Fairness Evaluation Experiments for Multicast Congestion Control Protocols

Karim Seada, Ahmed Helmy
Electrical Engineering-Systems Department
University of Southern California, Los Angeles, CA 90089
{seada,helmy}@usc.edu

Abstract - **Fairness to current Internet traffic, particularly TCP, is an important requirement for new protocols in order to be safely deployed in the Internet. This specifically applies to multicast protocols that should be deployed with great care. In this paper we provide a set of experiments that can be used as a benchmark to evaluate the fairness of multicast congestion control mechanisms when running with competing TCP flows. We carefully select the experiments in such a way to target specific congestion control mechanisms and to reveal the differences between TCP and the proposed multicasting protocol. This enables us to have a better understanding of the proposed protocol behavior and to evaluate its fairness and when violations can happen. To clarify our experiments we carry them on a single-rate case study protocol, *pgmcc*, using NS-2 simulations. Our analysis shows the strengths and potential problems of the protocol and point to possible improvements. Several congestion control mechanisms are targeted by the experiments such as timeouts, response to ACKs and losses, independent and congestion losses effect. In addition, we evaluate multicast mechanisms such as the effect of multiple receivers, group representative selection, and feedback suppression when there is network support.**

I. INTRODUCTION

Congestion control is a major requirement for multicast transport protocols in order to be safely deployed in the Internet [10]. It is believed that the lack of deployable and well-tested multicast congestion control mechanisms is one of the factors inhibiting the usage of IP multicast [18]. These mechanisms have to be fair to current Internet traffic.

In this paper we consider evaluating the fairness of multicast congestion control protocols, by providing a set of selected experiments and scenarios that target specific congestion control mechanisms. This facilitates a better understanding of these protocols in order to assess their safety and their effects on TCP. Fairness analysis is important for any kind of protocols that compete with TCP.

Our approach attempts to relate overall protocol behavior to individual protocol mechanisms by evaluating carefully selected scenarios. In our experience this often points to possible mechanistic modifications to improve the protocol performance. In this paper, we arrive at these scenarios based on our intuition and understanding of the protocol mechanisms. In addition, we are developing a methodology [14], based on the STRESS framework [8][9], to systematize this scenario selection process.

To clarify our experiments we have chosen, as a case study, a scheme called *pgmcc*. pgmcc [13] is a single-rate multicast congestion control scheme that is designed to be fair with TCP. We apply the experiments using NS-2 simulations. Our analysis shows the strengths and potential problems of the protocol and point to possible improvements. Some scenarios reveal TCP unfriendly behavior, due to high losses or feedback suppression. Also, poor performance, due to group representative switch has been observed.

The rest of this paper is outlined as follows. In Section 2 we provide an overview of multicast congestion control and pgmcc. In Section 3 we explain our experiments and the motivation behind them. In Section 4 we show the simulation results and analysis of the case study. Conclusions are presented in Section 5.

II. MULTICAST CONGESTION CONTROL

The design of a MCC (Multicast Congestion Control) protocol that provides high performance, scalability, and TCP-friendliness is a difficult task that attracts a lot of research effort. MCC can be classified into two main categories: single-rate and multi-rate. Single-rate has a limited scalability because all receivers must receive data at the same (slowest receiver) rate. It also suffers from feedback implosion problem and drop-to-zero problem [2] (where the rate degrades significantly due to independent losses by a large number of receivers). Multi-rate, where different receivers can receive at different rates, is more scalable but has other concerns such as the complex encoding of data, possible multiple paths in the layered approach, and the effects of receivers joining and leaving layers. TCP-friendly MCC can be classified into window-based and rate-based. Window-based has a similar congestion window control as TCP, while rate-based depends on the TCP throughput equation [11] for adjusting the transmission rate [7][17].

Another possible classification for single-rate protocols is whether they are representative-based or not. Non-representative-based protocols solve the scalability problems using some aggregation hierarchy. This requires complex building of the hierarchy and may need network support. The performance is still limited and [3] shows that even without losses, small variations in delay can cause fast performance degradation with the increase in number of receivers. Representative-based protocols provide a promising emerging approach to solve the scalability problems, where a small dynamic set of receivers is responsible for providing the feedback [4][5]. The main challenge is the dynamic selection of a good set of representatives in a scalable and efficient manner with appropriate reaction to changes in representatives. This still needs further investigation. Examples of single-rate representative-based protocols are *pgmcc* (window-based) [13] and *TFMCC* (rate-based) [18].

