Scaling Feedback to Very Large Groups

Jonathan Rosenberg
Bell Laboratories
jdrosen@bell-labs.com

Henning Schulzrinne
Columbia University
schulzrinne@cs.columbia.edu

February 13, 1997

Abstract

We describe, analyze and simulate algorithms that allow scaling of feedback in very large multicast groups with potentially millions of members, both for any-to-any feedback and any-to-one feedback. While the techniques generalize, our particular focus is on feedback mechanisms suitable for the real-time transport protocol (RTP). To suppress the initial flood of packets occuring when a large number of users simultaneously join a session, we propose a new, but backward-compatible algorithm, called reconsideration, which minimizes the initial transient. Another algorithm quickly "teaches" new arrivals to a group the current group size. We then describe polling (any-to-one) algorithms that quickly estimate multicast populations. Adaptive statistical sampling may be used to reduce the state storage overhead incurred by tracking large user populations for either polling or the reconsideration algorithms.

1 Introduction

It has been known for quite some time that network level multicast is the only way to efficiently allow for multipoint to multipoint communication among a large set of users. The result has been nothing less than a flood of activity on how to scale multicast to large groups; PIM [1], CBT [2], and SEAM [3] are the results of some of this activity. Less work, however, has focused on the impact of large scale multicast groups on *end-systems*. In many cases, end-systems must maintain state for each group member, or must transmit into the group at some rate inversely proportional to group size. Either scenario requires end-system capabilities to scale to large groups.

Perhaps the best example of this is QoS feedback among listeners in a cable TV system. There, the number of users can easily exceed 10^5. Each user must scale back their QoS reporting in order not to flood the quality monitoring station. Another application is described in [4], where users in a layered multimedia session perform "experiments" to probe the network for bandwidth by joining and leaving multicast groups. To avoid thrashing, users scale back the rate of experimentation based on the number of observed participants. Other applications, such as distributed real time measurements from a group of sensors, or bandwidth sharing in multiple access networks, are also examples where end-system scalability is an issue.

Although the problem is a general one, we focus here on Internet multicast groups which utilize RTP [5] for real-time communication. Since RTP requires end-systems to estimate the group size, and then scale back RTCP reporting based on the group size, it characterizes the problem we are trying to address. Furthermore, recent interest in using RTP for large cable networks makes our focus of practical concern.

There are a number of distinct, but inter-related problems which occur with large multicast groups for RTP:

- *Congestion*: In many cases, the access bandwidths for users will be small compared to network bandwidths (28.8 kb/s modems, for example can now handle multimedia RTP sessions when RTP header

compression [6] is used). It is also likely that users will join the groups almost simultaneously. This occurs when everyone's favorite TV movie comes on at 10 pm, and everyone quickly joins the multicast group to receive the program. It also occurs when session joins are in response to timed events, such as a multicast session announcement. The result is large groups with each member believing that the group size is actually small. Combined with slow access links, the result is a flood of RTCP reports, causing access link congestion and loss.

- *State Storage*: The simplest way to determine the size of the multicast group is to listen to broadcasts, and store state for each distinct user heard from. Clearly, this does not scale well to extremely large groups, which would require Megabytes of memory just to track users. Alternate solutions must be found, particularly for set top boxes, where cost is a significant issue.

- *Delay*: As the group sizes grow, the time between RTCP reports from any one particular user becomes very large, possibly exceeding the duration of group membership. This means that timely reporting of QoS problems will not occur, and the value of the actual reports is lost. We discuss this issue in more depth in Section 3.

It is our desire to address these problems via a unified mechanism which is appropriate for both large and small groups. There are a number of reasons for this, including ambiguity about the definition of a large group, difficulty in knowing the group sizes a priori, and complication of implementations.

We classify feedback in large multicast groups into two distinct techniques. The first we call *any-to-any*, whereby all participants in the group both send and receive feedback. The second approach we call *any-to-one*, where all users send feedback, but only one (or several) stations receive it. Both techniques suffer from the difficulties described above, but different solutions are appropriate for each case. This paper proceeds by first considering the any-to-any case, and proposing two novel solutions: reconsideration and sendmax. The paper then considers the any-to-one case, and examines polling as a solution. We conclude by discussing sampling as a solution for the state storage problem; this solution applies to both the one-to-any and any-to-any scenarios.

2 Any-to-Any Feedback

In the any-to-any scenario, feedback is sent by and received by all members of the multicast group. One may question why this approach is reasonable at all for large groups. There are, however, a number of reasons why it remains a good choice for eliciting feedback:

- There is no single point of failure. In the case where one user is responsible for receiving feedback and reporting the group size to the entire group, single failures are catastrophic.

- Any-to-any makes sense for small groups. Having a single mechanism which works for both small and large groups is particularly desirable, since in many cases group sizes are not known a priori.

- Any-to-one requires special servers to be deployed with the appropriate functionality. Any-to-any feedback can be used without any new hardware or software.

