# Congestion Control Performance of a Reliable Multicast Protocol

Dante DeLucia
HRL, LLC
3011 Malibu Canyon Road
Malibu CA 90265
email: dante@hrl.com

Katia Obraczka
USC Information Sciences Institute
4676 Admiralty Way Suite 1001
Marina Del Rey, CA 90292
email: katia@isi.edu

## Abstract

*This paper evaluates the congestion control performance of* Pseudofed*, a* **congestion-controlled***, reliable multicast transport protocol for bulk data transfer. Pseudofed's congestion control mechanism is based on the concept of* representatives*, a small, dynamic set of multicast group members. By reducing the congestion control problem to a bounded set of receivers, representatives allow the point-to-point congestion control model used by unicast protocols like TCP to scale to larger multicast groups. Other features that contribute to the scalability of Pseudofed's congestion control algorithm are: (1) attempting to distinguish between correlated and uncorrelated packet losses, (2) not requiring complete knowledge of the multicast group, and (3) not exchanging control communication with congestion-free subtrees.*

## 1. Introduction

The increasing popularity of group communication applications such as multi-party teleconferencing tools and information dissemination services has motivated the development of reliable multicast transport protocols layered on top of IP multicast for efficient multipoint data distribution. While TCP's point-to-point model treats multipoint data delivery as a collection of point-to-point flows, protocols using IP multicast avoid sending duplicate data repeatedly over the same network links.

The Internet relies on applications performing congestion control to react to network congestion and avoid congestion collapse. Most applications in use on the Internet employ TCP's congestion control algorithms [8]. To allow multicast protocols to be safely deployed on the Internet, it is imperative that they incorporate mechanisms for handling congestion. While many reliable multicast protocols have been proposed for the Internet, few of these protocols have considered congestion control.

This paper presents an evaluation study of the congestion control algorithm employed by *Pseudofed*, a congestion-controlled, reliable multicast transport protocol for bulk data transfer. Pseudofed resulted from incorporating the *representative-based* congestion control algorithm we describe below into the Multicast Dissemination Protocol (MDP) [11].

## 2. Congestion Control Algorithm

Our algorithm is based on the observation that in a multicast distribution tree a small set of bottleneck links cause the majority of congestion problems. The algorithm selects a small set of group *representatives* to represent the congested multicast subtrees. By concentrating congestion control efforts on the congested subtrees, representatives allow the source to apply a point-to-point congestion control model to large multicast groups.

Congestion is managed by combining packet drop and measured queueing delay delay. Queueing delay serves a twofold purpose. By itself, it allows us to react to congestion before packet losses occur. When combined with packet losses, it allows us to distinguish correlated and uncorrelated packet losses. If a packet loss occurs without detectable queueing, it is assumed to be uncorrelated. If queueing is detected, congestion avoidance measures are invoked. If a packet loss occurs in conjunction with measured queueing, congestion recovery is attempted by multiplicatively decreasing the transmission rate.

Probabilistic and representative suppression is used to control feedback generated by multicast group members. The suppression mechanism makes use of immediate feedback from dynamically selected representatives to allow loose constraints on feedback suppression timers. Consequently, the feedback timer protocol generates much less traffic by not requiring round-trip time (RTT) computation between all group members, or even between the source and all receivers.

## 2.1. Representatives, Data Source and Receivers

At any given time, a multicast group member can be acting either as a receiver, representative, or the data source.

**Representatives** Representatives play a key role in both feedback generation and congestion control efforts. Their feedback is immediate which allows the data source to respond to congestion quickly. Representatives are also instrumental in suppressing feedback from other receivers.

Figure 1 illustrates the concept of representatives using an arbitrary multicast distribution tree as example. Figure 1a shows the multicast tree with the source at the root of the tree and receivers at intermediate nodes and leaves. When our algorithm starts (Figure 1b), the representative set is empty and, since there is no indication of congestion, all participating receivers are eligible to become representatives. The source selects the receiver from which it receives feedback first (Figure 1c). Since receiver's feedback timers are initially set to an arbitrarily large value (currently, 1 second), the source will probably receive feedback from the closest receivers first.

When congestion is detected in the right subtree (Figure 1d), which is not yet covered by the current representative set, a new representative is selected to cover the newly congested subtrees (Figure 1e). Newly congested subtrees will likely generate more feedback when congestion is first detected since feedback control relies exclusively on probabilistic suppression until a representative is selected.

**Data Source** The source multicasts data at a variable rate. Based on feedback from the group, it adjusts the rate dynamically to avoid network congestion, while trying to make use of available network bandwidth.

The source is responsible for the representative selection process and for advertising the current representative set to the group. Based on feedback received, the source also computes the group's largest round-trip time (GRTT). We describe how the GRTT is computed in Section 2.2.

