Report Number: WUCS-00-03

2000-01-01

A Rate-based End-to-end Multicast Congestion Control Protocol

Sherlia Shi and Marcel Waldvogel

Current reliable multicast protocols do not have scalable congestion control mechanisms and this deficiency leads to concerns that multicast deployment may endanger stability of the network. In this paper, we present a sender-based approach for multicast congestion control targeted towards reliable bulk data transfer. We assume that there are a few bottleneck links in a large scale multicast group at any time period and these bottlenecks persist long enough to be identified and adapted to. Our work focus on dynamically identifying the worst congested path in the multicast tree and obtaining TCP-friendly throughput on this selected path. We device... **Read complete abstract on page 2.**

A Rate-based End-to-end Multicast Congestion Control Protocol

Sherlia Shi and Marcel Waldvogel

Complete Abstract:

Current reliable multicast protocols do not have scalable congestion control mechanisms and this deficiency leads to concerns that multicast deployment may endanger stability of the network. In this paper, we present a sender-based approach for multicast congestion control targeted towards reliable bulk data transfer. We assume that there are a few bottleneck links in a large scale multicast group at any time period and these bottlenecks persist long enough to be identified and adapted to. Our work focus on dynamically identifying the worst congested path in the multicast tree and obtaining TCP-friendly throughput on this selected path. We device novel selection (amongst receivers) and aggregation (over time) methods to achieve our goal. The response time of our protocol is then compatible to TCP once the worst path is identified. Only when switching between worst paths, the protocol response time is relaxed to multiple RTTs (less than 10) for the reasons of scalability and stability. We use the network simulator (NS2) to validate and evaluate our congestion control algorithm with both drop-tail and RED gateways.

A Rate-based End-to-end Multicast Congestion Control Protocol

Sherlia Shi and Marcel Waldvogel

WUCS-00-03

February 2000

Department of Computer Science
Washington University
Campus Box 1045
One Brookings Drive
St. Louis MO 63130

A Rate-based End-to-end Multicast Congestion Control Protocol

Sherlia Shi Marcel Waldvogel
Department of Computer Science
Washington University in St. Louis
{sherlia, mwa}@arl.wustl.edu

Abstract

Current reliable multicast protocols do not have scalable congestion control mechanisms and this deficiency leads to concerns that multicast deployment may endanger stability of the network. In this paper, we present a sender-based approach for multicast congestion control targeted towards reliable bulk data transfer. We assume that there are a few bottleneck links in a large scale multicast group at any time period and these bottlenecks persist long enough to be identified and adapted to. Our work focus on dynamically identifying the worst congested path in the multicast tree and obtaining TCP-friendly throughput on this selected path. We devise novel selection (amongst receivers) and aggregation (over time) methods to achieve our goal. The response time of our protocol is then compatible to TCP once the worst path is identified. Only when switching between worst paths, the protocol response time is relaxed to multiple RTTs (less than 10) for the reasons of scalability and stability. We use the network simulator (NS2) to validate and evaluate our congestion control algorithm with both drop-tail and RED gateways.

1 Introduction

Today's Internet applications, such as streaming media, on-line information retrieval and software or proxy caching updates, are demanding much higher bandwidth and much larger scale of distribution than ever. Multicast is an efficient method to disseminate data to a large number of receivers. Compared to using multiple unicast connections, a multicast connection reduces the transmission cost both at the data sources and in the network. However, the potentially large size of a multicast group and the heterogeneity among all receivers introduce hard problems in

scalably managing and controlling a multicast group in all aspects of multicast communication.

During the past few years, we have seen a lot of studies and improvement in multicast routing, address allocation and transport level error-recovery. Comparably, not as much mature work has been done in addressing the problem of multicast congestion control. Yet, congestion control is the key factor to the success of today's Internet. It ensures network stability and optimizes the network utilization by preventing applications from overloading the network. To facilitate the deployment of reliable multicast as a transport layer protocol, congestion control is an indispensable element that must be researched in greater depth. Although congestion control is required by all applications to ensure network safety, different applications have various constraints in speed, quality and consistency of data delivery. For example, real-time audio/video applications can trade quality for speed, while applications for software distributions can trade latency for reliability. For multicast congestion control, there is an additional challenge of meeting the heterogeneous conditions of different receivers. Hence, different approaches must be investigated for different class of applications.

In this paper, we set out to assess problems and solutions in multicast congestion control for a subset of multicast applications: reliable bulk data transfer. The example applications of this category include software distribution, web proxy updates and other synchronization and updates for data replication services. These applications usually transmit high volumes of data to medium- or large-sized groups and require reliable data delivery. However, there is no stringent requirement on the speed of data delivery, and whether all participants receive data at their maximum capabilities is not critical to the application. This gives us space for adapting data transmission rate according to network congestion among heterogeneous receivers.

We propose a sender-based congestion control scheme that tries to match the bandwidth condition on the *worst receiver path*. We define the *worst receiver* as the one that is downstream of a congested link that has the smallest bandwidth capacity among all multicast links. However, such a receiver may not be unique and may change dynamically. Hence, we devise novel selection (over receivers) and aggregation (over time) methods to identify such a *worst receiver*. We sketch the main issues addressed in our scheme as follows:

- **Metric for congestion indication:** the metric for congestion indication defines when a receiver should notify the sender of its congestion condition. Unlike multicast error control, in which receivers send feedback when they observe a packet loss. Multicast congestion control requires a metric that captures the degree of a receiver's congestion condition over a recent period of time. A single packet loss fails to indicate congestion reliably and will result volatile adaptation at the sender [2]. In our scheme, we choose the metric as a function of

receiver measured loss rate and round-trip time.

- **Feedback implosion control:** the classic problem in reliable multicast is how to deal with the potentially huge volume of feedback from all receivers. Various suppression mechanisms have been proposed in [6, 20, 19, 24] for multicast error control to reduce the number of NAKs for the same lost packet. Without re-inventing the wheel, we adopt these mechanisms to suppress congestion indications for the same congested link.
- **Responsiveness:** the responsiveness of a congestion control scheme is crucial to how the protocol will affect the network stability. Ideally, the sender should adapt to network changes within a round trip time or even less, decreasing its transmission rate as soon as the congestion builds up in the network and increasing its rate as soon as the congestion disappears. In order to achieve better responsiveness, a protocol should minimize the delay in congestion detection and rate adaptation. Yet, an over-responsive protocol may cause wide rate fluctuations and impede the network stability. In multicast, however, it is hard to identify the most congested link in a short time scale without causing instability due to the highly variant network traffic along different paths. Hence, in our closed-loop control, we trade off protocol responsiveness for scalability and correctness (in metric calculation), and adapt rate on the scale of a few round-trip times.
- **Fairness:** the fairness issue characterizes how the protocol co-exists and shares bandwidth on the bottleneck link with existing protocols, especially with TCP. Though TCP's congestion control mechanism is itself a moving target, a new mechanism still has to treat it fairly in order to become viable. Through simulation, we show that our protocol is able to achieve bandwidth sharing with TCP within a factor of 2.