Before considering our proposed solutions for the any-to-any case, we first review the current RTCP reporting algorithm.

2.1 Current RTCP Algorithm

Each user i in a multicast group using RTP, maintains a single state variable, the learning curve, which we denote $L(t)$. This variable represents the number of other users that have been heard from at time t. The state is initialized to $L(0) = 1$ when the user joins the group.

Each user multicasts RTCP reports periodically to the group. These reports contain information such as QoS feedback, user information, and messages for management of loosely controlled multimedia sessions. In order to avoid network congestion, the total amount of RTCP reports multicast to the group is set at 5% of the total multicast session bandwidth. To meet this criteria, each user increases the period of their reporting as the group size increases. Specifically, let C be RTCP interarrival time for the whole session, computed as the average RTCP packet size divided by 5% of the session bandwidth. Each user computes their deterministic interval as:

$$T_d = \max(T_{\min}, CL(t)),$$

where $T_{\min}$ is 2.5 s for the initial packet from the user, and 5 s for all other packets. To avoid synchronization, the actual interval is then computed as a random number uniformly distributed between 0.5 and 1.5 times T_d.

The algorithm for sending RTCP packets follows directly. The first packet from a user is scheduled at a time uniformly distributed between 1.25 and 3.75 seconds. We denote this time as t_0. All subsequent packets are sent at a time t_n equal to:

$$t_n = t_{n-1} + R(\alpha) \max(5, CL(t_{n-1})), \tag{1}$$

where we have defined $R(\alpha)$ as a random variable uniformly distributed between $(1-\alpha)$ and $(1+\alpha)$, where α equals $1/2$ in the current algorithm (we generalize because α has a strong impact on transient behavior).

When a large number of users all join the group at the same time (we call this a *step-join*), all schedule their packets within 2.5 s, and all are subsequently sent. The result is a flood of packets for 2.5 s, most of which are lost, causing subsequent floods of gradually decreasing intensity. The aim of the solutions proposed is to reduce this flood, and to ideally keep the aggregate RTCP packet rate at 5% of the session bandwidth at all times. It is an unfortunate consequence of mathematics that this desired behavior is in stark contrast to the desire for fast convergence and more frequent reporting from users. However, we feel that congestion avoidance is a much more important goal. This is particularly true when the convergence times exceed typical session durations, so that convergence is no longer even meaningful.

2.2 Alternate Solutions

A number of solutions for the scaling problems have been proposed, and many of these are orthogonal to our proposed solutions. They include adding summarizers to aggregate reports from regional or administrative areas, turning off RTCP reporting entirely, TTL scoping, using separate multicast groups for RTCP reports, and using the BYE message for QoS reporting, among others [7]. We note that many of these solutions are quite complicated, or require additional hardware, software, or protocol support. A complete evaluation of these alternatives, along with a comparison to our own, is done in [8], and we do not consider them further here.

2.3 Reconsideration and SendMax Algorithms

Our contribution is a new solution which we call *reconsideration*. This algorithm operates in two modes, conditional and unconditional. We first discuss conditional reconsideration.

At time t_n, as defined above, instead of sending the packet, the user checks if the group size estimate $L(t)$ has changed since t_{n-1}. If it has, the user reconsiders, and recomputes the RTCP interval (including the randomization factor) based on the current state (call this T'), and adds it to t_{n-1}. If the result is a time before the current time t_n, the packet is sent, else it is rescheduled for $t_{n-1} + T'$. Intuitively, this mechanism should help alleviate congestion by restricting the transmission of packets during the convergence periods.

In unconditional reconsideration, the user reconsiders independently of whether the number of perceived users has changed since the last report time. Experimental results in the next section demonstrated that this yields even better performance (in terms of congestion reduction) with step joins, with little effect on behavior after convergence.

Both modes of the algorithm are advantageous in that they do not require any modifications to the current RTCP protocol structure. In fact, they operate properly even when only a subset of the multicast group utilizes them. As more and more members of a group use the algorithm, the amount of congestion is lessened in proportion. This leaves open a smooth migration path which is absent for most of the other proposed solutions.

We introduce a third algorithm called *Sendmax*. In this algorithm, RTCP packets carry the group size estimated by the user sending the packet. Users receiving these packets log this quantity for each of the other users in the system, and the maximum of all group sizes received so far is computed (the user includes himself in the computation). The result is used for computing RTCP intervals, but the user continues to send his own perception of the group size in the RTCP packets.

This algorithm is most beneficial for step-joins occurring far apart in time (for example, every half-hour when a sitcom ends, and the viewers switch to a movie on another station). It assures that the subsequent step joins do not cause congestion. Unlike the reconsideration algorithm, however, it requires modification to the RTCP protocols, and is therefore not backwards compatible.

Both algorithms are safe, in the sense that it is impossible for a user to force other users to send at a faster rate by lying in their reports of estimated group size. However, in the Sendmax algorithm, a user can cause other users to report less frequently by sending a large estimated group size in their messages.

