Dynamics of the “pgmcc” Multicast Congestion Control Protocol

Chin-ying Wang and Sonia Fahmy
Department of Computer Sciences, Purdue University, West Lafayette, IN 47907–1398, USA
E-mail: fahmy@cs.purdue.edu

Abstract

Fairness to current unicast (point-to-point) Internet traffic is an important requirement of multicast (group communication) protocols. In this paper, we investigate the fairness (in terms of bandwidth use relative to TCP/UDP traffic) of the “pgmcc” protocol. Pgmcc is a promising multicast congestion control proposal, but it has not been extensively stress-tested. We investigate the performance of pgmcc when competing with bursty TCP and UDP flows in a scenario with multiple time-varying bottlenecks and round trip times. Our results indicate that pgmcc is a robust protocol, but is missing an algorithm for dynamically determining the timeout value, and an algorithm for smoother handling of switches among receiver representatives.

Keywords

multicast, congestion control, pgmcc, fairness, feedback aggregation

1. Introduction

Multicasting allows efficient information exchange among multiple senders and multiple receivers. Popular multicast applications include audio/video conferencing, distance learning, and distributed games. Pgmcc [1] is a single rate, representative-based multicast congestion control protocol designed to be fair to competing TCP flows. Pgmcc sets its transmission window size according to a representative called the “acker.” The acker is the receiver with the lowest rate (throughput) among all receivers within a group. A tight control loop is run between the acker and the sender [1].

Although pgmcc is one of the most promising multicast congestion control proposals, it has not been extensively tested. In this paper, we examine the pgmcc protocol as implemented according to the standard discussed at the IETF. In our first set of experiments, we demonstrate the feedback aggregation problem caused by the NAK suppression at routers and show its effect on acker selection. In the second set of experiments, the performance of pgmcc is evaluated when competing with TCP and UDP flows in a realistic scenario. The fairness of pgmcc to TCP flows is examined with different bottleneck link bandwidths. Simulation results show pgmcc may achieve higher throughput than competing TCP flows during acker switching, especially during the first few acker switches. In other cases, pgmcc performance degrades due to the fixed timeout interval used in pgmcc. Experiments are performed using ns 2.1b5, and pgmcc is implemented on top of the PGM [2] multicast transport protocol.

The remainder of this paper is organized as follows. Section 2 discusses reliable multicast protocols, specifically PGM and pgmcc. Section 3 examines the effect of feedback aggregation. Section 4 discusses simulation results of fairness among pgmcc and TCP. Future work is discussed in section 5.

2. Related Work

This section discusses reliable multicast protocols, including detailed descriptions of PGM and pgmcc.

2.1 Reliable Multicast Protocols

Figure 1 illustrates the operation of reliable multicast protocols. S represents a sender host, and each R represents a receiver host in a multicast group. S sends a single copy of every packet into the network. As the packet is forwarded by a router (the rectangles in figure 1 represent network routers), it is replicated when needed and forwarded via multiple outgoing links of the router. The packet should reach all the receivers in a group. In order to enforce reliability, each receiver has to provide some form of feedback to notify the sender whether the packet has been received. A receiver may send an ACK if it receives packets successfully, or it may send a NAK if a packet is assumed to be lost.

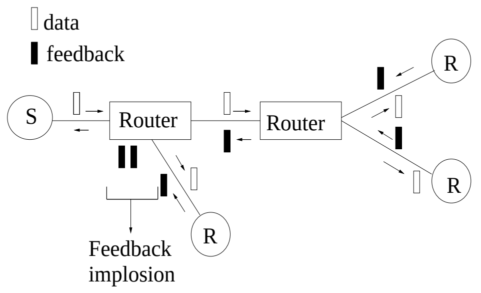


Figure 1: Reliable multicast protocol operation and the feedback implosion problem

Two important problems with reliable multicast protocols are depicted in figure 1 and figure 2. Since each of the receivers sends feedback to the sender, the sender may be overwhelmed by the implosion of ACKs/NAKs when the number of receivers becomes very large. Feedback uses bandwidth unnecessarily, and the sender is burdened with processing all the feedback packets.

