Internet Engineering Task Force
INTERNET-DRAFT
draft-ietf-avt-byerecon-00.ps

Audio/Video Transport wg
J. Rosenberg, H. Schulzrinne
Bell Laboratories/Columbia U.
November 13, 1997
Expires: May 1998

New Results in RTP Scalability

Status of this Memo

1 Abstract

Recently, a number of problems related to RTP scalability to large multicast groups have been identified. The main problem, congestion of RTCP packets due to near-simultaneous joins, has been resolved in previous work. In this document, we present some additional problems and describe the solutions. In particular, we discuss the problem of BYE floods and premature timeouts. To resolve the BYE flood problem, we propose a BYE reconsideration mechanism. To help alleviate premature timeouts, we propose a reverse timer reconsideration algorithm. Both algorithms are simple, and require minimal state and computation.

2 Introduction

Recently, a number of problems related to RTP [1] scalability to large multicast groups have been identified [2]. The most serious of these problems is RTCP congestion in simultaneous joins. In this scenario, a large number of users join a session at nearly the same time. This can happen if the join is an automatic response to an announced session on the mbone [3], or if RTP is being used for broadcast applications, and a popular show comes on. When this happens, each user joins the group believing that they are the only user. They then send their initial RTCP packets within a short period of time, causing a flood of RTCP reports. This can result in network and/or access link congestion. For modem dial-in users, the slow access links can cause congestion even with moderate simultaneous joins.

To combat this problem, an algorithm called reconsideration has been proposed [4] [5]. This algorithm causes backoff in the transmission of RTCP packets as group sizes increase.

However, several new problems have recently been discovered relating to RTCP scalability. These difficulties are similar to the step-join congestion problem, but different in that they require additional algorithms to resolve. This document describes the two new difficulties which have been encountered - BYE floods and premature timeouts.

3 BYE Floods

The BYE flood problem is very similar to the simultaneous join problem. Instead of many users joining at the same time, many users leave the group at the same time. Since an RTP client sends an RTCP BYE packet when it leaves the group, this causes a flood of BYE packets, which congests the network. Users can be expected to leave a group simultaneously for much the same reasons they might join simultaneously - an automatic leave as a result of the end of the session (as indicated in the SDP [6] announcement for the session), or because the "show" is over, and the users manually exit their applications.

A number of aspects of the BYE flood problem make it different than the simultaneous join problem. These must be taken into consideration when designing an algorithm to reduce the flood. We therefore state the goals of the BYE flood prevention algorithm as follows:

- Users often terminate their applications just after leaving the session. The algorithm must be aware of this possibility, and define the appropriate behavior if an application decides to terminate.
- The algorithm should behave gracefully; when very few users are leaving the group simultaneously, users should generally be allowed to send their BYE packets right away. It is only in the presence of a large number of BYE packets that the algorithm should "kick" in, and force users to hold back on sending their BYE packets.
- The algorithm should be simple, requiring minimal computation and storage.

We propose an algorithm called BYE reconsideration to accomplish these goals. The algorithm operates much like standard reconsideration. However, instead of counting other users, and using the resulting count as a multiplier for the packet transmission interval, the client counts BYE packets, and uses the number of BYE packets received thus far as the multiplier for the interval. The operation of the algorithm is as follows.

At some time t_l, the user decides to leave the session. The application first checks to see if it has ever sent an RTCP packet. If it has not, the application must not send a BYE packet. Instead, it should leave the session silently. Without having sent an RTCP packet, the BYE packet provides no useful information. Next, the application checks to see if the group size is less than some threshold, B_t (a value of 50 seems reasonable). If it is, the application may send a BYE packet immediately, and then leave the session. For small groups, BYE packets are of significant value (for loose session management), and sending them immediately is quite important. Since the group contains only a small number of participants, the flood of packets is limited.

