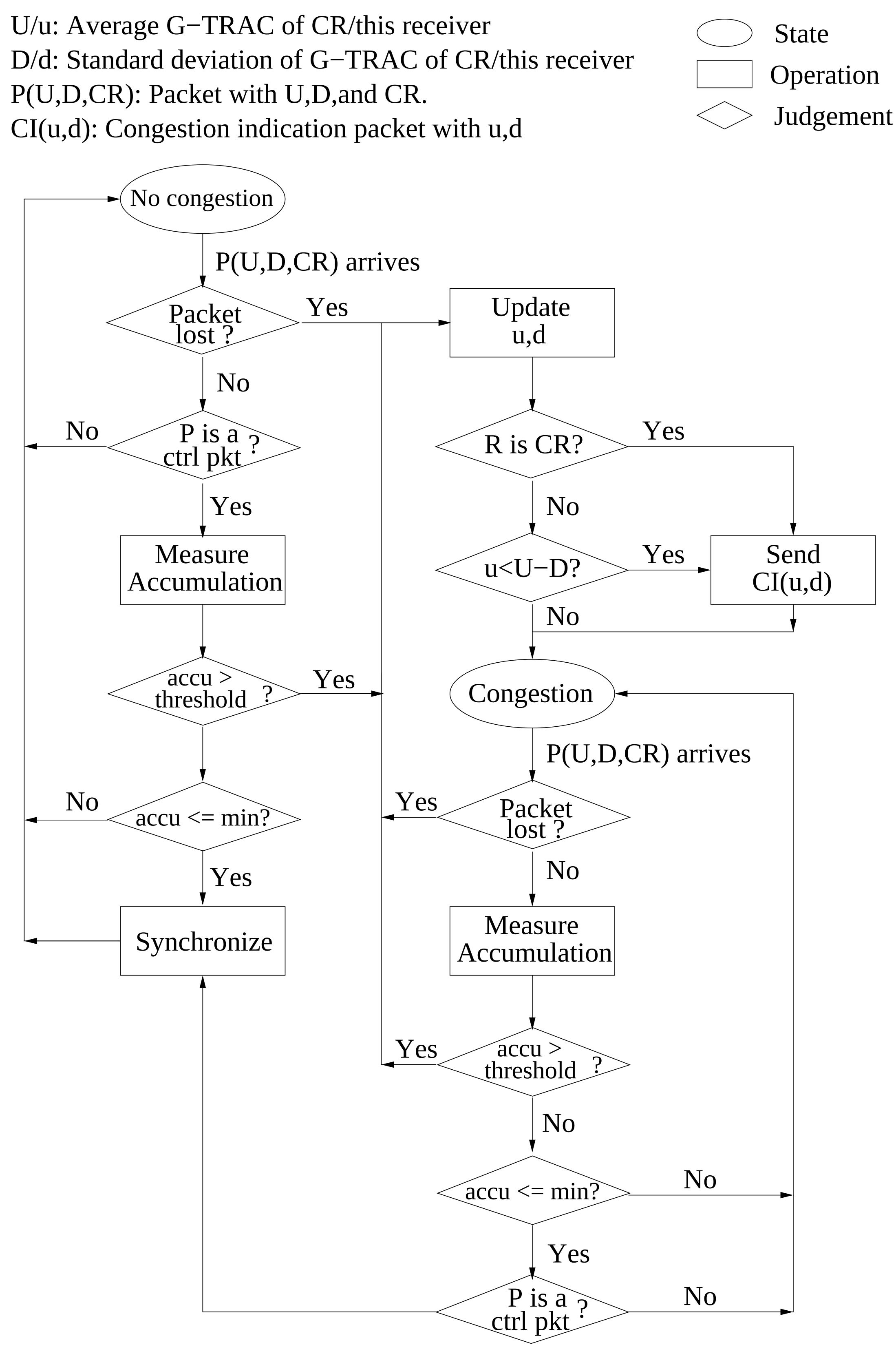


Figure 5.6: Receiver Operations

5.2.2.2 Feedback suppression

Even if a receiver detects congestion, it does not send CI if its average G-TRAC (u) is less than $U - D$, where U and D (the G-TRAC average and deviation of the CR) are multicast to receivers by source. If the CIs were sent, it would be discarded by the source anyway (Section 5.2.1.4). By this simple mean, a very large proportion of CIs are suppressed.

As we can see, both source operations and receiver operations are simple, and require only small constant number of states. That means MCA+ is easy to implement and deploy.

5.3 Simulations

We ran several *ns*-2 simulations to verify the performance of our scheme. The simulations include,

(1) Section 5.3.1: To verify the basic behaviors of MCA+ under simple situations.

(2) Section 5.3.2: To test the fairness between different MCA+ sessions in a linear network with multiple bottlenecks.

(3) Section 5.3.3: To verify that MCA+ is immune to Drop-to-Zero problem and effective at feedback suppression, and test the fairness between MCA+ sessions with multiple receivers and those with single receivers.

(4) Section 5.3.4: To verify that the source of MCA+ always adapts the sending rate to the most congestion bottleneck.

(5) Section 5.3.5: To test the performance of MCA+ in a heterogeneous and dynamic network.

In these simulations, the data packet size is 1000 bytes, initial RTT is 0.1 second. We used different bottleneck queue capacities to test MCA+ performance under situations with or without packet loss. To show the results clearly, we average the sending rates, the utilization rates and queue lengths over one-second periods.

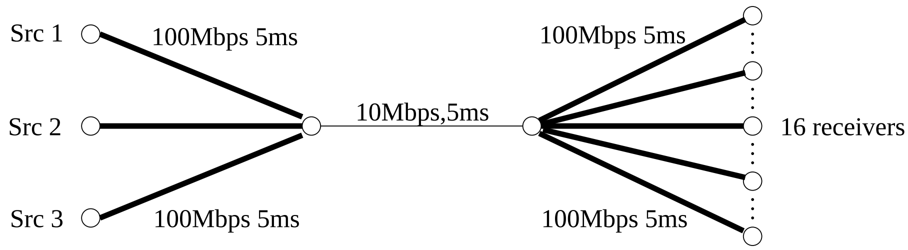


Figure 5.7: Single-Bottleneck Configuration with 16 Receiver Nodes

5.3.1 Basic Test on Simple One-Bottleneck Configuration

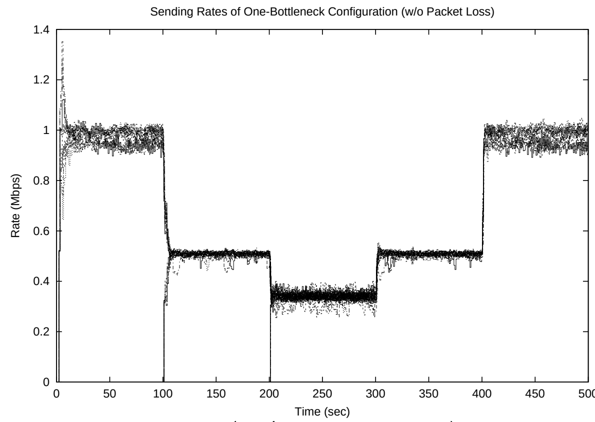
We first verify MCA+ performance on the simple topology in Figure 5.7. During different periods, there are 10 multicast flows from each source node to all 16 receiver nodes. The flows originated at Src 1 start at the beginning and end at 500th second, those originated at Src 2 start at 100th second and end at 400th second, those originated at Src 3 start at 200th second and end at 300th second. Therefore, in different periods, there may be 10, 20 or 30 flows sharing the 10Mbps bottleneck. The simulation time is 500 seconds.

The bottleneck queue capacity was set to 200 packets for lossless situation and 50 packets for lossy situation. As shown in Figure 5.8, under both situations, the transfer rates adapts to the bottleneck situation, while maintaining high bandwidth utilization and short queue.

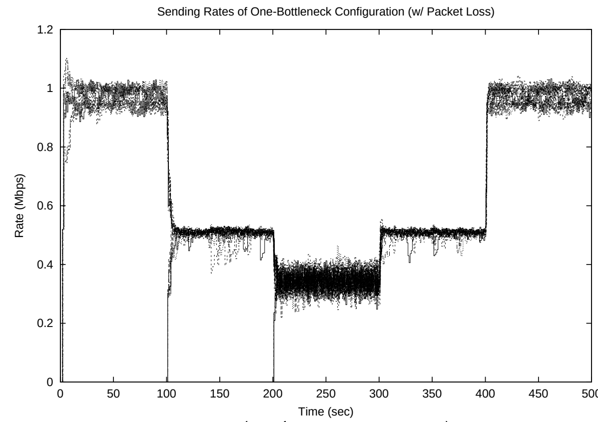
5.3.2 Fairness Test with Multiple Bottlenecks (Linear Network)

To check how MCA+ flows compete with each other when they pass different number of bottlenecks and what kind of fairness MCA+ can achieve, simulations were run on a multiple bottleneck topology (Figure 5.9).

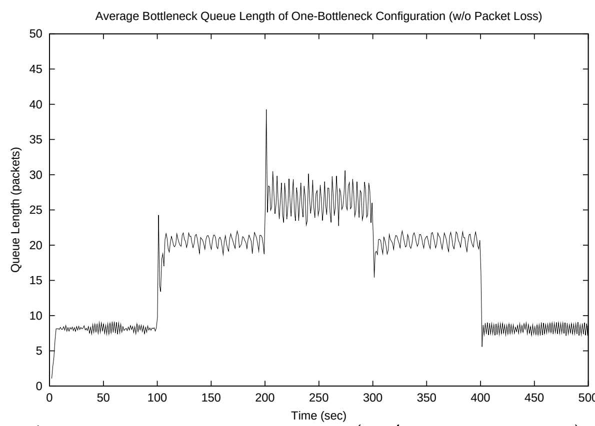
In this topology, there are 4 bottlenecks, each of 10Mbps bandwidth. Other links are of 100Mbps bandwidth. To reduce the effect of RTT, we set the bottleneck delays to 10 milliseconds and other delays to 1 millisecond. There are three types of flows: one-hop flows (i.e. flows going through one bottlenecks), two-hop flows and four-hop flows, as shown in Figure 5.9. Among them, for $i = (1, 2, 3, 4)$, 10 one-hop flows start at Src i and end at all the 16 receivers in Group i; for $i = (1, 3)$, 10 two-hop flows start at Src i and end at Group $i+1$; and there are 10 one-hop flows going from Src 1 to Group 4. Therefore, each bottleneck is shared by 30 flows. Bottleneck buffer size is set to 200 and 40 packets for lossless and lossy situation respectively. The simulation time is 400 seconds.



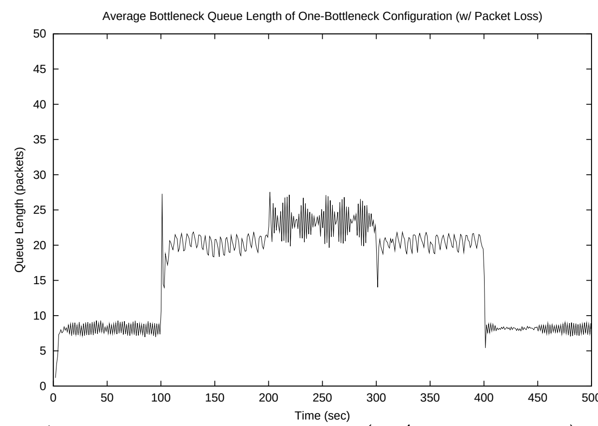
Rate (w/o packet loss)



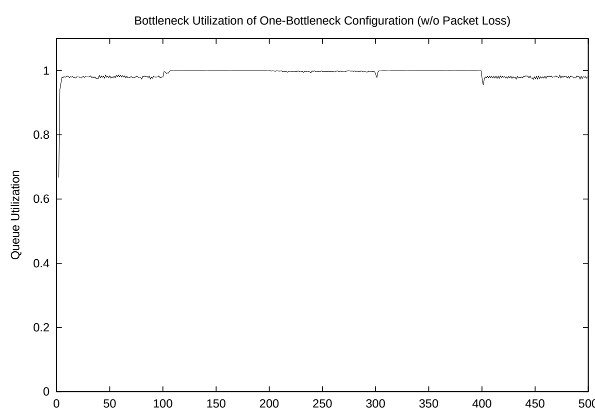
Rate (w/ packet loss)



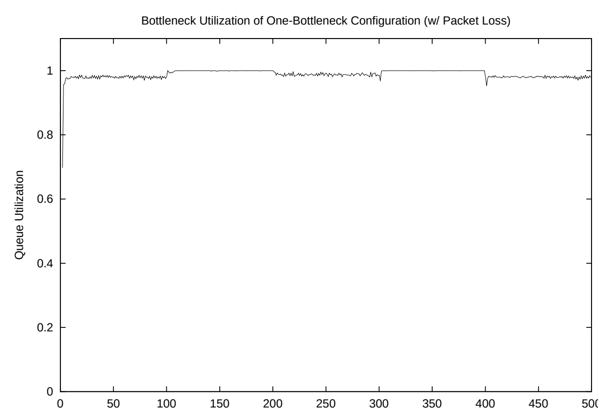
Average Queue Length (w/o packet loss)



Average Queue Length (w/ packet loss)



Bottleneck Utilization (w/o packet loss)



Bottleneck Utilization (w/ packet loss)

Figure 5.8: MCA+ Performance with Single Bottleneck and Dynamic Through Traffic
(MCA+ adapts to the bottleneck situation with high bandwidth utilization and low average queue length whether there is or is not enough buffer on the bottleneck to avoid packet losses.)