We will use pgmcc as our case study example, and in the rest of this section we will provide a brief description of pgmcc.

pgmcc [13] is a single-rate MCC scheme that is designed to be TCP-friendly. To achieve fast response while retaining scalability, a group representative called the *acker* is selected and a tight control loop is run between it and the sender. It is called the acker because it is the receiver that sends ACKs. Other receivers can send NACKs when they lose packets, if a reliable transport protocol is used[1]. pgmcc is used to implement congestion control in the PGM protocol [16].

The acker is the representative of the group. It is chosen as the receiver with the worst throughput to ensure that the protocol will be TCP-friendly. A window-based TCP-like controller based on positive ACKs is run between the sender and the acker. The feedback in pgmcc is provided in receiver reports that are used by the sender to estimate the throughput. They are embedded into the NACKs and ACKs and contain the loss rate and information for computing an estimate for the round trip time (RTT) of the sending receiver. There is a field in the ACK called the bitmask, which indicates the receive status of the most recent 32 packets and is included to help the sender deal with lost and out-of-order ACKs.

The most critical operation of pgmcc is the acker election and tracking. The sender selects the receiver with the worst throughput as the acker. When another receiver with worse throughput sends a NACK, an acker change may occur. The sender computes throughput from receivers' feedback and the simplified TCP-like formula: $T \alpha 1 / RTT\sqrt{p}$ where T is the throughput, RTT is the round trip time estimate and p is the loss rate [11]. For more details about pgmcc see [13].

III. Experiments Details and Motivation

In this section we discuss the experiments details and the motivation behind them. Each subsection contains a set of related experiments.

A. Experiment Set 1: Window and Timeouts

The first set of experiments contains simple topologies to compare the MCC protocol to different flavors of TCP in simple cases. The flavors are Reno, New-Reno, and SACK [6]. This comparison helps us understand the behavior of the protocol and the subtle differences between it and TCP. Two congestion control issues are targeted by this comparison: (1) reaction to losses and ACKs with its effect on the window size, (2) retransmission timeouts. TCP Reno is still the most widely deployed flavor in the Internet, but recent statistics show that TCP New-Reno and TCP SACK deployment is increasing [12]. New-Reno and SACK solve performance problems of TCP in case of multiple-packet loss in a window and they reduce the number of timeouts. When multiple packets are lost from a single window of data, New-Reno and SACK can recover without a retransmission timeout. With Reno and New-Reno at most one dropped packet is retransmitted per round-trip time, while SACK does not have this limitation [6]. This response to losses and ACKs has a major impact on the window size, and consequently on the fairness. According to [11] timeouts also have a significant impact on the performance of TCP Reno and they constitute a considerable fraction of the total number of loss indications. Measurements have shown that in many cases the majority of window decreases are due to timeouts, rather than fast retransmits. This experiment highlights the protocol policy in handling ACKs and timeouts, and which flavor it is closer to.

B. Experiment Set 2: Diverse Losses and Delay

This set of experiments addresses the effect of having multiple receivers with different losses and delay. We consider both independent and correlated (due to congestion) losses. The throughput of the MCC protocol when the receivers have different combinations of delay and loss rates (e.g. high loss, low delay vs. low loss, high delay) is compared to the competing TCP flows. There are several objectives behind this comparison: First, better understanding of the effect of losses, retransmissions, and delays with multiple receivers. Second, many MCC protocols use a TCP throughput equation to model the TCP behavior. This set of experiments evaluates the accuracy of the used equation. Third, the reaction to independent and congestion losses can show some of the protocol characteristics.

