Internet-Draft
draft-ietf-avt-reconsider-00.ps
Expires: January 1998

Jonathan Rosenberg
Lucent Technologies, Bell Laboratories
Henning Schulzrinne
Columbia University
July 1997

Timer Reconsideration for Enhanced RTP Scalability [1]

Filename: draft-ietf-avt-reconsider-00.ps

Status of This Memo

This document is an Internet-Draft. Internet-Drafts are working documents of the Internet Engineering Task Force (IETF), its areas, and its working groups. Note that other groups may also distribute working documents as Internet-Drafts.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress".

To view the entire list of current Internet-Drafts, please check the "lid-abstracts.txt" listing contained in the Internet Drafts Shadow Directories on ftp.is.co.za (Africa), ftp.nordu.net (Europe), munnari.oz.au (Pacific Rim), ds.internic.net (US East Coast), or ftp.isi.edu (US West Coast).

Abstract

RTP, the Real Time Transport Protocol, has gained widespread acceptance as the transport protocol for voice and video on the Internet. It provides services such as timestamping, sequence numbering, and payload identification. It also contains a control component, the Real Time Control Protocol (RTCP), which is used for loose session control, QoS reporting, and media synchronization, among other functions. The RTP specification describes an algorithm for determining the RTCP packet transmission rate at a host participating in a multicast RTP session. This algorithm was designed to allow RTP to be used in sessions with anywhere from one to a million members. However, we have discovered several problems with this algorithm when used with very large groups with rapidly changing group membership. One problem is the flood of RTCP packets which occurs when many users join a multicast RTP session at nearly the same time. To solve this problem, we present a novel adaptive timer algorithm called reconsideration. We present a mathematical analysis of this algorithm, and demonstrate that it performs extremely well, reducing the congestion problem by several orders of magnitude. We also back up these results with simulation.

[1] Note that this paper has also been submitted to Infocom 98

1 Introduction

There has recently been a flood of interest in the delivery of multimedia services on the Internet. The growing popularity of Internet telephony, streaming audio and video services (such as those provided by Real Audio) and the Mbone are all indicators of this trend. To support these applications, standards are being developed to insure interoperability. The ITU-T H.323 [1] specification for Internet telephony is gaining widespread acceptance among software vendors. The IETF is developing protocols such as SIP [2] for multimedia session initiation, and RTSP [3] for controlling multimedia servers on the Internet.

Interwoven with all of the above protocols is the Real Time Transport Protocol (RTP) [4]. It is used by H.323 terminals as the transport pprotocol for multimedia; both SIP and RTSP were designed to control multimedia sessions delivered over RTP. Its main function is to carry real time services, such as voice and video, over an IP network. It provides payload type identification so that the receiver can determine the media type contained in the packet. Sequence numbers and timestamps are also provided, so that packets can be reordered, losses can be detected, and data can be played out at the right speeds. RTP was designed to be easily used in multicast conferences. To this end, it guarantees that each participant in a session has a unique identifier called the synchronization source (SSRC). This identifier is carried in each packet, providing applications a way to de-multiplex packets from different users.

RTP also contains a control component, called the Real Time Control Protocol (RTCP). It is multicast to the same multicast group as RTP, but on a different port number. Both data senders *and* receivers periodically multicast RTCP messages. RTCP packets provide many services. First, they are used to identify the users in a session. One RTCP packet type, the Source Descriptor (SDES) contains the name, email address, telephone number, fax, and location of the participant. Another, the receiver report, contains reception quality reporting. This information can be used by senders to adapt their transmission rates or encodings dynamically during a session [5]. It can also be used by network administrators to monitor network quality [6]. It could potentially be used by receivers to decide which multicast groups to join in a layered multimedia session; such an application is similar to RLM [7]. Yet another RTCP packet type, the sender report (SR), is used to aid receivers in inter-media synchronization (lip sync), and to indicate transmitted bit rates, among other functions.

Since RTP was designed for multicast, it was engineered to work well with both large and small sessions. A typical “small” session might be a teleconference among five business executives, while a typical “large” session might be an Mbone broadcast of a shuttle launch, where group sizes of two hundred listeners have been reported [8]. As the demand for multimedia continues to grow, larger and larger group sizes will become commonplace. It is not difficult to envision Mbone concert broadcasts with thousands of members. It has even been suggested that RTP might be the transport protocol of choice for multicast distribution of multimedia in future cable networks, where tens of thousands of users might be the norm.

The principle difficulty in achieving scalability to large group sizes is the rate of RTCP packet transmissions from a host. If each host sends packets at some fixed interval, the total packet rate sent to the multicast group increases linearly with the group size, N. This traffic would quickly congest the network, and be particularly problematic for hosts connected through low speed dialup modems. To counter this, the RTP specification requires that end systems utilizing RTP listen to the multicast group, and count the number of distinct RTP end systems which have sent an RTCP packet. This results in a group size estimate, L computed locally at each host. The interval between packet transmissions is then set to scale linearly with L. This has the effect of keeping the total traffic in the multicast group constant, independent of the number of group members.

The above scaling mechanism works well for small to medium sized groups (up to perhaps a few hundred members). However, it suffers from problems when applied to larger groups, particularly ones whose group membership is dynamic. We have identified three inter-related problems which arise with large, dynamic multicast groups:

- *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 [9] is used). We also anticipate that many multicast RTP sessions will exhibit rapid increases in group membership at certain points in time. This can happen for a number of reasons. Many sessions have precise start times. Multimedia tools such as vat and vic can be programmed to join a session at the instant of its inception. Even without automation, users are likely to fire up their applications around the time the session is scheduled to begin. Such phenomena are common in current cable networks, where people change channels when shows begin and end. Studies have been performed to look at the group membership over time of some of the popular sessions on the Mbone [10][8]. These studies show exactly this kind of "step-join" behavior. The result of these step joins are inaccuracies in the group size estimates obtained by listening to the group. Each newly joined member believes that they are the only member, at least initially, and begins to send packets at a relatively fast rate. Combined with slow access links, the result is a flood of RTCP reports, causing access link congestion and loss.

  For example, consider an RTP session where the total RTCP rate is to be limited to 1 kb/s. If all RTCP packets are 1 kbit, packets should be sent at a total rate of one per second. Under steady state conditions, if there are 100 group members, each member will send a packet once every 100 seconds, and everything works. However, if 100 group members all join the session at about the same time, each thinks they are initially the only group member. Each therefore sends packets at a rate of 1 per second, yielding an aggregate rate of 100 packets per second, or 100 kb/s, into the group.