Determining the appropriate sending rate at the multicast sender is the second problem (shown in Figure 2). Each of the receivers in a multicast group may have a different capacity. The problem of determining the sending rate to achieve the “optimal” bandwidth usage depends on the application reliability semantics.

2.2 Pragmatic General Multicast (PGM)

PGM is a single-sender multicast protocol, providing a reliable service by using NAK-based retransmission requests. Feedback

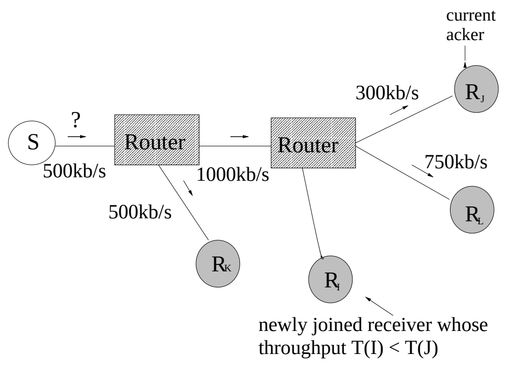


Figure 2: Acker selection in pgmcc

suppression allows PGM routers to only forward the *first* NAK to arrive at a router for each missing or corrupted packet [3]. An example data/feedback packet flow in PGM is depicted in figure 3. Each original data packet (odata) sent from the PGM sender is replicated at each router, and forwarded to each of the receivers in a group. If a receiver does not receive odata, it sends a NAK upstream. When the upstream router receives a NAK, it sends an NCF packet to the receiver indicating the reception of the NAK, and forwards only one NAK out of all the NAKs sent from different receivers in the same subtree for each of the lost/corrupted packets. Eventually, the sender will receive the NAK, and repair of the data (rdata) is transmitted only to the receivers who requested the retransmission.

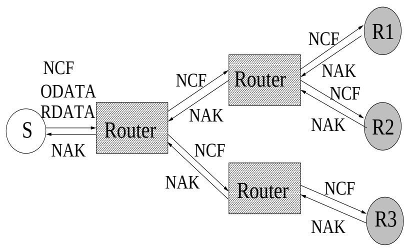


Figure 3: PGM protocol operation

2.3 Pgmcc Congestion Control

One method of adjusting the sending rate at the PGM sender is to pace the sender according to the “slowest” PGM receiver. Pgmcc paces the sending rate according to the receiver with the *worst throughput*, which serves as the group representative. This receiver is called the “acker.” The acker can change at any time since receivers are continuously joining and leaving the multicast group, and bottlenecks vary over time. A tight control loop is run between the acker and the sender. Only the acker sends ACKs to the sender to adjust the sender transmission window and token bucket. Other receivers may send NAKs when they lose packets. Both the loss rate and the round trip time (RTT) are needed to calculate the throughput of each of the receiver. This information is carried in both NAK and ACK packets, and the sender uses the throughput equation as specified in [1] to compute the throughput of each receiver. The acker is switched from one receiver to another if a receiver with a lower throughput is found. An example of acker switching is illustrated in figure 2.

A window based congestion control protocol similar to that used by TCP is run between the acker and the sender. In the protocol specified in [1], the sender maintains two state variables: a window W, and a token count T. W represents the number of packets in flight, while T is used to regulate the generation of data packets. One token is needed and consumed in order to transmit one data packet. Initially, both W and T are initialized to one. The values of W and T are updated with every ACK, NAK, timeout, and packet transmission [1].

3. Feedback Aggregation

In this section, we illustrate the effect of feedback aggregation on pgmcc performance. Due to the suppression of PGM NAKs containing RTT and loss rate information needed by the pgmcc sender to select the acker, incorrect acker switches may occur in certain cases. An example is shown in figure 4, where one PGM session runs pgmcc at the sender PS, and each of the four receivers PR*. There are two PGM routers, and all the links in this topology have the same bandwidth and delay. Among the four PGM receivers, we are interested in receivers PR1 and PR3. PR1 is closer to the sender and has a lower loss rate; PR3 is further away from the sender and exhibits a higher loss rate.