It is possible that a user has a lowball group size estimate if they leave a session quickly after joining it. If this session is large, and there are many users coming and going fairly quickly (typical of a channel surfer), it might appear that this can cause a steady flow of BYE packets. However, if these clients implement the forward reconsideration algorithms, they generally will have never sent an RTCP packet. This is because the new users' RTCP packet transmission will be constantly reconsidered, as the new user will be receiving RTCP packets at a steady rate from other users already in the group. This constant reception of packets will cause the new user to see a steady growth in the group size, causing its own RTCP packet transmission to be pushed into the future. Since a user who never sends an RTCP packet cannot send a BYE packet, this will generally cause these "channel surfers" to neither send RTCP SDES or RR information, nor a BYE packet.

If the user has sent an RTCP packet previously, and the group size exceeds B_t, the application computes a time interval T as:

$$T = R(1/2)max(T_{min}, n_l * C) \quad (1)$$

Where T_{min} is 2.5 seconds, $C = avgpktsz/(bw * .05)$, $R(1/2)$ is a random variable uniformly distributed between 1/2 and 3/2, $avgpktsz$ is the average size of all BYE packets received thus far, and bw is the session bandwidth. The average packet size is computed using the same exponential weighted average filter used to compute the average RTCP packet size in the current specification. The value is updated through the filter every time a BYE packet is received or transmitted (not when it is reconsidered).

The user then schedules the BYE packet to be sent at time $t_l + T$. Between t_l and this time, the user increments n_l for each BYE packet that is received. In this fashion, n_l counts the number of BYE's from other users since deciding to leave the session.

When this time arrives, the user recomputes T according to the previous equation. If $t_l + T$ is less than the current time, the BYE packet may be sent. If $t_l + T$ is more than the current time, the BYE packet transmission is rescheduled for time $t_l + T$. At that time, the computation and comparison are repeated. All along, n_l is incremented for each BYE packet received.

A BYE packet which is from an SSRC which already sent a BYE (a duplicate) is ignored. Furthermore, the application should not increment n_l if it receives a BYE from a user which has never sent an RTCP packet. Under normal situations, an application should never send a duplicate BYE packet, or send a BYE if an RTCP packet was never sent. However, a malicious user may send many BYE packets. If this check were not made, these BYE's would cause the variable n_l to increase, and effectively prevent any other user from sending a BYE.

If an application wishes to terminate before it can send a BYE RTCP packet according to these rules, it must not send a BYE packet. Instead, it should terminate silently. The BYE reconsideration algorithm will effectively re-allocate the bandwidth from users who leave without BYE's to those who wait around to send a BYE.

The effect of this algorithm is to restrict the BYE packet transmission rate to at most an additional 10% of the session bandwidth (assuming a very large simultaneous leave). At the same time, if only a few users are leaving the group (even for a large group), they will get to send their BYE packets in a timely fashion. This meets the design objectives described in the beginning of the section.

We ran numerous simulations to verify the performance of the algorithm. Even with as many as 10,000 users simultaneously leaving the session, the BYE reconsideration algorithm maintained the BYE transmission rate at 10%. This is demonstrated in Figure 1, which depicts the cumulative number of RTCP packets (BYE and others) send to the multicast group over time. At time 10,000, almost all of the users leave the group. The top line depicts the performance without BYE reconsideration, where some 10,000 BYE packets are sent all at once. The lower curve shows performance for BYE reconsideration. Note how there is only a small increase in packet transmission rates.

4 Premature Timeouts

We have observed a secondary effect when many users simultaneously leave a group. There are many applications where not all of the users will leave - some will stick around. An example is a