2.4 Simulations

We ran a number of simulations to examine the performance of the reconsideration and sendmax mechanisms. We consider the most important performance metric to be the packet send rate $r(t)$, which represents the total number of packets per second injected into the system by the users. A related metric of interest is the learning curve $L(t)$, which indicates the average estimated group size at time t_i. Again, while it is desirable to have $L(t)$ grow to the actual group size as rapidly as possible, it should not do so at a rate greater than $1/C$ packets per second, since this represents the desired RTCP packet rate. We therefore consider $L(t) = t/C$ as the ideal learning curve.

The model implicit in all of our simulations and analysis is depicted in Figure 1. All users are identical in their view of the network, and so we focus on the state evolution as seen by any particular user. Each user is connected to the network via an access link of 28.8 kb/s downstream. We assume upstream links are infinitely fast and never experience congestion. This is reasonable considering that congestion only occurs downstream in real systems. We model the network as providing full connectivity among all users in the group. Multicast join latencies are ignored; this is reasonable in protocols such as DVMRP [9] since initial packets are flooded. The network does introduce a delay of D seconds, where D is uniformly distributed between 0 and 600 ms, with an average of 300 ms. Each user has buffering of 100 kB on the downstream access link. We assume all RTCP packets are 1 kB in size.

Figure 2 and Figure 3 depict state evolution for a single user when 10,000 users simultaneously join a multicast group at $t = 0$. The figures depict the system with no reconsideration (the current specification), conditional reconsideration, and unconditional reconsideration. The graphs are plotted on a log-log scale