**Receivers** Upon receiving a data packet, receivers (including representatives) send feedback to the source in the form of positive and negative congestion indicators. Although these can be thought of as ACKs and NACKs, we will use the terms Congestion Clear (CC), and Congestion Indication (CI) to avoid confusion with reliability mechanisms. Unlike feedback from representatives, receiver feedback is subject to suppression. We discuss our suppression mechanism in Section 2.2 below.

CCs are used to detect congestion as it builds in the network before packet drops occur. CIs provide feedback in the case of packet drops, indicating that congestion has occurred and has been detected by the receiver.

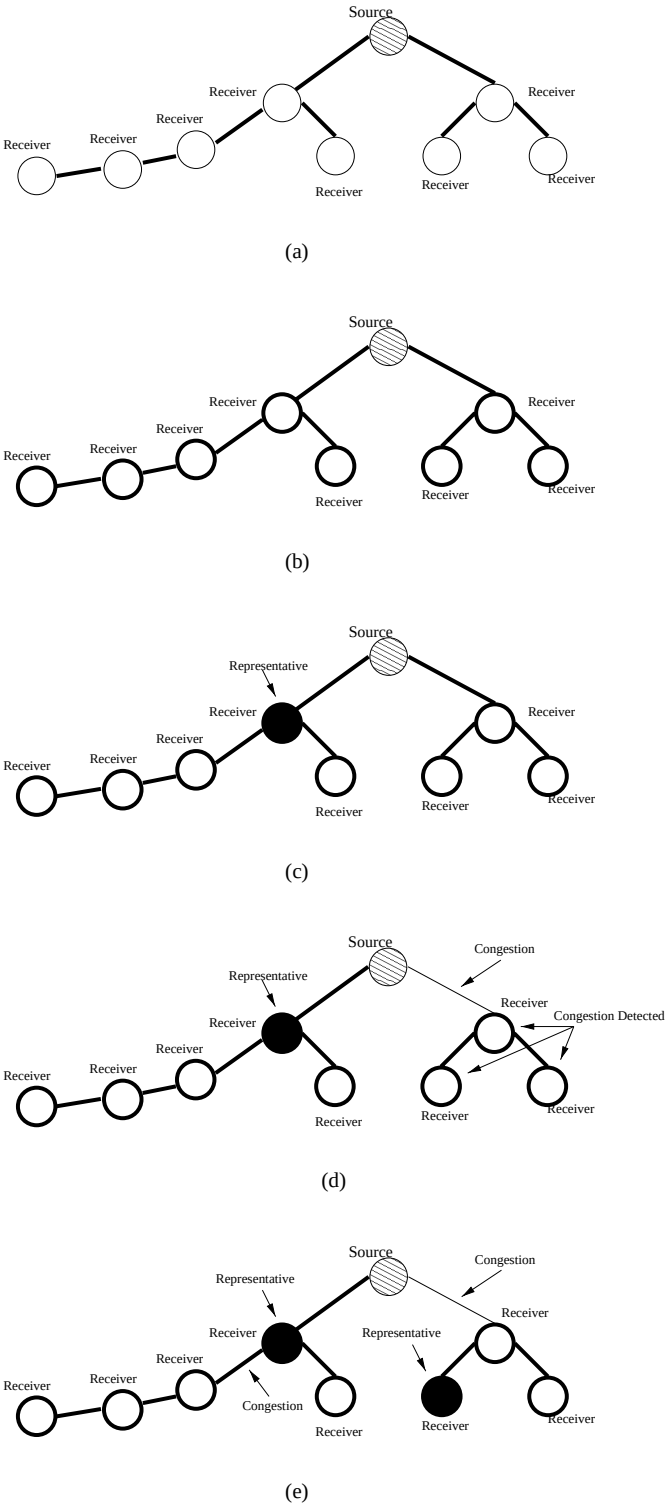



**Figure 1. An example multicast tree illustrating the concept of representatives.**

## 2.2. Feedback Generation and Control

The congestion control algorithm can be divided in two parts: (1) feedback generation and control and (2) source rate adjustment. Feedback generation and control is performed by representatives and receivers, while rate adjustment is performed at the source based on feedback from the group.

**Suppression** Suppression refers to the canceling of scheduled feedback in response to the receipt of another receiver's feedback. While non-representative receivers' feedback is subject to suppression, feedback from representatives is immediate and therefore not subject to suppression. Probabilistic suppression complements representative suppression: it limits feedback from multicast subtrees not yet covered by the current representative set.