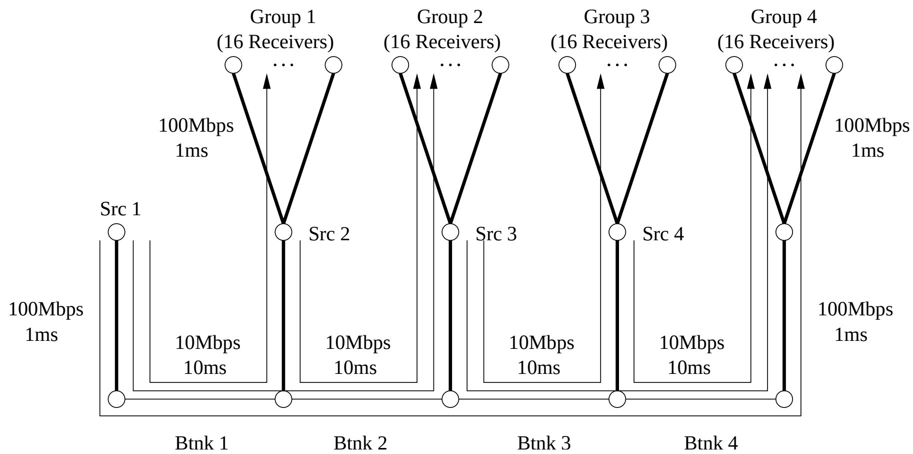


Figure 5.9: Linear Network: Multiple Bottlenecks Configuration

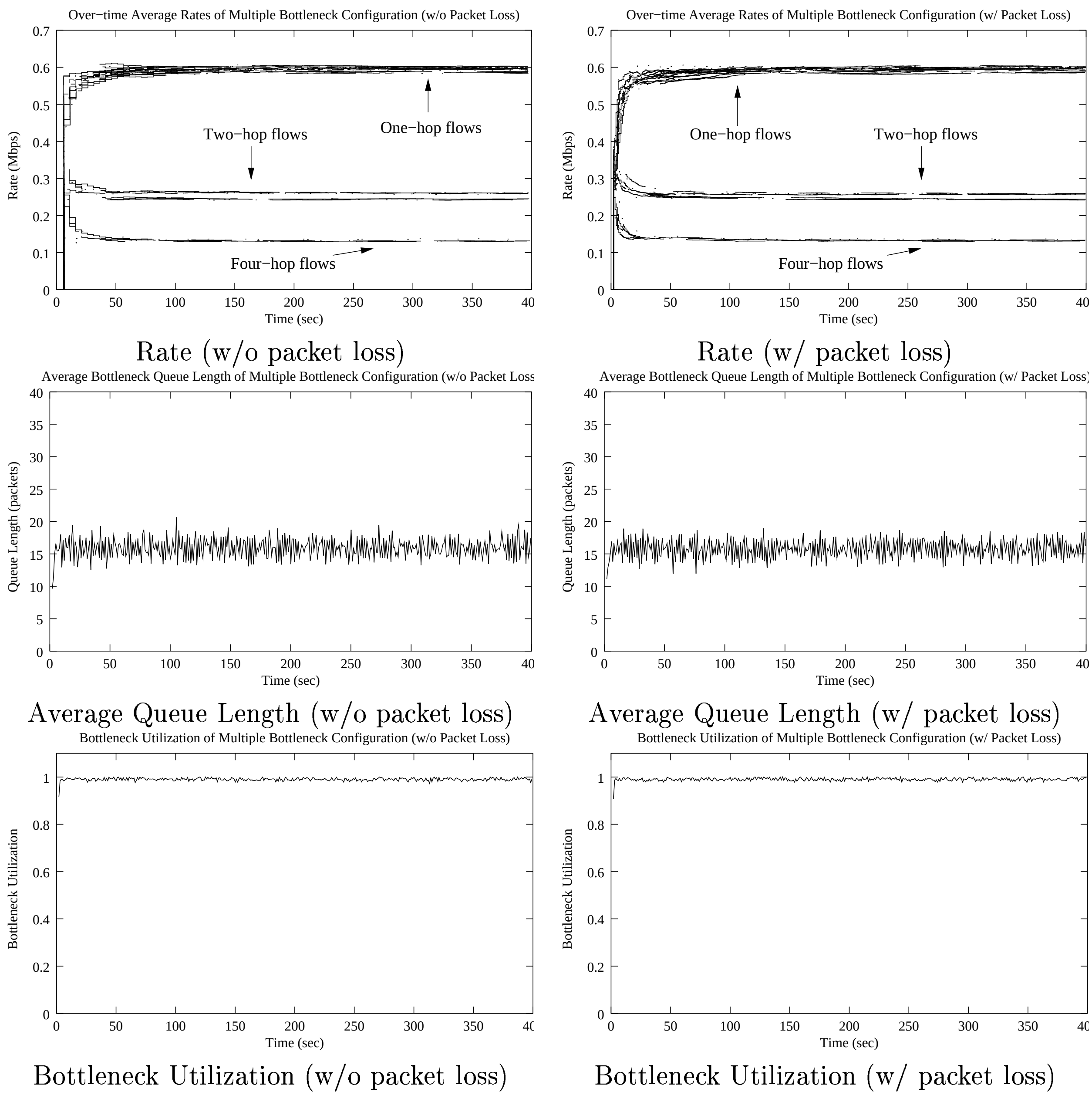
The average rates [28] in Figure 5.10 [29] show that the one-hop flows got approximately 2.5 times as much bandwidth as that by two-hop flows, while the two-hop flows got almost twice as much bandwidth as that by four-hop flows, which is close to what proportional fairness suggests theoretically. Again, the bottleneck queue length is low, and the utilization is high.

5.3.3 Test of Drop-to-Zero Avoidance and Friendliness to Unicast Flow, and Feedback Suppression

It is critical for a multicast congestion avoidance/control scheme to avoid reacting to more feedback than necessary, otherwise the sending rate will be constantly very low or even zero, which is known as the Drop-to-Zero problem. We designed a star topology in Figure 5.11 to generate asynchronous congestion on 64 different bottlenecks and checked the performance of MCA+. Bottlenecks of 1Mbps bandwidth and 5 millisecond delay are the links between the router and receivers. Between each pair of Source i and Receiver i, $i = 1 \ldots 64$, there are three unicast MCA+ flows (i.e. MCA+ flow with only one receiver). Also, there is a multicast MCA+ flow going from Source 65 to all 64 receivers. In consequence, each bottleneck is shared by four flows, which congest the link in an asynchronous manner. Bottleneck buffer size is set to 200 and 10 packets for lossless and lossy situation respectively. The simulation time is 400 seconds.

The over-time average rates (the mean and confidence interval of unicast flow rate are

[28] Average rate at time t is defined at the amount of data sent during $[0, t]$ divided by t.
[29] The results are of one randomly chosen bottlenecks. The situations of other bottlenecks are similar.



Rate (w/o packet loss) — Rate (w/ packet loss)

Average Queue Length (w/o packet loss) — Average Queue Length (w/ packet loss)

Bottleneck Utilization (w/o packet loss) — Bottleneck Utilization (w/ packet loss)

Figure 5.10: Fairness Results
(MCA+ achieves approximately proportional fairness.)

calculated with the samples of all 192 unicast flows) in Figure 5.12 show that the throughputs of the multicast MCA+ flow and the unicast flows are almost the same, no matter there is packet loss or not. The high bottleneck utilization and low average queue length shown in the figures are of one of the bottlenecks. Results of other bottlenecks are very similar.

In this simulation, under the lossy situation, the total number of feedback packets (CIs) sent by the multicast receivers is 5444, while the average number of CIs which would

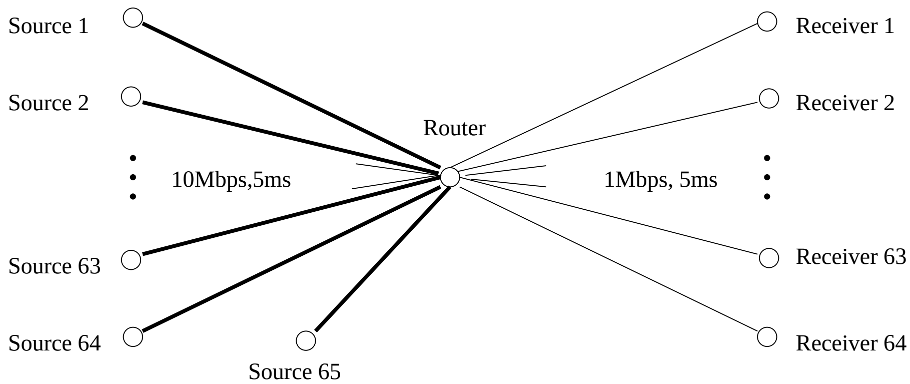


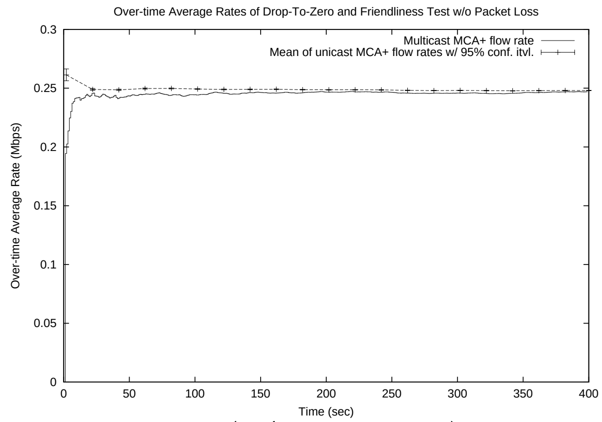
Figure 5.11: 64-Receiver Star Topology

have been sent per receiver without feedback suppression is 4978, and the average number of CIs sent by unicast flow receivers is 5057. We can see that the amount of CIs sent in the multicast session is very close to that by a unicast receiver, which indicates that *the feedback suppression mechanism in MCA+ is highly efficient.* For reference, under the lossless situation, the three numbers are 5605, 5135 and 5147 respectively, again close to each other.

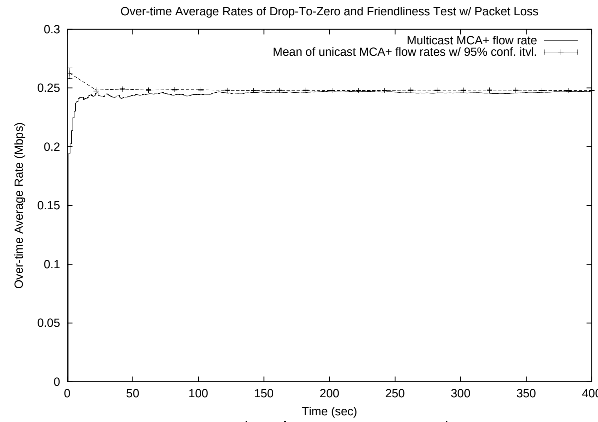
5.3.4 Test of Tracking The Most Congested Bottleneck

In this simulation, we changed the most congested bottleneck with an expected pattern. There is a multicast MCA+ flow from Source to all 32 receivers (Figure 5.13). During the whole simulation, one unicast MCA+ flow exists between Source and Receiver. At 200th, 400th and 600th second, we introduce 2, 3, 4 unicast MCA flows to the links between Source and Receiver 2, 3, 4 respectively. At 800th, 1000th and 1200th second, we stop the added flows in the reverse order. As the result, the most congested bottlenecks during different periods are as shown in Table 5.1. Bottleneck buffer size is set to 200 and 8 respectively for lossless and lossy situations.

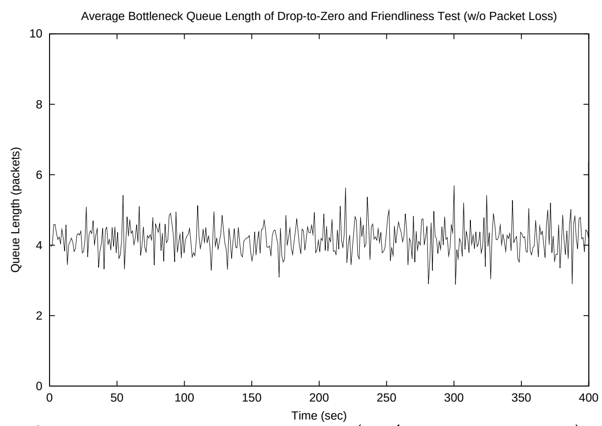
Rate changes as shown in Figure 5.14 indicate that the MCA+ flow always tracked the most congested bottleneck. There are more oscillations between 400 and 1000 than other time. It is because within that period the situations of the most congested bottleneck and the not-so-congested bottlenecks are close, and there are some back and forth CR switching. We focus on multicast behavior in this simulation, therefore we don't show the bottleneck utilization and queue length figures.



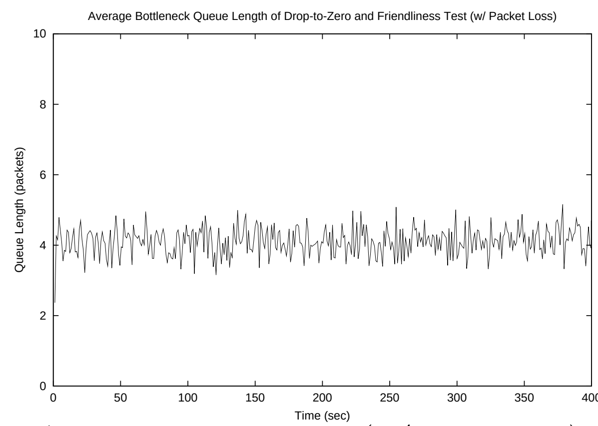
Rate (w/o packet loss)



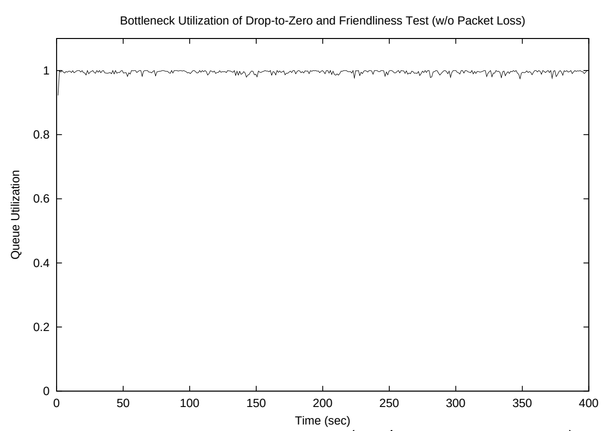
Rate (w/ packet loss)



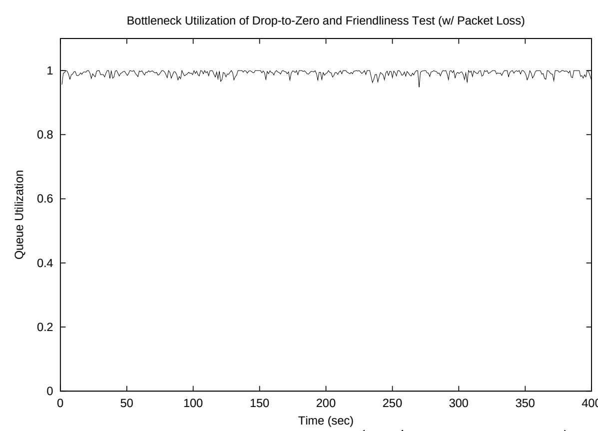
Average Queue Length (w/o packet loss)



Average Queue Length (w/ packet loss)



Bottleneck Utilization (w/o packet loss)



Bottleneck Utilization (w/ packet loss)

Figure 5.12: Drop-to-Zero Avoidance and Friendliness to Unicast Flows
(Multicast mca+ flow gets the same throughput as unicast flows do. Therefore, MCA+ is immune to drop-to-zero problem and is friendly to unicast flows.)

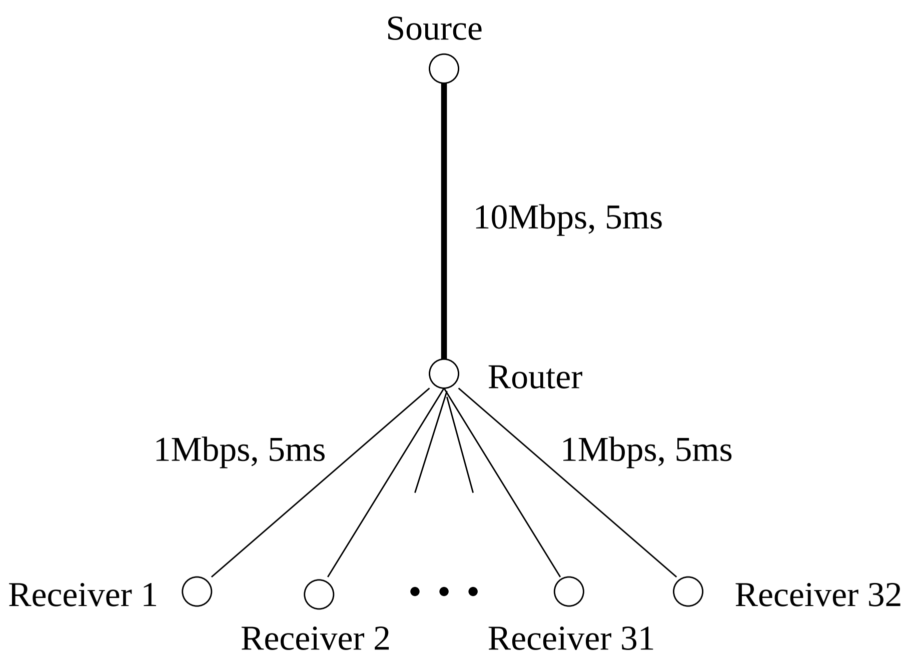


Figure 5.13: 32-Receiver Tree Topology

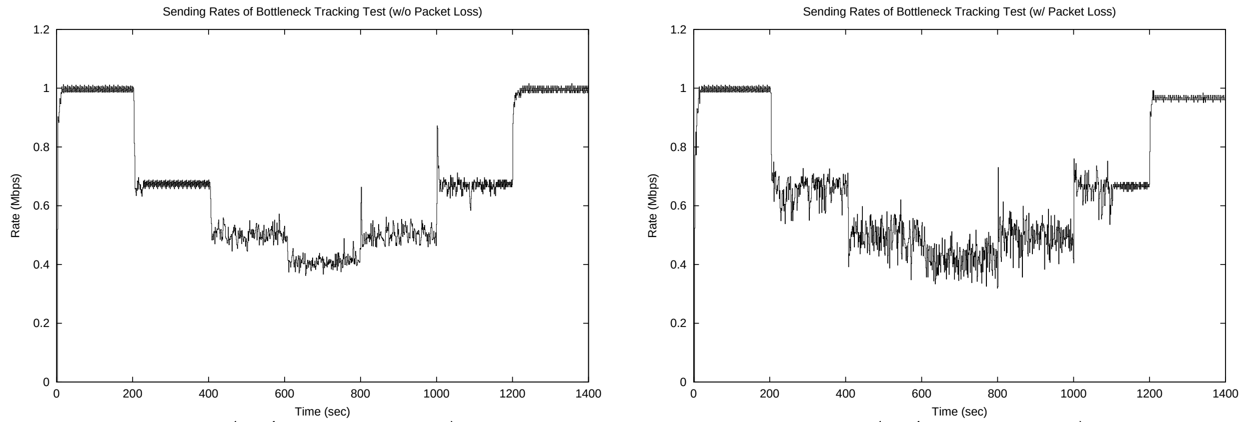
Table 5.1: Dynamics of Most Congested Bottleneck

Periods	Most Congested Link
[0, 200) and [1200, 1400]	Link 1
[200, 400) and [1000, 1200)	Link 2
[400, 600) and [800, 1000)	Link 3
[600, 800)	Link 4

(Link i is the link between Router and Receiver i.)