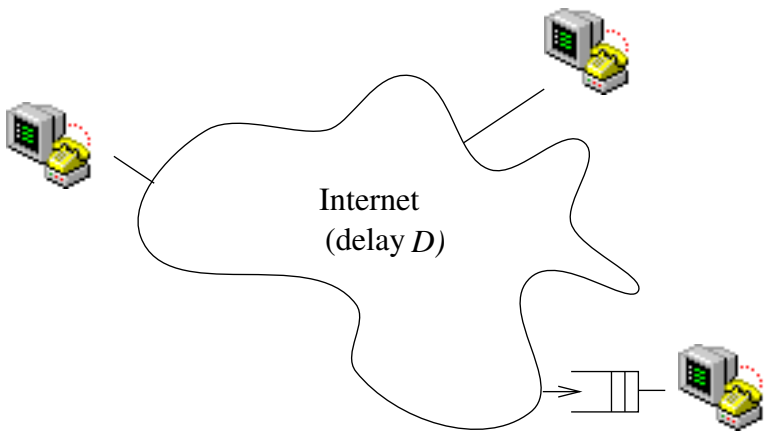

Figure 1: Network Model

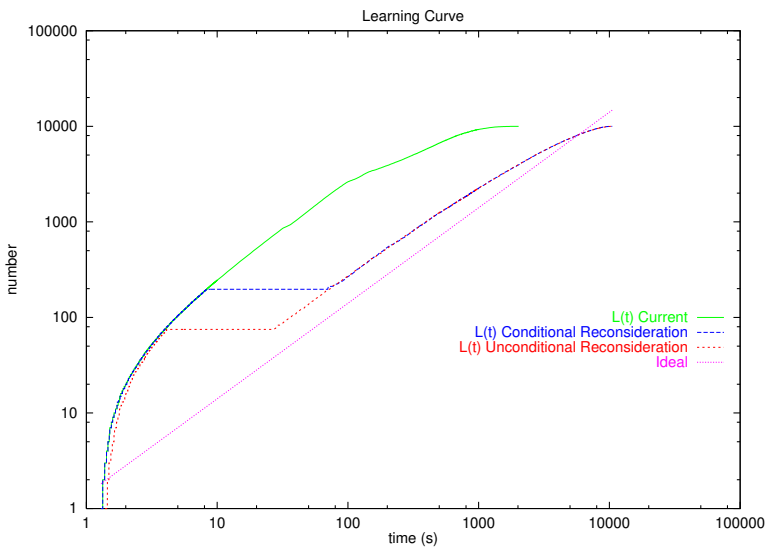

Figure 2: Learning Curve, N=10,000 step join

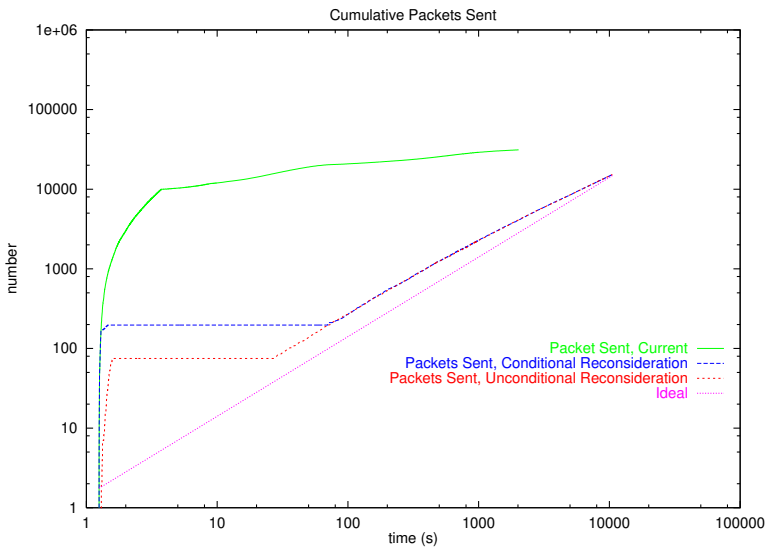

Figure 3: Total Packets Sent, N=10,000 step join

to emphasize the beginning and complete evolution of the system. Figure 2 depicts the learning curve, and Figure 3 shows the integral of $r(t)$, i.e., the total number of packets sent. Note the burst of packets sent in the beginning by the current algorithm. Exactly 10,000 packets are sent out in a 2.5 s interval. This is almost 3000 *times* the desired RTCP packet rate. However, this burst is reduced *substantially* by the reconsideration mechanisms. Conditional reconsideration causes only 197 packets to be sent over a 210 ms interval, and unconditional reconsideration causes merely 75 packets to be sent over a 327 ms interval. We also observed similar startup effects with a variety of different link speeds, delays, and delay distributions.

We noted that the startup burst with reconsideration was particularly disturbing when fixed delay distributions were used. Exponentially distributed delays exhibited nearly identical performance to a uniform distribution. We believe this is because the quantity of interest is the average time until the first few packets arrive. As the number of users grows, this average time decreases to zero for both exponential and uniform delay distributions but remains fixed for a fixed delay distribution.

We also ran simulations to observe performance in linear joins, where groups of users join the system at time Δ seconds apart, for some small Δ. In these cases, we observed that as soon as the number of users who join, times Δ, exceeds the network delays, the initial bursts in the reconsideration algorithms begin to disappear, whereas they remain for the current specification. We demonstrate this analytically in [8]. The conclusion is that the reconsideration algorithms perform even better in the case of even slightly staggered joins.

We have also observed that the reconsideration mechanisms cause a complete pause in packet transmissions after the initial spike. This pause lasts for a time proportional to the number of packets in the spike, giving network buffers time to clear. The phenomenon occurs because the spike of packets in the beginning causes the system to reconsider, and not send, all packets after the spike. A more detailed explanation of the phenomenon is given in Section 2.5.

We found, somewhat surprisingly, that the sendmax algorithm is not that effective. Consider the plot in Figure 4. For this plot, 5000 users join the system at time 0, and another 5000 at time 5000. This represents the "ideal" case where the sendmax algorithm should perform best. The curve depicts the cumulative number of packets sent around the second join time for both conditional reconsideration and conditional reconsideration with sendmax. Note in this case, the burst with conditional reconsideration only is quite small, perhaps 30 packets. This is because the reconsideration mechanism only needs one packet to kick the

algorithm into operation (it requires no packets when in unconditional reconsideration). The initial waiting period of 1.25 s is adequate to see one or two packets from older users, thus avoiding any large spikes. Of course, the sendmax algorithm performs as expected, so there is no burst at all. We noted, however, that the sendmax algorithm causes an increase in the rate further in the future. The reason is that all of the new users believe there are 5000 users, and all of the old users also believe there are 5000. Therefore, 10,000 users will schedule packets in an interval sized for 5000, causing a rough increase in sending rate by a factor of two until convergence.

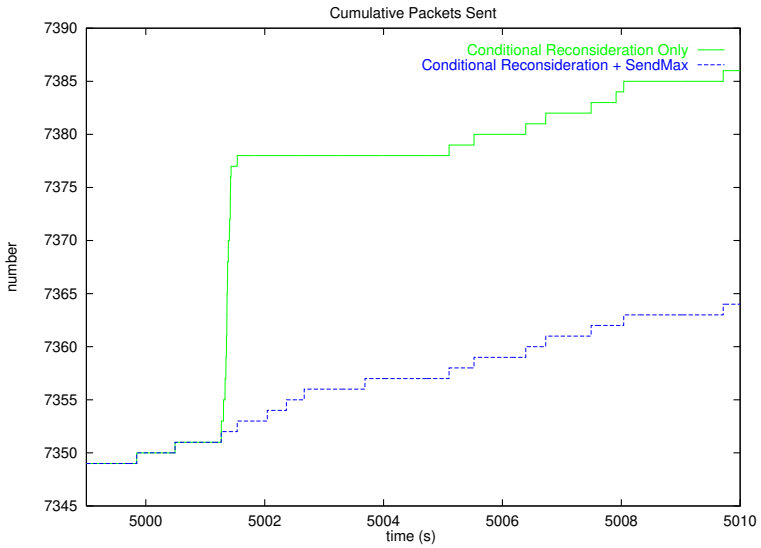

Figure 4: Cumulative packets sent, two step-joins

2.5 Analysis

In this section, we present a mathematical analysis of the reconsideration mechanism when used with a step-join. The analysis results in a differential equation which describes the learning curve. From this differential equation, the packet rate is evaluated.