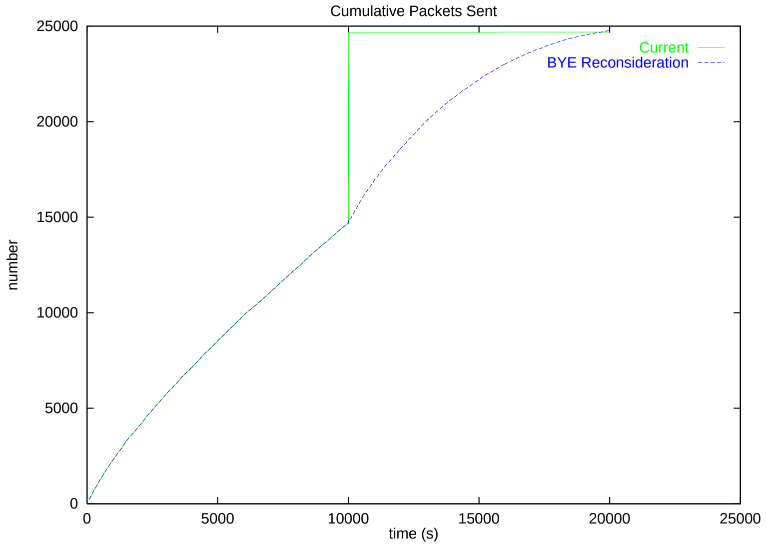

Figure 1: BYE Reconsideration Performance

distance learning application. There are perhaps several hundred students in the class. When the class ends, most of the students leave at about the same time. However, some stay behind to talk with the professor.

We have observed that rapid leaves can cause the remaining users to time each other out. It can take a significant amount of time for the users to return. This implies that each user will not see the other users, which is undesirable for the post-class discussion scenario just mentioned.

4.1 Quantifying the Problem

The difficulty is related to the way timeouts are handled. In the current specification [1], a user is timed out if they have not sent an RTP or RTCP packet within the last five RTCP intervals. In dynamic groups, the interval itself is dynamic. As many users leave a group, their BYE packets cause the group size estimate to rapidly decrease. This, in turn, decreases the timeout interval. If the number of users who leave the group is sufficiently large, the timeout interval may decrease so much that the remaining users will time out.

For example, consider a group of 505 users. If the total RTCP interval is to be limited to 1 packet per second, each user sends RTCP packets once every 505 seconds (on average). Assume user 1 last sent an RTCP packet at time 0. The user schedules the next RTCP packet for time 505. At time 490, 500 of the 505 members (not including user 1) leave the group, and send BYE packets (assume for the moment that there is no BYE flood prevention algorithm). Shortly thereafter (say time 500) the BYE packets have been received, and the remaining 5 users perceive the group size to be 5. Based on this, the timeout interval is 25 seconds. Any user who has not sent a packet since time 475 will therefore be timed out. User 1 last sent a packet at time 0, so they are timed out. In fact, odds are good that most of the remaining 5 users sent their last packet before time 475. Thus, every user will time out all of the other users. Furthermore, it may take a long time for those users to come back. Consider user 2, who did not leave the group, and who was unfortunate enough to have last sent an RTCP packet at time 450. They then scheduled their next RTCP packet for time

955 (since there were still 505 users at the time). After the exodus at time 500, user 2 will remain, but will not send the next RTCP packet until time 955 (unless the user sends data, in which case they will be known via the RTP packet).

The first question to ask is whether BYE flood prevention helps alleviate this problem. Since the algorithm is designed to reduce the flood of BYE packets, the group size cannot drop so rapidly. This does help, of course, but not completely. With network delays, there still can be spikes in BYE packets. It does not require many BYE packets for this phenomenon to surface; it only requires that the ratio of users left after the leave to the number before the leave be less than around 1/5. This can occur in both small and large groups alike.

Even assuming the rate of BYE packets is restricted to some constant factor, the efficacy of the timeout algorithm is reduced. To show this, we computed the number of packets sent by a user, on average, during the timeout interval, as the group size decreases linearly from some initial value N_s to some final value N_f. We denote the slope of the decrease by r users per second. We also define C as the multiplier to convert from group size to interval (so that if there are N group members, each member sends RTCP packets every CN seconds, on average), and M as the factor of 5 timeout multiplier. If r times C is much smaller than 1, and less than $(1/CM)(N_s/N_f - 1)$ the number of packets sent during the timeout window, is on average:

$$N_p = \frac{-1}{ln(1-rC)} ln(1+rCM) \tag{2}$$

and for rC larger than 1, but less than $(1/cM)(Ns/Nf - 1)$:

$$N_p \leq \frac{M}{1+rCM} \tag{3}$$

In the limit as r goes to zero (that is, nobody leaving the group), the number of packets is M (according to the first equation), as expected. However, this quantity decreases rapidly as the slope, r, increases relative to $1/C$. With the BYE prevention algorithm, the flood of packets can be shown to be bounded at an average rate of $2/C$ in the absence of network delays. With network delays, there can be a spike of packets, but following this spike, the BYE packet rate will also settle to $2/C$ initially, gradually decreaseing to $1/C$. Plugging in $r = 2/C$ into the second equation above, the number of packets sent is:

$$N_p \leq 5/11 \tag{4}$$

This means that each sender will send only half a packet during the timeout window, on average. In reality, this means that half of the users remaining will send 1, and the other half will not. Therefore, many users will still timeout. Those users which do manage to send a packet may still timeout if the packet is lost.

Note that both equations above rely on many users leaving the group at the same time (the constraint that $r < (1/MC)(N_f/N_s - 1)$). If the number who leave is small relative to the slope of the leave ($r > (1/MC)(N_f/N_s - 1)$), the effects are less severe.

4.2 Reverse Reconsideration

One of the major factors contributing to the premature timeout effect is the delay between when the group size decreases, and when users begin to send packets using the resulting smaller interval. In the example in the previous section, user 2 sent an RTCP packet at time 450, and scheduled the next for time 955. After the exodus at time 490, user 2 knows that the group size has dropped - but does nothing. The user instead waits until time 955, sends the packet, and then computes the next send time. Since the group size is now 5, user 2 will schedule the next packet 5 seconds later, on average. There is thus a 500 second "delay" between the exodus and when user 2 gets around to scheduling a packet using the new, smaller interval.

To resolve this problem, we propose an algorithm called reverse reconsideration. The idea is simple. If the group membership decreases, each user reschedules their next packet immediately. The packet is rescheduled for a time earlier than previously. The amount earlier is made to depend on how much the group size has decreased.

More specifically, assume that the last time a user sent an RTCP report is t_p. The next report is scheduled for time t_n, and t_c is the current time. Before the arrival of a BYE packet at the current time t_c, there were n_p users. There are now n_c users. Before the BYE, RTCP packet transmissions should be uniformly scheduled over time. That means that there should have been n_c packet transmissions scheduled between t_c and $t_c + C * n_c$. Now, however, the group size has decreased to n_p. This should cause there to be n_p packet transmissions scheduled between $t_c and C * n_p$. To accomplish this, every user should compress the interval between the current time and their next packet transmission by n_c/n_p. This implies that the next packet transmission should be rescheduled for time t_n:

$$t_n = t_c + (n_c/n_p)(t_n - t_c) \tag{5}$$

This new value for t_n has two key properties:

1. The new time is always earlier than the previous time.
2. The new time can never be before the current time.

The second property is key; it guarantees that there will not be a spike of packets transmitted due to a sharp decrease in group size.

On the surface, it would seem that an alternate algorithm might be a direct application of regular (or forward) reconsideration. Such an implementation might work as follows. At t_c, when the BYE arrives, the user recomputes the transmission interval T, based on the new group size n_c. This interval is then added to the previous packet transmission time t_p. If the result is a time before the current time, the packet is sent, else it is rescheduled for $t_p + T$. This algorithm does not work. Even a moderate decrease in the group size would cause many users to send their RTCP packets immediately, causing an additional spike. This is because this version of the algorithm does not maintain property 2 - the new transmission time can be before the current time.