Suppose the PGM sender begins to send data to all its receivers, and both PR1 and PR3 lose packet number 5. Due to the shorter delay to PR1, the router closest to the sender will receive the NAK from PR1 before receiving one from PR3. Hence, the router forwards the NAK sent from PR1 to the sender, and the NAK sent from PR3 is suppressed. Since pgmcc needs the loss rate and RTT carried in NAK packets to perform acker switching, the sender may select PR1 as the acker instead of PR3 at certain instances, even though PR3 clearly has lower throughput.

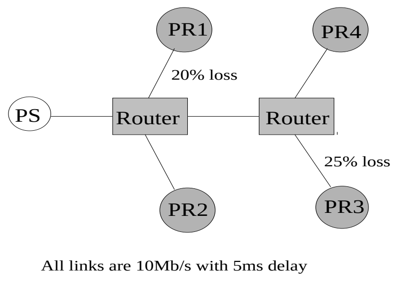
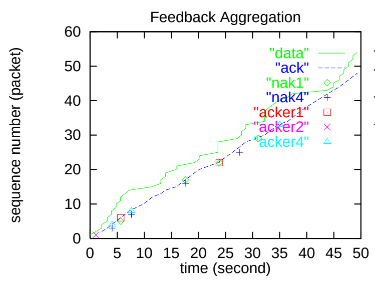
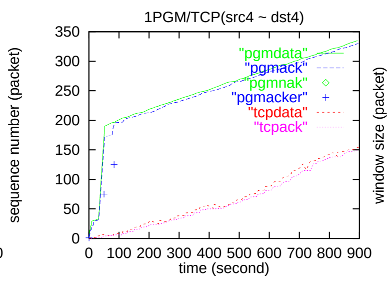


Figure 4: Simulation topology to examine feedback aggregation problems

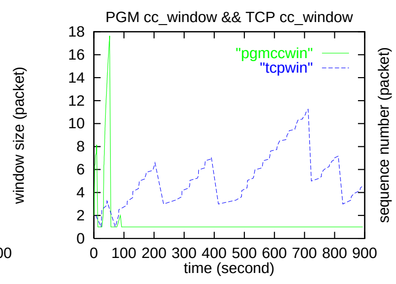
To verify this scenario, we simulate the scenario depicted in figure 4 (except PR3 and PR4 are switched) for 50 seconds. We plot the sequence numbers and acker switches in figure 5(a). In the figure, “data” is sent from the sender to the receivers, “ack” is the acknowledgment sent from the acker upon receiving a packet, “nak1” is the NAK sent from PR1, etc. “Acker1” shows that the current acker is PR1 at the specified time and “acker2” and “acker4” denote PR2 and PR4 are the ackers respectively. From the simulation results, we can see that PR2 is selected as the acker at the beginning of the simulation. About 4 seconds later, the sender receives a NAK from PR4, and it switches the current acker to PR4. The acker is switched to PR1 because the sender receives a NAK from PR1, and the one possibly sent from PR4 is suppressed. Thus, the acker is switched back to PR4 again at the 8th second. The sender switches the acker between PR1 and PR4 a number of times. Unnecessary acker switches occur between PR1 and PR4 although the acker should (on a larger time scale) always be PR4 which has a higher loss rate and higher RTT. Thus the time scale of pgmcc may be too fine, and coarser time scales may enhance stability.



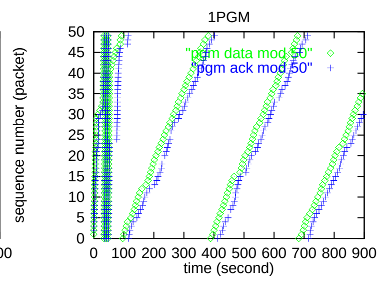
(a) Simulation results depicting feedback aggregation effects



(b) PGM/TCP throughput in highly congested network



(c) PGM/TCP sender congestion window in highly congested network



(d) PGM odata/ack sent/received at the sender in highly congested network

Figure 5: Simulation results depicting feedback aggregation and performance in a highly congested network