The system we consider here is one where all of the users join the network at the same time, $t = 0$. It is assumed that the network introduces neither delay nor loss, and that access links have infinite bandwidth. The result is that when a user sends an RTCP packet, it is received by all of the users simultaneously at the time it was transmitted. Although these assumptions are not valid for a real network, they nonetheless provide insight in the behavior of the reconsideration mechanism. Furthermore, we demonstrate in [8] that the addition of bandwidth limitations and delay affects mainly the startup behavior of the system.

We also assume that all users are receivers. In large cable networks, this assumption is generally true, as there is but one sender and many thousands of listeners. Again, we trace state evolution as seen by a particular user. The user estimate has converged when $L(t) = N$, the number of users actually in the group.

Whenever a packet is reconsidered, it is either sent, or it is not, depending on whether the newly computed send time is before or after the current time. We can therefore view the reconsideration mechanism as causing any packet to be sent with some probability P. In the most general case, P is a function of the current time t, the time of the last RTCP report, and the number of users observed at t, $L(t)$. Fortunately, the learning curve is only affected by packets which are initial, that is, sent by users which have not yet sent a packet. For all such users, the last report time is always $t = 0$, so that the send probability is a function of t and $L(t)$ only.

If we consider some small interval of time, the change in $L(t)$ is equal to the number of initial packets scheduled to be sent during this interval, times the probability of sending a packet in that interval. Based on this, we can immediately write the differential equation:

$$\frac{dL}{dt} = d(t)P(t, L(t)), \tag{2}$$

where $d(t)$ is the rate of packets scheduled for transmission during some time interval. What remains is the evaluation of the scheduled rate $d(t)$ and probability $P(t, L(t))$. We first consider the send probability.

Consider an initial packet scheduled to be transmitted by a user at time t. Since the number of perceived users, $L(t)$ has surely changed, this packet is reconsidered. At time t, the user perceives $L(t)$ other users in the system. It thus calculates a new packet interval, which is equal to:

$$T = R(\alpha) \max(T_{\min}, CL(t))$$

Since $CL(t)$ is larger than $T_{\min}$ most of the time, we ignore the $\max$ operator. Keeping in mind that the previous report time is always $t = 0$, we can immediately write the probability of sending a packet using Equation (1):

$$P_{\text{send}} = \frac{t - (1 - \alpha)CL(t)}{2\alpha CL(t)} \tag{3}$$

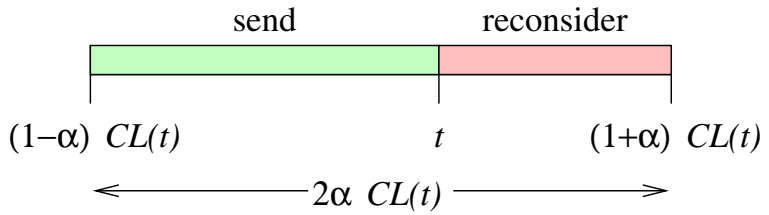

Figure 5: Computing P_{send} with reconsideration

The numerator represents the range of times in the interval widow which fall below the current time t, while the denominator represents the total range over which the times for the interval are selected. This is illustrated in Figure 1. Note that this probability only makes sense when t is between $(1 - \alpha)$ and $(1 + \alpha)$ of $CL(t)$. When t is to the left of the reconsideration window, the probability is zero, and when t is to the right of the window, it is one.

An important implication of this equation is that the send probability is zero when $t > (1 - \alpha)CL(t)$. This places an upper bound on the learning curve; if the learning curve should reach this bound, no initial packets would be sent, and the curve would remain flat until it fell back below this upper bound. We therefore define the *maximum learning curve* $L_{\max}(t)$ to be:

$$L_{\max}(t) = \frac{1}{(1 - \alpha)C}t \tag{4}$$

The actual learning curve $L(t)$ is always below $L_{\max}(t)$.

The next step is to compute the scheduled rate, which is significantly harder. Ideally, the scheduled rate at any point in time is the number of users in the system, N, divided by the average RTCP interval size perceived by those users at time t, namely $CL(t)$. However, there is a delay between the time when the users observe there to be $L(t)$ other users, and when packets are to be sent which were scheduled at t. On average, this delay is $CL(t)$. In other words, at time t, each user perceives $L(t)$ other users. When a packet

is sent at time t, the next RTCP packet is scheduled for time $t + CL(t)$. At that time, the scheduled rate of packets will be $\frac{N}{CL(t)}$. We thus have:

$$d(t + CL(t)) = \frac{N - L(t + CL(t))}{CL(t)}$$

Then, we define t' as the solution to $t = t' + CL(t')$, so that:

$$d(t) = \frac{N - L(t)}{CL(t')}$$

Note that it is always the case that $t' < t$ as long as $L(t) > 0$.