We should point out that the above issues are not at all independent, rather they interfere with each other closely and sometimes orthogonally. Our scheme therefore compromises among these objectives. We use simulation to study and evaluate our proposed scheme in NS2 [10]. Simulation results show that we are able to achieve TCP-fair throughput and be responsive to the network congestion even in large multicast groups with different link characteristics.

The rest of the paper is organized as follows. In Section 2, we present the background and our motivation for end-to-end congestion control. In Section 3, we classify problems in sender-based approach multicast congestion control, then in Section 4, we present our solutions and trade-offs in our design choices. In Section 5, we describe our scheme in detail and in Section 6, we use simulation to validate and evaluate the performance of our scheme. In Section 7, we discuss related work and conclude in Section 8.

2 Motivation and Background

2.1 Importance of End-to-end Congestion Control

There are two basic ways to deal with congestion in today's Internet: either application *actively* adapt to the available bandwidth, or the network enforces rate limits for every flow and therefore the applications *passively* adapt to the limited rate. The Internet is currently dominated by one reliable transport protocol – TCP, which is adopted by most of the applications running in the current Internet, such as World Wide Web, FTP, and Telnet. TCP provides end-to-end reliability and achieves network safety using end-to-end congestion control. These functionalities are delivered without additional network complexities or services but assuming that all end users are cooperative in gracefully increasing and decreasing their transmission rate based on the discovered available bandwidth.

The second way of dealing with congestion control is emerging in the form of integrated and differentiated services. These router-aided congestion control methods require flow reservation, profile specification and admission control ahead of the flow initialization. Furthermore, they also require routers to keep per-flow or aggregated flow states to enforce the reservations. The complexity of signaling and maintaining flow states have shifted the research focus from integrated services to differentiated services, which classifies flows into classes and provides statistical guarantee on a per-class basis instead of a per-flow basis. Although the differentiated services architecture has shown promises in its future, its deployment in a large infrastructure such as the Internet remains to be seen. We also observe that since differentiated services aggregate flows into classes, then within each class, flows must still be cooperative to each other and adaptive to the bandwidth allocated to that class, although their adaptive ranges can be different for each class. Therefore, we believe it is necessary and imperative to deal with congestion control in an end-to-end manner.

2.2 Approaches of End-to-end Multicast Congestion Control

Although end-to-end multicast congestion control is desirable, it is fundamentally a hard problem due to the heterogeneity of link bandwidth and delay on each individual receiver's path. This heterogeneity introduces an essential scaling issue: how can a sender decide the transmission rate if every receiver has a different capacity?

Two distinct ways of congestion control are introduced in the current reliable multicast literature: sender-based [4, 9, 15, 26, 22] and receiver-based [12, 14, 25]. The sender-based approach is essentially similar to what TCP does, the sender uses a single transmission rate

that affects all receivers and infers network congestion from feedback collected from receivers. This approach suites applications such as bulk data transfer, where the primary goal is to deliver data reliably to all receivers but some receivers may suffer delay in waiting for others. The sender-based approach also requires extreme careful design in dealing with problems such as implosion control and drop-to-zero problems, which we will address later in the paper.

In favor of other types of applications such as audio/video, which are sensitive to delay but tolerant to some amount of quality reduction, a receiver-based approach is also proposed. In this approach, data is organized into layers and transmitted onto different multicast groups. Receivers can thus choose how much data they can accept under the current network condition and only subscribe to those layers. For reliable bulk data transfer, problems stem from the receiver-based approach as data does not come with a natural layering, and it is hard to organize data into layers, yet maintaining data consistency and ordering. In addition, due to the complexity of encoding and decoding, the number of available layers is normally confined to be small, thus limiting the adaptive range of congestion control. The receiver-based approach also requires receivers to coordinate when join and leave a multicast group, causing more overheads and dependency on the underlying multicast routing protocols.

For the sake of simple explanation, in the rest of the paper, we generally refer to congestion control in the context of sender-based and end-to-end congestion control for reliable multicast.

3 Issues in Multicast Congestion Control

The primary goal of congestion control is to let the applications use the network resources efficiently by being responsive and adaptive to the network congestion occurred along the application's data path. Two fundamental issues arise when applying this principle to multicast congestion control: *scalability* and *fairness*.

3.1 Scalability

The scalability issue is essential to all multicast based protocols. A multicast congestion control protocol not only needs to scale to a large number of receivers but also needs to scale in a more heterogeneous environment with different link capacities and delays. Two resulting problems need thus be addressed: feedback implosion and rate drop-to-zero.

The implosion problem has been well explained in the literature on multicast error control, and various feedback suppression mechanisms have been introduced [6, 20, 19, 24]. However, all these mechanisms come with the cost of introducing extra delay in feedback. Feedback delay

directly contributes to the responsiveness of congestion control schemes, the longer the delay the less the responsiveness. This irresponsiveness of a multicast flow is especially dangerous to the network, as it potentially creates fluctuated link conditions along the whole multicast tree and may drive the network into instability. Additionally, in contrast to error control in which feedback is only triggered by packet losses discovered at receivers, in congestion control, the source needs constant feedback from the receivers to discover not only the congestion but the re-availability of resources as well. These continuous feedback should be well managed to avoid implosion and to achieve scalability, yet they should also be delivered in a timely manner for sender to react to network congestion.

The drop-to-zero problem is also known as loss multiplicity problem [2]. The problem arises when receivers use packet losses as congestion signals and the source uses these signals to regulate its transmission rate without proper aggregation. When packets are lost on multiple paths independently, receivers downstream of these paths will all send congestion signals to the source resulting in multiple rate drops at the source. In the current IP multicast model, the data source does not know the receiver topology, hence cannot aggregate the congestion signals over receiver locations. Generally, when there are multiple bottleneck paths, the source has to adapt to the sum of the congestion signals generated on these paths and its rate will be quickly throttled as the number of congested paths increases.