5.3.5 Test of Performance in Dynamic Network

It is also desirable to test the performance of MCA+ with some *unexpected* traffic patterns. We designed a network with heterogeneous delays, as shown in (Figure 5.15). Each link has 2Mbps bandwidth. Among all the links, 2 links at the first level, 4 links at the second level, and 8 links at the third level have 200ms delay, while all other links have 20ms delay. We arrange the links so that on any path between the source and a receiver, there is at most one link of 200ms delay. Furthermore, we generate dynamic traffic in this network. On each link, three unicast MCA+ flows are randomly turned on and off according to Pareto distribution with average period length of 60 and 30 seconds respectively. Moreover, all receivers except one in the multicast session join and leave randomly, again according to



Rate (w/o packet loss) Rate (w/ packet loss)

Figure 5.14: Responsiveness in Dynamic Network
(The sending rate of MCA+ always adapts to the most congested bottleneck.)

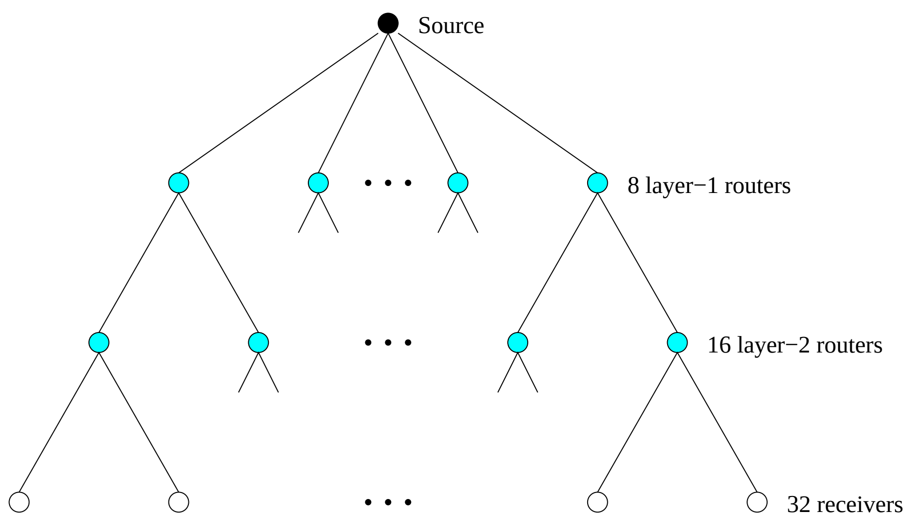


Figure 5.15: Heterogeneous Dynamic Network

Pareto distribution. The average in-session time is 60 seconds and average out-of-session time is 30 seconds. We keep one receiver always in the session so that feedback always exists and the sending rate won't increase infinitely. Bottleneck buffer sizes are set to 50 packets.

We ran the simulation for ten times. The average throughput of the multicast MCA+ flow is **232** Kbps, with standard deviation of **10** Kbps, which shows that MCA+ works well in a heterogeneous and dynamic environment. The average number of total feedback packets received by the multicast source is 3009, with standard deviation of 777, again showing the effectiveness of feedback suppression. The multicast behavior is still our major concern here, and we omit other statistics.

5.4 Summary

We have proposed MCA+, an end-to-end rate-based multicast congestion *avoidance* scheme. This scheme leverages the concept of *accumulation* developed in our prior work [110], which is defined as the number of buffered bits of a flow inside the network. By accumulation measurement extended to multicast, receivers detect congestion without necessarily inducing packet loss and send congestion indications (CIs) back to source for the purpose of rate control. Suppression filter is applied to those CIs before sending. The source keeps a record of the slowest receiver as congestion representative (CR), and only accepts its CIs for adapting the transfer rate according to AIMD rate control policy. Both CR switching and feedback suppression make use of a new metric, Good Throughput Rate At Congestion (G-TRAC), defined as the product of packet survival rate (one minus packet loss rate) and receiving rate during congestion. MCA+ does not require each receiver to *continuously* exchange packets with the source.

In brief, the scheme (1) does not suffer from Drop-to-Zero problem, and, (2) is friendly to unicast flows, (3) achieves high bottleneck utilization and low average queues, (4) is approximately proportionally fair. We expect the deployment scenarios for such a scheme to be at ISPs who can control their infrastructure (i.e. can manage buffers, isolate non-congestion avoidance flows) and want to gain efficiencies due to multicast on an edge-to-edge basis.

CHAPTER 6
Summary And Future Work

We summarize our contributions in this chapter and briefly discuss some directions of future research.

6.1 Summary

We have considered the multicast congestion management problem for several situations with different restrictions, and have provided *end-to-end* solutions to them without requiring coordination from within the network. These solutions show better performance than previous work.

For the situation where no support particular for congestion control is provided by receivers, we developed LE-SBCC (Chapter 2). In this scenario, receivers have facilities of multicast transport protocols (without built-in congestion control functions) such as receiving data and monitoring their quality, but they do not have any designs specific for congestion control purpose such as measuring available bandwidth. Due to the lack of receiver support, the congestion control facilities have to be deployed on source side, and the scheme has to be single-rate, i.e. all receivers receive at the same speed. In LE-SBCC, the source leverages the ACK or NAK common in many transport protocols to derive implicit congestion information, and filter them properly before using them for rate adaptation. To do the filtering, we design a unique filter cascade. This cascade can dynamically and effectively locate the most congested path in a multicast tree, and pass just enough congestion signals for rate adaption. We believe that it is the first purely source-based scheme to fully address all components of single-rate source-based multicast congestion control, i.e. *drop-to-zero* issues, *TCP friendliness*, RTT estimation and robustness issues.

For another situation where only one multicast group is allowed for a multicast session but we can assume support from receivers, we have proposed ORMCC (Chapter 3). It uses a novel metric called *Throughput Rate At Congestion* (TRAC) and distributes a portion of the task of selecting the most congested path to receivers without adding considerable control traffic. In fact, by using TRAC, the design eliminates the needs to measure RTT between the source and all receivers (except the most congested one), therefore suppresses this type of redundant control traffic required by other similar protocols. TRAC is also used to suppress feedback from other receivers. When the network is stable, the total feedback traffic volume of ORMCC remains approximately constant without regard to the multicast session size. The computation complexity and state requirement of both source and receiver side are small and constant, i.e. $O(1)$. As shown by different simulations, ORMCC exhibits superior performance compared to the best-known single-rate multicast congestion control schemes, PGMCC [94] and TFMCC [114].

For the third situation where there is no limitation on receiver support and group number, we have designed GMCC by extending ORMCC into a multi-rate framework. It provides a dynamic set of sub-sessions to receivers within an overall multicast session. In each of these sub-sessions, the sending rate is adjusted using ORMCC-like mechanism independently from other sub-sessions according to network congestion status. A receiver can join or leave these sub-sessions to get different sums of throughput rate according to the capacity of the path between the source and itself. By combining the fine-granularity control (the former technique) and the coarse-granularity control (the latter technique), it reduces the IGMP join and leave operations dramatically, and avoids redundant layer settings (e.g. number of layers). Consequently, the control traffic and the burden of intermediate routers are greatly reduced. We also proposed a novel technique called *probabilistic inter-layer bandwidth shifting* to distinguish *intra-layer congestion* from *inter-layer congestion* and explore hidden available bandwidth, which is a problem not addressed before. By qualitative discussion, we showed that GMCC has advantages over a recently published similar scheme SMCC [59].

For the situation where a single-rate *congestion avoidance* scheme is needed, we have MCA+. It includes a novel and scalable end-to-end mechanism to detect incipient congestion for multicast at receiver side based on a new concept called *accumulation*. It incurs shorter average bottleneck queue and achieves higher throughput, and is the first end-to-end multicast congestion avoidance scheme with good scalability to our knowledge.

To verify the performance of our schemes, we have run simulations in ns-2 [2]. To test the scalability of those schemes expected to scale well (ORMCC and GMCC), we have simulated them in ROSS [19] with thousands of receivers (MCA+ was not simulated with large scale setting because it has similar structure with ORMCC). We have also implemented

LE-SBCC and ORMCC on real systems to show their practical values.

Finally, we would like to mention that in the schemes we have developed, the rate adaptation module is independent of other parts. Therefore, different rate adaptation policies such as TFRC [33], binomial [5] and MCFC [40] can easily fit in. Other parts are then reusable.

6.2 Future Research

Due to its simplicity, single-rate multicast congestion control has been relatively well developed. The future research will hence focus on multi-rate aspect.

In GMCC, as shown in Chapter 4 Section 4.2.2, the responsiveness of receivers to more available bandwidth is not satisfactory. If we change the parameter settings, the responsiveness may be improved but more oscillations will come along. We would like to study the trade-off among different parameter settings in terms of responsiveness and oscillation, and find the optimal balance between them. Besides, it will be good for receivers to join a layer smoothly without incurring sudden increase of traffic.

We also want to study the parameters of probabilistic inter-layer bandwidth shifting. Different values may change the effectiveness of this technique (e.g. detect hidden bandwidth more quickly), but will affect other parallel flows at the same time (e.g. be more unfriendly to them). The balance point or range are desired.

Furthermore, TAF may not be the only choice that we can use in GMCC for the sake of CR selection and triggering join/leave operations. There are possibilities of using other metrics.

GMCC has been proposed for a situation where we can get unlimited groups for a multicast session. Although it is not a problem for SSM (source specific multicast) [44], in other scenarios it may not be feasible. Therefore, we need to extend GMCC to optimize its performance with limited layers. One possible method is to let the source adjust receivers' parameters and thus dictate their join and leave behavior. However, since there are not absolute metrics (e.g. bottleneck bandwidth) to dictate the optimization procedure, this is a challenging problem.

Studying GMCC in combination with specific types of applications is an interesting direction. For example, in a video streaming application, receivers may retrieve data from multiple different coding layers to improve the quality of received images. The coding layers here coincide the layers (i.e. sub-sessions) within GMCC. However, to seamlessly combine GMCC with video streaming applications, we must study how the video coding affects GMCC's sending rate regulation and vice versa. Moreover, currently GMCC receivers can only subscribe to layers accumulatively, while video streaming applications may require receivers to join discrete layers, presenting another challenging problem to GMCC.

In wireless networks that are becoming more and more popular today, multicast is a natural feature (at least true for now since we are using omni-directional antennas). Therefore, multicast has more prosperous future in wireless networks. However, the bandwidth resource is more scarce and precious in wireless networks, calling for more efficient congestion control algorithms. Due to the special characteristics of wireless networks, such as high packet error rate (and more packets are dropped for error but not for congestion) and node mobility, the multicast congestion control problem is more challenging.

In sensor networks that is a special form of wireless networks, multicast may also be utilized. Each node (sensor) is so simple and can only use so limited energy, that we cannot implement complex algorithms on them. Also the traffic pattern may be different, which requires congestion control algorithm to fit. After all, we believe there are much more challenging problems to find out in sensor networks.

Back to wired networks, although the deployment of multicast is growing [121], due to various reasons, IP multicast only cover a small portion of the Internet. People are exploring to use multicast on overlay networks (e.g. [4]) or the mix of them. We may exploit the results that we already have for IP multicast congestion control and provide solutions cheaper than a set of simply bundled unicast congestion control algorithms.

LITERATURE CITED

[1] Clio Albuquerque, Brett J. Vickers, and Tatsuya Suda, "Credit-based source-adaptive multilayered video multicast," *Performance Evaluation*, Volume 40, Issues 1-3 , March 2000, Pages 135-159

[2] S. Bajaj, et al, "Improving Simulation for Network Research", *Technical Report 99-702b, University of Southern California, March 1999, revised September 1999*

[3] A. Ballardie, "Core Based Trees (CBR) Multicast Routing Architecture," *IETF RFC 2201*, Sep. 1997.

[4] Suman Banerjee, Bobby Bhattacharjee, Christopher Kommareddy, "Scalable Application Layer Multicast", SIGCOMM 2002, August 2002

[5] D. Bansal and H. Balakrishnan, "Binomial Congestion Control Algorithms," *INFOCOM 2001*, Apr 2001.

[6] Anindya Basu and S. Jamaloddin Golestani, "Architectural Issues for Multicast Congestion Control," *NOSSDAV 1999.*

[7] S. Bhattacharyya et al, "A Novel Loss Indication Filtering Approach for Multicast Congestion Control," *J. of Comp. Commns*, Feb '01.

[8] S. Bhattacharyya, D. Towsley and J. Kurose, "Efficient Multicast Flow Control using Multiple Multicast Groups," *U.Mass, Amherst, CMPCSI Technical Report TR 97-15*, 1997.

[9] S. Bhattacharya, D. Towsley and J. Kurose, "The Loss Path Multiplicity Problem in Multicast Congestion Control," *INFOCOM '99*, March '99.

[10] S. Blake, D. Black, M. Carlson, E. Davies, Z. Wang, W. Weiss, "An Architecture for Differentiated Services," *RFC 2475*, IETF, December 1998

[11] Marjory S. Blumenthal, and David D. Clark, "Rethinking the design of the Internet: The end to end arguments vs. the brave new world," *ACM Transactions on Internet Technology,* 1(1): 70-109, Aug. 2001

[12] J.C. Bolot, T. Turletti, I. Wakeman, "Scalable Feedback Control for Multicast Video Distribution in the Internet," *SIGCOMM '94*, Aug '94.

[13] Ch. Bouras, A. Gkamas, "A Mechanism for Multicast Multimedia Data with Adaptive QoS Characteristics," *Lecture Notes in Computer Science*, Vol. 2213, pp. 74-88, 2001

[14] S. Bradner et al, "IETF criteria for evaluating reliable multicast transport and application protocols," *RFC 2357*, June '98.

[15] L. Brakmo and L. Peterson, "TCP Vegas: End to End Congestion Avoidance on a Global Internet," *IEEE JSAC,* Vol 13, No. 8, Oct 1995

[16] J.W. Byers, et al, "FLID-DL Congestion Control for Layered Multicast," *NGC*, Nov '00.

[17] J. Byers, G. Kwon, "STAIR: Practical AIMD Multirate Multicast Congestion Control", *NGC 2001*

[18] J. Byers, M. Luby, M. Mitzenmacher, "Fine-Grained Layered Multicast", *Infocom 2001*

[19] Christopher D. Carothers, David Bauer, Shawn Pearc, "ROSS: A High-Performance, Low Memory, Modular Time Warp System", *14th Workshop on Parallel and Distributed Simulation (PADS 2000)*, May 2000

[20] C. Casetti, M. Gerla, S. S. Lee, S. Mascolo and M. Sanadidi, "TCP with Faster Recovery," *Proceedings of MILCOM 2000*, Los Angeles, CA, October 2000

[21] S. Y. Cheung, M. Ammar, and X, Li, "On the Use of Destination Set Grouping to Improve Fariness in Multicast Video Distribution," *Proc. of INFOCOM'96*, March 1996

[22] Dah Ming Chiu, Miriam Kadansky, Joe Provino, "A Congestion Control Algorithm for Tree-based Reliable Multicast Protocols,", InfoCom 2002

[23] D. Chiu and R. Jain, "Analysis of the Increase/Decrease Algorithms for Congestion Avoidance in Computer Networks," *Journal of Computer Networks and ISDN*, Vol. 17, No. 1, June 1989, pp. 1-14

[24] R. Cruz, "Quality of Service Guarantees in Virtual Circuit Switched Networks," *IEEE Journal on Selected Areas in Communications*, 13(6):1048-1056, Aug 1995.