Experimental results presented in previous sections indicate that $L(t)$ is linear for small t (ignoring the transient behavior in the beginning). We therefore attempt to determine the slope of this line based on the differential equation. Substituting $L(t) = mt$ into (2):

$$m = \frac{N - L(t)}{CL(t')} \frac{1 - (1 - \alpha)Cm}{2\alpha Cm}$$

For small t, $L(t) < N$, so we can ignore the L in the first term's numerator. Thus:

$$\frac{2\alpha C^2 L(t')}{N} m^2 + m(1 - \alpha)C - 1 = 0$$

Note that since $t' < t$, and $L(t)$ is non-decreasing, $L(t') \leq L(t)$. Thus, for large N and small t, we can neglect the m^2 term, and we obtain the desired result:

$$m = \frac{1}{(1 - \alpha)C} \tag{5}$$

Not coincidentally, this is also the slope of the maximum learning curve. The equation indicates, therefore, that $L(t)$ grows at its maximum rate until the approximation is no longer valid, at which point its growth tapers off.

With a linear approximation to $L(t)$, we can now approximate the scheduled rate $d(t)$ (see [8] for details):

$$d(t) = \frac{N}{C\frac{1-\alpha}{2-\alpha}L(t)}$$

Plugging this back into the differential equation, the result is a traditional ODE. It turns out that $L(t)$ is the solution to a third order polynomial whose coefficients are powers of t. There exists an analytical solution, but it adds little insight. However, the ODE allows us to compute a numerical solution, which can be compared against the experiments. Figure 6 shows the learning curve, with 10,000 users joining at $t = 0$, for both analysis and simulation. Note that the differential equation is exact for the linear region of $L(t)$, and it is less than, but still close to, the experimental curve as it tapers off. The deviation is due to the approximations used to model the scheduled rate, which are exact only when $L(t)$ is linear.

With an understanding of the behavior of $L(t)$, we are now in a position to discuss the real quantity of interest; the aggregate bit rate generated by these sources as they move towards convergence. We call this quantity $r(t)$. Since the integral of this quantity is the total number of packets sent, we have, as an immediate consequence:

$$r(t) \geq \frac{d}{dt} L(t)$$

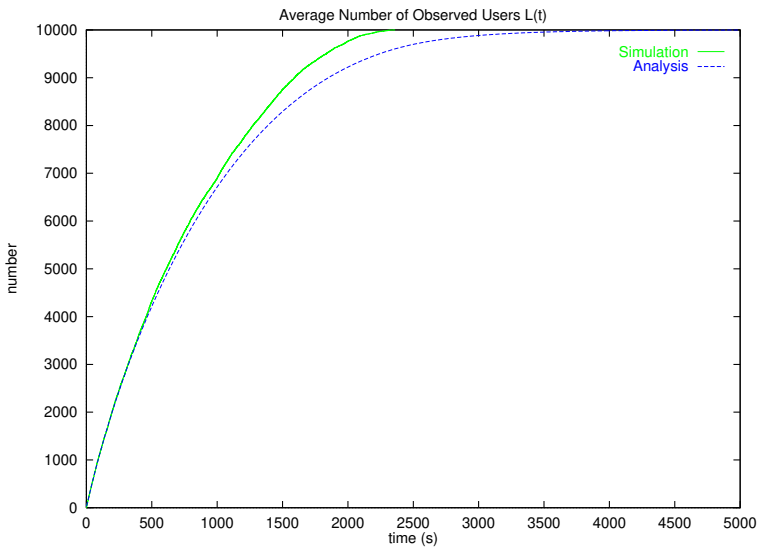

Figure 6: Experimental vs. analytical learning curve

Experimentally, we have observed that $r(t)$ is actually *equal* to the derivative of $L(t)$ for a large fraction of the time until convergence. The reason for this lies with the reconsideration mechanism's "favoring" of packets from users who have not yet sent a packet (initial packets). Consider two packets, both scheduled to be sent at some time t. One is an initial packet, and the other is from a user who has sent a packet previously. For the initial packet, the last report time is at $t = 0$, whereas for the other packet, the last report time is at some time t^*, not equal to zero. In the latter case, the bottom edge of the interval window is at $t^* + C(1-\alpha)L(t)$. Thus, the probability of sending an old packet at time t is:

$$P_{\text{sendold}} = \frac{t - t^* - C(1-\alpha)L(t)}{2\alpha C L(t)}$$

This quantity is *always* less than the send probability for initial packets as given in (3). In fact, for small t, $L(t)$ is equal to $t/C(1-\alpha)$. Plugging this in, we get that the numerator of the fraction is negative, so the send probability is exactly zero.[1]. Therefore, $r(t)$ is exactly equal to the derivative of $L(t)$ while $L(t)$ is linear. We expect it to continue to track the derivative closely even as $L(t)$ tapers off.