C. Experiment Set 3: Feedback Suppression

Most MCC protocols depend on the receivers' feedback in making decisions. Some multicast transport protocols have network support (e.g. by routers) to improve their performance. This support is normally in the form of feedback aggregation or suppression to avoid problems as ACK and NACK implosion. In this part we consider experiments to test the effect of feedback suppression on fairness. The experiments consist of topologies with critical receivers having their feedback suppressed. The definition of critical receivers depends on the protocol as will be shown in the next section. Feedback suppression affects the accuracy of decisions based on feedback. These experiments investigate the tradeoff between the amount of feedback and the correctness of decisions or computations.

D. Experiment Set 4: Group Representatives

Several MCC protocols use the idea of representatives to achieve scalability. Feedback is normally provided only by these special receivers. An important task is the selection and changing of the representatives. The experiments here target this operation by having configurations containing multiple sets of receivers that can be selected as representatives and having scenarios that trigger the changing between them. The aim of the experiments is to study the effect of these changes on the overall protocol operation and on its fairness to TCP.

[1] pgmcc can be used with both reliable and non-reliable transport protocols. Non-reliable protocols will also need to send NACKs from time to time for congestion control purposes.

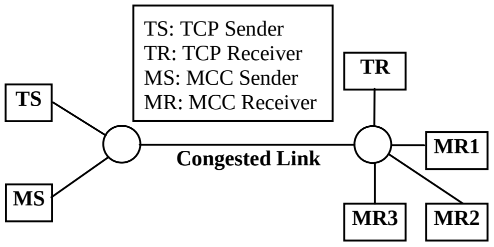

Fig. 1: TCP session competing with MCC session over a bottleneck link

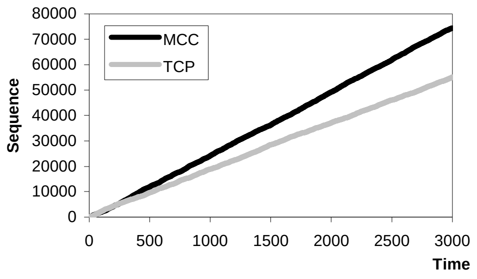

Fig. 2: Throughput of pgmcc vs. TCP Reno (*f*=74%)

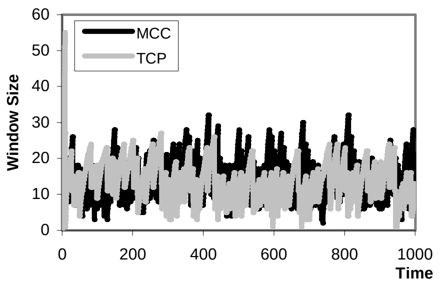

Fig. 3: Window size comparison of pgmcc and TCP Reno

IV. Case Study Simulations

In this section we apply our experiments to a case study scheme 'pgmcc'. We show the results obtained using the NS-2 simulator [1] to evaluate pgmcc and analyze its fairness with regard to TCP. Both TCP Reno and SACK are examined and they provide close results and similar conclusions (except for the first experiment, as will be shown). The source models used in the simulation are FTP sources with packet size of 1400 bytes. The links have propagation delay of 1ms, and bandwidth of 10Mb/s, unless otherwise specified. The queues have a drop-tail discard policy (RED was also examined and the results are similar) and FIFO service policy, with capacity to hold 30 packets. In the graphs we show the sequence numbers sent by the sender vs. the time. This has the same effect as showing the throughput. We provide a fairness metric f as the ratio between TCP and pgmcc throughput[2]. More details about the simulations are provided in [15].

A. *Experiment Set 1: Window and Timeouts*

In the first experiment we use the topology shown in Fig. 1 to test the fairness of pgmcc with the different TCP flavors in a very simple case, where we have only a single TCP session competing with a pgmcc session over a bottleneck link (500Kb/s, 50ms). pgmcc has a number of identical receivers, so anyone of them could be the acker.

Starting with TCP Reno and comparing the throughput of the TCP sender with the pgmcc sender we find in Fig. 2 that pgmcc is not fair to TCP Reno[3]. The reason for this behavior can be interpreted if we look more closely at how pgmcc works in comparison to TCP Reno. pgmcc times out after a stall when the ACKs stop coming in, and a long timeout expires. But there are no specific information about the exact timeout value for pgmcc and how it is determined. Without timeout pgmcc reacts to congestion by cutting the window in half similar to fast recovery in TCP. TCP on the other hand adjusts its timeout value depending on the measured RTT and the variance of the measured RTT values. In addition, ACKs in pgmcc are per-packet as in SACK, while in Reno ACKs are aggregate only, so for Reno to send an ACK for a packet, all packets in between have to be received. This has a large effect when multiple packets are dropped from a window.

