Staged Refresh Timers for RSVP

Ping Pan and Henning Schulzrinne

Abstract—**The current resource Reservation Protocol (RSVP) design has no reliability mechanism for the delivery of control messages. Instead, RSVP relies on periodic refresh between routers to maintain reservation states. This approach has several problems in a congested network. End systems send PATH and RESV messages to set up RSVP connections. If the *first* PATH or RESV message from an end system is accidentally lost in the network, a copy of the message will not be retransmitted until the end of a refresh interval, causing a delay of 30 seconds or more until a reservation is established. If a congested link causes a tear-down message (PATHTEAR or RESVTEAR) to be dropped, the corresponding reservation will not be removed from the routers until the RSVP cleanup timer expires.**

We present an RSVP enhancement called staged refresh timers to support fast and reliable message delivery that ensures hop-by-hop delivery of control messages without violating the soft-state design. The enhancement is backwards-compatible and can be easily added to current implementations. The new approach can speed up the delivery of trigger messages while reducing the amount of refresh messages. The approach is also applicable to other soft-state protocols.

Keywords—**RSVP; soft state; reliability; signaling.**

I. Introduction

The Reservation Protocol (RSVP) [1], [2] has been designed to exchange resource reservation information among routers in an internet. One of its advantages is that it relies on *soft state* to maintain reservation state in each router: Reservations will disappear by themselves if they are not refreshed periodically. This avoids orphan reservations and allows reservations to adapt quickly to routing changes, without involvement of the end systems. End systems send explicit tear-down messages to speed up the removal of reservations when routes change or the application exits.

RSVP sends its control messages as IP datagrams with no reliability guarantee. It relies on the 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.

Packet losses in the current Internet can be frequent, unfortunately. In today's Internet multicast backbone (Mbone), the packet loss rate [3] is approximately 1-2% on average, and can occasionally reach 20% or more on congested links. The existing RSVP message delivery mechanism will not work well in such an environment. For example, when a user tries to make a reservation over the network, if the *first* reservation request (RESV) is lost due to congestion, it will not be retransmitted over the congested link until the next refresh cycle arrives. The default refresh interval is 30 seconds.

Thus, the first few seconds of, say, a multimedia flow may experience degraded quality of service as packets are carried on a best-effort basis rather than as a reserved flow. Unfortunately, packet loss is more likely to delay reservations just when needed most, i.e., when packet loss rates for best-effort service are high.

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 and the Department of Electrical Engineering, Columbia University, New York, NY 10027. schulzrinne@cs.columbia.edu.

RSVP soft states are managed hop-by-hop, i.e., no network entities other than the node that sent the original refresh message can retransmit a refresh message. Thus, a user cannot accelerate the reservation process by retransmitting RESV messages.

RSVP also does not retransmit tear-down messages. If, for example, a user tries to remove a reservation, and the message (RESVTEAR) is lost, the reservation will remain in place until it times out, by default after 90 seconds. If holding a reservation incurs costs, the user will have to pay for the extra time that has been spent waiting for the reservation time-out. Also, network resources are used inefficiently. Network providers will have to account for this uncertainty in their billing policies.

In this paper, we propose a simple RSVP extension that provides a mechanism to deliver RSVP messages faster and more reliably, that is backward compatible with the existing implementations, and that reduces the number of refreshes among routers, contributing to protocol scalability.

II. Terminology

A. Sending and Receiving Nodes

A sending node is a router or host that generates RSVP messages. A receiving node is defined as the RSVP router or host that is one hop away from a sending node. In a shared-media or non-broadcast multiple access (NBMA) network such as an ATM subnet, a sending node may have multiple receiving nodes. In some cases, not all routers between sending and receiving nodes implement RSVP. We refer to these networks as non-RSVP clouds.

B. Trigger and Refresh Message

In RSVP, control traffic can be categorized into two types: trigger and refresh messages. Trigger messages are generated by an RSVP host or a router due to state changes. Such state changes include the initiation of a new state, a route change that altered the reservation paths, or a reservation modification by a downstream router. PATH, RESV, PATHTEAR and RESVTEAR serve as RSVP trigger messages.