Once $L(t)$ has converged to N, packets are sent at a rate of $1/C$. Therefore, r(t) exhibits a dual-constant behavior; it starts at $1/(1-\alpha)C$, stays there for some time, then reduces to $1/C$, where it remains from then on.

Of course, in a real network, delays are an important concern; in fact, they are primarily responsible for the transient congestion conditions seen in Figure 3. We have analyzed the impact of network delays in [8], and we only briefly mention the results here. For conditional reconsideration, the number of packets sent in the transient is roughly:

$$n = \frac{N}{2\alpha T_{\min}} \left(D + \frac{1}{m} \right),$$

where D is the fixed part of any network delays, and m is the link rate in RTCP packets per second. For unconditional reconsideration, the initial transient is much smaller. and it is proportional to the square of the network delays divided by $T_{\min}$, which is substantially less than for conditional reconsideration.

[1]Note that plugging in $L(t) = t/C(1-\alpha)$ to equation (3) yields a numerator of zero, and thus a probability of zero also. In fact, the send probability is zero only in the limit for $N = \infty$; it is slightly positive for all real cases. This is in contrast to the send probability for old packets, which is exactly zero for finite N.

3 Any-to-One Feedback

The preceding algorithms have solved the problem of RTCP packet storms for large groups and allow RTCP sessions to scale to very large sessions, based on packet rates alone.

Nevertheless, for large media distribution applications ("Internet cable TV"), multicast-based feedback has significant drawbacks:

- Since the interval between RTCP packets from a single source or single network is likely larger than the prune time-out, each RTCP packet will be flooded to all subnets if DVMRP is used.
- Since the learning rate of new modem-connected members at steady-state is only about 1 member/s, the estimation of the group size for most interesting cases is very inaccurate, particularly if users "channel surf".
- Even though the content of an RTCP packet can be encrypted with the public key of the monitor, addresses of listeners are still revealed to all members of the group. Using finger, it may be possible to link the IP address to individuals. (Note that encrypting SDES information also defeats parts of the RTP loop-detection information. The RR SSRC has to be in the clear to allow members to estimate group sizes.)
- If only a single receiver cares about receiver feedback, RTCP multicast creates unnecessary packet load in the network. In the "Internet TV" case, the server connection to the network is likely to be larger than a modem connection, so that it can handle a higher feedback rate than would be tolerable for the multicast case.

As an alternative to any-to-any feedback, the session monitor can send out requests to the group members to respond. Unless it is expected that only a single or very small number of receivers will respond [10], the request must contain an indication of a randomization interval spread out over which the receivers are to respond.

The polling request is most easily based on the SSRC identifier, which is known to be a random 32-bit integer. In the simplest poll, the monitor sends an n-bit number. If a receiver's uppermost n bits of the SSRC or the random number falls below that number, it considers itself selected and responds to the monitor. Alternatively, the poll can contain a bit pattern and a mask indicating which bits have to match or a range of numbers. With the latter two methods, it is easy to select many different receiver subsets of roughly the same size for each poll.

Repeated polling of the same receivers allows to judge the distribution in time of network impairments, that is, the monitor can distinguish if a single receiver saw a 10% loss over one minute or 1% loss over 10 minutes, with the perceptual results likely being a loss of usable reception in the first case and barely degraded reception quality in the latter.

A 16-bit threshold or mask together with a suitable long interval is probably sufficiently large even for $O(10^6)$ receivers, with only $O(15)$ responses at the smallest threshold.

It is very easy to estimate the accuracy of estimating the number of receivers based on a poll, as the number of responses is distributed as a Bernoulli random variable, with N tries and success probability $p = T/2^{32}$. If more than a few responses are expected, the DeMoivre-Laplace theorem [11, p. 49] tells us that the distribution of the number of responses will be approximately normally distributed with variance $Np(1-p)$ and mean Np as N grows larger.

Any polling scheme has a number of disadvantages:

- If the session size increases dramatically between polls, the polling probability p may be too large or the response interval R too small and cause excessive packet load. For the typical high-bandwidth

server and unicast responses, this may not be as dramatic as the consequence of multicast flooding mentioned earlier. This remains, however, the most serious limitation of any polling scheme. For some protection against prolonged packet storms, we can add an "emergency brake" by having the monitor send a poll with threshold zero and modifying receiver behavior such that a new poll stops any pending responses and re-starts the algorithm.

- As described, there can only be a single monitor. Clearly, it would be possible to have receivers multicast their responses instead, with increased danger of unleashing packet storms, both intentional (see next item) and accidental.
- It is easy to create a denial-of-service attack by directing a large percentage of receivers to send packets to a chosen host. Since the request source address is easily forged, it does not matter whether receivers only respond to the source of the poll or whether we allow "third-party polls". In any such scheme, receivers must also respond to polls that they can authenticate.