[25] Dante DeLucia, Katia Obraczka, "A Multicast Congestion Control Mechanism Using Representatives", *Proceedings of the IEEE ISCC 1998*

[26] S. Deering, "Host Extensions for IP Multicasting", *RFC 1112*, August 1989

[27] Christophe Diot, Brian Neil Levine, Brian Lyles, H. Kassan, Doug Balsiefien, "Deployment Issues for the IP Multicast Service and Architecture," *IEEE Network, special issue on Multicasting. January/February 2000.*

[28] Sudhir Dixit, "IP over WDM: building the next-generation optical Internet," by John Wiley & Sons, Inc. ISBN 0-471-21248-2, 2003.

[29] D. Estrin, D. Farinacci, A. Helmy, D. Thaler, S. Deering, M. Handley, V. Jacobson, C. Liu, P. Sharma, and L. Wei,"Protocol Independent Multicast - Sparse Model (PIM-SM) : Protocol Specification," *IETF RFC 2362*, June 1998.

[30] A. Fei, J. Cui, M. Gerla, M. Faloutsos, "Aggregated Multicast: an Approach to Reduce Multicast State", *Globecom 2001*

[31] William C. Fenner, "RFC 2236: Internet Group Management Protocol, Version 2", *IETF*

[32] Fethi Filali, Walid Dabbous, "A Simple and Scalable Buffer Management Mechanism for Multicast flows," *Proc. of ICNP 2002*, October 2002, Paris.

[33] Sally Floyd, Mark Handley, Jitendra Padhye, and Joerg Widmer, "Equation-Based Congestion Control for Unicast Applications," SIGCOMM 2000, August 2000.

[34] Sally Floyd, et al, "A Reliable Multicast Framework for Light-weight Sessions and Application Level Framing," *IEEE/ACM Transactions on Networking*, Volume 5, Number 6, pp. 784-803, December 1997.

[35] Sally Floyd, Kevin Fall, "Promoting the Use of End-to-End Congestion Control in the Internet", *IEEE/ACM Transactions on Networking*, August 1999

[36] T. Friedman, D. Towsley, "Multicast Session Membership Size Estimation," *IEEE Infocom*, New York, Mar. 1999

[37] Thomas T. Fuhrmann, Jorg Widmer, "On the Scaling of Feedback Algorithms for Very Large Multicast Groups", *Computer Communications*, 24(5-6): 539-547, Mar. 2001.

[38] Yung-Sze Gan, Chen-Khong Tham, "Loss differentiated multicast congestion control," *Computer Networks*, Volume 41, Issue 2 , 5 February 2003, Pages 161-176

[39] J. Golestani, "Fundamental Observations on Multicast Congestion Control in the Internet," *INFOCOM 1999*, March '99.

[40] S.J. Golestani and S. Bhattacharyya, "A Class of End-to-End Congestion Control Algorithms for the Internet", *Proceedings of ICNP*, 1998.

[41] R. Gopalakrishnan, James Griffioen, Gisli Hjalmtysson, Cormac J. Sreenan, and Su Wen, "A Simple Loss Differentiation Approach to Layered Multicast," *Proc. of INFOCOM'00*, March 2000.

[42] S. Gorinsky, K. K. Ramakrishnan, H. Vin, "Addressing Heterogeneity and Scalability in Layered Multicast Congestion Control," *Technical Report TR2000-31, Dept of Computer Sciences, Univ of Texas at Austin*, Nov 2000.

[43] S. Ha, K.-W. Lee and V. Bharghavan, "A Simple Mechanism for Improving the Throughput of Reliable Multicast." ICCCN'99, Boston, MA. October 1999.

[44] Hugh W. Holbrook, David R. Cheriton, "IP Multicast Channels: Express Support for Large-scale Single-source Applications," SIGCOMM 1999.

[45] V. Jacobson, "Congestion avoidance and control," *SIGCOMM*, Aug '88

[46] J. M. Jaffee, "Bottleneck flow control," IEEE Trans. Commun., vol. COM-29, pp. 954-962, July 1980

[47] Srinivasan Jagannathan, Kevin Almeroth, and Anurag Acharya, “Topology Sensitive Congestion Control for Real-Time Multicast,” *NOSSDAV 2000*

[48] T. Jiang, E.W. Zegura, and M.H. Ammar, “Inter-receiver fair multicast communication over the Internet,” *NOSSDAV 99*

[49] Kevin Fall, Sally Floyd, “Simulation-based Comparisons of Tahoe, Reno, and SACK TCP,” *Computer Communication Review*, V. 26 N. 3, July 1996, pp. 5-21

[50] M. Kadansky, D. Chiu, J. Wesley, and J. Provino, “Tree-based Reliable Multicast (TRAM)”, IEEE Internet Draft, draft-kadansky-tram-01.txt, September 1999.

[51] K. Kang, D. Lee, H. Y. Youn and K. Chon, “NLM: network-based layered multicast for traffic control of heterogeneous network,” *Computer Communications*, Volume 24, Issues 5-6 , 15 March 2001, Pages 525-538

[52] Koushik Kar, Saswati Sarkar, Leandros Tassiulas, “Optimization Based Rate Control for Multirate Multicast Sessions,” Proceedings of INFOCOM 2001, Alaska,

[53] Koushik Kar, Saswati Sarkar, and Leandros Tassiulas, “A Scalable Low-Overhead Rate Control Algorithm for Multirate Multicast Sessions,” *IEEE Journal on Selected Areas in Communications*, Vol.20, No.8, October 2002.

[54] S. Kasera et al, “Scalable Fair Reliable Multicast Using Active Services,” *IEEE Network Magazine*, January/February 2000.

[55] M. Kawada, H. Morikawa, and T. Aoyama, “”Cooperative Inter-stream Rate Control Scheme for Layered Multicast,” *Proceedings of Symposium on Applications and the Internet (SAINT2001)*, pp. 147-154, San Diego, CA, USA., January 2001.

[56] F. P. Kelly, A.K. Maulloo and D.K.H. Tan, “Rate control in communication networks: shadow prices, proportional fairness and stability”, *Journal of the Operational Research Society 49* (1998), 237-252.

[57] I. El Khayat and G. Leduc, “Congestion Control for Layered Multicast Transmission,” *Networking and Information Systems Journal*, vol. 33-4, 2000, pp. 559-573

[58] Ibtissam El Khayat, Guy Leduc, “A stable and flexible TCP-friendly congestion control protocol for layered multicast transmission,” *Proc. of 8th International Workshop IDMS'2001.*

[59] Gu-In Kwon, John Byers, “Smooth Multirate Multicast Congestion Control”, *IEEE INFOCOM '03*, April 2003

[60] A. Legout, E. W. Biersack, “Pathological Behaviors for RLM and RLC,” NOSSDAV 2000.

[61] Arnaud Legout, Jorg Nonnenmacher, and Ernst W. Biersack, "Bandwidth-Allocation Policies for Unicast and Multicast Flows," *IEEE/ACM Transactions on Networking*, Vol.9, No.4, August 2001.

[62] A. Legout, E. Biersack, "PLM: Fast Convergence for Cumulative Layered Multicast Transmission Schemes", *Proc. of ACM SIGMETRICS*, 2000

[63] Jiang Li, Shivkumar Kalyanaraman, "Using Average Attenuation Factor to Locate the Most Congested Path for Multicast Congestion Control", *Technical Report, CS, RPI*, 2003, available at http://www.cs.rpi.edu/~lij6/Research/papers.html

[64] V.O.K. Li, Zaichen Zhang, "Internet multicast routing and transport control protocols," *Proceedings of the IEEE*, Vol.: 90 Issue: 3 , March 2002 Page(s): 360 -391

[65] X. Li, S. Paul, P. Pancha, and M. Ammar, "Layered video multicast with retransmission (LVMR): evaluation of error recovery schemes," *Proceedings of the Sixth International Workshop on Network and Operating System Support for Digital Audio and Video*, St Louis, May 1997.

[66] Xue Li, Sanjoy Paul, Mostafa Ammar, "Multi-Session Rate Control for Layered Video Multicast," *Proc. of Multimedia Computing and Networking*, San Jose, California, USA, January 1999.

[67] Sam Liang, David Cheriton, "TCP-SMO: Extending TCP to support Medium-Scale Multicast Applications," InfoCom 2002

[68] J. C. Lin, S. Paul, "RMTP: A Reliable Multicast Transport Protocol," *Proceedings of IEEE INFOCOM '96*, Pages 1414-1424

[69] C. Liu, J. Nonnenmacher, "Broadcast Audience Estimation," *IEEE Infocom*, Tel Aviv, Israel, Mar. 2000

[70] Chao Liu, Xiuming Shan, "Self-suppressed nack-based multicast congestion control," *Proceedings of 10th International Conference on Telecommunications*, Volume: 1 , Feb. 23 - Mar. 1, 2003, Page(s): 456 -461

[71] Jiangchuan Liu, Bo Li, and Ya-Qin Zhang, "A Hybrid Adaptation Protocol for TCP-Friendly Layered Multicast and Its Optimal Rate Allocation," IEEE INFOCOM'02, New York City, June 2002

[72] S.H. Low, "A Duality Model of TCP and Queue Management Algorithms," *Proceedings of ITC Specialist Seminar on IP Traffic Measurement, Modeling and Management*, Sep 2000, Monterey, CA.

[73] S.H. Low, L.L. Peterson, and L. Wang, "Understanding Vegas: A Duality Model," *Proceedings of ACM SIGMETRICS*, Boston, MA, June 2001

[74] Michael Luby, Vivek K. Goyal, Simon Skaria, Gavin B. Horn, "Wave and Equation Based Rate Control Using Multicast Round Trip Time," *SIGCOMM* 2002.

[75] J. Macker, R. Adamson, "A TCP Friendly, Rate-Based Mechanism for Nack-Oriented Reliable Multicast Congestion Control", *Globecom 2001*

[76] Matthew Mathis, Jeffrey Semke, Jamshid Mahdavi, and Teunis Ott, "The Macroscopic Behavior of the TCP Congestion Avoidance Algorithm", *Computer Communications Review*, volume 27, number 3, Jul. 1997.

[77] S. McCanne, V. Jacobson, M. Vetterli, "Receiver-driven Layered Multicast," *SIGCOMM '96*, Aug '96

[78] William Mendenhall, "Introduction to Probability and Statistics: Third Edition", *Duxbury Press*, 1997

[79] J. Mo, R. La, V. Anantharam and J. Walrand, "Analysis and Comparison of TCP Reno and Vegas," *Proc. INFOCOM'99*, Mar 1999.

[80] Robert Morris, "Bulk Multicast Transport Protocol," *Proc. of IEEE INFOCOM'97*, April 1997.

[81] mrouted 3.9 beta3-1: ftp://ftp.rge.com//pub/communications/ipmulti/beta-test/mrouted-3.9-beta3.tar.gz, mrouted Linux patch: ftp://ftp.debian.org/debian/dists/potato/non-free/source/net/mrouted_3.9-beta3-1.diff.gz

[82] K. Nakauchi, H. Morikawa, and T. Aoyama, """Network-supported Rate Control Mechanism for Multicast Streaming Media," *Proceedings of Symposium on Applications and the Internet (SAINT2001)*, pp. 131-139, San Diego, CA, USA, January 2001.

[83] T. Nguyen, K. Nakauchi, M. Kawada, H. Morikawa, T. Aoyama, "Rendezvous Points Based Layered Multicast", *IEICE Trans. Commun.*, Vol. E84-B, No. 12, Dec. 2001.

[84] Jorg Nonnenmacher, Ernst W. Biersack, "Scalable Feedback for Large Groups", *IEEE/ACM Transactions on Networking*, 7(3): 375-386, Jun. 1999.

[85] Jitendra Padhye, Victor Firoiu, Don Towsley, Jim Kurose, "Modeling TCP Throughput: A Simple Model and its Empirical Validation," *SIGCOMM*, Aug. 1998.

[86] A. Parekh and R. Gallager, "A Generalized Processor Sharing Approach to Flow Control in Integrated Services Networks: The Single-Node Case," *IEEE/ACM Trans. on Networking*, 1(3):344–357, Jun 1993.

[87] Jon Postel, "Internet Protocol", *RFC 791*, Septermber 1981

[88] K. Ramakrishnan, S. Floyd, "A Proposal to add Explicit Congestion Notification (ECN) to IP," *RFC 2481*, Jan '99.

[89] S. Ramakrishnan, S. Kalyanaraman, J. Wen, H. Ozbay, "Effect of Time Delay in Network Traffic Control," Short Paper, *Automatic Controls Conference (ACC)*, 2001.

[90] H. Ramamurthy, A. Karandikar, R. Verma, "A grouping screme for reliable multicast congestion control," *The 8th International Conference on Communication Systems, 2002*, Volume: 2 , 25-28 Nov, 2002 Page(s): 938 - 942

[91] Injong Rhee, Nallathambi Ballaguru, George N Rouskas, "MTCP: Scalable TCP-like Congestion Control for Reliable Multicast," INFOCOM'99.

[92] I. Rhee, V. Ozdemir, and Y. Yi, "TEAR: TCP Emulation at Receivers – Flow Control for Multimedia Streaming", *NCSU Technical Report*, Apr. 2000.

[93] I. Rimac, W. Liese, J. Schmitt, R. Steinmetz, "Equation-based approach to TCP-compatible multicast congestion control for layered transmission in low-multiplexing environments," *Proceedings of the 2003 IEEE International Performance, Computing, and Communications Conference*, 2003, Page(s): 469-473

[94] L. Rizzo, "PGMCC: A TCP-friendly Single-Rate Multicast Congestion Control Scheme", *SIGCOMM '00*, Aug '00.

[95] Fan Rui, Cheng Shi-duan, "A multicast congestion control scheme for heterogeneous receivers," *Proceedings of International Conference on Communication Technology, 2003*, Volume: 2 , April 9 - 11, 2003 Page(s): 1749 -1753

[96] J. H. Saltzer, D. P. Reed, and D. D. Clark, "End-to-end arguments in system design," *ACM Transactions on Computer Systems*, pages 277-288, 1984.

[97] Saswati Sarkar, Tianmin Ren, Leandros Tassiulas, "Achieving Fairness in Multicasting with Almost Stateless Rate Control," Invited Paper Proceedings of SPIE, (Scalability and Traffic Control In IP Networks, II) Vol. 4868, pp. 16-30, ITCOM 2002, Boston,

[98] Saswati Sarkar, Leandros Tassiulas, "Back pressure based multicast scheduling for fair bandwidth allocation," Proceedings of INFOCOM 2001, Alaska,

[99] Saswati Sarkar and Leandros Tassiulas, "Distributed Algorithms for Computation of Fair Rates in Multirate Multicast Trees," *Proc. of IEEE INFOCOM'00*, March 2000.

[100] K. Seada, A. Helmy, "Fairness Analysis of Multicast Congestion Control: A Case Study on pgmcc," *Technical Report 01-743*, University of Southern California, CS Department, April 2001.

[101] M. Sedano, A. Azcorra, M. Caldern, "Performance of Active Multicast Congestion Control," *Lecture Notes in Computer Science*, International Workshop on Active Networks 2000, Tokyo, Japan, October 2000. ISBN 3-540-41179-8

[102] N. Shacham, "Multipoint Communication by Hierarchically Encoded Data," *Proc. of INFOCOM'92*, 1992.

[103] Puneet Sharma, Deborah Estrin, Sally Floyd, Van Jacobson, "Scalable Timers for Soft State Protocols", *Proceedings of IEEE INFOCOM*, Apr. 1997.

[104] S. Shi and M. Waldvogel, "A Rate-based End-to-end Multicast Congestion Control Protocol,"' *Proc. of IEEE Workshop in Enterprise Security (WETICE)*, MIT, USA, June 2001.

[105] Dorgham Sisalem, Adam Wolisz, "MLDA: A TCP-friendly Congestion Control Framework for Heterogeneous Multicast Environments," *IWQoS 2000*, June 2000, Pittsburgh.