4. Pgmcc Fairness Dynamics

In this section, we simulate pgmcc in more complex configurations. The objective of our experiments is to determine whether pgmcc is fair to TCP (i.e., is TCP friendly) in realistic scenarios. Figure 6 shows our simulation topology which includes 22 source nodes (S*) and 22 destination nodes (D*). The link between each node and router has a bandwidth of 150 kbps with 1 ms delay. The link bandwidths and link delays between routers are specified in table 1. 22 TCP flows run between source and destination nodes. TCP *NewReno* is used because NewReno and SACK are being deployed in the majority of web servers to provide better congestion control compared to other TCP versions [4]. We investigate the performance of the TCP flow from $S4$ to $D4$, which runs across the same links and nodes as the PGM receiver with the longest RTT. One UDP flow sending Pareto traffic runs across "Link 4" with a 500 ms on/off interval. All the routers use simple drop tail queues of size 120 packets. The PGM sender and receivers are located in the nodes labeled "PS" and "PR*" in Figure 6. All the simulations were run for 900 seconds.

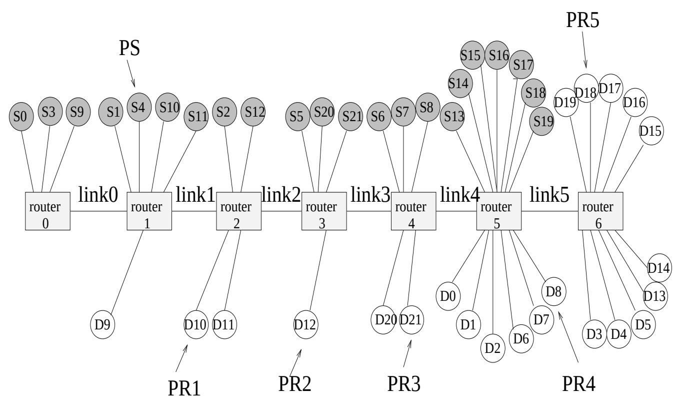


Figure 6: Simulation topology to investigate fairness

In each of the following experiments, we measure the goodput (as defined in [5]) which indicates bandwidth achieved at the receiver excluding duplicate packets. In all the experiments, the goodput for TCP flows from $S7$ to $D7$, from $S11$ to $D11$, and from $S21$ to $D21$ is almost 2/3 of the link bandwidth because the RTT of each of them is fairly short [6].

4.1 Experiment I: Highly Congested Network

The goodputs of the PGM session and the TCP session from $S4$ to $D4$ are shown in figure 5(b), and the sizes of their congestion windows are depicted in figure 5(c). In terms of the goodput, the goodput for each of the PGM receivers ranges from 3.95 to 4.26 kbps and the goodput for TCP receiver is 1.39 kbps. It is not surprising that goodput for both PGM and TCP flows is quite low because the links are congested and shared among many TCP and UDP flows. Figure 5(b) shows that for the first 50 seconds of the simulation, PGM has a much higher throughput than TCP. After 50 seconds, the slopes of the PGM flow and the TCP flow are similar. Both of the flows have low throughput. The reason for this is that the PGM window (figure 5(c)) increases only during the first 50 seconds of the simulation. The size of the PGM window drops to one several times in the first 100 seconds, and it remains one till the end of the simulation. On the other hand, the congestion window at the TCP sender increases slowly due to slow start at the beginning, but the TCP sender is able to send more data afterwards compared to the PGM sender due to a larger window throughout the rest of the simulation.

Observe that the PGM sender always chooses the receiver closest to it as the acker at the beginning of the simulation (since it is the first receiver it receives feedback from). Hence, $PR1$ is elected as the initial acker. Later, several packets are dropped at router 4 causing PR4 and PR5 to send NAKs for the same lost packets. The NAKs from PR5 are suppressed and only the NAKs sent from PR4 are forwarded to the sender. Hence, the acker is switched from PR1 to PR4 due to the higher loss rate of PR4 over PR1 perceived by the PGM sender. Finally, more packets are dropped at router 5 causing PR5 to send NAKs for the lost packets, so the acker is switched to PR5, which has the longest RTT and the highest loss rate.

