Staged Refresh Timers for RSVP

Ping Pan, Henning Schulzrinne

Abstract—**In the current RSVP design [2], there is no reliability mechanism for control message delivery. Instead RSVP relies on periodic refresh between routers to maintain reservation states. This approach has several potential problems in a congested network. End systems send PATH and RESV messages to set up RSVP connections. If the *first* PATH or RESV message is accidentally lost, an upstream router will have to wait for the entire duration of a refresh cycle to retransmit another copy of the lost message. If a tear-down message (a PATHTEAR or a RESVTEAR) is dropped at a congested link, a reservation will not be removed until the RSVP *cleanup timer* expires. The above problems may not be acceptable for certain applications. In this paper, we present a RSVP extension to support fast and reliable message delivery. The extension is backward compatible and is easy to be adapted to the current implementation. In the paper, we will address several design issues extensively.**

Keywords—**RSVP, soft state.**

I. Introduction

After years of design and development, RSVP [1], [2] along with PIM[4], [5] and several other new technologies will be gradually deployed to the Internet in the next few years. Continuous media applications such as teleconferencing and interactive multimedia games can interface RSVP to demand QOS from the network. Inside network, routers and switches will have traffic shaping and packet-level scheduling functionality to support various types of services. However, for further deploying and maintaining commercial networks, some of the Internet protocols need to be enhanced.

RSVP has been designed to exchange resource reservation information among routers. It has the advantages of

- supporting both unicast and multicast while independent from any specific routing protocol;
- accommodating heterogeneous receiver requirements by applying receiver initiated reservation;
- incorporating *soft state* to maintain states in each router.

However there are several subtle issues that RSVP has to overcome. One of such issues is its message delivery mechanism.

RSVP sends its messages as IP datagrams with no reliability guarantee. It relies on periodic refresh messages from hosts and routers to handle the occasional loss of a PATH or RESV message. Each RSVP host or router maintains a cleanup timer. A state is deleted if no refresh messages arrive before the expiration of a *cleanup timeout* interval.

In today's Internet backbone, as shown in Reference[7], the packet loss rate is around 1-2% on average, and can occasionally be as high as 20% on congested links. In this case, the existing RSVP message delivery method may not be acceptable. For example, a user is trying to make a reservation. If the *first* reservation request (RESV) from the user is lost due to network link noise or congestion, the message will not be retransmitted until the next refresh cycle arrives. By default, a refresh interval is 30 seconds. At the same time, the user will suffer from poor quality of service. In case of tearing down a reservation, if the message is lost, the user may have to end up paying for the extra time that is spent on reservation state time-out. The default time-out interval is 90 seconds.

In this paper, we propose a RSVP extension that can:

1. provide a mechanism to deliver RSVP message faster and more reliable.
2. is backward compatible with the existing implementations.
3. reduce the number of refreshes among routers.

II. RSVP Extension

A. Trigger Traffic vs. Refresh Traffic

In RSVP, control traffic can be generally categorized into two types: trigger and refresh.

Trigger messages are generated by a RSVP host or router due to state changes. A state change could be the initiation of a new state, a route change that altered the reservation paths, or a reservation modification by a downstream router.

Refresh messages are duplicated state information generated by a router for maintenance purposes. RSVP uses periodic refreshing to provide a level of robustness to make sure that hosts and routers have correct state information at all time. For instance, the RSVP daemon on a router is crashed or reset, and all RSVP state information are lost. Since its neighbor routers send the copies of RSVP state information in the refresh messages, the router can recover the lost states within one refresh interval.

Ideally, trigger messages should be delivered timely and reliably. The retransmission rate should be the range of one RTT (round-trip time). However, since RSVP routing interface[8] can discover most state changes, and the possibility of RSVP daemon crashing is fairly low in production network, the refresh interval can therefore be set to a very large value. Generally, a RSVP refresh interval should be in the range of hours.

B. Operation Outline

In the proposed RSVP extension, we designed a feedback mechanism for RSVP message delivery:

In each trigger message, an echo-request flag is set in the RSVP common header by a sending node. Upon reception, a receiving node has the option of sending an echo-reply message back to the sending node to acknowledge that the flows included in the message have been successfully received. When the sending node receives the echo-reply, it will automatically scale back the refresh rate to a slower value. If the sending node does not receive acknowledges, it will continue to generate refresh messages to the receiving node with a fast refresh rate.