- *State Storage*: In order to estimate the group size, hosts must listen to the multicast group and count the number of distinct end systems which send an RTCP packet. To make sure an end system is counted only once, its unique identifier (SSRC) must be stored. 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 memory is limited.

- *Delay*: As the group sizes grow, the time between RTCP reports from any one particular user becomes very large (in the example above, with 3000 group members, each would get to send an RTCP packet about once an hour). This interval may easily exceed the duration of group membership. This means that timely reporting of QoS problems from a specific user will not occur, and the value of the actual reports is lost.

In this paper, we consider only the first problem, that of *congestion*. It is our aim to solve this problem with a single mechanism, applicable to large groups and small alike. It is possible to develop solutions which work well for specific applications. For example, RTCP reporting can be disabled completely [11]. This, in fact, solves all three of the above problems. However, many applications require RTCP, and therefore this approach is not a general one. Another alternative is to use hierarchical summarizers. This helps improve convergence time, and may relieve congestion, but it requires special servers to be deployed. It is also not appropriate for small groups, and therefore does not work well as a universal solution to the congestion problem.

There has been a small amount of prior work on resolving difficulties with timers in Internet protocols. Most prominent among this work is [12] and [13]. Sharma et. al. consider how to scale soft state timers to varying link capacities and state quantities. Their work considers only the point to point case. In that scenario, any sharp increases in the amount of state to send (which is equivalent to the sharp increases of group membership we consider here) are known instantly by the sender, since all of the state resides there. The congestion problem which we treat here arises due to the *distributed* nature of the system and the multicast interconnect. In that scenario, a rapid change in group membership is represented by a change in group state distributed across many nodes. As such, our work can be viewed as a generalization of their's to distributed multicast groups.

In fact, our algorithm for controlling the congestion problem in RTP is applicable to other protocols and systems. An extension to the Service Location Protocol [14] has been proposed [15] which uses the reconsideration algorithm to control congestion in the multicast group used to disseminate information on network services. The algorithm is generally applicable to distributed systems where (1) control of bandwidth is desirable, (2) the bandwidth is used to transmit state, (3) the state is used to determine end system transmission rates, and (4) the state is dynamic. These constraints apply to BGP [16], for example, when a route server is used and update rates are to be controlled.

The remainder of the paper is organized as follows. In Section 2, we detail the current RTCP packet transmission algorithm. Section 3 describes the desired ideal behavior. Section 4 describes our solution, an algorithm called timer reconsideration, and shows its performance. Section 5 then analyzes the algorithm to provide more insight into its performance. Section 6 discusses the algorithms performance under steady state conditions, and Section 7 summarizes our work.

2 Current RTCP Algorithm

Each user i in a multicast group using RTP maintains a single state variable, the *learning curve*, which we denote by $L(t)$. 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. In order to avoid network congestion, the total rate of RTCP reports multicast to the group, summed across all users, is set at 5% of the total multicast session bandwidth (it is assumed in RTP that this quantity is known apriori). We define C as the average interval between arrivals of RTCP packets, across all users, into the group, so that C is the average RTCP packet size divided by 5% of the session bandwidth. To meet this criteria, each user computes a *deterministic interval*, which represents the nominal interval between their own packet transmissions required to meet the 5% constraint. This interval is given by [2]:

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

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. Assume a user joins at time $t = 0$. The first packet from that user is scheduled at a time uniformly distributed between 1/2 and 3/2 of T_{min} (which is 2.5 s for the first packet), putting the first packet transmission time 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:

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

where we have defined $R(\alpha)$ as a random variable uniformly distributed between $(1-\alpha)$ and $(1+\alpha)$. (α equals 1/2 in the current algorithm; we generalize because α has a strong impact on transient behavior). A pseudo-code algorithm describing the behavior upon expiration of the interval timer is given in Figure 1.

The difficuly arises when a large number (say, N) of users all join the group at the same time. We call this a *step-join*. Since all users start out with $L(t) = 1$, all schedule their first packet to be sent between $t = 1.25$ and $t = 3.75$, a fixed, 2.5 second interval. The result is a flood of N packets for 2.5 s, many of which are lost if the access bandwidth is low. Since group size estimates are based on the reception of these packets, losing them will continue to cause each user to have a low estimate of the actual group size. This will cause continued congestion until enough packets get through to make the group size estimates correct.

[2] In actuality, the RTP specification allocates 75% of the RTCP bandwidth to data senders, and the remaining 25% to listeners. In the remainder of the paper, we assume that everyone is a listener. This simplifies the analysis and simulations, all of which can be trivially extended to include the case where there are senders.

```
new_interval = C * current_group_size_estimate;
new_interval = max(new_interval, T_min);
new_interval = new_interval * random_factor;

send_packet();
schedule_timer(current_time + new_interval);
```

Figure 1: Current RTCP Algorithm

3 Ideal Behavior

The flood of packets caused by the current RTCP algorithm with a step join has both good and bad consequences. Clearly, the congestion which results is not desirable. However, the flood allows the end systems to very rapidly learn about the group sizes and group membership, which is desireable. There is a fundamental and unavoidable tradeoff between the convergence time (i.e., the time until the observed group size $L(t)$ equals the actual group size) and the bandwidth used to achieve convergence. What, then, represents the behavior which is desirable?

Our approach is to define the ideal behavior as the one where feedback into the group never exceeds its specified threshold (5% for RTCP). This implies that convergence times will grow as the group sizes grow. However, it is the most social solution, in the sense that it will never congest the network, no matter how large the group sizes become. If we define the ideal behavior as convergence within any amount of time that grows less than linearly with the group size, the result is a protocol that does not scale and can eventually result in congestion.

We also consider congestion avoidance to be more important because we expect many users to be connected via low speed dialup lines. In that case, bandwidth is at a premium, and it is in the self-interest of users to make the best use of it. Most users probably consider RTCP feedback much less important than the video or audio data itself, and therefore it is important to keep the feedback below the required 5%.

We now state the desired ideal behavior:

1. The learning curve $L(t)$ grows linearly at a rate of C users per second, until it reaches the group size N, at which point it becomes flat, and remains at N.