We now investigate why initial acker switches cause steep increase of the window at the PGM sender. Figure 5(d) illustrates the time that data packets (represented by the diamond shape) are sent and the acks are received by the PGM sender in this experiment. Each ack (sent from the current acker) is represented by a plus sign in the figure. The packet number (modulo 50 to make the figure more readable) is given on the y-axis. At the beginning of the simulation, PR1 is selected as the acker because it is closest to the sender. After sending packet number 173, the acker switched from PR1 to PR4 at time 50.222 second (the overlapped diamond and plus sign in Figure 5(d) at time 76.9 seconds indicates the acker switch). Because the RTT of PR4 is much longer than PR1, it takes

Table 1: Link bandwidths and delays between routers

Link	*Link 0*	*Link 1*	*Link 2*	*Link 3*	*Link 4*	*Link 5*
Bandwidth (kbps)	50	100	50	150	150	50
Delay (ms)	20	10	5	5	5	10

longer for PR4 to receive data packets sent from the sender than it does for PR1. Moreover, each data packet is marked with the current acker address in pgmcc. Hence, even though an acker switch occurs early, the previous acker (PR1) continues sending ACKs to the sender until reception of packet number 173. As a result, the new acker (PR4) only sends ACKs after the reception of packet number 174. In this experiment, PR4 began sending ACKs after 76.9292 seconds even though the acker switch occurred at time 50.222 seconds.

There are two consequences of this behavior. First, when an acker switch occurs, it means there is a receiver with a lower throughput than the current acker. However, as discussed above, the previous acker keeps on sending ACKs till the packet number is equal to the trail of the sender window at the time of the acker switch. For each ACK received at the sender side, the sender increases the token count T by one and increases the window accordingly. Thus, more data packets are sent by the sender. This leads to the second consequence which is that the network becomes even more congested. We observed the same behavior when the acker switched from PR4 to PR5.

The delay of sending ACKs from the new acker observed in this experiment is one of the causes of the sudden drop of the window size to one. Because pgmcc uses a fixed timeout interval to detect congestion, if the sender does not receive an ACK from the acker within the timeout specified, it drops the window size W to one and decreases the token count T to $\frac{-1}{2W}$ [1]. Revisiting figure 5(d), we see that the distance between the last overlapped cross and diamond at time 76.9 and the first non-overlapping cross indicates the time that the sender waits for the ACK for packet number 174 from the new acker (PR4). If the distance is longer than the timeout interval, which is true in this experiment, one or more timeouts occur, degrading PGM performance.

Another reason for the sudden drop of the window size to one is that the RTT of the current acker itself is sometimes simply *longer than the timeout interval*. In this case, the PGM sender will never be able to receive an ACK within the timeout time and will keep timing out, as shown in figure 5(c). This problem can be remedied by implementing a TCP-like retransmission timeout determination algorithm.

4.2 Experiment II: Medium Congestion

In this section, we maintain all simulation parameters unchanged except that we increase the bandwidth of the links between routers. Many experiments were run with bottleneck link bandwidths ranging between 2.5 and 3.5 times the original bandwidths shown in table 1. The results are similar, so we only discuss the results using bandwidth of 2.5 times, and 3.5 times the original bandwidth.

The throughput of both PGM and TCP flows and their windows sizes are shown in figure 7. We observe similar behavior to figure 5(b) in figure 7(a). Fewer timeouts occur in this experiment (figure 7(d)) compared to the highly congested network in the previous subsection. This is because we have increased the link bandwidths, so the time it takes for transmission of ACKs from the acker to the sender is shorter than that in the previous setting. However, the timeouts are still frequent, causing the window to drop to 1 because the RTT of PR5 is greater than the PGM sender timeout interval. The throughput of the TCP flow, on the other hand, is higher than the one in experiment I and has a higher slope. In terms of the goodput, the goodput of PGM receivers ranges from 4.77 to 4.91 kbps, and goodput of the TCP receiver is 4.97 kbps.

Figure 7(b) may appear different at first, but it is similar to Figure 7(a) if we had run the simulation longer. The goodput of each of the PGM receivers ranges between 22 and 22.4 kbps, and that of the TCP receiver is 7.24 kbps. The reason for the higher goodput for PGM receivers, in addition to the increase of the bandwidth, is that the acker was switched from PR3 to PR5 several times. As discussed above, acker switches take time and the sender window is increased meanwhile. Further, the two different branches have different throughputs. This effect is clearly shown in figure 7(d).