A staged refresh timer is used in each sending node (in case of receiving nodes not supporting the new extension): A sending node uses a very fast refresh timer to generate the first trigger message. As time progresses, the sending node will reduce the refresh rate exponentially until is reaches to a threshold. From

P. Pan is with the IBM Thomas J. Watson Research Center, Yorktown Heights, NY 10598. pan@watson.ibm.com.

H. Schulzrinne is with the department of computer science, Columbia University, New York, NY 10027. schulzrinne@cs.columbia.edu.

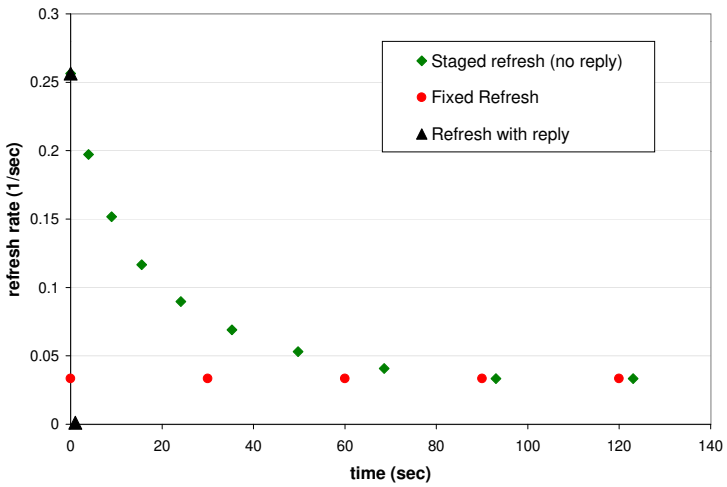

Fig. 1. RSVP refresh comparison.

that point on, the sending node will treat the receiving node as if it does not support the new RSVP extension, and will use a fixed refresh timer for PATH and RESV messages and stops re-transmitting tear-down messages.

The proposed extension requires several minor modifications in the current specification: a new bit is defined in the flag field of RSVP common header, and four new message types are created for echo-reply.

C. Time Parameters

The new extension will make the use of the following time parameters:

- **Fast refresh rate, Rf:** This is the initial refresh rate used by a sending node to transmit a trigger message. After the first message is sent, the node schedules a retransmission after 1/Rf seconds. The value of 1/Rf could be as small as one RTT (round-trip-time) between a sending node and a receiving node. However in case receiving nodes do not support the new extension, a sending node can use a slightly large value for 1/Rf to avoid unnecessary refresh messages flooding the link. A typical value for Rf can be once every 3 seconds.
- **Slow refresh rate, Rs:** Sending node refreshes with this rate when it determines that the receiving nodes support the new extension. The value of Rs is modifiable at each egress interface, and is set to be one refresh every 15 minutes by default.
- **Fixed refresh rate, R:** Initially, a sending node retransmits a trigger message if there are no replies from all receiving nodes. The refresh rate reduces exponentially. In case of PATH and RESV message, when the refresh rate reaches to the R, the sending node will use R as the rate to refresh receiving nodes from that point on. The default value for R is once every 30 seconds.
- **The incremental value, Δ:** This is used to gradually reduce the refresh rate at a sending node. The ratio of two successive refresh is $(1 + \Delta)$. Δ, by default, is set to be 0.30. It could be the same value as Slew.Max that has been defined in RSVP.

D. Staged Refresh

After a sending node transmits a trigger message, it will immediately schedule for a retransmission after 1/Rf seconds. If echo-replies are received, the sending node will switch the refresh rate to Rs. Otherwise, it will retransmit the message after $(1 + \Delta)$/Rf seconds. $(1 + \Delta)$ is the ratio between two successive retransmission. The staged retransmission will continue until either echo-replies are received, or the refresh rate has been reduced to the fixed refresh rate R.

After the staged retransmission rate has reached R, the sending node will start to refresh every 1/R seconds.