2. The bandwidth used by all feedback is always equal to C packets per second during the convergence period.

```
new_interval = C * current_group_size_estimate;
new_interval = max(new_interval, T_min);
new_interval = new_interval * random_factor;

if ((last_transmission + new_interval < current_time) ||
  (current_group_size_estimate ≤ previous_group_size_estimate)) {
    send_packet();
    schedule_timer(current_time + new_interval);
    last_transmission = current_time;
    previous_group_size_estimate = current_group_size_estimate;
}
else {
    schedule_timer(last_transmission + new_interval);
    previous_group_size_estimate = current_group_size_estimate;
}
```

Figure 2: Conditional Reconsideration

4 Reconsideration

Our contribution is a new solution which we call *reconsideration*. The effect of the algorithm is to reduce the initial flood of packets which occur when a number of users simultaneously join the group. 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*. This means that the user recomputes the RTCP interval (including the randomization factor) based on the current state (call this new interval 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 $t_{n-1} + T'$. In other words, the state at time t_n gives us potentially new information about the group size, compared to the state at time t_{n-1}. Therefore, we redo the interval computation that was done previously at time t_{n-1}, but using the new state. If the resulting interval would have caused the packet to be scheduled before the current time, we know that our interval estimate was not too low. If, however, the recomputation pushes the timer off into the future, we know that our initial timer estimate was computed incorrectly, and we delay transmission based on our new timer. A pseudo-code specification of the algorithm is given in Figure 2.

Intuitively, this mechanism should help alleviate congestion by restricting the transmission of packets during the convergence periods, where the perceived group sizes $L(t)$ are rapidly increasing.

In unconditional reconsideration, the user reconsiders independently of whether the number of per-

```
new_interval = C * current_group_size_estimate;
new_interval = max(new_interval, T_min);
new_interval = new_interval * random_factor;

if (last_transmission + new_interval < current_time)
{
    send_packet;
    schedule_timer(current_time + new_interval);
    last_transmission = current_time;
}
else {
    schedule_timer(last_transmission + new_interval);
}
```

Figure 3: Unconditional Reconsideration

ceived users has changed since the last report time. Thus, the RTCP interval is always recomputed, added to the last transmission time t_{n-1}, and the packet is only sent if the resulting time is before the current time. Clearly, when the group sizes are increasing, this algorithm behaves identically to conditional reconsideration. However, its behavior differs in two respects. First, consider the case where we have converged, and group sizes are no longer changing. In conditional reconsideration, no timer recomputation is done. But for unconditional, it is redone. Since group sizes have not changed, the deterministic part of the interval remains the same. However, the random factor is redrawn each time. This means that packets will be transmitted when the recomputed random factor is smaller than the previous factor, and packets will be delayed when the recomputed random factor is greater than the previous one. Note that since the random factor is of finite extent (between 1/2 and 3/2), packets are guaranteed to eventually be sent. However, the result is an average increase in the interval between RTCP packets.

The behavior of unconditional reconsideration differs during the initial transient as well. Consider N users who simultaneously join the group at time 0. They all schedule their first RTCP packets to be sent between $t = 1.25$ and $t = 3.75$. The users whose packets were scheduled earliest (at a time a little bit after $t = 1.25$) will not reconsider with conditional reconsideration, and will always send their packets. This is because no one else has sent any packets yet, and thus they have not perceived the group size to have changed. In fact, because of network delays, many users may send packets without reconsidering. Once the first transmitted packet has reached the end systems, conditional reconsideration "kicks in", since users will perceive a change in group size only then. With unconditional reconsideration, those first few users do not wait for the first packet to arrive before using the reconsideration algorithm. They will all recompute the timer. Obviously, the group size estimate hasn't changed, but the random variable will be redrawn. For the first few users, the random factor was initially extremely small (that's why they are the first few users to send). In all likelihood, when the factor is redrawn, it will be larger than the initial factor, and thus the resulting interval will be larger. This will delay transmission of RTCP packets for those users. As time goes on, it becomes less likely than the new random factor will be greater than the initial one. However, by then, any RTCP packets which may have been sent will begin to arrive, increasing the group size estimates for each user. In this fashion, unconditional reconsideration alleviates the initial spike of packets which are present in conditional reconsideration. These arguments are all quantified in later sections.

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.

4.1 Simulations

We ran a number of simulations to examine the performance of the reconsideration algorithms.

In our simulation model each user is connected to the network via an access link of 28.8 kb/s downstream (i.e., from the network to the user). We assume upstream links are infinitely fast, since congestion occurs only downstream. This congestion is due to the RTCP reports from all of the other users being sent to any particular user. Multicast join latencies are ignored; this is reasonable in protocols such as DVMRP [17] since initial packets are flooded. We assume that the network introduces a delay of D seconds, where D is uniformly distributed between 0 and 600 ms. Each user has a 100 kB buffer on the downstream access link. We assume all RTCP packets are 128 bytes in size.

Figure 4 and Figure 5 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, unconditional reconsideration, and the ideal behavior. The graphs are plotted on a log-log scale to emphasize the beginning and complete evolution of the system. Figure 4 depicts the learning curve, and Figure 5 shows the integral of $r(t)$, i.e., the total number of packets sent, given that $r(t)$ is the packet transmission rate into the multicast group. 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 improvements with a variety of different link speeds, delays, and group memberships.

We noted that the startup burst with reconsideration was particularly disturbing when network delays were deterministic instead of uniformly distributed. This is demonstrated in Figure 6, which looks at the cumulative number of packets sent when $10,000$ users simultaneously join at $t = 0$, using conditional reconsideration. In all cases, the mean network delay was 300ms, but the distribution varies. Exponentially distributed network delays exhibited nearly identical performance