Our explanation of the unfairness that is observed over long periods is due to these differences in handling timeouts and responding to ACKs and losses. By observing the window size changes in both of them (Fig. 3), we found that the pgmcc window is larger most of the time and it does not enter the slow start phase. We have also conducted several other experiments with changing the timeout value, we found that the results obtained depend heavily on this value. For example, if the timeout is set to a relatively small value this can cause TCP to have a much higher throughput. The appropriate value for timeout that achieves fairness depends on dynamic network conditions that change over time.

Next we try the same experiments with New-Reno and SACK. New-Reno and SACK reduce the timeouts and solve the performance problems when multiple packet are dropped from a window. Simulation results show that pgmcc is fairer (*f*=92%) with SACK and New-Reno [15].

To clarify more the effect of timeout and window size, we run the same experiment of TCP Reno with an adaptive timeout mechanism added to pgmcc. In this experiment pgmcc uses an adaptive timeout similar to that used in TCP and the reset of the timeout is controlled to be as close as possible to TCP Reno. It is reset only if there are no packets missing in the received bitmask (i.e. all packets are acked). Because of differences in RTT between different ackers, after a switch a fixed timeout is used until the adaptive timeout for the new acker is computed. Fig. 4 shows the result of pgmcc compared to TCP Reno after adding the adaptive timeout. The modified pgmcc is friendly to TCP Reno.

These experiments clarify some of the characteristics and design choices of pgmcc. It is similar to TCP SACK in handling ACKs and losses, and it avoids timeouts. Since the deployment of SACK is permitted and it is currently increasing, there is no requirement to degrade pgmcc to TCP Reno and these design choices seem to be correct.

B. *Experiment Set 2: Diverse Losses and Delay*

This experiment evaluates the effect of different combinations of RTT and loss rates on the protocol behavior and shows how accurate is the equation used for computing the throughput. In Fig. 5 we have two pgmcc receivers, one with high RTT (400ms) and low loss rate (.4% or 2%) and the other with lower RTT (200ms) and higher loss rate (1.6% or 8%). Losses in this experiment are considered to be independent and not correlated. This enables us to control the parameters accurately to have equal throughputs in both links

[2] The final sequence numbers in the graphs represent the aggregate throughput. So their ratio can be considered as the ratio between the average instantaneous throughputs.

[3] This experiment runs for 3000 seconds. When we run the experiment for 300 seconds, as in [13], the unfairness shown here was not clear [15].

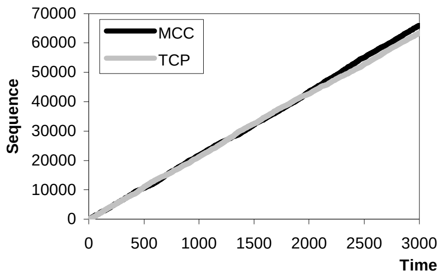

Fig. 4: Throughput of pgmcc with the adaptive timeout vs. TCP Reno (*f*=96%)

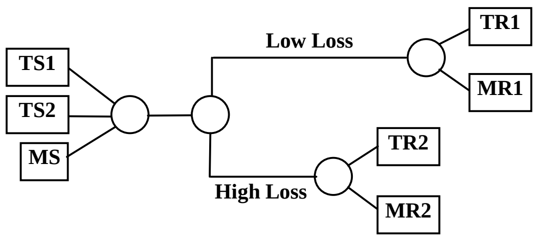

Fig. 5: MCC session with receivers having different delays and loss rates competing with TCP sessions