Non-representative feedback is scheduled over a random interval using *suppression* timers. If suppression timers are set long enough, feedback from representatives can traverse the entire group suppressing any non-representative feedback. Clearly, there is a tradeoff when setting suppression timers: the longer they are, the more feedback is suppressed. However, long suppression timers mean that feedback from receivers experiencing congestion in parts of the multicast tree not yet covered by the current representative set will take longer to get to the source.

**Suppression Timers** Suppression timers consist of two components. The first is a deterministic *wait* period, and the second is a random *suppression* interval. The purpose of the wait period is to allow time for representative feedback to traverse the group thereby suppressing feedback from non-representatives. The purpose of the suppression interval is to space out feedback responses and allow probabilistic suppression to reduce the amount of feedback.

The wait and suppression intervals, which are parameters used by our simulator, are set as a percentage of the estimated GRTT. The tradeoffs between various choices of wait and suppression intervals were studied in [6].

**GRTT Measurement** GRTT is maintained by keeping a table the longest RTTs heard. An entry will stay in the table for a specified period of time before it it timed out. The GRTT is the largest value in the table at any given time.

**Representative Selection** The source selects representatives based on feedback received. Over the life of a multicast group, the representative set changes to reflect congestion as it moves to different parts of the multicast distribution tree.

At startup, any receiver providing feedback is eligible for selection as a representative. This means that a receiver sending a CC may be selected as representative as long as no CI is received. As network conditions change, feedback received by the source is used to update the representative set.

Once the representative set is non-empty, if all feedback comes from current representatives, no new representative is selected. If the source receives feedback from non-representative receivers, these receivers are eligible to become representatives.

After a full representative set has been obtained, only CIs qualify a receiver for selection as a representative. In addition, when the representative set is full and a new representative is selected, an existing one must be ejected from the current set. To select a candidate for ejection from the representative set, we currently use a LRU algorithm based on the time since the last CI was sent. This criteria is based on the assumption that a representative that has not sent a CI recently is not currently experiencing congestion. To avoid excessive fluctuation of the representative set, the current implementation of our congestion control algorithm requires that representatives stay active for at least 4 GRTTs.

The source multicasts the current representative set to the group after a change in the set.

## 2.3. Rate Adjustment

The rate adjustment mechanism consists of two phases. The first is the "slow-start" phase, conceptually borrowed from TCP. As soon as congestion is detected, the algorithm reverts to its "congestion avoidance" phase.

Initially, the source transmission rate is set to a predefined minimum rate, $rate_{min}$. The transmission rate is never allowed to drop below this value. This minimum rate is administratively defined.

**Slow-Start** The slow-start phase attempts to quickly find the available network bandwidth. Initially the transmission rate is set to $rate_{min}$ and is increased by $\frac{1}{2}$ every $time\_interval$, where $time\_interval$ is a multiple of the current GRTT, or $time\_interval = I * GRTT$. Currently, $I$ is set to 4. Clearly, the higher $I$, the slower the rate is increased. The current rate increase is on the conservative side and attempts to avoid packet loss by increasing the rate too quickly.

Slow-start is terminated when a CI is received, or when congestion is detected using information gleaned from CCs. In either case, the transmission rate is reduced by half, and the algorithm switches to the congestion avoidance phase. We explain how we derive congestion information from CCs below.

**Congestion Detection** When the source receives a representative feedback message, it examines the message to

determine if congestion exists in the network.

We use a *delay-based* congestion metric inspired by TCP Vegas' congestion avoidance mechanism [1]. The delay-based metric measures the amount of data queued in the network. The number of packets queued in the network, $packets_{queued}$, is given by

$$packets_{queued} = \frac{(rtt_{current} - rtt_{min}) \times rate}{packet\_size} \quad (1)$$

where $rtt_{current}$ and $rtt_{min}$ are the worst and minimum RTT measured by the source so far, $rate$ is the current transmit rate, and $packet\_size$ is the size of a data packet.

If $packets_{queued} > \beta$, the source has an indication that congestion is building, and decreases its transmission rate. Otherwise, if $packets_{queued} > \alpha$ the rate is increased. There is an obvious tradeoff in setting $\beta$: if it is set too low, the congestion avoidance algorithm becomes too conservative. On the other hand, if it is set to a large number, congestion can lead to packet drops. We currently set $\alpha$ and $\beta$ to 1 and 3 packets respectively based on the TCP Vegas experiments reported in [1].

**Adjusting the Rate** We perform fine-grain rate adjustment, i.e., adjustment after each packet transmission, by controlling the rate indirectly via an *acceleration*. Each time a feedback packet is received, the acceleration is adjusted. Each time a packet is sent, the rate is adjusted according to the transmission time and the acceleration.