[106] T. Speakman, et al. "PGM Reliable Transport Protocol Specification", RFC 3208, December 2001

[107] D.L. Tennenhouse, J.M. Smith, W.D. Sincoskie, D.J. Wetherall, G.J. Minden, "A Survey of Active Network Research," IEEE Communications Magazine, pp. 80-86, January 1997

[108] Hong-Yi Tzeng, Kai-Yeung Siu, "On max-min fair congestion control for multicast ABR service in ATM," IEEE Journal on Selected Areas in Communications, Volume: 15 Issue: 3 , April 1997 Page(s): 545 -556

[109] L. Vicisano, L. Rizzo and J. Crowcroft, "TCP-like congestion control for layered multicast data transfer," *INFOCOM*, Apr '98.

[110] Y. Xia, et. al, "Accumulation-based Congestion Control,", submitted work, 2002. Available at http://www.ecse.rpi.edu/Homepages/shivkuma/ research/papers-rpi.html

[111] Huayan Amy Wang, Mischa Schwartz, "Achieving Bounded Fairness for Multicast and TCP Traffic in the Internet," SigComm 1998

[112] B. Whetten and J. Conlan, "A Rate Based Congestion Control Scheme for Reliable Multicast," *RMRG meeting*, Jul '98.

[113] Brian White et al, "An Integrated Experimental Environment for Distributed Systems and Networks (full report)", *Technical Report of University of Utah, May 2002; Revised version to appear at OSDI 2002, December 2002*

[114] Jorg Widmer, Mark Handley, "Extending Equation-based Congestion Control to Multicast Applications", *SIGCOMM 2001*, Aug. 2001.

[115] Linda Wu, Rosen Sharma. Brian Smith, "Thin Streams: An Architecture for Multicasting Layered Video", *Proc. International Workshop on Network and Operating System Support for Digital Audio and Video (NOSSDAV)*, St. Louis, Missouri, May 1997.

[116] Wei Wu, Yong Ren, Xiuming Shan, "Analysis on Adjustment-Based TCP-Friendly Congestion Control: Fairness and Stability", *Proceedings of the 26th IEEE Conference on Local Computer Networks (LCN'2001)*, Tampa, Florida, Nov. 2001.

[117] W. Wu, Y. Xu, J. Lu, "SRM-TFRC: A TCP-Friendly Multicast Congestion Control Scheme Based on SRM," ICCNMC'01, October, 2001

[118] Brett J. Vickers, Clio Albuquerque, Tatsuya Suda, "Source Adaptive Multi-Layered Multicast Algorithms for Real-Time Video Distribution," *IEEE/ACM Transactions on Networking (TON)*, December 2000, Volume 8 Issue 6

[119] K. Yano and S. McCanne, "A window-based congestion control for reliable multicast based on TCP dynamics," *ACM Multimedia*, 2000, pp. 249–258.

[120] http://www.cs.rpi.edu/~lij6/Research/ormcc/ormcc.html

[121] http://www.multicasttech.com/status/

APPENDIX A
Pseduo Code of LE-SBCC

event: Send packet:

Record $T_{send}[j]$ for seq j

event: LI[i][j] received for from receiver i for seq j:

```
call estimateRTT()
call LI2LEfilter()
if passed
   call maxLPRFilter()
   if passed
      call ATFilter()
      if passed
         rate = rate/2
      endif
   endif
endif
```

estimateRTT() {

$RTT_{current} = T_{current} - T_{send}[j]$

$\delta = SRTT - RTT_{current}$

$SRTT = RTT_{current} + 0.125 * \delta$

$\sigma = \sigma + (0.125 * (|\delta| - \sigma)))$

}

LI2LEfilter() {

if $((t - T_{lastPass}[i]) > SRTT + 2\sigma)$

$T_{lastPass}[i] = t$

pass LI as LE

else

filter LI

endif

}

maxLPRFilter() {

update X_i, $\max X_i$, ΣX_i

$P(accept) = \frac{\max X_i}{\Sigma X_i}$

Accept LE w/ probability $P(accept)$

}

ATFilter() {

if $(CongestionFlag == TRUE)$

filter LE

else {

$SilenceFlag = TRUE$

$SilencePeriodTimer = RTT/2 + 2\sigma$

$CongestionFlag = TRUE$

$CongestionEpochTimer = SilencePeriod + SRTT + 4\sigma$

Accept LE

}

}

event: $CongestionEpochTimer$ expires:

$CongestionFlag = FALSE$

event: $SilencePeriodTimer$ expires:

$SilenceFlag = FALSE$

event: $RateIncreaseTimer$ expires:

if $(CongestionFlag == FALSE)$

$rate+ = \frac{MSS}{SRTT+2\sigma}$

else {

if $(SilenceFlag == TRUE)$

no data transfer

else

no increase in rate

}

$RateIncreaseTimer = RTT + 2\sigma$

APPENDIX B
ORMCC Algorithm

B.1 Source Operations

Variables :

λ : Transmission rate

μ : Throughput rate at congestion (TRAC) in the received CI(μ)

$E(\mu^{cr})$: Average TRAC of the CR

μ_{σ}^{cr} : Deviation of the CR TRAC

s : Packet size

RTT_{max} : Maximum RTT

RTT_{cr} : RTT between the source and the CR

T^{cr} : CR response time when the bottleneck is fully loaded

$E(T^{cr})$: Average of T^{cr}

T_{σ}^{cr} : Deviation of T^{cr}

cr_valid : Indicates whether the CR is valid

R : The receiver sending the received CI(μ)

Initialization:

cr_valid = **false**

$RTT_{max} = 0$

Event every RTT_{cr}**:**

if *There is no rate reduction within the recent* RTT_{cr} **then**

$\lambda \leftarrow \lambda + s/RTT^{cr}$

if $\lambda \geq E(\mu^{cr}) + 4\mu_{\sigma}^{cr}$ **and** *CR checking timer is not running* **then**

Start CR checking timer with time length $E(T^{cr}) + 8T_{\sigma}^{cr}$

$t \leftarrow$ the current time

endif

endif

Event when the CR checking timer expires:

cr_valid $\leftarrow$ **false**

Send packet:

if *cr_valid is* **true then**

Send a packet with real $E(\mu^{cr})$ and μ_σ^{cr}

else

Send a packet with invalid values for $E(\mu^{cr})$ and μ_σ^{cr}

endif

Subroutine : CutRate ()

if λ *has not been cut within the most recent* RTT_{cr} **then**

$\lambda \leftarrow \min(\lambda, 0.75\mu)$

Stop CR checking timer

endif

Event upon receipt of CI(μ):

if R *is* CR **then**

cr_valid $\leftarrow$ **true**

if *CR checking time is running* **then**

$\Delta \leftarrow$ current time $- t$

Update $E(T^{cr})$ and T_σ^{cr} with Δ

endif

Update $E(\mu^{cr})$ and μ_σ^{cr} with μ

Update RTT_{cr}

if $RTT_{max} < RTT_{cr}$ **then**

$RTT_{max} \leftarrow RTT_{cr}$

endif

Stop CR checking timer

do CutRate ()

return

endif

/* The CI(μ) is NOT from CR if reach here */

if *cr_valid is* **false then**

Choose R as the CR

Start CR grace period as $2RTT_{max}$

else if *In CR grace period* **then**

if *The RTT sample measured by this CI(μ) is larger than* RTT_{cr} **then**

Choose R as the CR

endif

/* NOT in CR grace period */

else if $\mu < E(\mu^{cr}) - \mu_\sigma^{cr}$ **then**

Choose R as the CR

endif

if *CR has been changed at the receipt of this CI(μ)* **then**

cr_valid $\leftarrow$ **true**

Update $E(\mu^{cr})$ and μ_σ^{cr} with μ

Update RTT_{cr}

if $RTT_{max} < RTT_{cr}$ **then**

$RTT_{max} = RTT_{cr}$

endif

do CutRate ()

endif

B.2 Receiver Operations

Variables :

μ : A throughput rate at congestion (TRAC) sample

$E(\mu)$: Average TRAC of this receiver

$E(\mu^{cr})$: Average TRAC of the CR

μ_σ^{cr} : Deviation of the CR TRAC

Event upon receipt of a packet:

if $E(\mu^{cr})$ *and* μ_σ^{cr} *has been changed* **then**

Update the local copy of $E(\mu^{cr})$ and μ_σ^{cr}

endif

if *This packet indicates packet losses* **then**

Meaure μ and update $E(\mu)$

if $E(\mu^{cr})$ *and* μ_σ^{cr} *are invalid* **or** $E(\mu) < E(\mu^{cr}) - \mu_\sigma^{cr}$ **then**

Send CI(μ)

endif

endif

APPENDIX C
Theoretical Analysis of ORMCC Properties

C.1 Capability of Tracking The Slowest Receiver

In this part, we are going to show that an ORMCC flow always track the slowest receiver, i.e. the receiver behind the most congested path. For convenience, we are going to refer the path between the source and the CR as *Representative Path*.

Let's consider a multicast session using ORMCC. Suppose there are N ($N > 1$) different paths on the multicast tree. Let R_i be the receiver behind path i. Without loss of generality, assume R_1 is the current CR. The source will choose another receiver R_j ($j \neq 1$) as the new CR only if R_j sees a lower throughput rate at congestion (TRAC) than that seen by R_1. To see when a R_j will see a lower TRAC on average, first we are going to calculate the TRACs on all paths from 1 to N, given the instantaneous ORMCC sending rate at which a burst of packet losses begins.

For the analysis, we have the following definitions (all i are from 1 to N):

$\lambda_{i,t}$: Instantaneous sending rate of the ORMCC flow at time t on path i.

$\mu_{i,t}$: Instantaneous throughput rate of the ORMCC flow corresponding to $\lambda_{i,t}$.

$\lambda^o_{i,t}$: The sum of instantaneous sending rates of all other flows sharing the bottleneck on path i at time t.

W_i : Bandwidth of the bottleneck on path i.

Q : Buffer size of a bottleneck. Here we assume that all bottlenecks have the same buffer size. If a queue is constantly non-zero, we will treat the part which is emptied and (partly) filled as the whole queue.

s : Packet size. We assume that all the packet sizes are equivalent.

RTT_i : RTT of path i.

Δ : The sending rate increment of the ORMCC flow per unit time. $\Delta = s/RTT_1 > 0$.

Δ_i : The sum of the sending rate increments per unit time of all but the ORMCC flows sharing the bottleneck on path i, without packet losses occuring at the bottleneck, assuming all do AIMD. In reality, most likely Δ_i changes randomly, therefore we consider its average value in an aggregate sense. $\Delta_i > 0$.

$\gamma_i : \gamma_i = \Delta_i/\Delta$.

Moreover, we assume that data are sent bit-by-bit evenly, sending rates are increased continuously, as well as that all packet losses are due to congestion. Also, drop-tail buffer management is assumed for bottlenecks.[30]

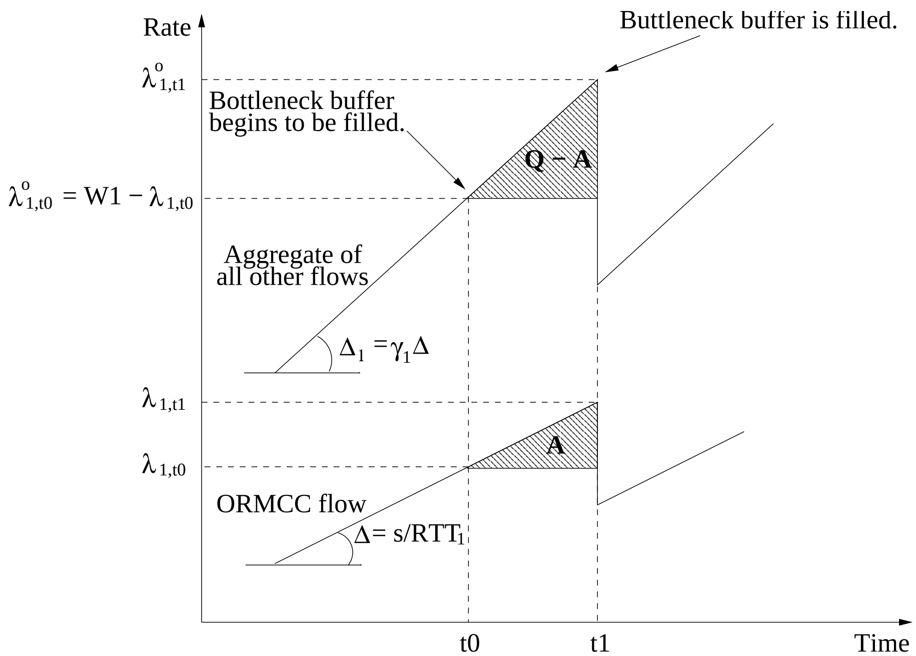


Figure C.1: Evolution of ORMCC Sending Rate on The Representative Path

Let's consider path 1 first (Figure C.1). Suppose at time t_1, there is a burst of packet losses. The bottleneck queue must be full at this moment. The sum of sending rates of all the flows going through the bottleneck, $\lambda_{1,t_1} + \lambda^o_{1,t_1}$, must be larger than the bottleneck bandwidth W_1. Recall that $\Delta > 0$ and $\Delta_1 > 0$, meaning that without packet losses at the bottleneck, the sum of sending rates keep increasing. Consequently, at an earlier moment t_0, the sending rate sum must be equal to W_1, i.e.,

$$\lambda_{1,t_0} + \lambda^o_{1,t_0} = W_1 \tag{C.1}$$

Since the sending rate of the ORMCC flow grows by Δ per unit time,

$$\lambda_{1,t_1} = \lambda_{1,t_0} + (t_1 - t_0)\Delta \Rightarrow \lambda_{1,t_0} = \lambda_{1,t_1} - (t_1 - t_0)\Delta \tag{C.2}$$

[30]Although our analysis is based on drop-tail routers, ORMCC also works well with RED routers. It has been confirmed by simulations, though for space reason, the results are not included.

$\lambda^o_{1,t}$ grows by $\Delta_1 = \gamma_1\Delta$ per unit time, therefore,

$$\lambda^o_{1,t_1} = \lambda^o_{1,t_0} + (t_1 - t_0)\Delta_1 = \lambda^o_{1,t_0} + (t_1 - t_0)\gamma_1\Delta \tag{C.3}$$

From (C.1), (C.2) and (C.3), we have,

$$\begin{aligned}\lambda^o_{1,t_1} &= W_1 - (\lambda_{1,t_1} - (t_1 - t_0)\Delta) + (t_1 - t_0)\gamma_1\Delta \\ &= W_1 - \lambda_{1,t_1} + (t_1 - t_0)(1 + \gamma_1)\Delta \end{aligned} \tag{C.4}$$

We also assume that at t_0, the bottleneck queue size is zero. Since at t_1, the queue is full, the queue is filled by sending rate increments during $[t_0, t_1]$. Recalling that the total sending rate grows by $\Delta + \Delta_1 = (1 + \gamma_1)\Delta$ per unit time, and the assumption of all flows' doing AIMD, we have,

$$\frac{1}{2}(t_1 - t_0)^2(1 + \gamma_1)\Delta = Q \Rightarrow t_1 - t_0 = \sqrt{\frac{2Q}{(1 + \gamma_1)\Delta}}$$

Together with (C.4),

$$\lambda^o_{1,t_1} = W_1 - \lambda_{1,t_1} + \sqrt{2\Delta Q(1 + \gamma_1)} \tag{C.5}$$

Assuming all flows going through the bottlenck have the same priority, since at time t_1 the bottleneck is working at its full load, we know

$$\begin{aligned}\mu_{1,t_1} &= \frac{\lambda_{1,t_1}}{\lambda_{1,t_1} + \lambda^o_{1,t_1}} W_1 \\ &\overset{(C.5)}{=} \frac{\lambda_{1,t_1}}{1 + \frac{1}{W_1}\sqrt{2\Delta Q(1 + \gamma_1)}}\end{aligned} \tag{C.6}$$

On any other path j ($j = 2 \dots N$), since the ORMCC source ignores the congestion indications on this path (Figure C.2), the sending rate of the ORMCC flow still grows by Δ per unit time. With t_1 of the same meaning as before, according to a derivation similar to that above, we have,

$$\mu_{j,t_1} = \frac{\lambda_{j,t_1}}{1 + \frac{1}{W_j}\sqrt{2\Delta Q(1 + \gamma_j)}} \tag{C.7}$$

Consider λ_{i,t_1} ($i = 1 \dots N$). Assume that the sending rate of the ORMCC flow varies between λ_{min} and λ_{max} (Figure C.2), then λ_{i,t_1} is a sample value of a random variable Λ_i

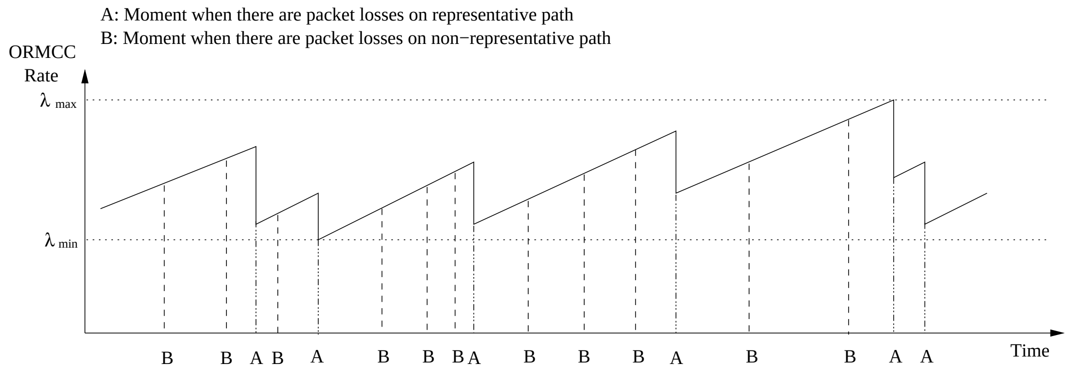


Figure C.2: ORMCC Source Only Considers The Congestions on The Representative Path for Rate Adaptation

with sample space as $[\lambda_{min}, \lambda_{max}]$. Assuming Λ_i's are identically distributed, their expected values are the same, i.e. $E(\Lambda_i) = E(\Lambda_j)$ $(i \neq j)$.