Refresh messages, on the other hand, contain replicated state information generated by a router to maintain state. As indicated in the introduction, RSVP periodically refreshes state for robustness. For instance, if the RSVP daemon on a router crashes and resets, it loses all RSVP state information. However, since its neighbor routers send copies of RSVP state information periodically, the router can recover the lost states within one refresh interval. A refresh message can be either a PATH or RESV message.

The RSVP routing interface [4] can detect state changes, so that refresh messages are not needed to update router reservation states. If the RSVP daemon is reasonably reliable, refresh messages are more of a safety mechanism than actually used for network operation and can thus be sent very infrequently,

in the range of hours instead of 30 seconds. This greatly reduces the traffic and processing impact of RSVP messages and makes RSVP signaling at least as efficient as circuit-switched setup protocols. However, this requires that trigger messages are delivered reliably.

III. RSVP Extensions

A. Outline of Operation

We propose the following feedback mechanism for RSVP trigger message delivery: When sending an RSVP trigger message, a node inserts a new echo-request flag into the RSVP common header of the message. Upon reception, a receiving node acknowledges the arrival of the message by sending back an echo-reply. When the sending node receives this echo-reply for a PATH or RESV message, it will automatically scale back the refresh rate for these messages for the flow. If the trigger message was a flow tear-down, no more tear-down messages are sent, just as in the current RSVP specification. Until the echo reply is received, the sending node will retransmit the trigger message. The interval between retransmissions is governed by a *staged refresh timer*. The staged refresh timer starts at a small interval which increases exponentially until it reaches a threshold. From that point on, the sending node will use a fixed timer to refresh PATH and RESV messages and stop re-transmitting tear-down messages. This mechanism is designed so that the message load is only slightly larger than in the current specification even if a node does not support this staged refresh timer.

The proposed mechanism requires several minor modifications to the current version of RSVP: a new bit is defined in the flag field of the RSVP common header, and four new message types are created for echo-reply. The echo reply messages are simple copies of the message to be confirmed, with the message type changed. While PATH messages are generated end-to-end, PATH echo-replies are hop-by-hop, using the previous hop (PHOP) field from the message.

B. Time Parameters

The new extension makes the use of the following time parameters:

Fast refresh interval R_f: R_f is the initial retransmission interval for trigger messages. After sending the message for the first time, the sending node will schedule a retransmission after R_f seconds. The value of R_f could be as small as the round trip time (RTT) between a sending and a receiving node, if known. Unless a node knows that all receiving nodes support echo-replies, a slightly larger value of, for example, 3 seconds is suggested. This is the value used in the examples in this paper.

Slow refresh interval R_s: The sending node retransmits with this interval after it has determined that the receiving nodes support the RSVP echo-reply. To reduce the number of unnecessary refreshes in a stable network, R_s can be set to a large value. The value of R_s can be set for each egress interface. Throughout the remainder of the paper we assume a value of 15 minutes for R_s.

Fixed refresh interval R: A node retransmits the trigger message with the interval $(1 + \Delta)^i R_f$ until the refresh interval reaches the fixed refresh interval R or an echo reply has been received. If no reply has been received, the node continues to retransmit refreshes every R seconds. We choose a value for R of 30 seconds, the same value as the refresh interval in the current RSVP specification.

Increment value Δ: Δ governs the speed with which the sender increases the refresh interval. The ratio of two successive refresh intervals is $(1 + \Delta)$. We arbitrarily set Δ to 0.30, which is also the the same value as the Slew.Max parameter that has been defined in RSVP to increase the retransmission and timeout interval for long-lived flows using local repairs.

C. Staged Refresh

After a sending node transmits a trigger message, it will immediately schedule a retransmission after R_f seconds. If it receives echo-replies, the sending node will change the refresh interval to R_s. Otherwise, it will retransmit the message after $(1 + \Delta)R_f$ seconds. The staged retransmission will continue until either echo-replies are received, or the refresh interval has been increased to R_f.

The refresh interval for each refresh cycle i can be described as:

$$R_i = \begin{cases} (1+\Delta)^i R_f & \text{if no reply and } i \le I \\ R & \text{if no reply and } i > I \\ R_s & \text{after echo reply} \end{cases}$$

where

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

is the number of retransmissions before the interval reaches the fixed refresh interval R.