There is one additional aspect to the reverse reconsideration algorithm that must be considered - how it interacts with forward reconsideration. Consider the following example. There are 100 users in a group. The constant C is equal to 1 packet per second. At time 0, user A sends an RTCP packet, and schedules the next for time 100. At time 50, 50 users leave the group. User A executes the reverse reconsideration algorithm, and reschedules their packet for time 50 + (50/100)(100 - 50) = 75. At time 60, one more user joins the group. At time 75, user A executes the forward reconsideration algorithm (we assume conditional reconsideration). Since the group size has increased (51 vs. 50), user A recomputes the interval - which is now 51 seconds on average. This is then added to the previous transmission time, which is time 0. The result is $t = 51$, significantly earlier than the current time $t = 75$. This will cause the user (and in fact, all other users) to send their packets immediately, even if the group size further increases. The problem is that while we have adjusted the next packet transmission time, t_n, with reverse reconsideration, we have not adjusted the value of the previous packet transmission time, t_p. This quantity is used for forward reconsideration, and must be adjusted as well in order to maintain consistency.

The adjustment algorithm is simple. The value for t_p is updated when t_n is updated by reverse reconsideration. Its value is adjusted to:

$$t_p = t_c - (n_c/n_p)(t_c - t_p) \tag{6}$$

The nature of this adjustment is the same as for t_n. In the previous example, it would have caused t_p to be adjusted from 0 to (50 - (50/100)(50 - 0)) = 25. At time 75, when the user is performing the forward reconsideration algorithm, the interval $T = 51$ is added to $tp = 25$, yielding $t = 76$, slightly ahead of the current time $t = 75$, as expected (since the group has only increased by one member).

4.3 Performance

What is the improvement due to reverse reconsideration? We have been able to prove that the number of packets sent during the timeout window has an achievable lower bound of:

$$N_p = (1/rC) * ln(1 + rCM) \tag{7}$$

This time, independent of the relative value of rC. Again, this holds only when the actual number of users who leave is a significant fraction of the current group size ($N_f/N_s < 1/(1 + MCr)$). Assuming that BYE reconsideration is being used, the BYE rate is limited to around $r = 2/C$ when network delays are small. Plugging this in:

$$N_p = 1/2ln11 = 1.19 \tag{8}$$

This means that a user will send 1.19 packets on average, during the timeout window. This is to be compared to the situation before reverse reconsideration, where $N_p = 5/11$. Therefore, reverse reconsideration affords a factor of three improvement in performance. Now, no users will timeout under normal circumstances (on average). However, a single packet loss may cause a user to timeout prematurely.

5 Conclusion

We presented two additional problems with RTP scalability - BYE floods and premature timeouts. BYE floods are caused when many users simultaneously leave a group. To fix the problem, we presented an algorithm called BYE reconsideration, which works much like forward reconsideration, but for BYE packets. The second problem, premature timeouts, is a secondary effect, but still problematic. To help resolve it, we presented an algorithm called reverse reconsideration, which shows a threefold factor of improvement. Both algorithms are extremely simple, requiring no additional memory beyond forward reconsideration, and requiring a single O(1) computation upon reception of a BYE packet.

References

[1] H. Schulzrinne, S. Casner, R. Frederick, and V. Jacobson, "RTP: a transport protocol for real-time applications," Tech. Rep. RFC 1889, Internet Engineering Task Force, Jan. 1996.

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

[3] M. Handley, "SAP: Session announcement protocol," Internet Draft, Internet Engineering Task Force, Nov. 1996. Work in progress.

[4] J. Rosenberg and H. Schulzrinne, "Timer reconsideration for enhanced RTP scalability," Internet Draft, Internet Engineering Task Force, July 1997. Work in progress.

[5] J. Rosenberg and H. Schulzrinne, "Timer reconsideration for enhanced rtp scalability," in *To appear in Proceedings of IEEE Infocom '98*, 1998.

[6] M. Handley and V. Jacobson, "SDP: Session description protocol," Internet Draft, Internet Engineering Task Force, Mar. 1997. Work in progress.

6 Authors' Addresses

Jonathan Rosenberg
Bell Laboratories, Lucent Technologies
101 Crawfords Corner Rd.
Holmdel, NJ 07733
Rm. 4C-526
email: jdrosen@bell-labs.com

Henning Schulzrinne
Columbia University
M/S 0401
1214 Amsterdam Ave.
New York, NY 10027-7003
email: schulzrinne@cs.columbia.edu