The refresh rate at each refresh cycle can be described as:

$$R_i = \begin{cases} (1+\Delta)^{-i} Rf & \text{if no reply, and i} \leq \text{I} \\ R & \text{if no reply, and i} > \text{I} \\ Rs & \text{if reply} \end{cases}$$

where,

$$I = \frac{\log \frac{R}{Rf}}{\log(1+\Delta)}$$

I is the number of retransmissions before the rate is reduced to the fixed refresh rate R. After sending I messages, if the trigger message is either RSVP PATH or RESV message, the sending node will switch to refresh rate R. For RSVP tear-down messages (PATHTEAR and RESVTEAR), the sending node will stop re-transmitting.

In Figure 1, we illustrated how various refreshing mechanism operates. The fast refresh rate Rf is once every 3 seconds. The slow refresh rateRs is once every 15 minutes. The fixed rate R is once every 30 seconds. The incremental value Δ is 0.3. The figure shows that having the new extension supported on both sending and receiving nodes reduces the number of refreshes. If the receiving nodes do not support the new RSVP extension, or refuse to reply to the trigger messages, the sending node will generate several refreshes within a short period of time, and the

refresh rate will eventually converge to the fixed refresh rate R. This behavior certainly is an advantage in dealing with congestion: sending nodes always transmit several copies of a trigger message to increase the likelihood of success. In the figure, the sending node transmit seven messages before reducing its refresh rate to the fixed rate R. In a more stable network, users can always speed up the convergence time by using larger Δ value.

The implementation of staged refresh is quite simple. A sending node can use the following algorithm when the RSVP refresh timer for state i is expired:

R_i: the current refresh rate
R: the fixed refresh rate
Δ: the incremental rate

if $(R_i > R)$
    $R_i = \frac{R_i}{1+\Delta}$
    send out a refresh message;
    wake up state i after $\frac{1}{R_i}$ seconds;
    exit.
else
    $R_i = R$
    if(the state is for a tear-down message)
        clean up state i;
        exit.
    else
        send out a refresh message;
        wake up state i after $\frac{1}{R_i}$ seconds;
        exit.

Asynchronously, when the sending node determines the receiving nodes support the new extension, it can change the refresh rate R_i to the slow rate Rs for state i.

III. Special Considerations

A. Backward Compatibility

Backward compatibility is one of the main objectives in our design. One cannot assume that both sending and receiving nodes on a link will support the extension simultaneously.

In our design, sending nodes rely on echo request/reply mechanism to "learn" about the status of receiving nodes. If the receiving nodes do not support the new extension, the existing RSVP functionality will not be effected.

B. *Compute* Cleanup Timeout *Values*

Each RSVP PATH (or RESV) message carries a refresh rate in its TIME_VALUES object. The refresh rate can be used by receiving nodes to compute the *cleanup timeout* interval. Generally, the *cleanup timeout* interval is a multiple of refresh rate.

In the staged refresh design, a sending node initially inserts the slow refresh timer, 1/Rs, in the PATH or RESV message. After the actual refresh rate has been reduced to the fixed rate R, the sending node will put 1/R in the refresh messages from that point on.

As a result, for receiving nodes that support the extension, they will have a *cleanup timeout* interval based on Rs. For receivers that do not support the extension, their *cleanup timeout* will be based on R, eventually.

C. Handling of Tear-down Messages

In RSVP, PATHTEAR and RESVTEAR are used to tear-down path and reservation states, respectively. In the current RSVP design, tear-down message is generated once from a sending node. If the message is accidentally dropped along the way, the receiving nodes will have no knowledge of any tear-down request. The reserved resource will eventually be released after the *cleanup timer* is expired.

In the new extension, we have altered the processing rules of tear-down messages. Instead of deleting the state after a tear-down message has been generated, a sending node will release all resource allocated to the state, and mark the state as *pending*. The state information is saved for message retransmission. The entire state information will be removed when echo-replies are received, or when the sending node realizes the receiving nodes do not support the extension.

D. Operation in NBMA Environment

For a multicast RSVP session in a non-broadcast multiple access (NBMA) network (such as ATM), a sending node may not know the total number of receiving nodes for a PATH or PATHTEAR message at an egress interface. Therefore, a sending node cannot simply switch to a slower refresh timer based on echo-replies.