Figure 1 illustrates how various refresh mechanisms The figure shows that the number of refreshes is reduced if both sending and receiving node support the new extension. If the receiving nodes do not reply to a trigger message, the sending node generates several refresh messages until the refresh interval converges to the fixed refresh interval R. While incurring additional overhead, these retransmissions increase the likelihood that the reservation state will be established even in a lossy network. In the figure, the sending node transmit seven messages before reaching the refresh interval R. With a larger value of Δ, R_f could be decreased to accelerate state establishment.

The implementation of staged refresh is simple. A sending node can use the following algorithm when the RSVP refresh timer for state (flow) k has expired:

if $(R_k < R)$
    $R_k \rightarrow R_k(1 + \Delta)$
    send out a refresh message;
    wake up in state k after R_k seconds;
    exit.
else
    $R_k \rightarrow R$
    if (the state k is for a tear-down message)
        clean up state k;
        exit.
    else
        send out a refresh message;
        wake up state k after R_k seconds;
        exit.

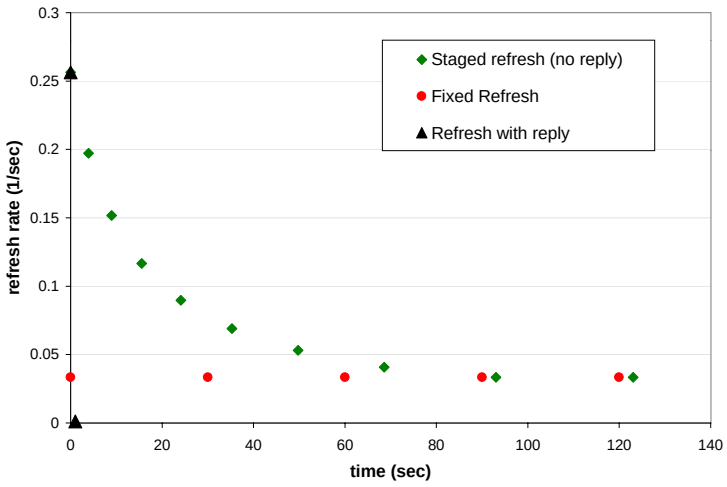

Fig. 1. RSVP refresh comparison ($R_f = 3$ s, $R_s = 15$ min, $R = 30$ s, $\Delta = 0.3$)

Asynchronously, when a sending node receives echo-replies from the receiving nodes, it will change the refresh interval R_k to R_s for state k.

IV. 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 the current RSVP specification, sending nodes refresh the soft states with fixed timers. In our design, sending nodes rely on echo request/reply mechanism to "learn" about the status of receiving nodes. If a sending node does not receive echo replies from the receiving nodes after several tries, it will assume the receiving nodes do not support the new extension, and switch its refresh interval to a fixed value. The RSVP operation is not affected at the receiving nodes.

B. Computing Cleanup Timeout Values

Each RSVP PATH and RESV message carries a refresh interval in its `TIME_VALUES` object. Receiver nodes use the refresh interval to compute the cleanup timeout interval that governs the lifetime of reservation state that has not been refreshed. Generally, the cleanup timeout interval is a small multiple of refresh interval.

In the staged refresh design, a sending node initially places the slow refresh timer, R_s, in the PATH or RESV message. For the receiving nodes that do not support the new extension, the sending node will insert R in the refresh messages after the actual refresh interval has been increased to R. If the receiving nodes do support the new extension, they will set the cleanup timeout interval based on R_s.

C. Handling of Tear-Down Messages

RSVP uses PATHTEAR and RESVTEAR messages to tear-down path and reservation states, respectively. According to the current specification, sending nodes only generate one tear-message per flow. If the message is accidentally dropped along the way, the reserved resource will not be released until the cleanup timer expires. However, receiving duplicate tear-down messages at a receiving node should not impact the operation of RSVP in a proper implementation.

In our RSVP extension, we have altered the processing rules for tear-down messages at the sending node. Instead of deleting the state after a tear-down message is sent, a sending node will release all resource allocated to the state, and mark the state as *closing*. 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 that the receiving nodes do not support the extension.