Let receiver i be the receiver behind path i. μ_{i,t_1} $(i = 1 \dots N)$ is the TRAC measured at receiver i. According to (C.7), μ_{i,t_1} is a function of λ_{i,t_1}, and thus is a random sample. Denote the corresponding random variable as U_i. Assuming that W_i, Q, s are constant, and that γ_i and RTT_1 in steady state have small deviations and thus can be treated as constant, we have,

$$\begin{aligned} U_i &= \frac{\Lambda_i}{1 + \frac{1}{W_i}\sqrt{2\Delta Q(1+\gamma_i)}} \\ E(U_i) &= \frac{E(\Lambda_i)}{1 + \frac{1}{W_i}\sqrt{2\Delta Q(1+\gamma_i)}} \end{aligned}$$

As designed in ORMCC, for $j = 2 \dots N$, only upon detection of $E(U_j) < E(U_1)$ (the average TRAC of the current CR receiver 1) will receiver j send congestion indication (CI) packets back to the source, which then update the congestion representative (CR) to receiver j. From the expression of $E(U_i)$ above, we have,

$$\begin{aligned} & E(U_j) < E(U_1) \\ \Leftrightarrow \quad & \frac{E(\Lambda_j)}{1 + \frac{1}{W_j}\sqrt{2\Delta Q(1+\gamma_j)}} < \frac{E(\Lambda_1)}{1 + \frac{1}{W_1}\sqrt{2\Delta Q(1+\gamma_1)}} \\ \Leftrightarrow \quad & \frac{W_j}{\sqrt{1+\gamma_j}} < \frac{W_1}{\sqrt{1+\gamma_1}} \qquad \text{since } E(\Lambda_j) = E(\Lambda_1) \end{aligned} \tag{C.8}$$

We can see that $W_i/\sqrt{1+\gamma_i}$ $(i = 1 \dots N)$ indicates the degree of congestion on the bottleneck of path i. In fact, if the bottleneck has less bandwidth, i.e. W_i is smaller, $W_i/\sqrt{1+\gamma_i}$ has a lower value; if more flows are sharing a bottleneck, the sum of their per-unit-time rate increments Δ_i is higher, $\gamma_i = \Delta_i/\Delta$ is then larger, which in turn also makes $W_i/\sqrt{1+\gamma_i}$ lower. Therefore, (C.8) actually shows that as long as a non-representative path (path j) experiences a more serious congestion than the representative path (path 1) does, the receiver behind path j will see lower average TRAC $E(U_j)$, and will send CI(μ)s back to the source, making the source change CR. Namely, *an ORMCC flow always tracks the slowest receiver.*

C.2 TCP-Friendliness on Representative Path

By representative path, we mean the path which the congestion representative (CR) is behind. In the following, we are going to show that an ORMCC flow is friendly to a TCP flow on the representative path, by showing that they get approximately equal share of the bottleneck bandwidth, with the assumption that their RTT estimations and packet sizes are the same. More strictly speaking, we want to show that, with proper choice of rate reduction factor β for ORMCC, V_t^{MCC}/V_t^{TCP} oscillates around 1, where V_t^{MCC} and V_t^{TCP} denote the sending rates of the TCP flow and ORMCC flow at time t respectively. Those two flows are assumed to be the only flows on the representative path. A sample of the rate evolution is given in Figure C.3.

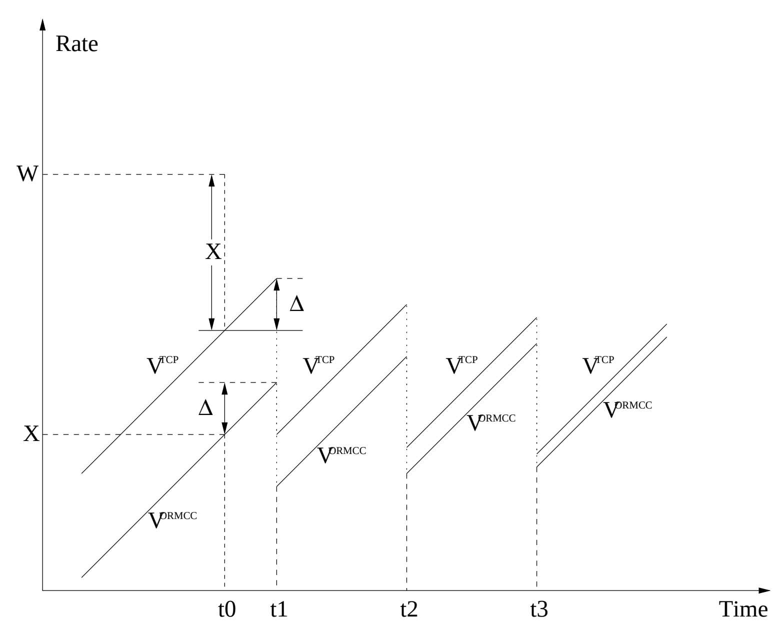


Figure C.3: Evolution of The Sending Rates of TCP and ORMCC Flows

Like other TCP throughput analysis papers [85] [76] have done, our analysis focuses only on TCP's congestion avoidance behavior. During congestion avoidance period, when without packet losses, a TCP source increases its congestion window by $1/N$ packet upon the receipt of per ACK, where N is the current congestion window size. A TCP source transmit all the packets in its congestion window in one RTT, therefore, the window grows by 1 packet per RTT, [31] which corresponds to the fact that its sending rate is increased by s/RTT per RTT, where s is the packet size. An ORMCC source increases its sending rate at the same pace, as covered in scheme description. At packet loss, a TCP source will reduce its congestion window by half, which is equivalent to cutting its sending rate by half.

Assume that congestion is the only reason for packet losses. Let W be the bottleneck bandwidth. It is obvious that packet losses can occur only if $V_t^{TCP} + V_t^{MCC} \geq W$. Suppose some packets are lost and both flows reduce their transmission rates at t_1 (Figure 2.13). Before the losses, since both V_t^{TCP} and V_t^{MCC} keep increasing, there must be a moment t_0 when $V_{t_0}^{TCP} + V_{t_0}^{MCC} = W$. For short, let $V_{t_0}^{MCC} = X$, then $V_{t_0}^{TCP} = W - X$. For the first step of analysis, we will show that with appropriate β,

$$\begin{cases} X < W - X \Rightarrow X/(W-X) < V_{t_1}^{MCC}/V_{t_1}^{TCP} \\ \\ X > W - X \Rightarrow X/(W-X) > V_{t_1}^{MCC}/V_{t_1}^{TCP} \end{cases} \tag{C.9}$$

Let the moment just before the rate reduction at t_1 be t_1'. Because the TCP and ORMCC flows share the same path, we assume that they detect packet losses and reduce transmission rates approximately at the same time. For the TCP flow, suppose that at t_1', its transmission rate has been increased by d since t_0, i.e.

$$V_{t_1'}^{TCP} = W - X + d$$

After a reduction by half,

$$V_{t_1}^{TCP} = \frac{V_{t_1'}^{TCP}}{2} = \frac{W - X + d}{2}$$

[31] We assume that a TCP receiver sends an ACK per received packet.

Since the ORMCC flow increases its rate at the same pace, we have,

$$V_{t'_1}^{MCC} = X + d$$

Assume that both flows have the same priority and are almost synchronous, i.e. their packets are forwarded by the bottleneck with the same probability. In consequence, at t'_1, the ORMCC CR sees an approximate receiving rate of

$$\frac{V_{t'_1}^{MCC}}{V_{t'_1}^{MCC} + V_{t'_1}^{TCP}} W = \frac{X+d}{W+2d} W$$

According to the rate adaptation policy of ORMCC,

$$V_{t_1}^{MCC} = \beta \frac{X+d}{W+2d} W$$

Therefore,

$$\frac{V_{t_1}^{MCC}}{V_{t_1}^{TCP}} = \beta \frac{X+d}{W+2d} W \Big/ \frac{W-X+d}{2}$$

Now let's compare $X/(W-X)$ and $V_{t_1}^{MCC}/V_{t_1}^{TCP}$.

$$\begin{aligned} & \frac{X}{W-X} - \frac{V_{t_1}^{MCC}}{V_{t_1}^{TCP}} \\ = & \frac{2}{W-X+d} \\ & \cdot \left[\left(\frac{1}{2} - \frac{\beta W}{W+2d} \right) X + \left(\frac{X}{2(W-X)} - \frac{\beta W}{W+2d} \right) d \right] \end{aligned} \tag{C.10}$$

Since $W > X$ and $d \geq 0$, $2/(W-X+d) > 0$, and the positivity of (C.10) is decided by its second factor between square brackets. If we choose a value for β carefully so that,

$$\beta = \frac{1}{2} \frac{W+2d}{W}$$

The second factor of (C.10) becomes

$$0 \cdot X + \frac{d}{2} \left(\frac{X}{W-X} - 1 \right) = \frac{d}{2} \left(\frac{X}{W-X} - 1 \right) \tag{C.11}$$

It is easily seen that, if $X > W-X$, (C.11) > 0 so that (C.10) > 0; while if $X < W-X$, (C.11) < 0 so that (C.10) < 0. That is exactly what we want for (C.9).

With (C.9) eastablished, we can go further. Assume that at t_i, $i = 1 \ldots \infty$, both the TCP flow and the ORMCC flow reduce their rates (Figure 2.13), and their rates after reduction are $V_{t_i}^{TCP}$ and $V_{t_i}^{MCC}$ respectively. Also assume that $t_{i,0}$ is the closest moment before t_i so that $V_{t_{i,0}}^{TCP} + V_{t_{i,0}}^{MCC} = W$. Recall the meanings of X and $W - X$, (C.9) actually indicates that,

$$\begin{cases} V_{t_{i,0}}^{MCC} < V_{t_{i,0}}^{TCP} \Rightarrow V_{t_{i,0}}^{MCC}/V_{t_{i,0}}^{TCP} < V_{t_i}^{MCC}/V_{t_i}^{TCP} \\ \\ V_{t_{i,0}}^{MCC} > V_{t_{i,0}}^{TCP} \Rightarrow V_{t_{i,0}}^{MCC}/V_{t_{i,0}}^{TCP} > V_{t_i}^{MCC}/V_{t_i}^{TCP} \end{cases} \tag{C.12}$$

Let's consider the situation that $V_{t_{i,0}}^{MCC} < V_{t_{i,0}}^{TCP}$ first, which is shown in Figure 2.13. Note that for any i,

$$V_{t_{i+1,0}}^{MCC} = V_{t_i}^{MCC} + A, \;\; V_{t_{i+1,0}}^{TCP} = V_{t_i}^{TCP} + A \quad A > 0$$

So,

$$\frac{V_{t_i}^{MCC}}{V_{t_i}^{TCP}} < \frac{V_{t_i}^{MCC} + A}{V_{t_i}^{TCP} + A} = \frac{V_{t_{i+1,0}}^{MCC}}{V_{t_{i+1,0}}^{TCP}} \overset{(C.12)}{<} \frac{V_{t_{i+1}}^{MCC}}{V_{t_{i+1}}^{TCP}}$$

Similarly, if $V_{t_{i,0}}^{MCC} > V_{t_{i,0}}^{TCP}$,

$$\frac{V_{t_i}^{MCC}}{V_{t_i}^{TCP}} > \frac{V_{t_{i+1}}^{MCC}}{V_{t_{i+1}}^{TCP}}$$

As the result, if the ORMCC flow rate is less than that of the TCP flow, it will grow until it exceeds the latter; likewise, if the ORMCC flow rate is more, it will get less and less until it is below the TCP flow rate. Hence, V_t^{MCC}/V_t^{TCP} oscillates around 1. In conclusion, we say that ORMCC is approximately TCP friendly, given that the rate reduction factor β is properly chosen.

With regard to the value of rate reduction factor β, recall that above in the analysis, we need to have that

$$\beta = \frac{1}{2}\frac{W + 2d}{W}$$

Since $d \geq 0$, β needs to have a value greater than 0.5. Consider the fact that TCP uses ACKs to measure RTTs. It can have lower RTT estimation than that of ORMCC which uses NAKs for this purpose, as we discussed in Section 3.1.6. Thus, TCP can increase sending rate faster than ORMCC. For compensation, ORMCC at packet losses can reduce its transmission rate by less using larger value of β. In our implementation, we use a value of 0.75 and it works fine in simulations.

C.3 Immunity To Drop-to-zero Problem

The cause of drop-to-zero problem is the asynchronous packet losses on multiple paths. If a multicast source reduces the transmission rate too much on the losses, the rate will stay very low or even converge to zero. However, in nature, the source in ORMCC adapts the transmission rate according to the congestion on one single path while ignoring that on all others, there will be no drop-to-zero problem for ORMCC. In more details, if a receiver other than the current congestion representative (CR) sees a packet loss rate lower or equal to that by CR, it won't send CI(μ)s back to the source. The source won't see any CI(μ)s from it, thus of course won't reduce the transmission rate. Even if the source gets CI(μ)s from different receivers because there is a change of the most congested bottleneck, once it chooses a receiver as the new CR after a very short period of time (several RTTs), it will ignore CI(μ)s from all other receivers. Consequently, we can say that ORMCC is immune to drop-to-zero.

C.4 Effectiveness of Feedback Suppression

Without support from internal nodes, which is the situation that we assume for reality, most multicast feedback suppression schemes (e.g. [34], [103], [114], [37], [84]) use random timers for delaying receivers' feedback before sending them. However small it is, there is some feedback latency which may bring performance penalty. Since our feedback suppression is not based on timers, it does not suffer from this problem. Also, there is no need to know or estimate the total number of receivers like [37]. Moreover, we are going to show below that in ORMCC, the total number of feedbacks (i.e. CI(μ)s) sent to the source by all receivers in a multicast session, is independent of the total number of receivers. Instead, it depends on the switching frequency of the most congested bottleneck, as well as the number of receivers behind the new most congested bottleneck, plus the minimum RTT between them and the source. For convenience, we use the acronym *MCB* for *most congested bottleneck* in the following discussion.

We assume that there is only one MCB at any moment[32]. To begin the calculation,

[32]There can certainly be multiple bottlenecks which have similar degree of congestion and are all most congested. However, the discussion still holds.

the following notations are needed[33].

N : Total number of receivers behind the new most congested bottleneck (MCB).