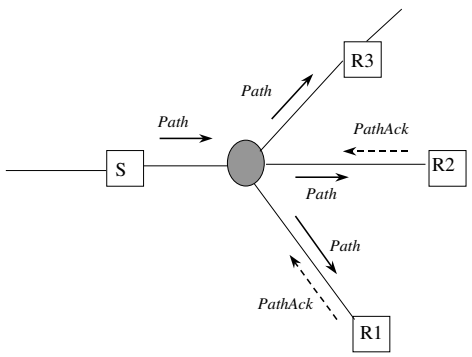

Fig. 2. Path message in NBMA.

For example, as shown in Figure 2, if the receiving node R3 does not support the new RSVP extension, the sending node S should not change to a slower refresh rate, even through it has received echo-replies from R1 and R2.

In this case, a sending node has the following options:
1. Query local database such as ARP or MARS server to find out the exact number of the next-hop receivers. It will switch to a slower refresh rate until it has received echo-replies from all known receiving nodes.
2. Since PATH messages are mainly used for traffic advertisement purposes, the sending node may not need to run staged refresh timers for PATH.

However, in case of PATHTEAR message, a sending node always has the information of all the receiving nodes that have made reservations. The following rules can be used:

- A sending node stops re-transmitting PATHTEAR messages once it receives echo-replies from all its known next-hop receivers at an egress interface.
- Otherwise, the sending node generates PATHTEAR messages using staged refresh timer until the refresh rate is reduced to the fixed refresh rate R. Then it stops re-transmitting.

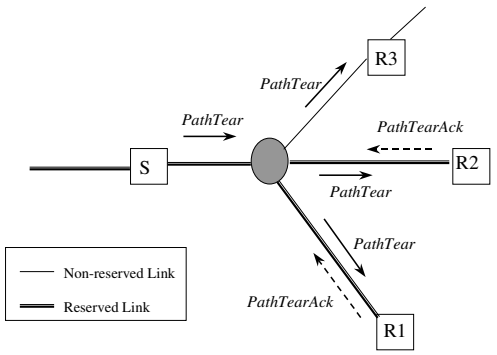

Fig. 3. PathTear message in NBMA.

An example is shown in Figure 3. R1, R2 and R3 are the receiving nodes to S. Initially, the sender S had the reservation state information for receiving node R1 and R2. Since R3 did not make any reservation, S would not know the existence of R3 from its RSVP database. After sending the first PATHTEAR message, S will retransmit the message until it has received echo replies from R1 and R2. After which, S stops generating PATHTEAR messages.

IV. Advantages

A. Reduce Message Loss Probability

Adding feedback mechanism in RSVP message delivery reduces the message loss probability. Figure 4 shows the accumulative loss probability for both fixed and staged refresh mechanism. The fixed refresh rate is 1/30. The initial fast refresh rate is 1/3. The incremental value Δ for staged refresh is 0.3. The message loss probability for a single message to 2% from reference [7].

The reason behind the loss reduction is that if more messages are sent within a given time, the delivery will be more likely to success.

In the example, within the first 60 seconds, six refreshes are sent if staged refresh timer is used, compare with only two refreshes with fixed refresh timer. The packet loss probability is reduced to the neighborhood of 10^{-5} compare with 10^{-2} with fixed refresh rate.

B. Reduce Protocol Overhead

When a RSVP flow is established using the staged refresh timers, the *soft state* refresh frequency is reduced.

In Table 5, we show the protocol overhead for a single RSVP flow. For a trigger RSVP RESV message (150 byte), if the sending node uses the fixed refresh timer (refresh rate = 30 seconds), it will generate 300 bytes in the first 60 seconds, and 18 Kbytes in an hour. Using staged refresh timer (Rf = 1/3, Rs = 1/900, R = 1/30 and Δ = 0.3), if the receiving node supports the extension, the number of bytes to be transmitted in an hour is only 900 bytes. Also notice, even if the receiving node does not support the new extension, the amount of data being transmitted over an hour is nearly the same comparing with the fixed refresh timer case.

For a link that requires to manage thousands of RSVP flows, protocol reduction is clearly an advantage over the fixed refresh timer.