By increasing the bottleneck link bandwidth, the throughput of both the PGM and TCP flows increases. From this set of experiments, we conclude that the PGM flow outperforms the TCP flow during initial acker switching, but the TCP flow has a higher throughput if the timeout interval at the sender does not adapt to the increase of the acker RTT.

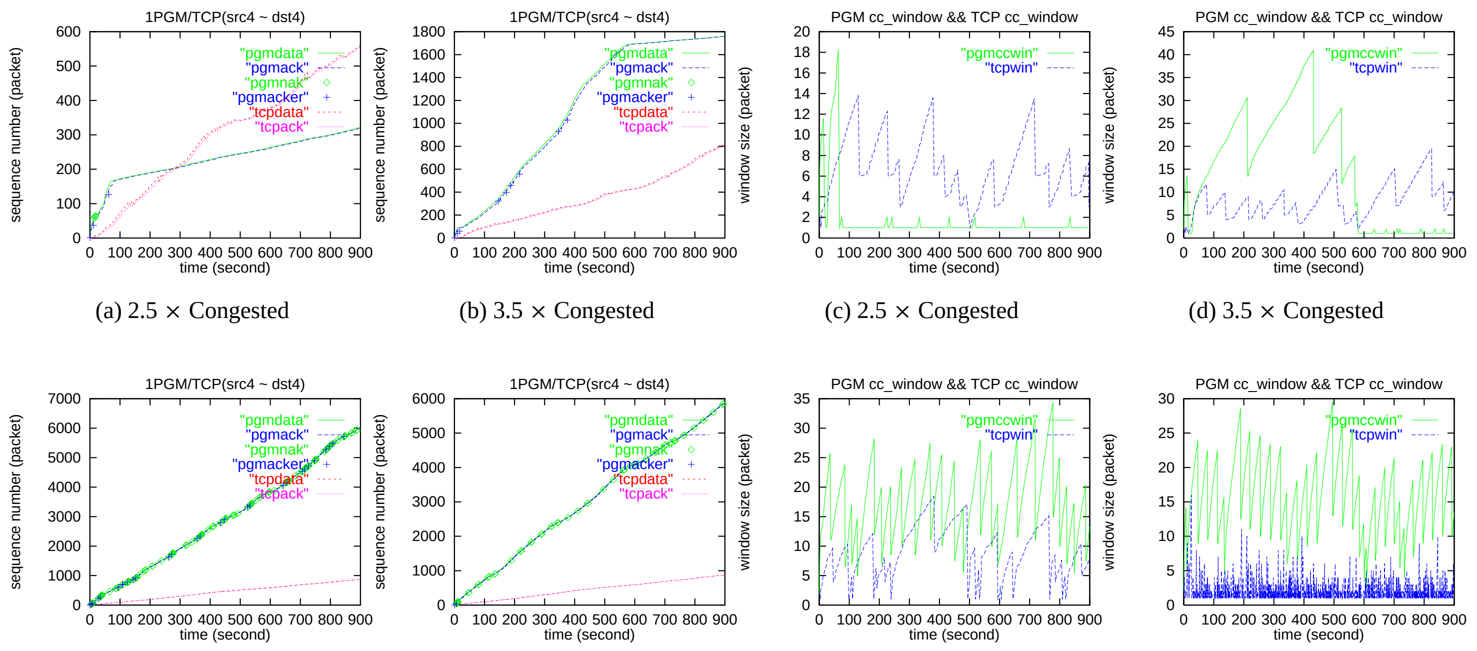
4.3 Experiment III: Uncongested Network

In this section, we retain all the parameter values of previous experiments but we increase the bandwidth of the links between routers. Many experiments were run with various bandwidths, but, since the results are similar, we only show the results using bandwidths 10 times and 80 times the original bandwidths in table 1. The throughput of both PGM and TCP flows and their windows sizes are shown in figure 7(e)–(h). From the figure, we find that the acker switches back and forth between PR3 and PR5 due to the closeness of the throughputs of PR3 and PR5. Feedback suppression does not cause problems here because PR3 and PR5 lose different packets. As expected, the PGM sender window continues to increase some time after acker switch. In terms of goodput, the goodput for each of the PGM receivers ranges from 74.15 to 76.56 kbps, and for the TCP receivers it is around 7.8 kbps in figures 7(e) and (f).