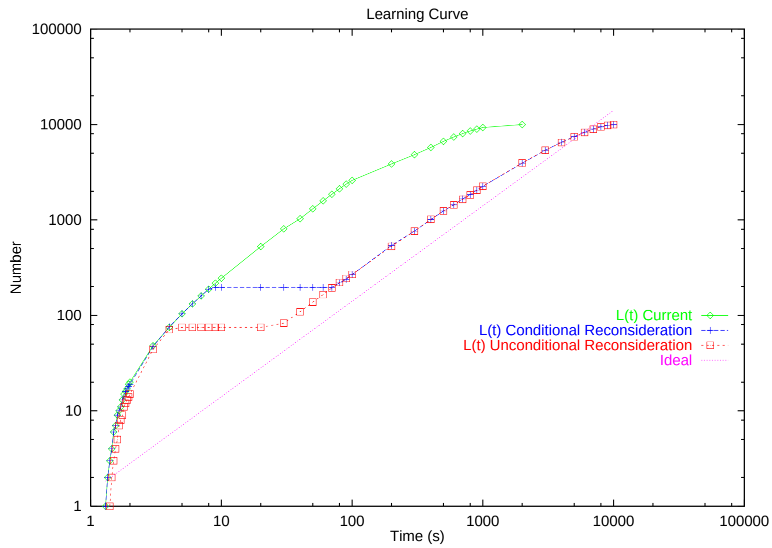

Figure 4: Learning Curve, step join with N=10,000

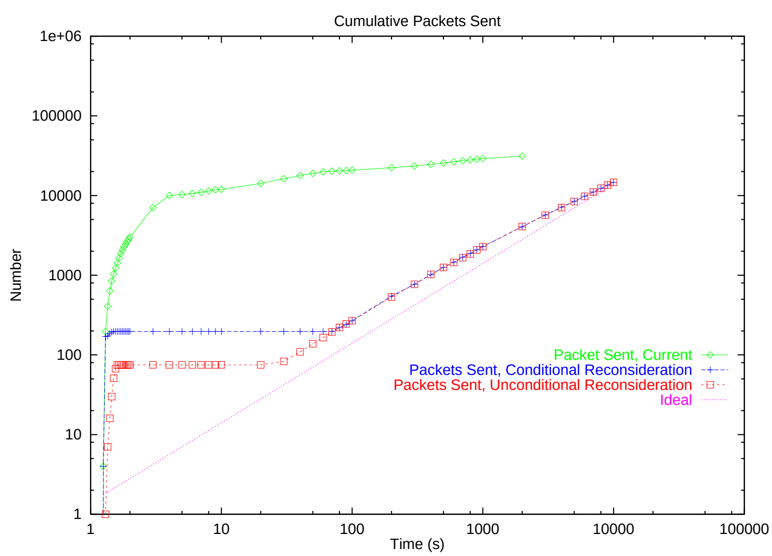

Figure 5: Total Packets Sent, step join with N=10,000

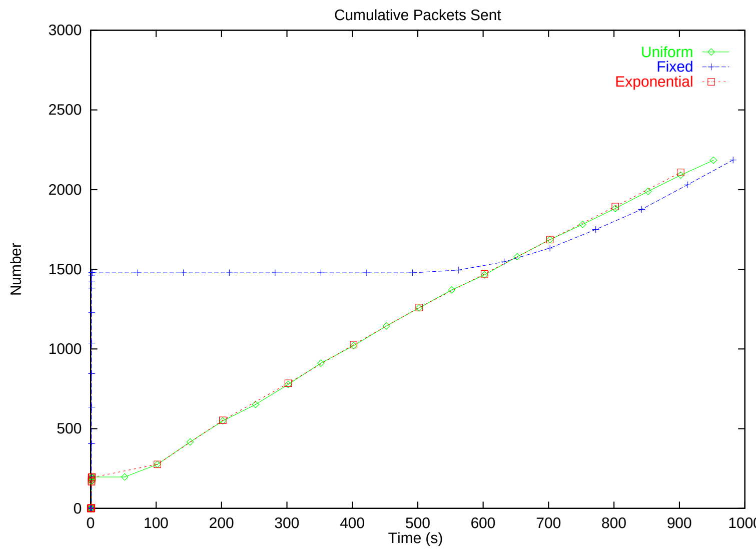

Figure 6: Effect of Delay Distribution on Transient for Conditional Reconsideration

to a uniform distribution. Later sections will demonstrate that the spike is dependent on the amount of time until the first packet arrives. As the number of users in the step join becomes large, the number of users who send their packets within the first ϵ seconds after $t = 1.25$ grows large for any ϵ. Consider an ϵ much smaller than typical network delays, say 10 μs. As far as computing arrival times at end stations, these packets can be treated as though they were all sent at the same time. The amount of time until the first of these packets arrives at any end system is thus the *minimum* network delay experienced by all of those packets. If the network delays are exponential, the expected minimum of N exponential random variables goes to zero as N grows. The same is also true for a uniform random variable. For a deterministic variable, this is not the case; the minimum is always the same. Therefore, the performance is worse for network delays which are fixed.

We have also observed that the reconsideration mechanisms cause a complete pause in packet transmissions after the initial spike. This pause (which we call the "plateau effect") lasts for a time proportional to the number of packets in the spike. This has both positive and negative implications. On the plus side, it gives network buffers time to clear. However, it also causes the send rate to deviate from our desired fixed $1/C$ packets per second. 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 5. However, after the spike and plateau, the packet rate behaves fairly well, sending packets at a nearly constant rate.

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. The results are shown in Figure 7 and Figure 8. Both plots depict the cumulative number of packets sent by all users. The simulation parameters

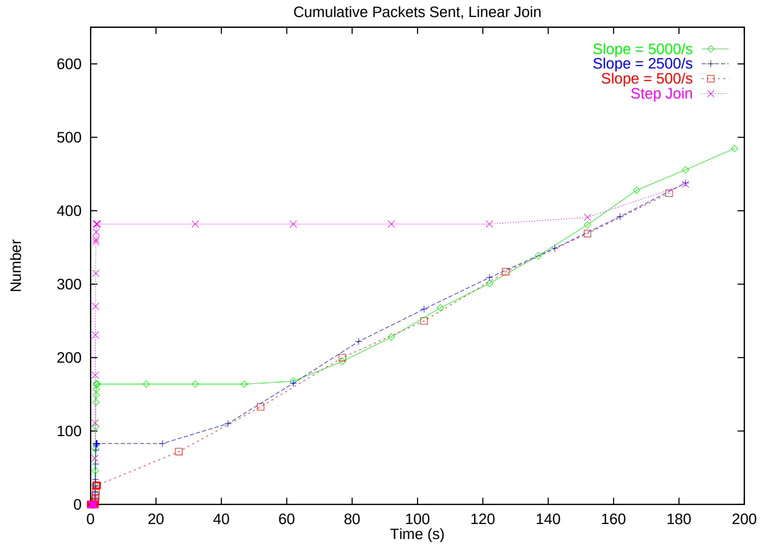

Figure 7: Linear Joins: Conditional Reconsideration

are identical to the above cases, except network delays are deterministic 300 ms. The first plot depicts conditional reconsideration, and the second, unconditional. In all cases, 2500 users join the system, but the rate that they do so is varied. Both plots depict the step join, and joins at a rate of 5000, 2500, and 500 users per second. The plots indicate that linear joins quickly eliminate the initial transient of packets and the plateau period, with the reduction being better for unconditional reconsideration.