C. Simple to Process

The newly defined echo-reply messages are simple to generate and easy to process. The format of echo-reply messages can be designed in such a way that a router can build an echo-reply message by simply copying the appropriate objects from the echo-request message.

The new algorithm used by RSVP timers is given in Section II-D. It takes literally a few dozen lines of C code to implement.

V. Conclusion and Future Works

We believe that RSVP message delivery mechanism requires some degree of reliability guarantee. One way of improving reliability is to grant some minimal bandwidth for RSVP messages to protect them from congestion losses, as being suggested in Reference[2]. However it may require additional functions at both sending and receiving nodes. And it is not clear how it can be backward compatible.

In this paper, we presented a mechanism that enhances the current RSVP message delivery and is completely backward compatible. As shown in the paper, staged refresh timer is easy to be supported in a RSVP router, and saves both processing and bandwidth overhead.

Staged refresh timer algorithm is currently being implemented in IBM Research's router software platform, and will be tested on several router/switch prototypes.

Acknowledgments The authors wish to thank Shai Herzog for the insightful discussion at the early stage of this work. Many thanks to Roch Guérin for his insightful comments.

References

[1] L. Zhang, S. Deering, D. Estrin, S. Shenker, and D. Zappala. RSVP: A New Resource ReServation Protocol. IEEE Network Magazine, September 1993

[2] R. Braden, L. Zhang, S. Berson, S. Herzog, and S. Jamin. Resource ReSerVation Protocol (RSVP) - Version 1 Functional Specification. INTERNET DRAFT, November 1996. Work in progress.

[3] R. Braden, and L. Zhang. Resource ReSerVation Protocol (RSVP) - Version 1 Message Processing Rules. INTERNET DRAFT, October 1996. Work in progress.

[4] S. Deering, D. Estrin, D. Farinacci, V. Jacobson, C. Liu, L. Wei, P. Sharma, and A. Helmy. Protocol Independent Multicast (PIM): Motivation and architecture. INTERNET DRAFT, March 1996 Work in progress.

[5] S. Deering, D. Estrin, D. Farinacci, V. Jacobson, C. Liu, L. Wei, P. Sharma, and A. Helmy. Protocol Independent Multicast (PIM), sparse mode protocol: Specification. INTERNET DRAFT, March 1996 Work in progress.

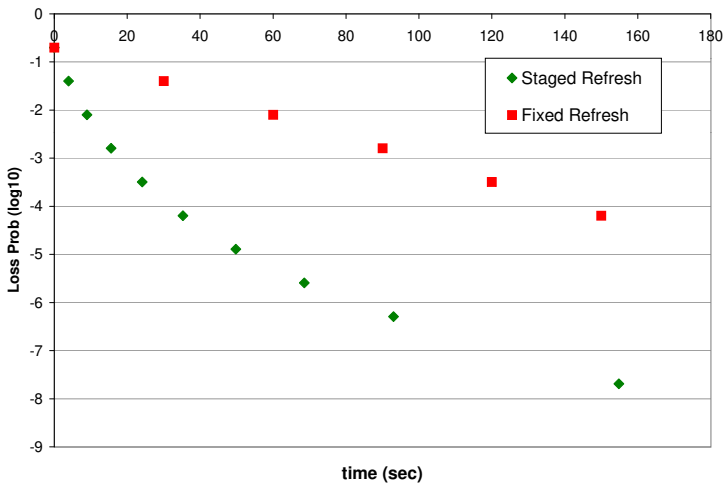

Fig. 4. Message loss probability comparison.

	Bytes Tx in first 60 seconds	*Bytes Tx in 60 minutes*
Fixed Refresh	300	18,000
Staged Refresh (no reply)	900	18,600
Staged Refresh (with reply)	300	900

Fig. 5. Protocol overhead comparison.

[6] D. Mitzel, D. Estrin, S. Shenker, and L. Zhang. An Architectural Comparison of ST-II and RSVP. Infocom '94, June 1994

[7] M. Yajnik, J. Kurose, and D. Towsley. Packet Loss Correlation in the MBone Multicast Network. November 1996.

[8] D. Zappala. RSRR: A Routing Interface For RSVP. INTERNET DRAFT, November 1996. Work in progress.