R_i : Receiver i behind the new MCB. ($i = 1 \dots N$)

RTT_i : RTT between the source and R_i.

RTT_i^f : Forwarding (downstream) part of RTT_i.

RTT_{min} : The minimum of all RTT_i's.

p : Packet loss rate seen by receivers behind the new MCB.

v : Average transmission rate of the ORMCC flow.

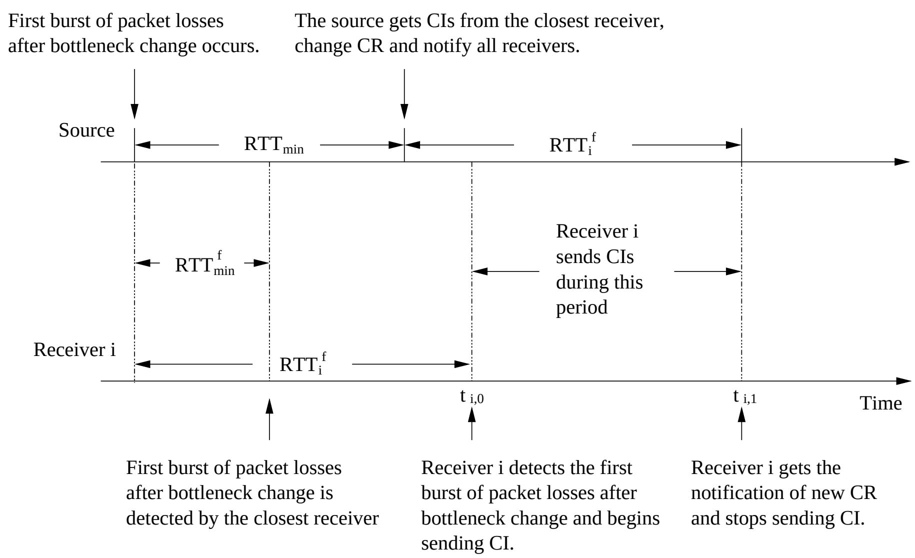


Figure C.4: Feedback Suppression Mechanism

Whenever there is a new MCB, according to the previous discussion of ORMCC's following the most congested path, only those receivers behind the new MCB will send CI(μ)s back to the source, and all of them except one will stop sending CI(μ)s once one of them is chosen as the new CR. More specifically, the source will first see the CI(μ)s from the receiver with RTT_{min}, then change CR and tell all receivers of the change. For any R_i except the new CR, the duration of sending CI(μ)s is between the moment $t_{i,0}$ when they first detect packet loss after bottleneck change and the moment t_{i_1} when they know the new CR. According to Figure C.4, $t_{i_1} - t_{i_0} = RTT_i^f + RTT_{min} - RTT_i^f = RTT_{min}$. Therefore, before a new CR is decided, the number of CI(μ)s sent from this receiver i is thus $pvRTT_{min}$,

[33] Since the receivers involved here are all behind MCB, we can assume that they see the same degree of congestion and thus the same packet loss rates.

and the total number of CI(μ)s sent by all receivers behind the new MCB is,

$$\sum_{i=1}^{N} pvRTT_{min} = vpN \cdot RTT_{min}$$

Once a new CR is decided, only one receiver, namely the new CR, will send CI(μ)s. Let's call the period between two successive MCB switchings *MCBSP (MCB switching period)*. During a MCBSP of length t, the total number of CI(μ)s sent to the source is, assuming $t \geq RTT_{min}$,

$$vpN \cdot RTT_{min} + vp(t - RTT_{min}) = vp(t + (N-1)RTT_{min})$$

If a Poisson process with parameter λ[34] is assumed for MCB switching, for a multicast session of duration T, on average, the total number of CI(μ)s transmitted is approximately,

$$\lambda T \cdot vp \left(\frac{1}{\lambda} + (N-1)RTT_{min}\right) \tag{C.13}$$

We can see that, for a ceratin T,

$$\begin{aligned} & \lim_{\lambda \to 0} \lambda T \cdot vp \left(\frac{1}{\lambda} + (N-1)RTT_{min}\right) \\ = \quad & vpT + \lim_{\lambda \to 0} \lambda vpT(N-1)RTT_{min} = vpT \end{aligned} \tag{C.14}$$

That means, if during a multicast session, there is no MCB switching, the total number of CI(μ)s transmitted is approximately equal to the number of CI(μ)s sent from a single receiver behind the MCB. To make it clearer: if the MCB does not change during a multicast session, the volume of feedback is on the same level of unicast feedback!

Also, from (C.13), we find that the total number of transmitted CI(μ)s is independent of the total number of receivers in a multicast session (Note that N is (C.13) is not the total number of receivers but the number of receivers behind the MCB). It depends on how fast MCB switches and the amount of receivers behind the new MCB, as well as the smallest RTT between those receivers and the source. Usually, MCB switches only once in many multiple RTT_{min}'s, and the amount of receivers behind the new MCB is much less than the overall number. Moreover, RTT_{min} is almost a negligible duration. Consequently, our

[34]Considering the reality, we can assume that MCB switching does not occur too frequently and $1/\lambda \geq RTT_{min}$.

feedback suppression mechanism is effective.

Finally, we must say that due to measurement errors in reality, the total number of CI(μ)s sent can be a little higher than what we have derived here. However, the difference won't be significant.

APPENDIX D
MCA

In this appendix, we present the work leading to MCA+ in Chapter 5. Since the theory background is the same as MCA+, we omit it here.

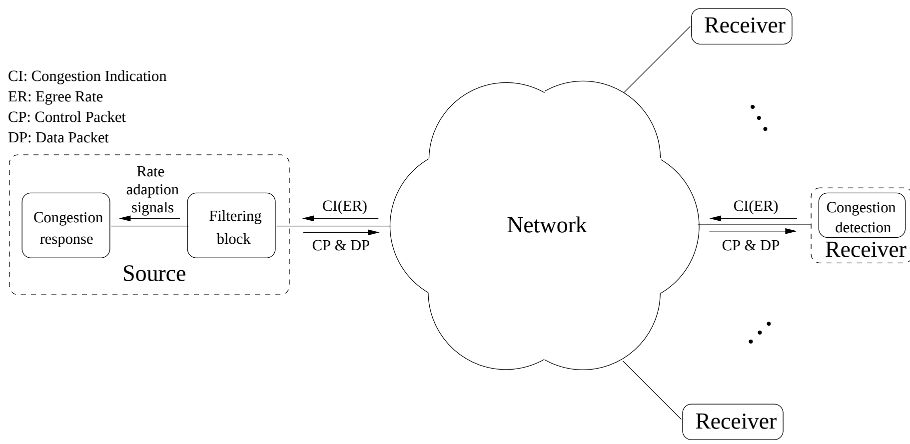


Figure D.1: Multicast Congestion Avoidance Model

MCA consists of three key building blocks (Figure D.1): a *congestion detection* block at receivers, a *filtering block* at the source to discriminate between competing feedback from receivers, and a *congestion response* block which implements a rate-increase/decrease policy. Congestion detection is based upon the “accumulation” measure. Congestion detection triggers feedback, which is sparse in the sense that at most one feedback is generated per measurement period (unlike multiple loss indications generated during packet loss in “congestion control” schemes). Congestion feedback to senders can be in the form of single-bit congestion indication (CIs) or as a multi-bit output rate measure. The two different feedback models (bit-based or explicit rate-based) leads to two different schemes: bit-based and explicit rate-based. These schemes essentially have different designs of the filtering and congestion response policy blocks. The explicit rate feedback can be leveraged to reduce the state requirements at the sender to $O(1)$.

Simulation results show that both schemes avoid the drop-to-zero problem [114, 94, 9]. Drop-to-Zero is the problem of reacting to *more feedback* indications than necessary leading to a beat-down of the multicast flow's rate[114, 94, 9]. This occurs because the multicast flow receives feedback indications from multiple paths and may not filter them sufficiently. TCP-unfriendliness is the problem of reacting to *less feedback* than a hypothetical TCP flow would on the worst loss path [14, 94, 114]. Though the congestion detection model is incompatible with that of TCP (and we cannot directly demonstrate fairness with TCP) we demonstrate fairness with similar *unicast* congestion avoidance flows.

D.1 Accumulation Measurement and CI Generating Algorithm

Suppose we begin at time t_0. Let T be the value of control packet interval. The behavior of the source is simply to send out a control packet (CP) to the receivers at $t_0 + iT$ ($i = 0, 1, 2...$). The receivers execute the following algorithm whenever a CP arrives, with the variable *accu* recording the accumulation:

t	:	current time
i	:	the sequence number of CP
T	:	control packet interval
t_s	:	time of the most recent synchronization point (SP)
seq_s	:	CP sequence number of the most recent SP
$accu$	:	accumulation in bytes
$accu_g$	:	global accumulation in bytes
H_thresh	:	high threshold of accumulation.
L_thresh	:	low threshold of accumulation.

(*Synchronization point (SP)* is the point at which we assume no packet backlog on the path from the source to the receiver.)

1. If the CP is the very first one since t_0,

   Set: $accu_g = 0$, $accu = 0$, $t_s = t$, $seq_s = i$. (SP)

   Return.

Endif

2. If $t < t_s + (i - seq_s)T$,

   Set: $accu_g = 0$, $accu = 0$, $t_s = t$, $seq_s = i$. (SP)

   Return.

Endif

3. Set $accu =$ the bytes received within $(t_s + (i - seq_s)T, t] + accu_g$.
4. If $accu \geq H\_thresh$,

Send a CI back to the source.

Else if $accu > L\_thresh$,

Do nothing.

Else if $accu$ has ever exceeded H_thresh since t_s,

Set: $accu_g = accu$, $accu = 0$, $t_s = t$, $seq_s = i$. (SP)

Else

Do nothing.

Endif

Return.

The algorithm above assumes zero packet loss. To make it robust, a receiver also send CIs upon detection of packet losses. Although packet losses may decrease the accumulation seen by the receiver and hide congestion, the congestion detection by packet losses compensate for it. Besides, the error of $accu$ measured at the arrival of i^{th} CP won't be carried over to next measurement if the receiver does not see any loss during $(t_s + (i + 1 - seq_s)T, t]$. In addition, at the arrival of i^{th} CP, if the receiver has seen any losses during $(t_s+(i-seq_s)T, t]$, it won't do re-synchronization.

However, if the route between the source and the receiver changes to be longer (although it does not happen too often), our scheme won't be able to re-synchronize. That is an issue for our future research.

D.2 Bin-CI Scheme

We now describe filtering and congestion response policies at the source. If the feedback upon congestion detection is a single bit, i.e., a binary congestion indication (CI), the scheme is called *Bin-CI*. We shall see that the tradeoff between explicit rate feedback and single bit feedback is simplicity of feedback vs complexity of state at the source. For this *Bin-CI* scheme, we largely leverage LE-SBCC (Chapter 2) for a similar case of binary feedback carrying packet loss indications (LIs) instead of congestion indications (CIs). There are a cascade of three filters (CI2CE, MaxLPRF, ATF) into which CIs are fed, and two modules of RTT estimation and rate adaption, as follow.

<u>CI2CE FILTER</u> Whenever a CI from receiver i arrives, the source checks the current time t_2. Let t_1 be the time when last CI from i was accepted. If $t_2 - t_1 \leq RTT + 2 \times \sigma$ [35], the

[35] σ is the mean deviation of RTT samples.

CI is rejected. Otherwise, it is accepted as a new congestion event (CE) from receiver i and passed to the next filter MaxLPRF.

MaxLPRF FILTER Let the total number of CEs from receiver i be X_i. Any CE is passed with probability of $(\max X_i)/\sum_i X_i$, i.e., the MaxLPRF passes on the average, $\max X_i$ CEs out of a total of $\sum_i X_i$ CEs.

In addition to the above probabilistic behavior, MaxLPRF maintains two accounting variables: P_d and V_d. P_d is set zero at initialization and *each time the rate is reduced in the rate adaptation module (see below)*. Whenever a CE arrives, P_d is incremented by $(\max X_i)/\sum_i X_i$. If P_d was below 1 prior to incrementing and is at least 1 after incrementing, we set V_d as the current data transfer rate. These accounting variables are used in the rate-adaptation module (see below).

ATF FILTER When a CE arrives at ATF, the current time t_2 is checked against the time t_1 when last CE (from any receiver) was passed by ATF. The new CE is passed if $t_2 - t_1 \leq RTT + 4 \times \sigma$. This guarantees that at most one rate deduction is performed in any one RTT.

RTT ESTIMATION At the arrival of a CI, (1) if it is triggered by a CP, the RTT sample RTT_s is the difference between the current time t and the departure time of the CP triggering the CI, (2) if it is triggered by a packet loss and not a retransmitted one, RTT_s is the difference between t and the transmit time of the lost packet. With RTT_s, the RTT is updated as $RTT = 7/8 \cdot RTT + 1/8 \cdot RTT_s$.

RATE ADAPTATION During the periods of no congestion (i.e. no CE), the data transfer rate V_s is incremented by $S/SRTT$ every $SRTT$ [36] (where S is the data packet size). An exponentially weighted moving average (EWMA) of the rate-increments, V_e is maintained as $V_e = \alpha V_e + (1 - \alpha) S/SRTT$ [37].

If a CE passes the filter cascade, the rate V_s is adjusted in the following way:

1. If $P_d < 1$, $V_s = \beta(V_s - V_e)$.
2. If $P_d \geq 1$, $V_s = \beta(V_d - V_e) - (V_s - V_d)$.

In our simulations, β is 0.9. When $P_d < 1$, V_s is deducted by the amount of V_e first, and then multiplicatively reduced by β. This ensures that, with a high probability, drain capacity is provisioned for the accumulation incurred. When $P_d \geq 1$, it means that the source

[36] $SRTT = RTT + 2 \times \sigma$
[37] α is 7/8 in our simulations.

responds late (since the ATF may have filtered a CE passed by MaxLPRF). Therefore, the source goes back to the rate V_d where it should have been, responds as described earlier, and cancels excessive increment due to late response.

D.3 ER-CI Scheme

The *ER-CI* scheme leverages the multi-bit egress rate (ER) information in the feedback message (also called as a CI for convenience) to reduce the state requirements at the source to O(1). The filtering block in the *ER-CI* scheme calculates an EWMA of the ER fed back by receivers. The EWMA estimate $V_{fe} = \gamma V_{fe} + (1-\gamma)V_f$ where V_f is the egress receiving rate in the feedback message. An ER error estimate σ_v is also calculated: $\sigma_v = \gamma\sigma_v + (1-\gamma)|V_{fe} - V_f|$ [38]. Let R be the the receiver whose CI was accepted most recently. An arriving CI is accepted if any of the following conditions is met:

(1) The CI is the very first one received by the source,

(2) The CI is from receiver R,

(3) V_f satisfies $V_f < V_{fe} - 3\sigma_v$,

(4) $t_2 - t_1 > RTT + 7\sigma$, $V_s \geq V_{fe} + 3\sigma_v + S/SRTT$,

where V_s is the data transfer rate, S is packet size,

t_2 is the current time,

t_1 is the time when last CI was accepted.

When a CI is accepted, the source data transfer rate V_s is updated as $V_s = min(V_s, \beta V_f)$, $(\beta < 1)$. The decrease factor β is the same as that of the *Bin-CI* scheme. If there is no congestion detected or the CI is filtered by the algorithm above, V_s grows by $S/SRTT$ every $SRTT$.