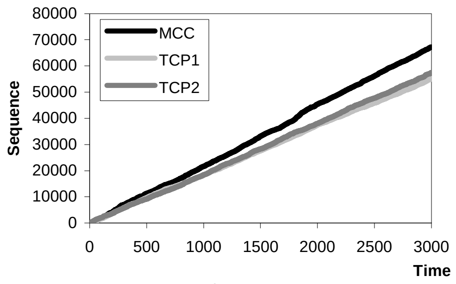

Fig. 6: Throughput of pgmcc vs. the two TCP sessions with low loss rate (*f1*=82%, *f2*=85%)

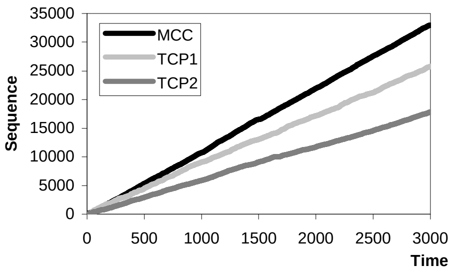

Fig. 7: Throughput of pgmcc vs. the two TCP sessions with high loss rate (*f1*=78%, *f2*=54%)

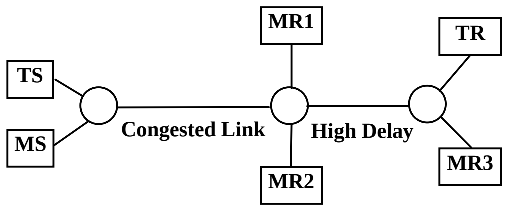

Fig. 8: MCC session with receivers having the same loss rate, but different delays

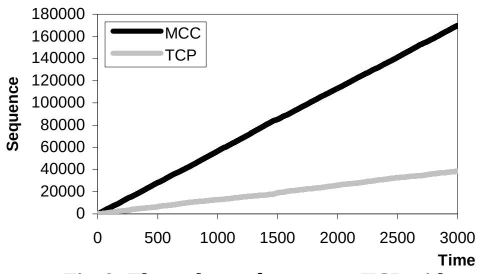

Fig. 9: Throughput of pgmcc vs. TCP with NACK suppression (*f*=23%)

and evaluate the results in this case.

In Fig. 6 we see that pgmcc and the two TCP sessions have close throughput[4]. The loss rates here are .4% for the low loss link and 1.6% for the high loss link. In Fig. 7 we are using a loss rate of 2% for the low loss link and 8% for the high loss link, which causes pgmcc to be unfair to the high loss rate TCP session. The reason for that is mainly because of the difference in reacting to packet losses. In pgmcc the reliability window is separated from the congestion window and the handling of acknowledgements is different. Unlike TCP, in pgmcc the sender keep sending new packets, even if there are previous lost packets not received yet. This separation between reliability and congestion seems to be unavoidable in order to achieve an acceptable performance in reliable multicast. In addition, according to [11] the simplified equation used for computing the throughput is for fast retransmission only and it does not take timeouts into account. It also overestimates the throughput for losses above 5% and so it is suitable only when loss rates are below 5% and no timeouts happen.

This experiment shows that at high loss rates pgmcc can be unfair to TCP due to the ignoring of previously lost packets by congestion control, and due to the inaccuracy in the throughput equation. We performed also experiments for correlated losses. Due to space limitations the results are presented in [15].

C. *Experiment Set 3: Feedback Suppression*

In this experiment we are testing the use of feedback suppression in the routers and its effect on congestion control. In PGM [16], if feedback aggregation is used, the first instance of a NACK for a given data segment is forwarded to the source and subsequent NACKs are suppressed. Using the topology in Fig. 8 we find that feedback aggregation will cause pgmcc to be unfair to TCP, because the worse receiver MR3 will always have its NACKs suppressed (the link leading to MR3 router has 50 ms delay). The throughput of pgmcc and TCP without network support is similar to experiment 1. In Fig. 9 we see that with network support, pgmcc gets much higher throughput than TCP.

This experiment shows that feedback suppression can cause pgmcc to be unfair to TCP. Accordingly we recommend that some changes are needed in the way feedback aggregation is performed with pgmcc. A solution for that is to store both the loss ratio and RTT for each NACK and to compare the throughputs using these values. This solution may solve the problem, but it increases storage and computation overhead in the routers. We propose a low overhead solution for that problem by random suppressing of NACKs in the router. The router will suppress NACKs only with some probability. This will give the worst receiver NACKs some chances to reach the sender. There will be a tradeoff here between the amount of feedback suppressed and the accuracy of acker selection.