3.2 Responsiveness and Fairness

In today's Internet, TCP is the dominant transport protocol and its success largely attributes to its congestion control and error control mechanisms [11]. Consequently, it is important to design a multicast congestion control scheme which coexists and shares the bandwidth fairly with TCP. In reliable bulk data transfer, the fairness is defined as to achieve TCP-compatible throughput on the worst sender-to-receiver path. The responsive time of TCP's window based congestion control mechanism is typically one RTT (fast retransmission) or one retransmission time-out. As we have pointed out earlier, the delay in the feedback makes a multicast congestion control scheme hard to respond as fast as TCP, and therefore, fairness cannot always be achieved in a very short period (at most one RTT). Furthermore, the TCP congestion control scheme is tightly coupled with its error control scheme. A TCP receiver uses the left edge of the window (the highest sequence number of continuously received data) to ACK to the source instead of the right edge (the highest received data sequence number), and at a retransmission time-out event, it reduces the congestion window size to one segment size. In the case of multicast, however, the sender is not necessarily informed of all packet losses and packets can be retransmitted locally.

Hence, a coupling of error control and congestion control will only add additional complexity and unscalability to the protocol. It is possible that local retransmissions may well endanger the already congested path by injecting more packets into it, and care must be taken to limit these local retransmissions. However, it is still an open issue as how local retransmissions should be limited. The decoupling of the error control and congestion control implies that in time of severe congestion when the packet loss ratio is very high, the different degree of responsiveness taken by multicast and TCP congestion control will result in some degree of unfairness.

It was also pointed out in the last RMRG(Reliable Multicast Research Group) meeting [23], that we may just have to live with the slow responsiveness of multicast flow and only try to achieve fairness with TCP in a long run. Yet, it is not clear how the network and other flows will sustain during this period of overload and how long the period should be.

In the next section, we introduce our approach which tries to identify the worst receiver path and adapts the data rate to this path in a TCP-similar manner. Because we single out one worst path at a time, relatively fast feedback can be obtained on this path and the flow can therefore respond quickly to congestion. In addition, each receiver estimates its own capacity and compares with that of the worst receiver. The capacity of a receiver is calculated as a function of experienced loss rate and RTT, which we will detail later. This provides a richer feedback to the source for switching between receivers to adapt its rate and significantly reduces the drop-to-zero effect. We employ an additive increase/multiplicative decrease rate adaptation algorithm at the sender, which is similar to that of TCP, to achieve fair bandwidth sharing.

4 Design Assumptions and Solutions

In this section, we first outline our main assumptions of the network model that our protocol is designed to operate on and then our key ideas that solve the problems described in the previous section. For simplicity reasons, we describe our scheme in an one-to-many scenario. A many-to-many case can be generalized by running a separate instance of protocol for each data source.

4.1 Network and Application Model

We assume in our model that there are a few bottlenecks within a multicast group and they persist for a period of time long enough for the data source to adapt to. These bottlenecks are links with offered load near their capacity, creating a few congested paths to downstream receivers within a multicast group. The nature of the network traffic is unpredictable and

dynamic, suggesting that these bottlenecks can change from one to another and the degree of congestion on each path can vary over time. However, studies in [8, 16] show that there are typically a few "hot spots" in the network that are significantly more congested than the others and these few bottlenecks usually remain for a noticeable time.

We also assume that the application specifies its rate adaptation range and takes care of dynamic group membership. For example, if a path is so severely congested that the application cannot tolerate the low adapted data rate, members downstream of the path may need to be dropped out. But this should be decided and performed by the application itself, while the congestion control protocol will simply adapt to as low rate, possibly zero, as the receiver feedback indicates.

4.2 Solution Outline

We propose a rate-based and sender-based multicast congestion control protocol that relies only on end-to-end feedback. In our design, we use a combination of distributed receiver feedback suppression and sender feedback aggregation scheme to handle shared and independent congestion, and identify the worst receiver path; we adopt the addictive increase and multiplicative decrease rate adaptation algorithm (AIMD) to achieve TCP-fairness on the worst path. The main ideas are outlined below:

- **Agent architecture:** An *agent* is defined as a receiver downstream of the most congested data path. There is a single agent among all receivers at any time instance. An agent sends positive or negative feedback(PF/NF) to the source indicating the congestion condition of its represented path. Such an agent is dynamically selected and it helps the source to adapt to the worst path in a timely manner.
- **Agent selection:** The basic criterion of selecting an agent is that it must be located in the most congested subtree. Among those, the closest receiver to the bottleneck is ideally the best agent since it senses the path condition faster than the rest of receivers, hence sending feedback more quickly. However, the selection mechanism must not cause any implosion problem. In our scheme, each receiver independently decides when to become an agent based on their estimation of its path condition and sends a *congestion notification*(CN) to the source. A suppression mechanism is then applied to avoid implosion of CNs within a subtree and an aggregation method at the source to single out the worst receiver from multiple independent CNs.

- **Feedback metric:** Each receiver estimates its *capacity* as an indication of the degree of congestion on its own path and decides locally when to send a CN. This capacity estimation is a function of both measured loss rate over a recent period and measured round-trip time. It avoids largely the drop-to-zero problem and avoids false alarms such as packet errors or random losses by averaging losses over time, thus mitigating the effect of each individual packet loss.
- **Rate adaptation:** We use a rate-based AIMD adaptation algorithm on the identified worst path to achieve congestion control and TCP-fairness. This choice is mainly for simplicity, since the sender can use a single parameter, the transmission rate, across all receivers. On the contrary, a window-based scheme has extra complexity in maintaining and synchronizing the congestion window across all receivers. In addition, a rate-based approach is more friendly to the network, when a window-based scheme generates data bursts periodically.

5 Protocol Details

In this section, we elaborate our solution in details and also discuss some unsolved issues up front. Our congestion control mechanism builds on top of any existing error control protocol. Instead of devising a parallel suppression mechanism to the one already existing in error control, we re-use it for the suppression of congestion notifications. Although there are differences in the implosion and exposure control for these error control protocols, the impact on our scheme is very small so we do not discuss them in great details.

5.1 Identifying bottleneck under shared and independent congestion

Most of the issues we discussed so far stem from the fact that neither the source nor any of the receivers is able to distinguish shared congestion from independent congestion. The occurrence of shared congestion results in feedback implosion as receivers do not know whether they are within the same subtree. On the other hand, independent congestion contributes largely to the drop-to-zero problem, since the source cannot effectively aggregate all the congestion notifications. Figure 1 shows the three categories of congestion: independent, shared and a combination of both.