We have done some analysis to examine how the behavior of reconsideration changes under linear joins. Our analysis has shown 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. All other aspects of the system performance (including long term growth of $L(t)$) are identical to the step-join case.

5 Analysis

In this section, we present a mathematical analysis of the reconsideration mechanism. We first consider the case where there are no network delays. This results in a differential equation which describes the learning curve. The analysis also applies to networks with delay, but only models the post-transient behavior of the system. However, this is sufficient to compute the post-transient packet rate and system convergence times. We then extend this analysis to the case of network delays, and derive expressions which describe the transient spikes and plateaus in the learning curve. We also analytically demonstrate the reasons for improved performance from unconditional reconsideration, which only exists in the presence of network delays.

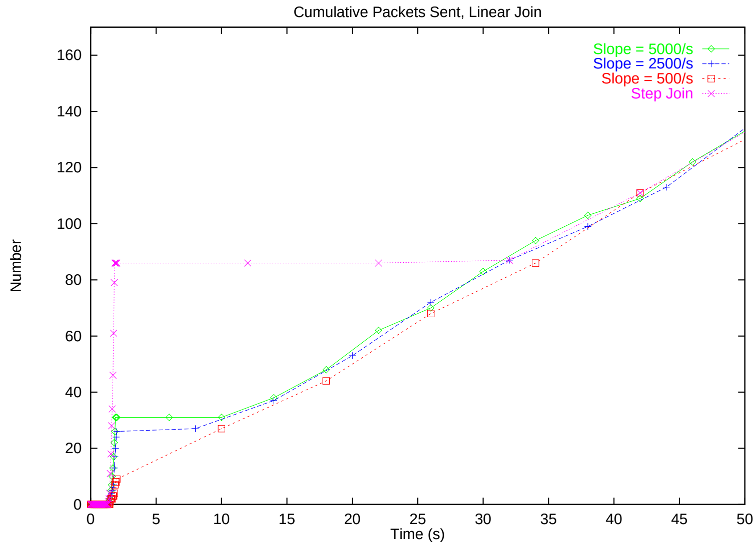

Figure 8: Linear Joins: Unconditional Reconsideration

5.1 No Delay

We consider a system 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.

In the model considered here, all users will have exactly the same state (in terms of $L(t)$) at all times. Thus, 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 initialized to $t = 0$, 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

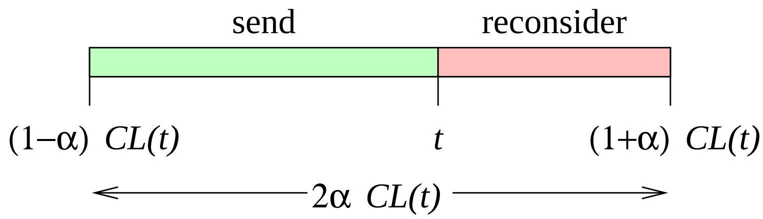

Figure 9: Computing P_{send} with reconsideration

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 over any time interval, this packet is reconsidered [3]. 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):

$$P_{\text{send}} = \tfrac{t-(1-\alpha)CL(t)}{2\alpha CL(t)} \quad (1-\alpha)CL(t) < t < (1+\alpha)CL(t) \tag{3}$$

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 9. 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. In the ideal case, the rate that packets have been scheduled at should equal the number of users in the system, N, divided by the average RTCP interval size perceived by those users at time t, namely $CL(t)$. At any point in time the fraction of packets to be sent which are initial is $(N-L)/N$. Thus, the

[3] It is for this reason that we make no distinction between conditional and unconditional reconsideration here

scheduled rate of initial packets is roughly given by:

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

The curves of Figure 4 show that the reconsideration algorithms exhibit linear behavior between roughly $t = 100$ and $t = 9000$ (thus ignoring the transient behavior in the beginning few seconds). We therefore attempt to determine the slope a of this line based on the differential equation. Substituting $L(t) = at$ into (2):

$$a = \frac{N - L(t)}{CL(t)} \frac{1 - (1-\alpha)Ca}{2\alpha Ca}$$

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

$$\frac{2\alpha C^2 L(t)}{N} a^2 + a(1-\alpha)C - 1 = 0$$

Thus, for large N and small t, $L(t) \ll N$, and we can neglect the a^2 term, and obtain the desired result:

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

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.

As mentioned previously, the equation for the scheduled rate $d(t)$ is very approximate. We have done some more extensive analysis, and found that a slightly better approximation is given by:

$$d(t) = \frac{N - L(t)}{C\frac{1-\alpha}{2-\alpha}L(t)} \tag{6}$$

This is of the same form as the previous equation, but tends to model the nonlinearities of the system better.

Now, with the density and send probabilities computed, we can write the final differential equation, which is:

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

This ODE allows us to compute a numerical solution, which can be compared against the simulations. Figure 10 shows the learning curve, with 10,000 users joining at $t = 0$, for both analysis and simulation. In the simulation, however, we take into account non-zero delays and finite link speeds; network delays are a deterministic 300 ms, and link speeds are 28.8 kbps. Note that despite this change in assumptions, the analytical expression *still* comes extremely close to the experimental

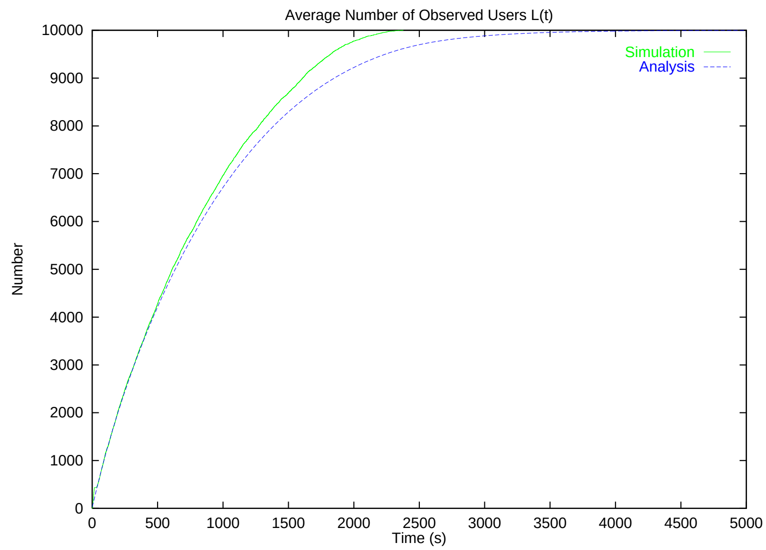

Figure 10: Experimental vs. analytical learning curve

for a large portion of the convergence period. In particular, it is very close during the period of linearity of $L(t)$ and less accurate afterwards. In addition, the differential equation does not capture the behavior of $L(t)$ for $0 \leq t \leq 20$, where the experimental curve exhibits the spike and plateau (this is difficult to see in Figure 10 because of the x axis scale).