The reason why PGM outperforms TCP appears to be the selection of PR3 as the acker throughput most of the simulation. Both the TCP receiver and PGM receiver PR5 are connected to router 6. PR5 is the acker for only 173 seconds while PR3 is the acker for 736 seconds of the simulation time. PR3 has a short RTT and reasonable loss rate. Therefore, even though increasing the link bandwidth will increases the throughput of both PGM and TCP flows, the PGM flow outperforms the TCP flow in this case.

5. Conclusions and Future Work

In this paper, we have investigated the fairness and dynamics of the pgmcc single-rate multicast congestion control protocol. Our simulation results show that pgmcc flows initially send more than competing TCP flows due to the rapid opening of the PGM sender window between initial acker switches. If the acker selection process stabilizes and a PGM receiver with a very long RTT is selected to be the acker, timeouts severely degrade the performance of the PGM flow. A TCP-like retransmission timeout computation



(a) 2.5 × Congested (b) 3.5 × Congested (c) 2.5 × Congested (d) 3.5 × Congested

(e) 10 × Congested (f) 80 × Congested (g) 10 × Congested (h) 80 × Congested

Figure 7: PGM/TCP throughput and window size with medium and no congestion

mechanism can remedy this problem. With the use of a timeout interval selected according to the acker RTT, the PGM sender can distinguish between situations of real congestion and late ACKs received from an acker with a long RTT. In an uncongested network, the PGM flow may outperform a competing TCP flow if frequent acker switches occur between ackers of different throughputs.

We plan to examine various application reliability semantics to see how the pgmcc protocol fits in the unreliable (or not fully reliable) multicast protocol context. The PGM multicast protocol provides reliability in the transport layer as specified in [2] and illustrated in Figure 3. On examining the pgmcc implementation, we find that if a PGM receiver loses a data packet, it only sends a NAK back to the sender once. If the NAK get lost or corrupted before it gets to the sender, or if the NCF sent to acknowledge the NAK reception is lost or corrupted before it gets to the receiver, the receiver which originally sent the NAK will wait for a retransmission timeout. Then, the receiver reschedules the retransmission timeout up to ten times waiting for the repair, instead of resending the NAK. If, after rescheduling the retransmission timeout for ten times, the repair is not received, the receiver treats the packet as unrecoverable. This essentially means that this flavor of PGM is not fully reliable. It is not completely unreliable though, because NAKs are sent, and when a receiver receives the repair from the sender, it does not check whether the repair is needed or not (e.g., if the repair must be received within a certain amount of time and it is useless otherwise). We plan to experiment with various reliability semantics, and examine their effect on the pgmcc congestion control algorithm, especially on acker selection with insufficient NAKs.

Acknowledgments

The authors would like to thank Gianluca Iannaccone and Luigi Rizzo for providing us with their pgmcc implementation. This research is sponsored in part by *Tektronix*, and by the *Schlumberger* Foundation technical merit award.

6. References

[1] L Rizzo, “pgmcc: a tcp-friendly single-rate multicast congestion control scheme,” in *Proceedings of the ACM SIGCOMM*, August 2000.

[2] D. Farinacci, A. Lin, T. Speakman, and A. Tweedly, “PGM reliable transport protocol specification,” Internet draft, March 2000.

[3] K. Miller, “Multicast networking and applications,” Addison Wesley Longman, Inc., 1999.

[4] K. Fall and S. Floyd, “Simulation-based comparisons of Tahoe, Reno, and SACK TCP,” in *ACM Computer Communication Review*, July 1996, vol. 26, pp. 5–21.

[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] J. Padhye, V. Firoiu, D. Towsley, and J. Kurose, “Moldeling TCP throughput: A simple model and its empirical validation,” in *Proceedings of the ACM SIGCOMM*, September 1998, vol. 28, pp. 303–314.