D. *Experiment Set 4: Group Representatives*

This experiment shows the effect of using group representatives and changing them. In pgmcc we evaluate the effect of acker switching using also the topology of Fig. 8, but with a higher delay (200ms) in the link leading to the MR3 router. No suppression will happen in this case because the retransmissions will reach MR1 and MR2 router before the NACK of MR3 reaches there. In PGM retransmissions are directed by the router only on those links that sent the NACKs, and these retransmissions delete the NACKs states from routers. As shown in Fig. 10, the throughput of pgmcc becomes too low, and the TCP throughput is much higher. This does not constitute a fairness problem, but a performance degradation problem for pgmcc.

[4] We set the parameters of RTT and loss rates to let the two TCP sessions get the same throughput, according to the TCP equation.

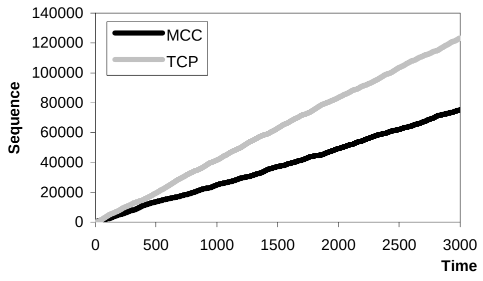

Fig. 10: Throughput of pgmcc vs. TCP due to acker switches (*f*=164%)

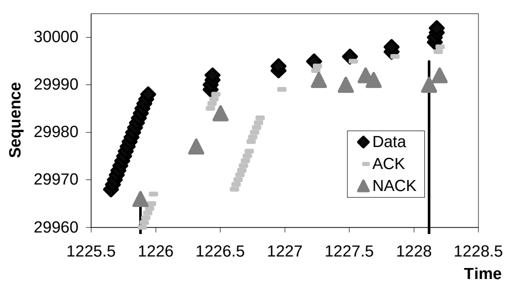

Fig. 11: Detailed sequence of pgmcc packets during acker change

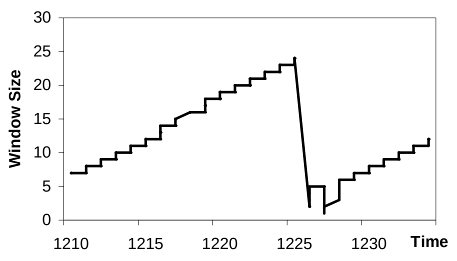

Fig. 12: Window size changes of pgmcc session, during an acker change

The reason for this bad performance under the given topology is the acker switching between a high RTT receiver and a low RTT receiver. By looking at acker switches in detail we found that two switches happen in succession close to each other. The first NACK from the closer receiver causes an acker change then the other NACK causes another change for the far one when it arrives. This pattern repeats with packet losses. It is interesting to look at why acker changing causes this bad performance (in other experiments it has no effect). By taking a more detailed look in Fig. 11 to observe what happens between two acker switches (a vertical line means an acker switch), we find that after the switch to the close receiver, new ACKs arrive before old ACKs. The old ACKs that arrive at 1226.5 do not cause new packets to be sent which means that they do not generate new tokens. Later, when new ACKs arrive the window start at slow rate, which means that, it has been cut. Fig. 12 shows how the window is cut at 1226.5. The reason for that is due to the out-of-order ACK delivery and the reactions taken accordingly by the sender. Wrong loss detections can be interpreted, because ACKs for old packets have not arrived yet. Also on a loss detection the sender try to realign the window to the actual number of packets in flight, which will not be interpreted correctly after the switch, because there are still packets and ACKs in flight to and from the old acker.

To solve this problem the sender needs to keep track of the recent history of acker changing and the ACKs sent by each acker. In addition the bitmask provides information about the recent packets received by the acker. Accordingly the sender can adjust its window and avoid these problems.

This experiment shows that acker switching between receivers with large difference in delay degrades the performance of pgmcc. This problem will be more common on larger scales.