If the acceleration is positive, the rate increases. If it is negative, the rate decreases. CIs decrease the acceleration, while CCs increase it. A CI combined with a queueing indicator will immediately set the acceleration to a negative value. This allows independent handling of multiple CIs. CCs will have the effect of gradually overriding any negative acceleration due to CIs. The more CCs that are received, the faster the negative acceleration is overcome.

In designing a rate adjustment mechanism for our congestion control protocol we chose to take a practical approach and use feedback from our simulation experiments to tune the general linear increase-multiplicative decrease rate adjustment approach. The resulting rate adjustment mechanism works as follows.

When a **CI combined with queueing indication** is received, the acceleration is set to decrease the rate by half in the next $time_interval$, where $time\_interval = I * GRTT$. Like in the slow-start phase, $I$ is set to 4. The new acceleration is computed as follows:

$$accel = -\frac{rate}{I \times GRTT} \quad (2)$$

On receipt of CC indicators, the transmit rate is increased by $\delta_{rate}$ within the next GRTT if no queueing is detected. Since the source may receive multiple CC indicators, we want to increase by $\delta_{rate}$ only if every representative sends a CC. Consequently, in response to any one CC, we increase the acceleration by:

$$\delta_{accel} = \frac{\delta_{ratemax} \times \frac{rate}{GRTT} - accel}{representative\_set\_size} \quad (3)$$

In the case where queueing is detected by a CC, the acceleration should decrease, but not as sharply as in the case where a CI is received. A CC with queueing indication decreases the acceleration as follows:

$$accel = -I * \frac{\delta_{ratemax} \times rate}{GRTT} \quad (4)$$

## 3. Congestion-Controlled, Reliable Multicast

Pseudofed is a combination of the MDP framework [4] and the independently developed representative-based congestion control mechanism just described. Instead of the fixed rate used in MDP, the rate is controlled by the congestion control mechanism and is adjusted after each packet transmission.

### 3.1. The Multicast Dissemination Protocol (MDP)

The Multicast Dissemination Protocol (MDP) [11] provides a reliable multicast framework for file distribution. MDP evolved from the IMM image multicaster, a protocol used in disseminating satellite image files over the MBone.

The MDP sender fragments the file to be transmitted into a sequence of MDP maximum data units (MDUs) which are multicast to the group using the UDP/IP multicast suite. MDUs are sent at the transmission rate set by the sender application, which can also control the interval between file transmissions. MDP receivers assemble the received data units into the original file, which can be archived or processed using image viewers or text processing tools.

MDP's recovery mechanism works in rounds: after transmitting a file, the source asks receivers for retransmission requests. A receiver who has detected sequence number gaps schedules a retransmission request which gets multicast if no other receiver has requested the same retransmission. MDP also provides an operation mode in which receivers may request repairs at any time during the transmission of a file. The MDP source can also request ACKs from receivers, who respond with information about their current state.

MDP allows sites to join and leave the multicast group. New sites announce their presence by multicasting a join packet to the group. Currently, MDP does not have any mechanism to prevent late comers from requesting retransmission of old packets.

### 3.2. Congestion Control and MDP Interface

Pseudofed uses MDP lost packet reports as the basis for CC and CI information. Fields where added to MDP headers to compute round trip time. For CC, the lost packet reports simply report zero packets lost, but otherwise are the same.

## 4. Simulations

Internet multicast is in a state of flux, so we cannot assume that whatever conditions we study now will necessarily match the future state of the Internet multicast infrastructure. What we can do is study our protocol under a variety of network conditions and evaluate its performance.

In this section we present the results of simulation experiments we conducted to evaluate Pseudofed's performance. Simulations allow us to run controlled experiments with Pseudofed in larger networks and explore the various dimensions in the multicast congestion control design space. Our simulation parameters include different link bandwidths and propagation delays, different competing traffic patterns, various multicast group densities and representative set sizes. We simulated a variety of network conditions including highly congested networks where link bandwidths vary by an order of magnitude or more. We implemented Pseudofed in the NS [12] network simulator, which has been used in performance studies of TCP [7] and multicast protocols [13].

### 4.1. Experimental Setup

We employ two basic experimental setups. The first is a simple two-node, single-hop topology. This simple topology allows us to examine the basics of the rate adjustment algorithm operating in a one-to-one communication mode with and without TCP. The second setup consists of a number of more complex topologies in which the performance of the protocol under multicast conditions is studied.

**Topologies** In the two-node topology, the multicast and TCP sources were located at one node, and the receiver and sink at the other.

For the second set of experiments, we used Georgia Institute of Technology's Internetwork Topology Models (GT-ITM) [17, 3] tool to generate random topologies of up to 100 nodes [1].

In simulations, bandwidths of 5 and 100 Mbits/sec were assigned to stub-to-stub, and transit-to-transit links respectively. Stub-to-transit links used lower bandwidths (56 Kbits/sec, 128 Kbits/sec, and 1 Mbits/sec).