D. Operation in an 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 the longer refresh timer R_s based on having received echo-replies.

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 the longer refresh interval R_s, even though it has received echo-replies from R1 and R2.

In this case, a sending node has two alternatives:

- It can query a local database such as the ARP or MARS server to find out the exact number of the next-hop receivers. It then switches to a longer refresh interval after receiving echo-replies from all receiving nodes.

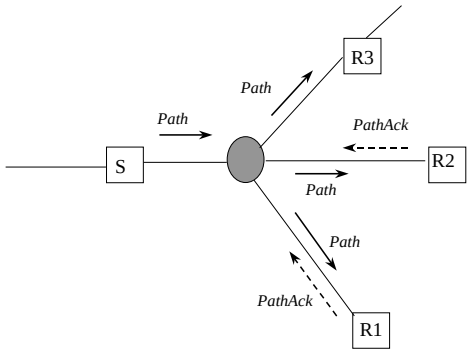

Fig. 2. PATH messages in an NBMA network

• Since PATH messages are mainly used for traffic advertisement purposes, the sending node may not need to use staged refresh timers for PATH messages. In an NBMA network, the staged refresh time mechanism would only make sense for the message delivery of RESV, RESVTEAR and PATHTEAR messages.

In case of PATHTEAR message, a sending node always knows 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 interval is increased to the fixed refresh rate R. Then it stops re-transmitting.

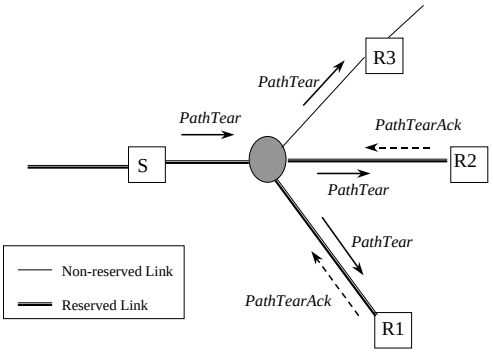

Fig. 3. PathTear message in an NBMA network

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.

V. Evaluation

A. Reduced 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. We assume the message loss probability for a single message is 20% [3]. In the example, four refreshes are sent within the first 30 seconds if a staged refresh timer is used, compared with only one refresh with the fixed refresh timer. The probability that no reservation is established after half a minute is reduced to the neighborhood of $3 \cdot 10^{-4}$ compared with $4 \cdot 10^{-2}$ with the current fixed timer. For a loss rate of 2%, the failure probabilities are $3 \cdot 10^{-9}$ and $4 \cdot 10^{-4}$, respectively.

B. Reduced Protocol Overhead

	60 s	60 min
Fixed refresh	300	18,000
Slewed refresh	300	1,950
Staged refresh (no reply)	900	18,600
Staged refresh (with reply)	300	900

Fig. 5. Protocol overhead (bytes sent) for different RSVP timer mechanisms

When RSVP trigger messages have been acknowledged by echo-replies, the soft state refresh frequency is reduced. In Table 5, we show the protocol overhead for a single RSVP flow, using the same parameters as before. We also compare this to the mechanism described in [2, p. 57], which increases the refresh interval by a slew factor (here, 0.3)[1]. If the receiving node supports the staged timer extension, the number of bytes to be transmitted in an hour is only 900 bytes. Even if the receiving node does not support the new extension, the amount of data being transmitted over an hour is nearly the same compared with the fixed refresh timer case.

For a link that requires to manage thousands of RSVP flows, protocol overhead 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 block. We added staged refresh timers to IBM's RSVP implementation, requiring only a few dozen lines of extra C code.

VI. Conclusion and Future Work

We believe that RSVP message delivery mechanism requires some degree of reliability guarantee to make RSVP useful for individual applications rather than reserving "pipes". One way of improving reliability is to grant some minimal bandwidth

[1] As far as we know, this has not been implemented anywhere.