We believe that network delays only impact the behavior of $L(t)$ when they are on the order of $CL(t)$. This is somewhat intuitive; the timescale of transmission events for any particular user is $CL(t)$. If network delays are much smaller than this, they are almost instantaneous as far as sending packets goes, and therefore do not affect the system behavior. It is for this reason that network delays only impact the learning curve during the first minute or so.

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:

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

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 is that the reconsideration mechanism favors 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 a non-initial packet at time t is:

$$P_{\text{sendold}} = \frac{t - t^* - C(1-\alpha)L(t)}{2\alpha CL(t)} \tag{7}$$

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 to (7), we get that the numerator of the fraction is negative, so the send probability is exactly zero.[4]. 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 $1/C$ with conditional reconsideration. With unconditional reconsideration, this rate is somewhat less. 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.

The final step is to approximate the convergence time. Unfortunately, the precise time depends on the non-linear regime of $L(t)$, which we cannot capture adequately. However, we can bound the convergence time by assuming linear behavior until $L(t)$ equals N. Since the actual $L(t)$ is less than this curve, the convergence time T_c is easily bounded on the left by:

$$T_c \geq NC(1-\alpha)$$

This bound grows linearly with the group size, as expected.

It is possible to compute an upper bound as well. Consider the last initial packet to be sent. Before it is sent, $L(t) = N - 1$. As long as the send probability is less than one, it is possible that this last initial packet will not be sent. But, according to (3), the send probability is one when $t > (1+\alpha)CL(t)$. This means that the last initial packet must be sent as soon as $t = (1+\alpha)C(N-1)$. This gives us an upper bound of:

$$T_c \leq NC(1+\alpha)$$

5.2 Modeling Delay and Loss

In this section, we consider the reconsideration algorithm in the presence of network delay and link bottlenecks. We compute the size of the spike during the initial transient, and the duration of the plateau. We also demonstrate the superiority of unconditional reconsideration in reducing these startup effects.

The spike and plateau are easily explained. At $t = 0$, all N users join the system. They schedule their packets to be sent between $(1-\alpha)T_{min}$ and $(1+\alpha)T_{min}$. At time $(1-\alpha)T_{min}$, packets begin to be sent. Lets say the network introduces a delay of D seconds. This means that no packets will

[4]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 non-initial packets, which is exactly zero for finite N.

arrive at any end system until time $(1-\alpha)T_{min} + D$. During these D seconds, many packets will be sent by end-systems, causing the initial spike of packets. After D seconds, this burst of packets will arrive. This causes a sharp increase in the perceived group size $L(t)$. This, in turn, increases the packet transmission interval, and moves the left hand side of the interval window well beyond the current time, so that $P_{send} = 0$. The result is a complete halt in transmissions until real time catches up with the left hand side of the reconsideration window.

This qualitative description of the system is easily quantified. For a large enough N, the flood of packets starting at time $(1-\alpha)T_{min}$ will saturate the access links D seconds later, independent of whether conditional or unconditional reconsideration is used. While the links remain saturated, packets arrive at a continuous rate at the link speed, which we denote as m packets per second. We can therefore express the arrival time of the n^{th} packet as:

$$t_n = (1-\alpha)T_{min} + D + \frac{n}{m} \tag{8}$$

Since each packet arrival increases $L(t)$ by one, we can set n equal to $L(t)$ in the above equation and solve for $L(t)$:

$$L(t) = m(t - (1-\alpha)T_{min} - D) \tag{9}$$

This flood of packets will cause the learning curve $L(t)$ to advance very quickly, beyond its maximum as given in (4). When the learning curve exceeds this maximum, all sending will stop. Call this stopping time t_{stop}. It can be obtained as the solution to:

$$(1-\alpha)CL(t_{stop}) = t_{stop} \tag{10}$$

$$t_{stop} = (1-\alpha)T_{min} + D + \frac{(1-\alpha)T_{min} + D}{(1-\alpha)Cm - 1} \tag{11}$$

We can then plug this into (9) and solve for the number of packets which have arrived at this point, n_{stop}:

$$n_{stop} = \frac{(1-\alpha)T_{min} + D}{(1-\alpha)C - 1/m} \tag{12}$$

The next step is to determine the number of packets sent up to this point. This figure differs based on whether the reconsideration mechanism is conditional or unconditional. We first look at conditional.

The number of packets sent consists of two terms. Before the arrival of the first packet (at time $(1-\alpha)T_{min} + D + 1/m$), all packets scheduled to be sent are actually sent, since no users have observed a change in the group size (which would activate the reconsideration mechanism). The number of packets sent is then the density of packets scheduled to be sent (which is $N/2\alpha T_{min}$) times the amount of time until the first packet arrives. We call this quantity n_{senta}, and it is:

$$n_{senta} = \frac{N}{2\alpha T_{min}} \left(D + \frac{1}{m} \right) \tag{13}$$

Once the first packet arrives, reconsideration kicks in, and not all packets will be sent. Each will be sent with some probability, P. Unfortunately, this is not the same probability P_{send} as defined in Equation 3. That equation ignored the max operator, assuming $L(t)$ was large most of the time. This is not true in the very beginning, where it takes a few packets to increase $CL(t)$ beyond T_{min}. We assume that once enough packets have arrived to do this, the result will be to move the left hand side of the reconsideration window ahead of the current time (this is true when $D < C$). In other words, we assume the left hand side of the reconsideration window is always at $(1-\alpha)CT_{min}$ until t_{stop}.