[1] These are reasonably large networks given the resources available.

For our experiments, we used 5 different randomly generated transit-stub topologies. The average propagation delay on stub-to-stub, transit-to-stub, and transit-to-transit links were 5, 20, and 100 milliseconds, respectively.

**Multicast Group Description** Multicast group members are placed at the stub nodes. In each of the five topologies used, the multicast source and receivers were randomly chosen among stub nodes. The multicast group density establishes the percentage of stub nodes that are members of the multicast group. In our simulations, we used multicast group densities of 25 and 75%.

**Cross Traffic** In the two-node topology, traffic consists of a TCP connection running concurrently with Pseudofed for 300 seconds.

In the transit-stub topology setup, the cross traffic patterns we use consist of a mix of uniformly distributed short and long TCP sessions. A short TCP session is created every 0.1 second on average and sends 10 Kbytes of data. Long TCP sessions send 100 Kilobytes of data and are created at a rate of 1 every 10 seconds. The total number of TCP sessions for our 180 second experiments were 1800 10K sessions, and 18 100K sessions. The TCP sources and sinks were randomly chosen from among the stub nodes in our topology.

Our goal in using cross traffic as background noise is twofold: (1) perturb the multicast flow to evaluate how quickly it reacts to congestion and how well it adapts to changes in bandwidth availability; (2) evaluate how well link bandwidth is utilized and shared with competing traffic: we want to ensure that the multicast flow does not starve competing traffic and vice-versa. We should also point out that in our simulations we do not model lossy links. Packet losses are solely due to congestion.

We used TCP-based cross traffic as a yardstick for our congestion control algorithm. TCP's congestion control mechanisms have been well tuned for an environment like the Internet, and showing that our algorithm behaves well

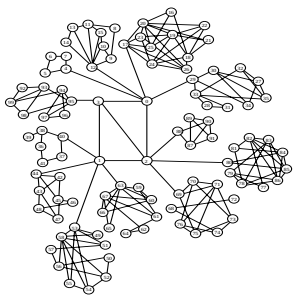


**Figure 2. Transit-Stub Topology**

when competing for bandwidth with a reasonably conservative algorithm is promising.

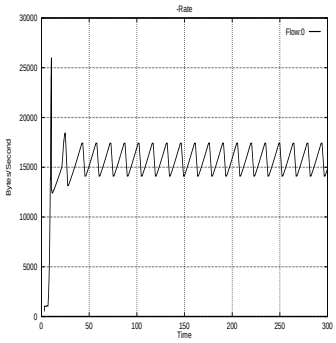
### 4.2. Simulation Results

**Two-Node Topology** These experiments evaluate Pseudofed's congestion control operating in a one-to-one communication mode competing for bandwidth against TCP flows. These runs illustrate the ability of the congestion control algorithm to find and maintain the maximum allowable rate and share bandwidth with competing TCP traffic.

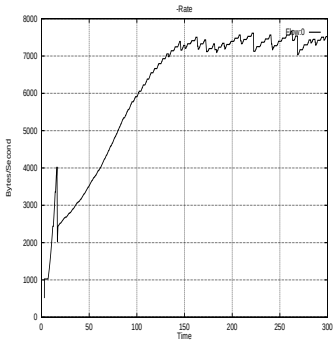
Figure 3(a) shows the source transmission rate over time when a single Pseudofed flow has no competition for bandwidth. The congestion control algorithm quickly finds the bottleneck bandwidth and then oscillates around it for the duration of the flow. The statistics in Table 1 confirm that Pseudofed's average throughput of 15,396 bytes/sec (approximately 123 Kbits/sec) closely approximates the link's physical bandwidth. Also note that no packets are dropped.

When competing with TCP (a single Pseudofed flow and single TCP session were started at the same time and run concurrently for the duration of the experiment), we observe from Table 1 that Pseudofed's average throughput drops to approximately 48 Kbits/sec, or less than $1/3$ of its throughput without TCP. This shows that the congestion control algorithm relinquishes bandwidth in the presence of competing traffic. In fact, it lets TCP grab around $2/3$ of the link bandwidth, on average.

Figure 3(b) shows that half-way through the experiment, Pseudofed's rate stabilizes around 57.6 Kbits/sec. Note that both Pseudofed and TCP drop ratios are still low.




(a) Transmission rate of a single Pseudofed flow.




(b) Transmission rate of Pseudofed flow competing with one TCP flow.