Independent congestion: For independent congestion, we use a two-step suppression: firstly each receiver sends a CN if and only if it has worse capacity then the current agent, for example, if A is an agent, and B's measured capacity approximates to A's then B will not send a CN. This first step significantly reduces the number of congestion notification from the number of

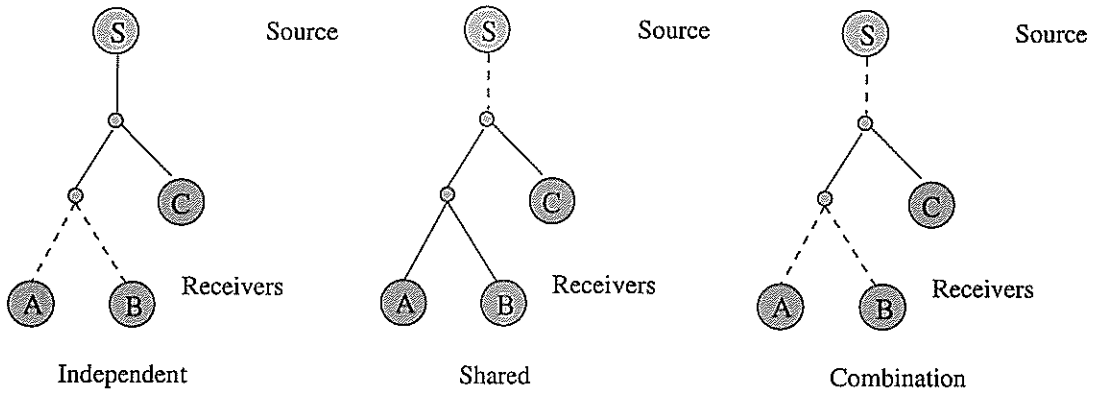

Figure 1: Independent and Shared Congestion

congested subtrees to the number of heavily congested subtrees; secondly the sender selects the worst among all receivers who send a CN, as a new agent. For example, if both A and B simultaneously detect severe congestion and both send CN, then the sender will only choose one of them, the worse one.

Shared congestion: Shared congestion happens when there are multiple receivers downstream a common congested link. Generally, every receiver in this congested subtree is eligible to send feedback to the source, but the ideal receiver to feed the source should be the one closest to the bottleneck, since it is the one that first detects the packet loss and is able to send the source the quickest feedback on the condition of the bottleneck link, thus minimizing delay. In Figure 1, assuming all links have the same delay, then receiver C should be ideal to send feedback. However, the estimated capacity at each receiver is a function of both RTT and loss rate, so receiver A, B will then have worse capacity even though they are not the optimal agents. The problem can be solved if receiver C's congestion notification can always suppress A's and B's. This is possible in a hierarchical suppression method such as RMTP [20], LMS [19] etc., where the designated receiver can distinguish if CNs from downstream are the same as its own by looking at the sequence number. For schemes like SRM [6], where randomness is used in a flat topology, then we have to depend on the accuracy of SRM's timer estimation among A, B and C such that C's notification timer always fires beforehand and will reach A and B in time to suppress their notifications. Therefore, there is fuzziness in selecting the best agent, however, it does not affect the correctness of determining the bottleneck link.

Combination of independent and shared congestion: In time of receivers experiencing both independent and shared congestion, our scheme still converges to the receiver with the worst capacity. As shown in Figure 1(c), initially when no agent is selected, all three receivers

are eligible for sending CNs. Initially the sender may select C as the agent since C's CNs reach first. Some of A's and B's CNs are suppressed by C's CNs due to shared congestion, but since A and B are experiencing higher loss rate, they are generating more CNs when C doesn't and in turn become the new agent. When either A or B becomes agent, C will stop sending CNs since it discovers itself no longer the worst receiver.

5.2 Feedback Mechanism

In the current Internet, congestion is usually detected by packet loss. A naive approach of reporting congestion might be sending a negative acknowledgment to the source on detecting a packet loss. However, this scheme does not scale for two reasons: (1) for congestion control, we not only need to know when the congestion happens, but also when the resources become available again, so the source's rate can ramp up again to efficiently use the resources. If a simple NAK-based scheme is adopted, the source must suffer delay in detecting the resource re-availability, typically by a long time-out of lack-of-NAK, and cannot use the resources efficiently; (2) a NAK-based congestion signal suffers from the drop-to-zero problem, since in a large multicast group, every single packet may have a high probability of getting lost on at least one of the paths. A source does not have enough information from the NAKs to aggregate them, resulting in unnecessary bandwidth throttling. Therefore, the source needs a richer set of information. In addition, for scalability reasons, this information must be calculated by each receiver locally.

There are two parameters to decide a worst receiver: round trip time and loss rate. It is obvious that the packet loss rate directly measures the link condition towards the receiver, however, it is less obvious that the round trip time also affects the choice of the worst receiver. This is because in closed loop control, the feedback time controls how fast the rate can be adjusted. If the adjustment range is the same, the faster the rate oscillates, the higher the throughput. In addition, TCP also provides fairness proportional to round-trip time. If a close-by receiver is experiencing high loss rate, while a receiver further away is experiencing moderate loss rate, adapting to the close-by receiver may result higher data rate than what TCP would achieve on the longer path, hence causing unfairness. We calculate the capacity as below:

$$Capacity = 1/(RTT * sqrt(lossrate))$$

The time period of calculating this capacity directly affects the responsiveness of the congestion control scheme. If the time is too long, the estimated congestion degree is smoothed too much and the flow will be unresponsive. On the other hand, if the time is too short, the

receiver does not have sufficient information to obtain an accurate estimation while filtering out the noise, causing the flow to be over-responsive. A pure NAK-based approach can be viewed as such an example: it uses one packet time to report the capacity of zero or one and cannot filter out any noise caused by loss variance.

Note that our capacity equation bears assemblance to the steady-state TCP throughput equation [13]. This is no coincidence. Indeed, we try to closely model the equivalance of TCP throughput, and hence the choice of square root of loss rate as a parameter.

5.3 TCP-like Rate Adaptation Algorithm

Once an agent is selected, it sends positive or negative feedback to the source every RTT. The source uses AIMD to adapt its transmission rate: the rate is increased one packet every RTT, where the RTT is reported by the agent; and the rate is decreased to half upon receiving a negative feedback. The agent's PF/NF includes its current estimated capacity, measured round trip time and the sequence number of the last missing packets. The source only adapts to PF/NF with the sequence number higher than that of the first data packet sent after last rate adjustment. This delay in action ensures that the source only adapts to newly experienced congestion. In addition, it increases its rate only if a PF indicates an increasing capacity. In other words, if there is queue building up in the network resulting in an increasing RTT and decreasing capacity, the source anticipates it and does not increase its rate.

Therefore, during the lifetime of the bottleneck link, if additional traffic is created, the source will detect the decreased link capacity reported by the agent, since it is experiencing either higher queuing delay or higher loss rate. On the other hand, if some flow is terminated on the bottleneck link, the source will detect the increased link capacity from the agent.