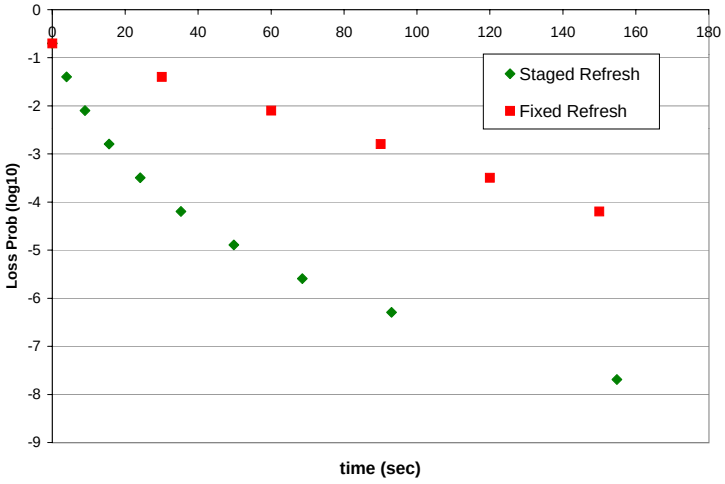

Fig. 4. Comparison of message loss probability as a function of time for fixed and staged refresh

for RSVP messages to protect them from congestion losses, as suggested in the RSVP specification [2]. However, this may require additional functionality at both sending and receiving nodes and does not help if RSVP messages have to traverse non-RSVP clouds. It is also not clear how this can be achieved in a backward-compatible manner.

In this paper, we presented a mechanism called staged refresh timers that enhances the current RSVP message delivery and is completely backward compatible. Staged refresh timers are easy to add to RSVP router and host implementations and save both processing and bandwidth overhead.

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

The staged refresh timer mechanism is an example of state management that falls somewhere between "classical" handshake-based reliability as found in ATM signaling, for example, and purely timer-based soft-state protocols such as the original RSVP proposal [1], delta-t [5] or IGMPv1 [6]. An approach similar to staged refresh is also being used by the Session Initiation Protocol [7] to confirm state establishment. Generally speaking, experience has shown that state management is greatly simplified by requiring only one message (in each direction) to establish state, rather than going through several intermedia states. State establishment messages should be idempotent and should contain a globally (spatially and temporally) unique state label, so that retransmissions of the same message can be ignored.

Since only neighboring routers are involved in the reliability mechanism described here, these routers can easily estimate round-trip times, thus further tightening the retransmission interval, if desired.

While staged refresh timers improve scalability, RSVP remains a rather complex protocol. Alternative approaches to reserve reservation [8], [9] may offer better scaling properties.

Acknowledgments. The authors wish to thank Shai Herzog for the useful discussions at the early stages of this work and 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*, vol. 7, pp. 8–18, Sept. 1993.

[2] R. Braden, L. Zhang, S. Berson, S. Herzog, and S. Jamin, "Resource reservation protocol (RSVP) – version 1 functional specification," Internet Draft, Internet Engineering Task Force, June 1997. Work in progress.

[3] M. Yajnik, J. Kurose, and D. Towsley, "Packet loss correlation in the MBone multicast network," in *Proceedings of Global Internet*, (London, England), Nov. 1996.

[4] D. Zappala, "RSRR: a routing interface for RSVP," Internet Draft (expired), Internet Engineering Task Force, Nov. 1996. Work in progress.

[5] R. W. Watson, "The Delta-t transport protocol: Features and experience," in *First IFIP WG6.1/WG6.4 International Workshop on Protocols for High-Speed Networks* (H. Rudin and R. Williamson, eds.), (Zürich, Switzerland), pp. 3–17, May 1989.

[6] S. Deering, "Host extensions for IP multicasting," Request for Comments (Standard) STD 5, RFC 1112, Internet Engineering Task Force, Aug. 1989. (Obsoletes RFC0988).

[7] M. Handley, H. Schulzrinne, and E. Schooler, "SIP: Session initiation protocol," Internet Draft, Internet Engineering Task Force, July 1997. Work in progress.

[8] D. Clark and J. Wroclawski, "An approach to service allocation in the internet," Internet Draft, Internet Engineering Task Force, Aug. 1997. Work in progress.

[9] P. P. Pan and H. Schulzrinne, "YESSIR: A simple reservation mechanism for the Internet," in *submitted for publication*, 1998.