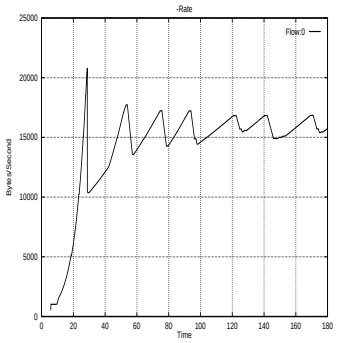
**Figure 3. Two-node topology over a 128 Kb/s link.**

| | Flow | Throughput | Drop Ratio |
|---|---|---|---|
| Pseudofed Only | Pseudofed | 15,396 | 0.0 |
| TCP Only | TCP | 15,943 | 0.016 |
| Combined | Pseudofed | 6,083 | 0.076 |
| | TCP | 10,467 | 0.039 |

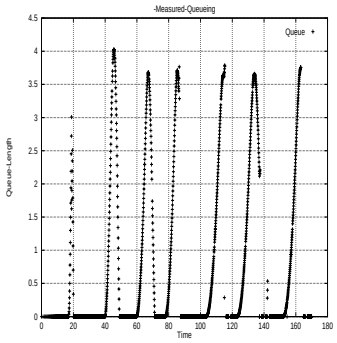
**Table 1. Average throughput (bytes/sec) and byte drop ratio over two-node topology.**

**Transit-Stub Topologies** We use the transit-stub topology scenarios to evaluate Pseudofed's operation in multicast mode. As before, we tested the congestion control algorithm with and without TCP cross traffic. While the experiments without cross traffic evaluate how well Pseudofed performs on uncongested links, the experiments with TCP cross traffic study Pseudofed's behavior in the presence of congestion. The goal of these simulations is to ensure that the congestion control algorithm detects congestion buildup and backs off the transmission rate. Yet, the congestion-controlled multicast flow should still grab a reasonable portion of the bandwidth.

**Single Multicast Flow** The first row in Table 2 shows the average throughput and byte drop ratio of Pseudofed with no competing TCP traffic. These averages were computed over five runs using randomly-generated transit-stub topologies with 128 Kbits/sec bottleneck links and 75% multicast group density. Pseudofed's average throughput of approximately 126 Kbits/sec is very close to the optimum bandwidth yet causing very few packet drops. The graphs in Figure 4 show the results of a sample run. They confirm that the congestion control algorithm is able to find the bottleneck link bandwidth (in this case, it oscillates around 128 Kbits/sec) reasonably quickly. Although the bottleneck link utilization is kept reasonably high, multicast packet drops are quite low (16 packet drops out of 1629 packets sent, or a total of 132 multicast data drops out of 141,298 packet




(a) Transmission rate.




(b) Measured Queueing (in packets).

**Figure 4. Single Pseudofed flow over transit-stub topology.**

hops). From Figures 4(a) and (b), we observe that there is a close relationship between the measured queueing and the variations in Pseudofed's transmission rate: every increase/decrease in measured queueing is quickly followed by a decrease/increase in the rate.

We observe the same trend for 25% multicast group densities, as well as different bottleneck link bandwidths.

Next, we evaluate the behavior of Pseudofed's congestion control algorithm when competing with TCP cross traffic. From the summaries in Table 2, we observe that both Pseudofed and TCP acquire a fair share of the bandwidth. Yet, the average drop ratios for both TCP and Pseudofed are kept reasonably low. The reported TCP throughput is computed as the average throughput over all TCP connections. Pseudofed's transmission rate slow-starts and achieves approximately 7.2 Kbytes/sec 30 seconds into the run. The congestion control algorithm then switches to congestion avoidance mode (due to the receipt of a CI) and the rate drops sharply to around 3.7 Kbits/sec, and then gradually increases.

| | Flow | Throughput | Drop Ratio |
|---|---|---|---|
| Pseudofed | Pseudofed | 12,567 | 0.009 |
| TCP Only | TCP | 15,197 | 0.036 |
| Pseudofed/TCP | Pseudofed | 8,175 | 0.013 |
| | TCP | 8,477 | 0.155 |

**Table 2. Average throughput (bytes/sec) and byte drop ratio over 5 randomly-generated transit-stub topologies.**

**Spatially Uncorrelated Loss** To test Pseudofed's response to spatially uncorrelated loss, we use a transit-stub topology. Instead of having the access to the transit network be the bottleneck, we make the intra-stub links the bottleneck links. This simulates a network in which losses occur primarily at the edges. We set the transit links to have bandwidths of 100Mbits/s, and the stub links to 128 Kbits/s.