The agent's report is aggregated over one RTT, that is if there are multiple losses over one RTT, it only sends one NF. An NF has precedence over a PF, so the source receives one PF or NF every RTT. Thus, the response time to a packet loss event is one RTT which is similar to the time needed in TCP's fast retransmission algorithm. For small bursty losses within one RTT, TCP only halves its window size once according to the fast recovery algorithm, while in our scheme, the rate is reduced once since only one loss event will be reported. However, a large burst of losses typically causes TCP's retransmission timer to expire and reduces the congestion window size to one segment. In our scheme, large bursts may cause multiple rate drops but may not result in as conservative a rate as TCP's.

5.4 Open Issues

There are still several open issues that we are aware of, but are left alone in our design, including: the choice of initial data rate and RTT, the measurement of RTT and the control of local retransmission rates. These issues are very important in actual protocol implementation and deployment. We discuss them briefly below:

Start-up behavior

The choice of an initial transmission rate and the default RTT has been raised during a recent RMRG meeting. It is generally agreed an RTT of 500ms is big enough to cover the span of current Internet. If we start transmission at 1 packet/RTT and increase the rate 1 packet/second every 500ms, suppose we have a 1Mb link and use packet size of 1KB, it will take about 32 seconds before the sender is able to saturate the link. The same problem was presented 10 years ago in the design of TCP initialization method and the solution is to use an exponential increase whenever the window size is dropped to one. Yet, an exponential increase may not be suitable for multicast since TCP usually suffers much higher losses during its slow start period and this situation may be amplified even more in a multicast environment.

An ideal solution is to let all routers on the multicast tree to select a rate it is able to handle based on its history and propagate back the minimum rate to the source. However, the policies involved in such a design are beyond the scope of the paper. In our simulations, we always choose an initial rate such that the source is able to saturate the link in a small time interval.

RTT measurement

In our scheme, each receiver needs to maintain an average of round trip time measurements to calculate the capacity. This assumes synchronized clock and symmetric links between the data source and the receiver. Unfortunately, these two conditions may not be true in some situations, such as when satellite links are involved. In [1, 17], hierarchy-based round trip time measurement without the above assumptions are proposed. Since such a measurement happens outside the loop of data rate control, we believe it, as well as other future solutions, can be directly plugged into our scheme. One must note that this is not a problem for agents, because they send feedback frequently to the source, thus allowing the sender to measure the RTT and relay back the information.

Error control and retransmissions

As we mentioned earlier, retransmissions can be harmful to an already congested link if they are not rate limited. If the data source is the only one retransmitting, then the source should account for the retransmissions as part of its original data transmissions. If local retransmission

is allowed, ideally these retransmission should only affect the local subtree and the receiver who sends the retransmission should limit the transmission rate to some small percentage of the data rate. Since packet losses are rarely consist of big burst, we imagine limiting the retransmission rate to be small is acceptable and will not increase the recovery delay significantly.

6 Simulation Study

We have implemented our multicast congestion control scheme on top of the Scalable Reliable Multicast protocol (SRM) in Ns2. We adopt the random timer based suppression method of SRM to send CNs and keep all other aspects of SRM including error recovery, session messages unchanged.

The main metric we are interested in is the throughput delivered to each receiver, with varying multicast group sizes and competing traffic. We compare with TCP performance to study the fairness issues. The other metric we are interested in is the impact of traffic and network dynamics.

6.1 Multiple Competing SRM/CC Flows

In this experiment, we construct a simple scenario to understand the basic behavior of our congestion control scheme. We initiate 20 SRM/CC flows at random time for 0 to 1 second, sharing a single bottleneck link of 10Mb/s and using 1KB packets. We vary the round trip time from 60ms to 600ms and the simulation runs for 100 seconds. Figure 2 shows a scattered throughput plot of each flow with drop-tail queue on left and RED on right.

The ideal fair bandwidth sharing for each flow is 64KB/s. Figure 2 shows that when the round-trip time is low, bandwidth is more equally shared among all flows. With an increasing RTT, the dispersion of throughput also increases. The maximum throughput ratio is a factor of 2.5 when RTT equals to 600 ms. This ratio is true for both drop-tail and RED, with RED having a slightly wider dispersion. The increase of RTT attributes to feedback delay and flow response time, resulting in higher loss rate. Meanwhile, the variances of SRM suppression timers also increase since these timers scale on receiver RTTs. We believe these variance are the major cause of throughput deviations. The highest loss rate among all flows during the entire simulation is around 3% (not shown) which is quite good given the additional delay in feedback.

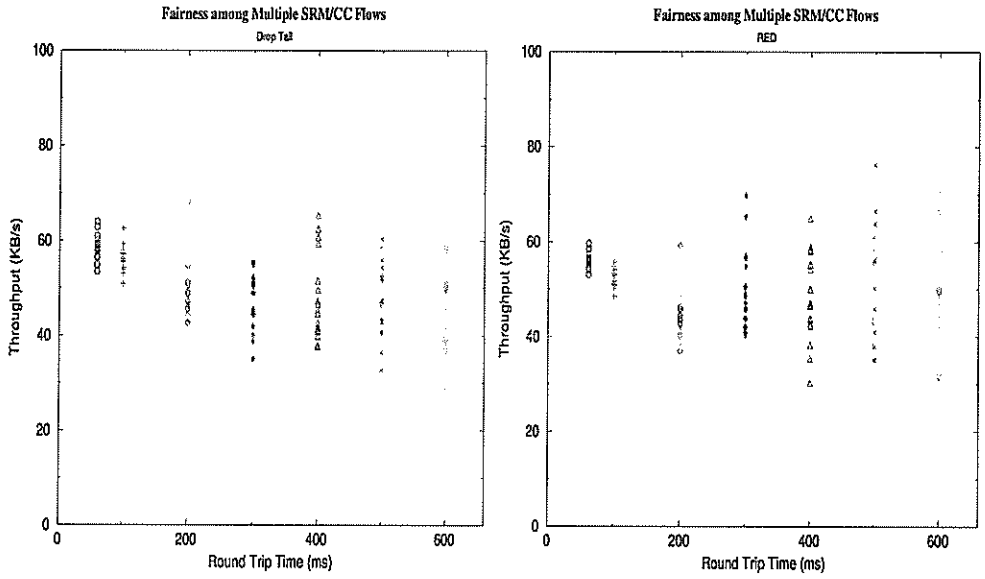

Figure 2: Fairness among Multiple SRM/CC Flows

6.2 TCP Fairness

In this experiment, we study the TCP-fairness aspect in a single bottleneck network model. We use TCP-Reno as the base-line TCP. The topology we use is shown in Figure 3. All traffic sources are at the left side of the bottleneck link, and receivers on the right. We keep the bandwidth of the bottleneck link to be proportional to the number of flows sharing the link so that in the ideal case, each flow should always get the same amount of throughput regardless of other varying parameters.