With this in mind, the send probability between the arrival of the first packet, and the stopping of transmission, is given by:

$$P_{send} = \frac{t-(1-\alpha)T_{min}}{2\alpha T_{min}} \tag{14}$$

The number of packets sent is given by the integral of the scheduled packet rate times the send probability:

$$n_{sentb} = \int_{(1-\alpha)T_{min}+D+1/m}^{t_{stop}} d(t)P_{send}dt \tag{15}$$

Since the density is $N/2\alpha T_{min}$ during this time period of interest, the number of packets sent is obtained by:

$$n_{sentb} = \int_{(1-\alpha)T_{min}+D+1/m}^{t_{stop}} \frac{N}{2\alpha T_{min}} \frac{t-(1-\alpha)T_{min}}{2\alpha T_{min}} dt \tag{16}$$

This integral results in a growth in the number of sent packets as t^2 until complete cutoff at t_{stop}. The solution to the integral is:

$$n_{sentb} = \frac{N}{8\alpha^2 T_{min}^2} \left(\left(\frac{(1-\alpha)T_{min}+D}{(1-\alpha)Cm-1} + D \right)^2 - \left(D + \frac{1}{m} \right)^2 \right) \tag{17}$$

And the total number of packets sent, using conditional reconsideration, during this transient spike is:

$$n_{sent} = n_{senta} + n_{sentb} \tag{18}$$

These analytical results are compared with simulation in Figure 11. The figure displays the cumulative number of packets sent for a step join. For the simulation, 100,000 users join the system at $t=0$. Network delays are deterministic and equal to 300 ms, and link speeds are 28.8 kbps. The plot shows only the initial transient. The linear and then t^2 behavior is clear from the simulation. Our approximation for both n_{senta} and n_{sentb} is quite good. The analysis also predicts that sending will stop at $t_{stop} = 1.72s$, which agrees with the simulation. Also note that the number of packets sent is dominated by the n_{senta} term.

For unconditional reconsideration, the number of packets sent during the transient is different. In the conditional case, the total consisted of two parts; one before the arrival of the first packet (as the reconsideration mechanism had not "kicked in" yet), and one after. In the case of unconditional, we do not need to wait for the arrival of a packet for the mechanism to activate. Therefore, the number of packets sent is given by an equation similar to that for n_{sentb} above. It is the integral of the scheduled rate, times the send probability. In this case, the integral is between $(1-\alpha)CT_{min}$

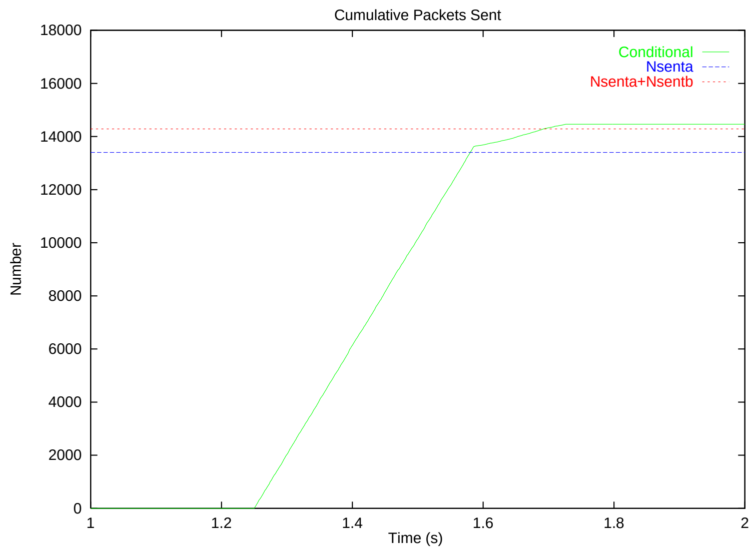

Figure 11: Transient with Conditional Reconsideration

and t_{stop}, instead of just between the arrival of the first packet and t_{stop}. The number of packets sent for unconditional is therefore:

$$n_{sent} = \int_{(1-\alpha)T_{min}}^{t_1} \frac{N}{2\alpha T_{min}} \frac{t-(1-\alpha)T_{min}}{2\alpha T_{min}} dt \tag{19}$$

Solving, we obtain:

$$n_{sent} = \frac{N}{8\alpha^2 T_{min}^2} \left(\frac{(1-\alpha)T_{min}+D}{(1-\alpha)Cm-1} + D \right)^2 \tag{20}$$

This quantity is small compared to n_{senta} for conditional reconsideration, thus the improved performance. These results are compared with simulation in Figure 12. The simulation model is identical to that in Figure 11, except unconditional reconsideration is used. As the plot indicates, only the t^2 behavior is present here. The total number of packets sent during the transient is much reduced, and reasonably well predicted by our analysis.

The next step is to determine the duration of the plateau period. Packet sending will start again when the current time catches up with the left hand side of the interval window, which will have quickly advanced to $(1-\alpha)Cn_{sent}$. The time at which this happens, t_{start} is:

$$t_{start} = (1-\alpha)Cn_{sent} \tag{21}$$

For conditional reconsideration, if we assume $n_{sent} \approx n_{senta}$, we obtain:

$$t_{start} = \frac{C(1-\alpha)N}{2\alpha T_{min}} \left(D + \frac{1}{m} \right) \tag{22}$$

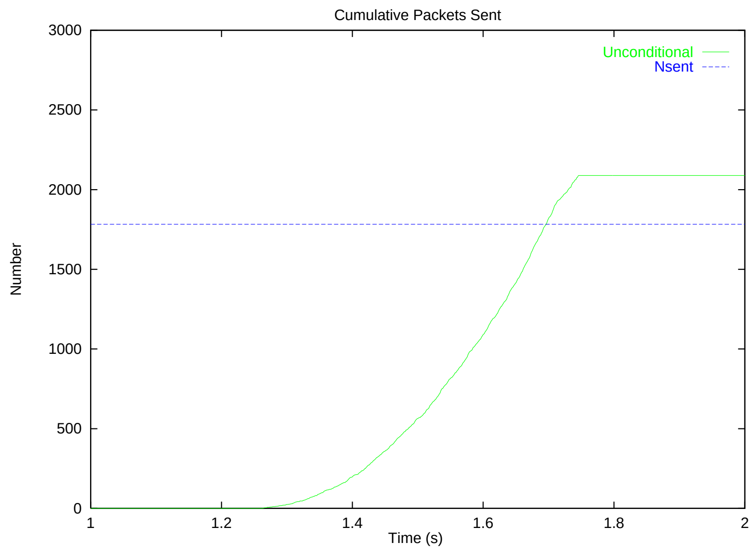

Figure 12: Transient with Unconditional Reconsideration

Group Size N	Conditional		Unconditional	
	n_{sent}	T_{plat}	n_{sent}	T_{plat}
1000	143	49 s	18	5 s
10000	1430	506 s	178	61 s
100000	14305	5083 s	1784	632 s

Table 1: Transient Behavior for Various Group Sizes

The duration of the plateau period itself is given by:

$$T_{plat} = t_{start} - t_{stop} \tag{23}$$

Table 1 lists the values of the parameters derived above for various group sizes. In all cases, $\alpha = 1/2$, $T_{min} = 2.5$, $C = .711s$, and $D = 300ms$. The unconditional mechanism provides clear gains in terms of reducing the number of packets sent during the transient, and the duration of the plateau effect.