Packet drops occur on 85% of the links in the 75-member multicast tree. Despite this large, uncorrelated loss ratio, we still obtain acceptable performance (see Figure 4.2). Table 3 shows that the average throughput of the non-backbone TCP traffic is not seriously impacted by the presence of the multicast flow.

| | TCP Throughput | TCP Drop Ratio |
|---|---|---|
| Pseudofed/TCP | 232,130 | 0.16 |
| TCP Only | 258,092 | 0.12 |

**Table 3. Spatially uncorrelated losses: average TCP throughput (bytes/sec) and byte drop ratio.**

## 5. MBone Experiments

Experiments on the MBone allow us to evaluate Pseudofed in yet another challenging environment that is difficult to simulate. The MBone is a virtual multicast network with multicast capable subnets connected via unicast tunnels. Traffic on the MBone consists primarily of fixed-rate video and audio sessions with no congestion control. Consequently, parts of the MBone suffer severe congestion with no relief via congestion control. This environment could be considered extremely artificial and subject to radical change in the future.

Although in our simulations we have studied Pseudofed under several network environments, MBone experiments

| | |
|---|---|
| ISI Loss Ratio | 0% |
| UCSB Loss Ratio | 0% |
| GaTech Loss Ratio | 20% |
| Average Sending Rate | 19,181 bytes/sec |

**Table 4. Statistics for MBone experiment.**

give us a reality check. Several experiments were carried out on the MBone as proof of concept.

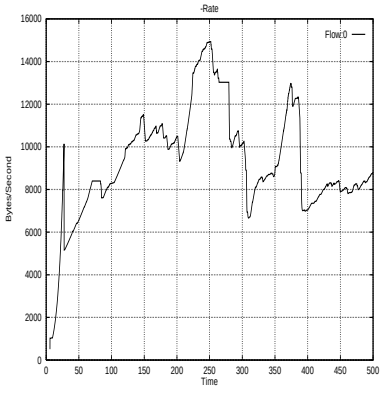
For these preliminary experiments, we used a small, geographically diverse group of sites located at: USC, ISI, UCSB, and Georgia Tech. While this small group does not begin to exercise scalability issues, it does help exercise some of the uncorrelated loss problems that occur in the MBone.

The experiment consisted of sending a single large file of 5,590,335 bytes from USC to the other three sites. Table 4 summarizes some statistics for the experiment.

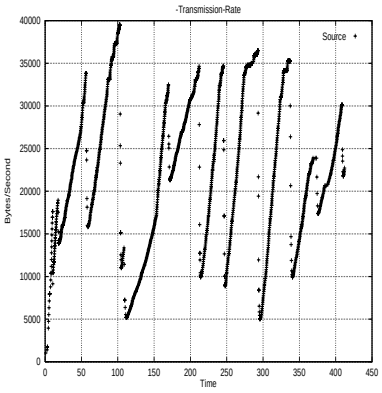
As can be seen from the graph in Figure 5(b), the GRTT estimates fluctuate considerably. A closer examination of the queueing estimates (Figure 5(c)) shows how closely these estimates are related to the GRTT estimates. At the discontinuities in the GRTT estimates, the queueing estimates change. When the GRTT drops drastically, the queueing estimates change enough to cause the rate to decrease. A more stable method of GRTT would probably give a more stable rate, since the rate adjustment is dependent upon stable queueing estimates.

## 6. Related Work

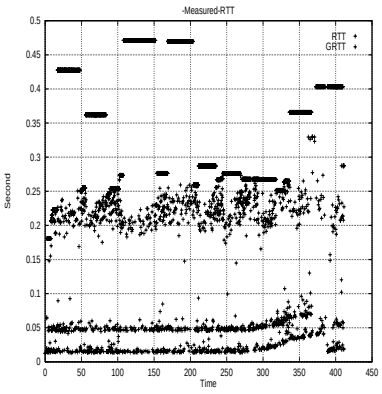
Reliable multicast has been a topic of intense research and development efforts over the past couple of years. Both the Internet Engineering and Internet Research Task Forces (IETF and IRTF) have been heavily involved in coordi-



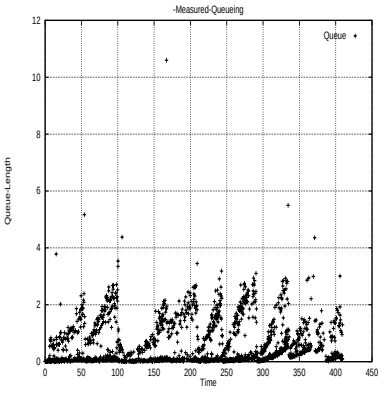

(a) Transmission rate.




(b) Transmission Rate.




(c) Round Trip Time.




(d) Measured Queueing (in packets).

**Figure 5. MBone experiment**

nating multicast transport protocol research, development, standardization, deployment, and technology transfer activities. Surveys of the state-of-the-art of reliable multicast include [10, 14, 9, 5, 16, 15].