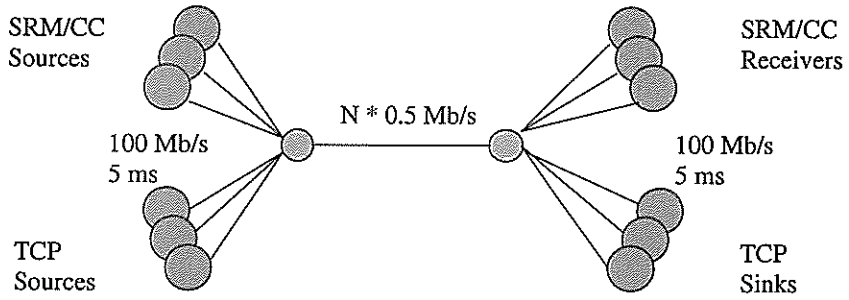

Figure 3: Topology for TCP-fairness Simulation

Due to the excessive memory consumption of NS, we were not able to construct large multicast groups while simultaneously running many SRM flows. Instead, we keep each group size small with two receivers only but varying the number of competing flows from 10 to 80. Half of these flows are TCP and the other half SRM/CC. All flows are started randomly during the

first 5 seconds and the simulation runs for 120 seconds.

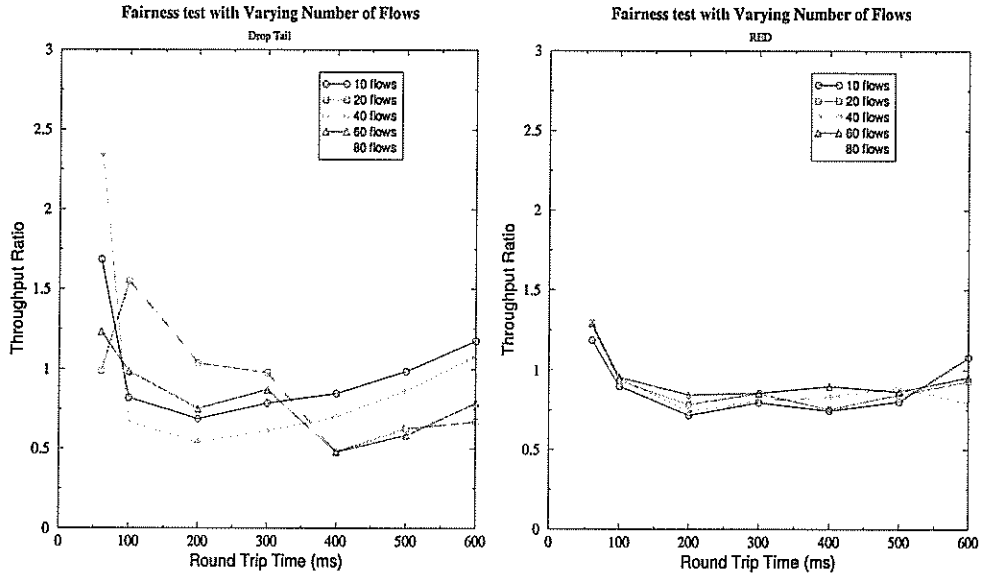

Figure 4: TCP Fairness

Figure 4 shows the average throughput ratio between SRM/CC and TCP with drop tail queues and RED. We observe that the throughput ratio lies between 0.5 and 2.5. This ratio is compatible with that of the previous experiment, meaning that SRM/CC is able to treat TCP fairly. We also observe that when RTT is small, SRM/CC is more aggressive while TCP tends to get higher throughput with larger RTTs. When RTT is small, TCP's congestion window is small because of the low bandwidth-delay product, therefore, multiple losses in a single window create more time-outs at the source resulting slow start and low throughput. On the other hand, once the TCP congestion window opens up high, the fast retransmission helps TCP recovering lost packets in about 1 RTT, so the multiple-loss impact on TCP is very little. In this case the TCP becomes more aggressive than SRM/CC. This is typically why the majority of the points in figure 4 passing the 100ms RTT mark, are less than 1.

We repeated the same set of test for RED gateways. For RED, the minimum threshold is kept at 5 packets, the maximum threshold at 20 packets and the queue weight at 0.003. Clearly, RED is able to achieve better fairness than drop-tails queues. The early warning of the incipient congestion and the smaller averaging queuing delay reduce the bursty loss for TCP, and reduce the measured RTT variance for SRM/CC. This results in more stable behavior and brings closer the throughput of the two types of flows.

We conclude from this test that SRM/CC is able to achieve good fairness with TCP and

scales well with large number of flows. However, TCP operates in two phases during congestion: slow start and fast retransmission. This non-uniformity is not necessarily a desired feature for congestion control, but it certainly makes it hard for other type of congestion control to achieve complete fairness with TCP.

6.3 Impact of Network Dynamics

In this experiment, we examine the impact of network and traffic dynamics on source's rate adaptation. Figure 5 shows the topology used in this experiment. We uses CBR sources to create dynamic traffic on the intermediate links. The size of SRM/CC multicast group is 20 with each receiver link delay uniformly distributed between 5ms and 100ms. The traffic dynamism are as follows: at t = 0, SRM/CC source start transmission; at t = 20, CBR flow 1 transmits at 20KB/s; t=40, CBR flow 2 transmits at 40KB/s; t=60, CBR flow 3 transmits at 30KB/s; t=80, CBR flow 2 stops. We add zero-mean noises of uniform distribution over [-0.5, 0.5] to the inter-packet transmission time of CBR flows. We tests both drop-tail and RED queue at the bottlenecks. The link bandwidth settings are shown in Figure 5. All other parameters are the same as before. The simulation runs for 100 seconds.

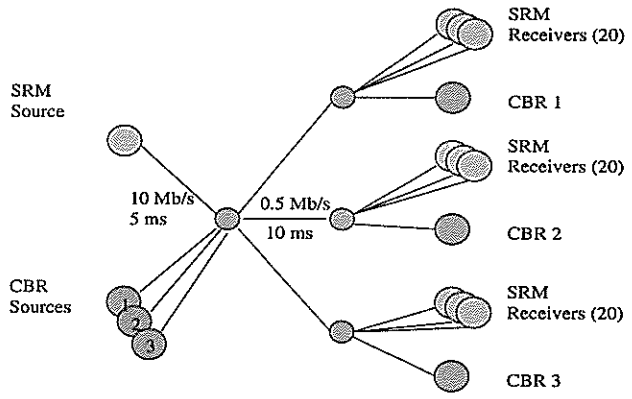

Figure 5: Simulation Topology for Dynamic Network

Figure 6 shows the source rate adaptation over time, the dotted line shows the available bandwidth. We observe that the rate oscillation of SRM/CC follows the available bandwidth closely regardless the RTT difference among downstream receivers. This clearly shows that SRM/CC is able to detect the switch of congestion path and react to it rapidly. The early warning from the RED routers seem to create more oscillation. At time t=80 seconds, when CBR flow 2 stops, both graphs show a spike. This is because that at this instance the agent is temporarily downstream of a "good" path and the source is getting the feedback from the