V. CONCLUSIONS

We have presented a set of carefully designed experiments to evaluate multicast congestion control protocols. These experiments clarify the operational details of the protocol by targeting specific mechanisms. They also show the differences with TCP and the related fairness issues. We carried the experiments on a case study protocol, pgmcc. Some problems have been found due to high losses, feedback suppression, and group representative switch. Improvements are proposed to cope with some of the problems, such as random suppression of NACKs, sender response after representative switches, and the adaptive timeout in case fairness to TCP Reno is required. We recommend researchers to consider our scenarios in addition to existing scenarios, and hope that this with the methodology presented in [14], be part of an evaluation framework to expedite the development and standardization of multicast congestion control protocols.

REFERENCES

[1] L. Breslau, D. Estrin, K. Fall, S. Floyd, J. Heidemann, A. Helmy, P. Huang, S. McCanne, K. Varadhan, Y. Xu, and H. Yu. "Advances in Network Simulation." *IEEE Computer,* May 2000.

[2] S. Bhattacharyya, D. Towsley, and J. Kurose. "The Loss Path Multiplicity Problem for Multicast Congestion Control." *IEEE Infocom, March* 1999.

[3] A. Chaintreau, F. Baccelli, and C. Diot. "Impact of Network Delay Variation on Multicast Session Performance with TCP-like Congestion Control." *IEEE Infocom,* April 2001.

[4] D. DeLucia and K. Obraczka. "Multicast feedback suppression using representatives." *IEEE Infocom,* April 1997.

[5] D. DeLucia and K. Obraczka. "A Multicast Congestion Control Mechanism for Reliable Multicast." *IEEE ISCC,* June 1998.

[6] K. Fall and S. Floyd. "Simulation-based Comparison of Tahoe, Reno, and SACK TCP." *Computer Communication Review,* July 1996.

[7] S. Golestani and K. Sabnani. "Fundamental Observations on Multicast Congestion Control in the Internet." *IEEE Infocom,* March 1999.

[8] A. Helmy, D. Estrin, and S. Gupta. "Systematic Testing of Multicast Routing Protocols: Analysis of Forward and Backward Search Techniques." *IEEE ICCCN,* October 2000.

[9] A. Helmy, S. Gupta, D. Estrin, A. Cerpa, and Y. Yu. "Systematic Performance Evaluation of Multipoint Protocols." *IFIP FORTE/PSTV,* October 2000.

[10] A. Mankin, A. Romanow, S. Bradner, and V. Paxson. "IETF Criteria for Evaluating Reliable Multicast Transport and Application Protocols." *RFC 2357,* June 1998.

[11] J. Padhye, V. Firoiu, D. Towsley, and J. Kurose. "Modeling TCP throughput: A Simple Model and its Empirical Validation." *ACM SIGCOMM,* September 1998.

[12] J. Padhye and S. Floyd. "On Inferring TCP Behavior." *ACM SIGCOMM,* August 2001.

[13] L. Rizzo. "pgmcc: A TCP-friendly Single-Rate Multicast Congestion Control Scheme." *ACM SIGCOMM,* August 2000.

[14] K. Seada, S. Gupta, and A. Helmy. "Systematic Evaluation of Multicast Congestion Control Mechanisms." *SCS SPECTS,* July 2002.

[15] K. Seada and A. Helmy. "Fairness Evaluation Experiments for Multicast Congestion Control Protocols." *Technical Report 02-757, University of Southern California, CS Department,* March 2002.

[16] T. Speakman, J. Crowcroft, J. Gemmell, D. Farinacci, S. Lin, D. Leshchiner, M. Luby, T. Montgomery, L. Rizzo, A. Tweedly, N. Bhaskar, R. Edmonstone, R. Sumanasekera, and L. Vicisano. "PGM Reliable Transport Protocol Specification." *RFC 3208,* December 2001.

[17] J Widmer, R Denda, and M Mauve. "A Survey on TCP-Friendly Congestion Control." *IEEE Network Magazine,* May 2001.

[18] J. Widmer and M. Handley. "Extending Equation-based Congestion Control to Multicast Applications." *ACM SIGCOMM,* August 2001.