4 Group Size Estimation

Estimating large group sizes is difficult for two reasons. Using the current RTCP mechanism, it takes a very long time until a single receiver has heard from all other receivers. For the modem case, the learning rate is only about one user per second. Secondly, a receiver would have to remember all SSRC values to make sure that a report from the same SSRC is not counted twice.

One answer might be to estimate the group size by only looking at a single source. One could imagine, for example, that a receiver would not send a report until it has heard from the same source twice. In that interval, it would count the number of reports received, and then use that as an estimate of the group size. That would ensure that a new arrival would immediately adjust its report rate to somewhat less than the steady-state rate. Unfortunately, the algorithm has some drawbacks. First, it is not safe. A malicious user can send packets close together, fooling others into believing the interval is shorter (and thus the group size smaller) than it really is. Furthermore, there is no guarantee that the source chosen ever sends twice; the source may leave the group before sending the second packet.

Another approach which avoids having to keep state for each member in the group is to count only a selected subset of the SSRC space and extrapolate the total population based on that. A receiver would set an upper bound on the number of distinct SSRCs that it wants to track. If this number is being reached, it reduces the probability of accepting a new SSRC. It also removes all sources that do not meet the new, stricter matching criterion. If the number of sources drops below a threshold, the acceptance probability is lowered again. After lowering the acceptance probability, the estimate should, however, still be no lower than that computed using the old, higher acceptance probability. [8] contains details on an efficient implementation which avoids sorting.

5 Summary

RTP was meant to support real-time communications ranging from two-party telephone calls to broadcast applications with very large user populations. It incorporates an adaptive feedback mechanism that allows scaling to moderately sized groups, but shows a number of deficiencies once the group size exceeds on the order of a thousand. The problems can be summarized as initial transient, convergence delays and state storage problems. Combining the algorithms proposed, we have solved all three problems. Reconsideration prevents large bursts of traffic when the user population changes rapidly and allows any-to-any feedback to be used, if with decreasing information content, for groups of any size. This proposal is backward compatible with the current specification. If it is important to estimate group sizes quickly, a polling mechanism

may be preferable. We described an adaptive mechanism where the polling agent can trade of accuracy and traffic. Finally, instead of maintaining state for each of possibly millions of session members, a simple adaptive sampling technique allows to estimate group sizes based only on a few hundred entries. Polling and the any-to-any feedback may be used together or as alternatives. All algorithms are applicable to generic learning and bandwidth-sharing applications. One could, for example, imagine using reconsideration for having a large number of constant-rate sources fairly share a single broadcast medium (e.g., a wireless network or an Ethernet-like LAN).

The algorithms are currently being implemented as part of a generic RTP library.

6 Acknowledgments

The authors wish to thank Steve Casner, Sanjay Mithal, and T.V. Lakshman for their insightful comments and discussion.

References

[1] S. Deering, D. Estrin, D. Farinacci, M. H. A. Helmy, V. Jacobson, C. Liu, P. Sharma, D. Thaler, and L. Wei, "Protocol independent multicast-sparse mode (PIM-SM): Motivation and architecture," Internet Draft, Internet Engineering Task Force, Oct. 1996. Work in progress.

[2] A. Ballardie, S. Reeve, and N. Jain, "Core based trees (CBT) multicast - protocol specification," Internet Draft, Internet Engineering Task Force, Sept. 1996. Work in progress.

[3] M. Grossglauser and K. Ramakrishnan, "SEAM: A scheme for scalable and efficient ATM multipoint-to-multipoint communication," Contribution 96-1142, ATM Forum, 1996. To appear in Infocom'97.

[4] S. McCanne, V. Jacobson, and M. Vetterli, "Receiver-driven layered multicast," in *SIGCOMM Symposium on Communications Architectures and Protocols*, (Palo Alto, California), Aug. 1996.

[5] H. Schulzrinne, S. Casner, R. Frederick, and V. Jacobson, "RTP: a transport protocol for real-time applications," Request for Comments (Proposed Standard) RFC 1889, Internet Engineering Task Force, Jan. 1996.

[6] S. Casner and V. Jacobson, "Compressing IP/UDP/RTP headers for low-speed serial links," Internet Draft, Internet Engineering Task Force, Nov. 1996. Work in progress.

[7] B. Aboba, "Alternatives for enhancing RTCP scalability," Internet Draft, Internet Engineering Task Force, Jan. 1997. Work in progress.

[8] Anonymous, "Scaling feedback to very large groups," tech. rep., Withheld for Anonimity, Feb. 1997.

[9] T. Pusateri, "Distance vector multicast routing protocol," Internet Draft, Internet Engineering Task Force, Sept. 1996. Work in progress.

[10] J.-C. Bolot, T. Turletti, and I. Wakeman, "Scalable feedback control for multicast video distribution in the internet," in *SIGCOMM Symposium on Communications Architectures and Protocols*, (London, England), pp. 58–67, ACM, Aug. 1994.

[11] A. Papoulis, *Probability, Random Variables, and Stochastic Processes*. New York, New York: McGraw-Hill Book Company, 2nd ed., 1984.