6 Steady State Behavior

It is important to consider the behavior of the reconsideration algorithms when the learning curve has reached steady state (i.e., $L(t) = N$). The ideal behavior is for the total send rate of the group to be $1/C$ RTCP packets per second, equally divided among all users.

There are actually two situations which can be reasonably deemed as steady state. The first of these is a group size which remains exactly fixed. However, in real systems, users come and go, so a second definition of steady state is a group whose membership oscillates slightly about some large value.

We ran simulations to examine performance of the algorithms under both of these conditions. In first, fixed-group size scenario, both conditional reconsideration and the current RTCP algorithm both generate packets at the desired rate of $1/C$ per second. We found, as expected, that the packet rate was reduced in the unconditional case, and packets were sent at $.82/C$ packets per second, a reduction by 18%.

We also performed a stochastic analysis of the unconditional algorithm in steady state. The analysis demonstrated that the packet intervals, instead of being uniformly distributed between 1/2 and 3/2 of the deterministic interval, were distributed with a density of $(y-1/2)e^{y-1/2}dy$ between 1/2 and 3/2. The result is that the packet rates are reduced by $1-\frac{1}{e-3/2}$, or 18%, matching our simulations exactly.

In the second, slightly oscillating scenario, unconditional reconsideration and the current algorithm performed identically to their behavior in the first scenario. Conditional reconsideration, however, exhibited an average packet rate of $.91/C$, a reduction by 9%. This makes sense. When a user is about to send an RTCP packet, half the time the group size is larger than when the last packet is sent, activating the reconsideration. The other half of the time, the group size is slightly less, and the packet is sent, as if there were no reconsideration. Thus, the packet rate should be halfway between unconditional and the current algorithm.

We also ran some simulations to investigate the fairness properties of the algorithm. By fairness, we mean the variation of the number of packets transmitted per user, across all users. In a perfectly fair system, all users should have transmitted the same number of packets. We found all algorithms, including the current RTCP algorithm, to be extremely fair, with coefficients of variation below 0.005 after about an hour of running time.

Finally, we investigated the impact of reconsideration on synchronization. The problem of synchronization in the Internet was studied by Van Jacobson and Sally Floyd in [18]. Their study focused on the synchronization of periodic routing messages, such as those generated by RIP or IGRP. However, they generalize their results to any system which is characterized by their periodic messages model. Fortunately, the RTCP feedback mechanism fits perfectly into this model, making their results directly applicable here. Although reconsideration reduces the randomness of the interval, the reduction is negligible compared to the amount required to induce synchronization.

7 Summary and Future Work

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 congestion, convergence delays and state storage problems. We have solved the congestion problem via a simple algorithm called reconsideration. Both analysis and simulation have shown that the algorithm reduces the initial congestion by orders of magnitude under a variety of conditions. Furthermore, the algorithm is backwards compatible with the existing RTCP algorithm, allowing for a simple migration path.

The reconsideration algorithm has been implemented as part of a generic RTP Library, and is now operational in several common Mbone tools, such as rat and Nevot. It has also been proposed to the IETF as an improvement to the RTP specification, and is likely to be incorporated into the next release.

Future work involves considering the problem of simultaneous leaves, to which reconsideration cannot be directly applied. More work is also needed to solve the other RTP scalability problems.

8 Acknowledgments

The authors wish to thank Steve Casner, T.V. Lakshman, Sanjay Mithal, Daniel Rubenstein, Bernhard Suter, and Don Towsley for their insightful comments and discussion.

References

[1] ITU-T, *Recommendation H.323 - Visual Telephone Systems and Equipment for Local Area Networks which Provide Non-Guaranteed Quality of Service*, February 1996.

[2] M. Handley, H. Schulzrinne, and E. Schooler, “SIP: Session initiation protocol,” Internet Draft, Internet Engineering Task Force, Dec. 1996. Work in progress.

[3] H. Schulzrinne, “A real-time stream control protocol (RTSP’),” Internet Draft, Internet Engineering Task Force, Dec. 1996. Work in progress.

[4] 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.

[5] I. Busse, B. Deffner, and H. Schulzrinne, “Dynamic QoS control of multimedia applications based on RTP,” *Computer Communications*, vol. 19, pp. 49–58, Jan. 1996.

[6] J. Robinson and J. Stewart, “Multimon - an ipmulticast monitor.” The MultiMON web page can be found at http://www.merci.crc.doc.ca/mbone/MultiMON.

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

[8] K. Almeroth and M. Ammar, “Multicast group behavior in the internet’s multicast backbone (mbone),” *IEEE Communications Magazine*, vol. 35, June 1997.

[9] 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.

[10] K. Almeroth and M. Ammar, "Collecting and modeling the join/leave behavior of multicast group members in the mbone," in *Proceedings of the Symposium on High Performance Distributed Computing*, (Syracuse, NY), pp. 209–16, IEEE, Aug. 1996.

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

[12] P. Sharma, D. Estrin, S. Floyd, and V. Jacobson, "Scalable timers for soft state protocols," in *Proc. of IEEE Infocom*, 1997.

[13] P. Sharma, D. Estrin, S. Floyd, and V. Jacobson, "Scalable timers for soft state protocols," technical report, University of Southern California, June 1996.

[14] J. Veizades, E. Guttman, and C. Perkins, "Service location protocol," Request for Comments (Proposed Standard) RFC 2165, Internet Engineering Task Force, June 1997.

[15] J. Rosenberg, H. Schulzrinne, and B. Suter, "Wide area service location," (internet draft), Internet Engineering Task Force, July 1997. Work In Progress.

[16] Y. Rekhter and T. Li, "A border gateway protocol 4 (BGP-4)," Request for Comments (Draft Standard) RFC 1771, Internet Engineering Task Force, Mar. 1995. (Obsoletes RFC1654).

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

[18] S. Floyd and V. Jacobson, "The synchronization of periodic routing messages," *IEEE/ACM Transactions on Networking*, vol. 2, pp. 122–136, Apr. 1994.