The filter block is required because too many rate reductions ($V_s = min(V_s, \beta V_f)$) would eliminate the chance of rate increment and hence beat-down a multicast flow having many branches. The filter block attempts to keep track of only one of the receivers seeing the smallest egress rate (ER). Condition (1) is for initialization. Condition (2) means that if the receiver R whose CIs the source accepted most recently keeps sending CIs, the source will always accept them. To avoid neglecting other receivers, condition (3) accepts feedback from other receivers provided that the ER fed back $V_f < V_{fe} - 3\sigma_v$, i.e., the ER fed back is

[38] γ is 7/8 in our simulations.

statistically significant. Condition (4) is a statistical safeguard against the case of the receiver R disappearing, while the other feedback rates remain within the range $[V_{fe} - 3\sigma_v, \infty)$. Condition 4 implies that the source has not seen CIs from the receiver R for a long enough period $(RTT + 7\sigma)$, as well as that the transfer rate has been increased significantly $(V_s \geq V_{fe} + 3\sigma_v + S/SRTT)$ and congestion feedback is being received from some other receiver. We choose $RTT + 7\sigma$ as a long enough period after we consider the following extreme case:

> As usual[45], the round trip time estimate RTT is updated as $RTT = \theta RTT + (1 - \theta T)$, where T is a new sample. The error σ is maintained as $\sigma = \theta\sigma + (1 - \theta)|T - RTT|$ before RTT is updated by T.
>
> Let $RTT = t$, $\sigma = 0$. Suppose the next sample is $t + \delta$, $(\delta > 0)$. Then RTT becomes $\theta t + (1-\theta)(t+\delta) = t + (1-\theta)\delta$, the error $\sigma = \theta 0 + (1-\theta)|t+\delta-t| = (1-\theta)\delta$. To let $[RTT - x\sigma, RTT + x\sigma]$ include the sample of $t + \delta$, solve the equation of $RTT + x\sigma \geq t + \delta$, we have $x \geq \theta/(1-\theta)$. Since we use $\theta = 0.875$ in our scheme, $x \geq 0.875/0.125 = 7$ and we pick the smallest value. If $\delta < 0$, the discussion is similar.

D.4 Simulation Results

We ran several *ns*-2 simulations to verify the performance of our scheme. The simulations include (1) Simple Multicast Configuration, (2) Multiple Bottlenecks (Linear Network), (3) Drop-to-Zero Avoidance Testing. In these simulations, the data packet size is 1000 bytes. The bottleneck buffer size is 1MB which is sufficient to avoid any packet losses in all simulations. Queue graphs show that the real queue is far smaller. When the bottleneck buffer size is smaller, there can be packet losses while our scheme still performs well. However, we omit those results here.

D.4.1 Simple Multicast Configuration

Consider the simple multicast topology in Figure D.2. At time=0, there is only one multicast flow, with the source on Node 1, two receivers on Node2 and 3 respectively. At t=30s, another multicast flow is added with the same source-receiver set. At t=60s, the third multicast flow is added, again with the same source-receiver pattern.

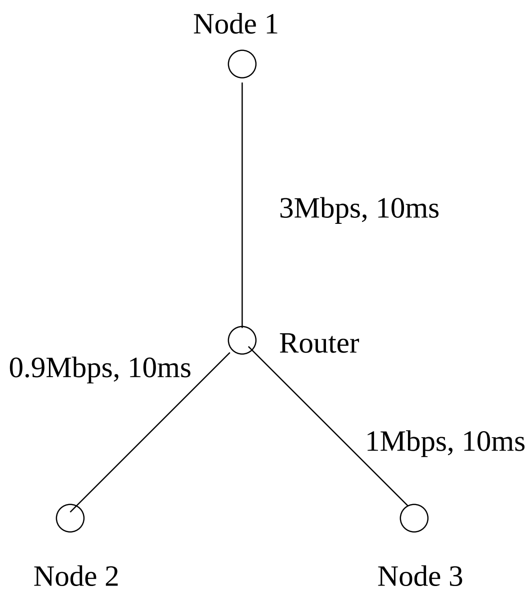
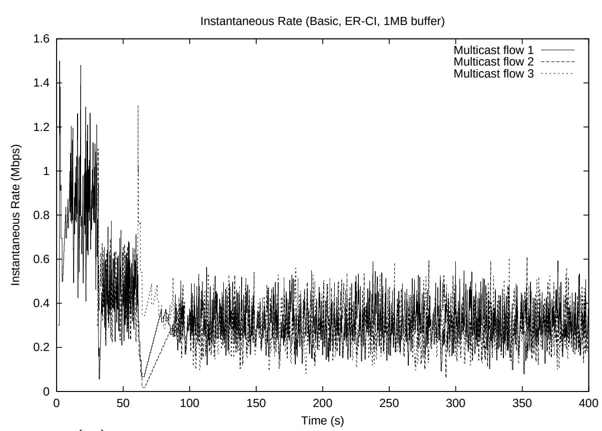
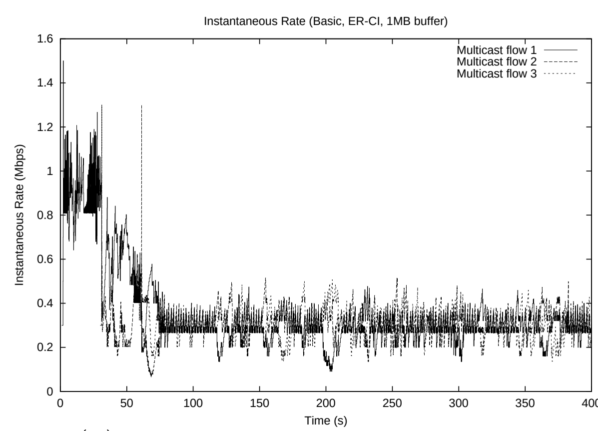


Figure D.2: Simple Multicast Configuration

Figures D.3(a) and (b) shows the performance of *Bin-CI* and *ER-CI* respectively. In both cases, observe that for t in [0s,30s], the rate oscillates around 0.9 Mbps. For t in [30s, 60s] the rates are around $0.9/2 = 0.45$ Mbps showing that the two multicast flows compete fairly. For $t > 60s$, the rates oscillate around $0.9/3 = 0.3$ Mbps, again with the rates being shared fairly.



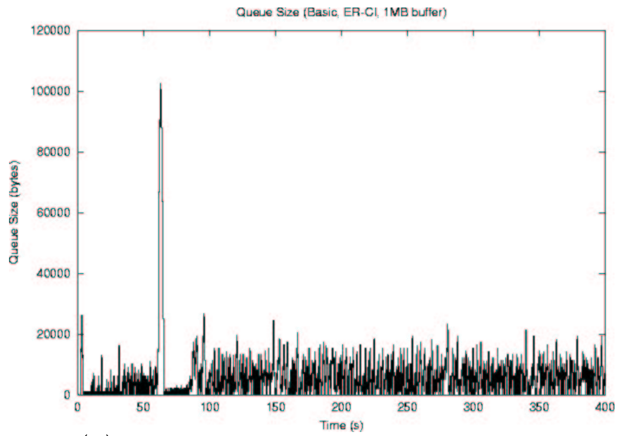
(a) Bin-CI, 1MB bottleneck buffer



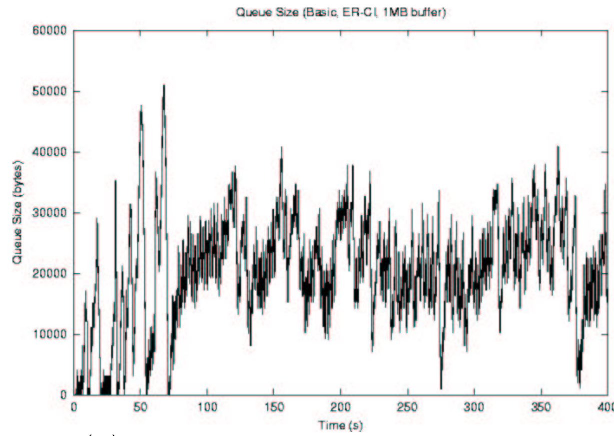
(b) ER-CI, 1MB bottleneck buffer

Figure D.3: Simple Multicast Configuration Results (Rates)
(Both Bin-CI and ER-CI can follow bottleneck dynamics.)

The queue sizes of different simulations are shown in Figure D.4 and the utilization/queue data is summarized in Table I. We can observe that the bandwidth utilization is high (above 80%) while the average queue length is low (up to 20 packets). Also observe that the utilization in the *ER-CI* scheme is in general higher than that in the *Bin-CI* scheme.



(a) Bin-CI, 1MB bottleneck buffer



(b) ER-CI, 1MB bottleneck buffer

Figure D.4: Simple Multicast Configuration Results (Queues)
(Both Bin-CI and ER-CI have relatively low average bottleneck queues, while Bin-CI is better than ER-CI.)

Table D.1: Average Queue Size and Utilization
(ER-CI is better at queue utilization while Bin-CI is better at queue length.)

	Average Bottleneck Queue Size (bytes)	Bottleneck Utilization
Bin-CI, 1MB bottleneck buffer	5745.93	83.0221%
ER-CI, 1MB bottleneck buffer	20359.8	98.0692%

D.4.2 Multiple Bottlenecks: Linear Network

The linear network is a popular multi-bottleneck configuration. We extend this configuration for multicast as shown in Figure D.5. There are three flows running on the configuration of Figure D.5. One multicast flow goes from Node 1 to Node 4 and 5, two single-receiver multicast flows go from Node 2 to Router 2 and from Node 3 to Node 4 respectively. In this configuration, if the flow that traverses multiple bottlenecks gets a larger share, it reduces the overall network capacity. Different notions of fairness define how much the long flow can get. *Proportional fairness* implies that the the long (multicast) flow should get one-third of the bottleneck bandwidth whereas *Max-min fairness* suggests a share of one-half.

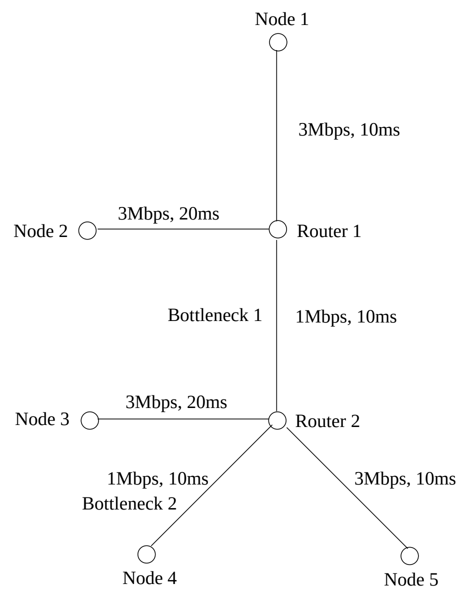
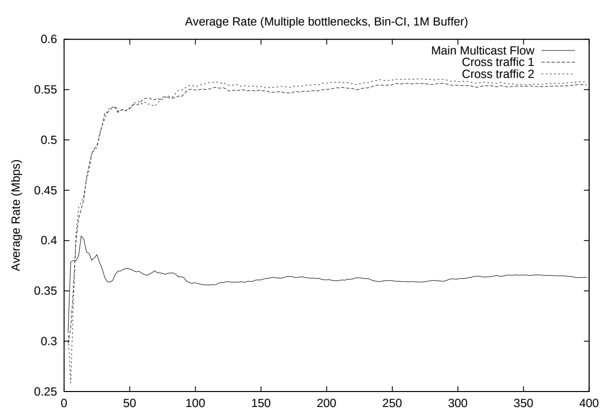
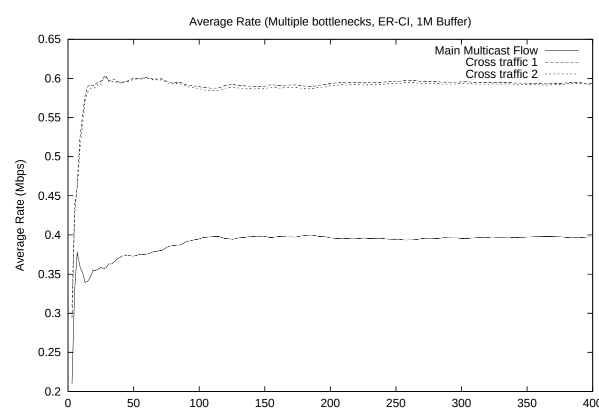


Figure D.5: Multiple Bottlenecks: Linear Network

The average rate [39] graph (Figure D.6) shows that the multicast flow gets more than 1/3 of the bottleneck bandwidth (the proportional fairness share), but less than 1/2 of the bottleneck bandwidth (the max-min fairness share).



(a) Bin-CI, 1MB bottleneck buffer



(b) ER-CI, 1MB bottleneck buffer

Figure D.6: Linear Network Results (Average Rates)
(MCA achieves a fairness between proportional and max-min.)

[39] Average rate = (amount of data sent between time 0 and t) / t, where t is the sampling time.

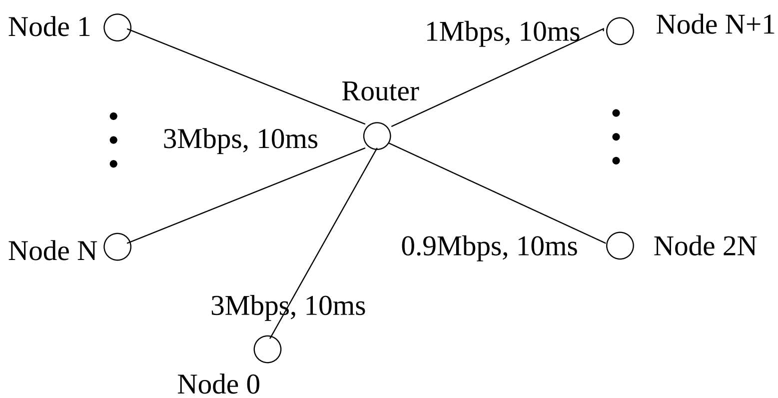
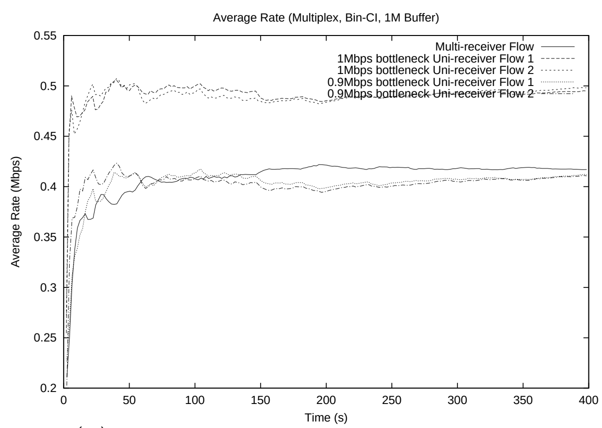


Figure D.7: Star Topology Configuration

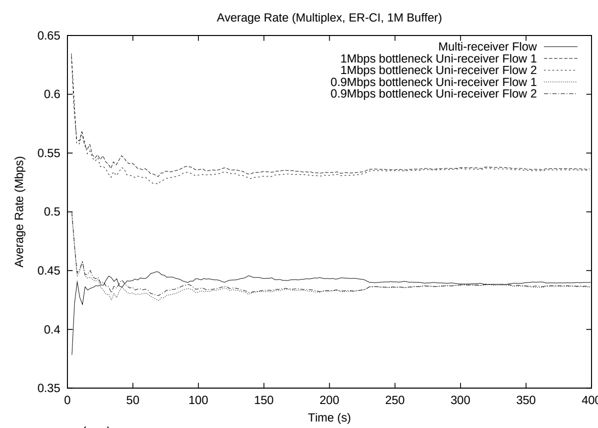
D.4.3 Star Topology: Drop-to-Zero Avoidance Testing

To test the immunity of the scheme to the drop-to-zero (DTZ) problem [94], we use the star topology (Figure D.7). The DTZ problem occurs when multiple paths in a multicast tree experience different congestion levels asynchronously, and the source reacts to more congestion feedback than necessary leading to a beat-down of transmission rate.

In the star topology, node i sends data to Node $i + N$ ($i = 1..N$), $N = 16$, thus generating background traffic on each path. A multicast flow has as its source, Node 0, and receivers, Node $N + 1$ to $2N$. The links between Router and Node j ($j = N + 1..3N/2$) have bandwidth of 1Mbps; and the links between Router and Node j ($j = 3N/2 + 1..2N$) have bandwidth of 0.9Mbps. The multicast flow should compete fairly with those single-receiver flows on 0.9Mbps bottlenecks. Indeed, figure D.8 shows that the multicast and the unicast flows sharing the 0.9Mbps bottleneck achieve equal rates around [0.4,0.45]Mbps. This demonstrates fairness and the drop-to-zero immunity of the schemes.



(a) Bin-CI, 1MB bottleneck buffer



(b) ER-CI, 1MB bottleneck buffer

Figure D.8: Average Rate of 16 Multiplexed Flows in Star Configuration
(MCA avoids drop-to-zero. Multicast MCA flows are fair with unicast MCA flows.)