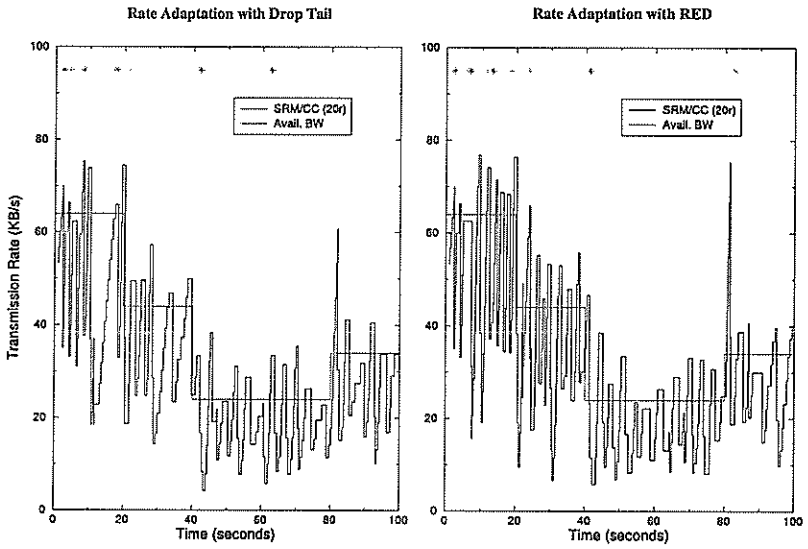

Figure 6: Response time of SRM/CC

mis-informed agent. This mistaken period lasts about 1 seconds, or about 20 RTT.

At the top of each figure, we also listed the switches of agent at the sender side. Commonly in both drop-tail and RED, SRM/CC is able to pinpoint an agent behind the bottleneck quickly when traffic changes. There are some oscillations between agents during the first 20 seconds when all three links are experiencing same degree of congestion. This leads to our next experiment on whether SRM/CC have excessive agent oscillations when all links are suffering from similar degree of congestion.

When sender switches agent, it drops its transmission rate to half. Hence, agent oscillations may lead to extreme low throughput. These oscillations are due to the large variance of loss rates on each individual path which causes receivers unable to measure accurately their loss rate over a short period time. We examine the impact of network loss variance on our congestion control scheme. In this experiment, we use a star topology of 40ms round-trip time and 0.5Mb/s links with one multicast group of 50 receivers. On each path, random loss is created with mean p varied from 1% to 10%. Each random loss is a uniform distribution over $[0, 2 * p]$.

Figure 7 shows the throughput comparison of TCP under the same loss rate and SRM/CC. It shows that SRM/CC does exhibit some performance degradation when loss rate is low. Since the lower the loss rate, the longer the time period needed to measure it accurately. However, as loss rate increases, SRM/CC behaves more stable than TCP and throughput of two flows are closer when loss rate is around 5%. In [21], Paxson suggested that loss bursts exhibit a "heavy-tailed" distribution, indicating immense variability over both small and large time scale. If this

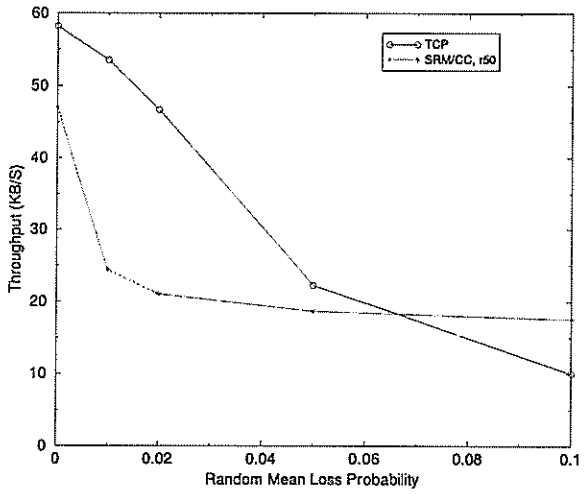

Figure 7: The Drop-to-Zero Problem

observation is true, then any end-host based measurement cannot be accurate unless using a prolonged period which is not suitable for congestion control. We are currently underway to research other mechanisms that may help in producing more precise measurement.

7 Related Work

There are several sender-based multicast congestion control scheme presented in [4, 7, 9, 26, 22]. In [4], Delucia and Obraczka presented a similar model to identify independent bottlenecks and letting representatives in each of these subtrees to send feedback. Although the basic model is similar to ours, the methods of identifying the bottlenecks, suppression of congestion signals and the rate adaptation mechanism are completely different from ours. In [4], feedback is in the form of ACKs and NAKs, and is multicast to the entire group for the sake of suppression of shared congestion signals. The ACK/NAKs that escape the subtree and reach the source are likely from independent subtrees, and are selected by the source as representatives. Once the representatives are selected, the source adopts a TCP-Vegas [3] like rate adaptation algorithm. We argue that this scheme is vulnerable to independent packet losses. In a large network, it is highly possible that each subtree will suffer some degree of independent packet losses. If there are only a limited number of representatives, they cannot cover every subtree. Thus, any uncovered receiver may send NAKs upon detection of packet loss, causing sender's rate be throttled. Furthermore, the suppression method they use is also not efficient. In order to let representatives' feedback traversing the group and suppress others, the feedback timer at each

receiver is set based on the longest RTT of the group instead of each receiver's own RTT. Even so, since all feedback is multicast, there is at least one feedback received by everybody in the group for every packet. This high volume of feedback traffic is undesirable in a geographically large multicast group.

Mark Handley, et al presented a different approach for TCP-friendly reliable multicast congestion control in RMRG [9, 26]. They propose to use the TCP throughput approximation equation [18] at each receiver to estimate its current reception rate and feed this information back to the source. The source then simply chooses the slowest rate and adjusts its transmission rate. Since the throughput approximation is based on steady state analysis, and the receivers have to calculate their loss fractions over a relative long period (tens of RTT) in order to achieve an accurate rate estimation. The idea behind [9] is to relax the responsiveness as compared to TCP and only achieve TCP-fairness over a long term average. The robustness and reliability of the throughput equation in a real network is still under investigation. However, given the different prototypes of TCP implementations and the heterogeneity of network environment, a single form of equation will be hard to model all these situations accurately. This puts the scalability and reliability of any protocols based on such equation in doubt. Additionally, the impact of low responsiveness on TCP performance also needs further study and investigation. One possibility is that since TCP responds to congestion much faster, it may be starved of bandwidth, while the scheme in [9] may not detect any serious congestion at all. The same mistaken perception can happen in rediscovery of the bandwidth availability as well. The trade-off between responsiveness and accuracy therefore needs to be carefully examined.