Recent work on multicast congestion control has been carried out in the context of the IRTF Reliable Multicast Working Group (RM-WG). References to multicast control research efforts as well as other initiatives under the RM-WG can be found in [2].

## 7. Future Work

We have shown that Pseudofed can co-exist with a single TCP connection in a simple network, and that it can operate in more complex, heavily congested networks. Further simulations need to be conducted to investigate further to find whether there are scenarios in which unfairness occurs with either TCP or Pseudofed. While, overall network performance characteristics are encouraging, more attention needs to be given to the performance of individual flows and how they compete on congested links.

To date we have only performed simple Mbone experiments as a sanity check on our algorithm. More extensive testing needs to be performed to either validate our simulations or identify weaknesses.

The rate adjustment algorithm used in Pseudofed needs to be more sensitive to link bandwidth. While it can be tuned to perform well for a particular target bandwidth, it needs to be able to automatically tune itself for different bandwidths. This tuning will probably be a function of the GRTT, packet size, and current source transmission rate.

## 8. Conclusions

Our simulations studies of Pseudofed's performance under a simple two nodes topology and heavily congested transit stub topologies show that it can use available bandwidth, yet still coexist with TCP traffic. Pseudofed can also perform well even with uncorrelated loss in the multicast tree.

Simulation results show that Pseudofed's congestion control algorithm responds to congestion in a timely fashion, yet makes use of available bandwidth. Simulations also show that Pseudofed can co-exist with TCP: it backs off in the presence of TCP traffic but still keeps a reasonable share of the bandwidth. Even in networks with high uncorrelated loss, Pseudofed manages to obtain a reasonable share of the network bandwidth while allowing general TCP traffic a reasonable portion of the bandwidth and without incurring high packet losses. As proof of concept, we conducted experiments with Pseudofed on the MBone. These preliminary MBone experiments demonstrate Pseudofed's ability to operate in a diverse, uncontrolled network environment.

We anticipate that its versatility and ability to operate under adverse network conditions will allow Pseudofed's congestion control algorithm to be employed in a wide range of multicast environments.

## References


[1] L. Brakmo, S. O'Malley, and L. Peterson. TCP Vegas: New techniques for congestion detection and avoidance. *1994 ACM SIGCOMM Conference*, pages 24–35, May 1994.

[2] M. by Mark Handley. The reliable multicast research group. Available from http://www.east.isi.edu/rm, 1997.

[3] K. Calvert and E. Zegura. Georgia tech internetwork topology models. http://www.cc.gatech.edu/fac/Ellen.Zegura/graphs.html.

[4] W. Dang and T. Nielsen. The imm protocol. On-line source code from ftp://ftp.hawaii.edu/paccom/imm-3.3, May 1994.

[5] B. DeCleene, S. Bhattacharaya, T. Friedman, M. Keaton, J. Kurose, D. Rubenstein, and D. Towsley. Reliable multicast framework (rmf): A white paper. March 1997.

[6] D. DeLucia and K. Obraczka. Multicast feedback suppression using representatives. *Proc. of the IEEE Infocom'97, Kobe, Japan*, April 1997.

[7] K. Fall and S. Floyd. Simulation-based comparisons of tahoe, reno, and sack tcp. *ACM Computer Communications Review*, July 1996.

[8] V. Jacobson. Congestion avoidance and control. *ACM SIGCOMM 88*, pages 273–288, 1988.

[9] J. Knoght. Multicast transport protocols. http://hill.lut.ac.uk/DS-Archive/MTP.html, 1997.

[10] B. Levine and J. Garcia-Luna-Acceves. A comparison of known classes of reliable multicast protocols. *Proceedings of the International Conference on Network Protocols (ICNP)'96*, October 1996.

[11] J. Macker and W. Dang. The multicast dissemination protocol (mdp) framework. Internet Draft, Internet Engineering Task Force, draft-macker-mdp-framework-00.txt, November 1996.

[12] S. McCanne. ns - LBNL network simulator. Available from http://www-nrg.ee.lbl.gov/ns/.

[13] S. McCanne and V. Jacobson. Receiver-driven layered multicast. *1996 ACM Sigcomm Conference*, pages 117–130, August 1996.

[14] T. Montgomery. Reliable multicast links. http://research.ivv.nasa.gov/RMP/links.html, October 1997.

[15] K. Obraczka. Multicast transport mechanisms: A survey and taxonomy. *IEEE Communications Magazine*, January 1998.

[16] Z. Whang, J. Crowcroft, C. Diot, and A. Ghosh. Framework for reliable multicast application design. *Proceedings of the HIPPARCH'97*, 1997.

[17] E. Zegura, K. Calvert, and S. Bhallacharjee. How to model an internetwork. *Proc. of the IEEE Infocom'96, San Francisco, CA*, April 1997.