Both of the above works use a rate-based adaptation algorithm, while in [7], Golestani investigated the fairness relationship between window-based and rate-based schemes. They proposed a hierarchical approach of window-based congestion control. The idea is to keep a distinct window for every receiver so as to carry a sustainable throughput adapted to the slowest receiver. This hierarchical approach can also be used separately for feedback consolidation and RTT estimation. However, as the writing of this paper, we are not aware of any formal evaluation or simulation of the proposed scheme in [7].

8 Conclusions and Future Work

In this paper, we have presented a multicast congestion control scheme based on dynamically identifying the congested subtrees in the network and adjusting data rate according to feedback from these subtrees. Suppression and aggregation of feedback on both receiver and sender side

avoids the implosion problem, still the agent based model ensures that feedback is propagated to the sender in a timely manner. The simulation shows promising results in scalability and in achieving TCP-fairness.

Although we have focused our study in an end-to-end based approach, the same model can be extended to other multicast congestion control framework as well. For example, if the routers are to keep states for a multicast session, then the router that is mostly congested can act as an agent sending feedback to the end nodes, possibly using the ECN bit [5]. In this case, the capacity is not necessarily be strictly proportional to the square root of loss rate and RTT, but can be calculated in accordance with the queue management algorithms or with additional policies. In our scheme, we have adopted the AIMD algorithm for it is proven to be safe to the network. Additionally, for fairness reasons, we have chosen the increase and decrease factor of the AIMD algorithm to be same as that of TCP. However, we found that the two phases of TCP's congestion control makes it very hard for any other type of congestion control to be completely fair with it. The congestion avoidance of TCP during which the TCP congestion window reduces to one segment size is very conservative and usually results in very poor performance. On the other hand, the reducing to half semantic results in wide oscillations in transmission rate. These features of TCP are developed during its historical design, whether any new protocol should conform to these design simply for the sake of pure fairness is an open question. There is one possibility of deploying multicast as a separate flow class in the differentiated service architecture. Then, the flow adaptive range is limited to be smaller and the increase and decrease factor can then be changed to smaller values in order to achieve smoother and healthier rate adaptation and better link utilization.

References

[1] A. Basu and S. J. Golestani. Estimation of Receiver Round Trip Times in Multicast Communications. *RMRG Meeting*, December 1998.

[2] S. Bhattacharyya, D. Towsley, and J. Kurose. The Loss Path Multiplicity Problem in Multicast Congestion Control. Technical report, University of Massachusetts, August 1998.

[3] L. Brakmo, S. O'Malley, and L. Peterson. TCP Vegas: New techniques for congestion detection and avoidance. In *ACM SIGCOMM*, 1994.

[4] D. DeLucia and K. Obraczka. A Multicast Congestion Control Mechanism for Reliable Multicast. In *Proc. IEEE INFOCOM*, 1997.

[5] S. Floyd and K. Fall. Promoting the Use of End-to-End Congestion Control in the Internet. *IEEE/ACM Transactions on Networking*, August 1999.

[6] S. Floyd, V. Jacobson, C. Liu, S. McCanne, and L. Zhang. A Reliable Multicast Framework for Light-weight Sessions and Application Level Framing. In *Proc. ACM SIGCOMM*, 1995.

[7] S. J. Golestani. Fundamental Observations on Multicast Congestion Control in the Internet. In *Proc. IEEE INFOCOM*, 1999.

[8] M. Handley. An Examination of MBone Loss Distributions. *Presentation at the 3rd RMRG meeting*, February 1998.

[9] M. Handley and S. Floyd. Strawman Specification for TCP Friendly Reliable Multicast Congestion Control (TFMCC). *RMRG Meeting*, December 1998.

[10] http://www mash.cs.berkeley.edu/ns.

[11] V. Jacobson. Congestion Avoidance and Control. In *Proc. ACM SIGCOMM*, 1988.

[12] S. Kasera, J. Kurose, and D. Towsley. Scalable Reliable Multicast Using Multiple Multicast Channels. Technical Report 96-73, University of Massachusetts, October 1996.

[13] J. Mahdavi and S. Floyd. TCP-Friendly Unicast Rate-Based Flow Control. *Technical note sent to the end2end-interest mailing list*, January 1997.

[14] S. McCanne, V. Jacobson, and M. Vetterli. Receiver-driven Layered Multicast. In *Proc. ACM SIGCOMM*, 1996.

[15] T. Montgomery. A Loss Tolerant Rate Controller for Reliable Multicast. Technical report, NASA-IVV-97-011, 1997.

[16] A. Odlyzko. The Internet and Other Networks: Utilization Rates and Their Implications. Technical report, AT&T Research Labs, September 1998.

[17] V. Ozdemir, S. Muthukrishnan, and I. Rhee. Scalable, Low-Overhead Network Delay Estimation. In *Proc. IEEE INFOCOM*, 2000.

[18] J. Padhye, V. Firoiu, D. Towsley, and J. Kurose. Modeling TCP Throughput: A Simple Model and Its Empirical Validation. In *Proc. ACM SIGCOMM*, 1998.

[19] C. Papadopoulos, G. Parulkar, and G. Varghese. An Error Control Scheme for Large-Scale Multicast Applications. In *Proc. IEEE INFOCOM*, 1998.

[20] S. Paul, K. K. Sabnani, J. Lin, and S. Bhattacharyya. Reliable Multicast Transport Protocol (RMTP). In *Proc. IEEE INFOCOM*, 1996.

[21] V. Paxson. End-to-End Internet Packet Dynamics. In *Proc. ACM SIGCOMM*, 1997.

[22] I. Rhee, N. Ballaguru, and G. N. Rouskas. MTCP: Scalable TCP-like Congestion Control for Reliable Multicast. In *Proc. IEEE INFOCOM*, 1999.

[23] RMRG Meeting, http://www.east.isi.edu/RMRG/newindex.html, December 1998.

[24] T. Speakman, D. Farinacci, S. Lin, and A. Tweekly. PGM Reliable Transport Protocol. *Internet Draft*, August 1998.

[25] L. Vicisano, L. Rizzo, and J. Crowcroft. TCP-like congestion control for layered multicast data transfer. In *IEEE INFOCOM*, 1998.

[26] B. Whetten and J. Conlan. A Rate Based Congestion Control Scheme for Reliable Multicast. *Technical White Paper*, 1998.