Enhancing RSVP Scalability

Ping Pan
Bell Laboratories
101 Crawfords Corner Road
Holmdel, NJ 07733
pingpan@research.bell-labs.com

Henning Schulzrinne
Computer Science Department
Columbia University
New York, NY 10027
schulzrinne@cs.columbia.edu

December 5, 2000

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.

1 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.

The use of *soft states* in RSVP can result in both reliability and scaling problems.

The reliability problem occurs when RSVP messages are lost during transmission. Packet losses in the current Internet can be frequent on some links, unfortunately. In one extreme case, it was measured in 1996 [3] that the packet loss rate in the Internet multicast backbone (Mbone) [4] was approximately 1-2% on average, and could occasionally reach 20% or more on congested links. The existing RSVP message delivery mechanism will not work well in an such 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.

Another problem related to reliability is that RSVP 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.

The scaling problem is linked to the resource requirements in terms of bandwidth usage and processing overhead. The resource requirements increase proportionally with the number of RSVP sessions. Each session requires the generation, transmission, reception and processing of PATH and RESV messages per refresh period. Supporting a large number of sessions, and the corresponding volume of refresh messages, presents a scaling problem.

One way to address the reliability of RSVP signaling is to decrease the refresh period on all network nodes. This can improve the probability that state will be installed in the face of message lose, but at the cost of increasing refresh message volume and associated processing requirements and aggravate the scaling problem. On the other hand, a simple way to reduce the effect of scaling problem in RSVP is to increase the refresh period on all network nodes. However, this increases the time it takes to synchronize reservation state.

In this chapter, we present a simple RSVP extension called *staged refresh timer*, that provides a mechanism to deliver RSVP messages faster and more reliably, and has the property of reducing the number of refreshes among network nodes, contributing to protocol scalability. We will also briefly describe two other mechanisms, *message bundling* and *summary refresh*, that can further reduce the processing cost of refresh messages. All the extensions described here are designed to be backward compatible with the existing RSVP implementations.

2 Terminology

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*.

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 process 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.

3 RSVP Staged Refresh Timer Extension

A common practice in supporting message delivering reliability is to use some type of feedback or acknowledgement mechanism between senders and receivers. However, using feedback between RSVP end-users, as being proposed in [5], won't work unless we significantly change the original protocol. This is because RSVP soft states are managed hop-by-hop, and no network entities other than the node that sent the original refresh message can retransmit a refresh message. That is, end users have no control over the message delivery process in the network. They cannot accelerate delivery by simply retransmitting RSVP messages. Hence,we argue that *the only way to accomplish reliable RSVP message delivery is to rely on some hop-by-hop feedback mechanism.*

RSVP relies on underlying routing protocols to setup reservation path. Routing protocols can detect most of the state changes in the network. When a network failure that results in reservation path modification occurs, router's routing process can upcall the RSVP process to adjust the effected reservations by initiating new RSVP requests while removing the old states. In RSVP, this procedure is called *local repair*. This implies that *once reservation states are installed, there is no reason for RSVP to frequently update router reservation states.* If the RSVP process 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.

In this section, we will describe a reliable trigger message delivery mechanism base on the above observation.

3.1 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 retransmission 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 several new RSVP objects need to be created for echo-request and echo-reply. Although PATH messages are generated end-to-end, PATH echo-replies are hop-by-hop, using the previous hop (PHOP) field from the message.

3.2 Time Parameters

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

Fast refresh interval $\mathcal{R}_f$: $\mathcal{R}_f$ is the initial retransmission interval for trigger messages. After sending the message for the first time, the sending node will schedule a retransmission after $\mathcal{R}_f$ seconds. The value of $\mathcal{R}_f$ 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.

Slow refresh interval $\mathcal{R}_s$: 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, $\mathcal{R}_s$ can be set to a large value. The value of $\mathcal{R}_s$ can be set for each egress interface. We choose 15 minutes as the default value.

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 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.

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

3.3 Staged Refresh Timer Algorithm

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

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

$$R_i = \begin{cases} (1+\Delta)^i \, \mathcal{R}_f & \text{if no reply and } i \leq I \\ \mathcal{R}_c & \text{if no reply and } i > I \\ \mathcal{R}_s & \text{after echo reply} \end{cases}$$

where

$$I = \frac{\log \frac{\mathcal{R}_c}{\mathcal{R}_f}}{\log(1+\Delta)}$$

I is the number of retransmission before the interval reaches the fixed refresh interval $\mathcal{R}_c$.

The implementation of staged refresh is quite simple. The router algorithm operates on two sets of parameters. A set of "global" timing variables consists of $\mathcal{R}_f$, $\mathcal{R}_s$, $\mathcal{R}_c$ and Δ. Each reservation state, k, need to maintain the following attributes:

k.refresh-timer	the state refresh timer.
k.cleanup-timer	the state expiration timer.
k.type	reservation state type: either Active or Tear-Down

k.refresh-timer governs when to send a refresh message. k.cleanup-timer is reset to zero upon the reception of a refresh message. Figure 1 is the router's algorithm at sending nodes. We only show the procedures that are relevant to RSVP time management. As shown in the figure, if the value of $\mathcal{R}_f \, (1+\Delta)^i$ exceeds $\mathcal{R}_c$ after i tries, we will switch the refresh timer to $\mathcal{R}_c$.

In a network that has many RSVP reservations, routers may have to use a large number of timers to support the staged refresh timer algorithm. This can be a problem on some router OS's. In addition, conventional timer routines can be very CPU intensive. To overcome this problem, we can implement various hashed or hierarchical timing wheels, such as the one being described in [6], to manipulate a large number of soft-state timers with a handful of *real* OS timers.

In the IBM's implementation, the staged refresh timer mechanism was based on three hashed timing wheels, one for each of the fast, slow and fixed refresh cycles. At message processing time, each reservation state is inserted to one of the timing wheels. Each timing wheel is driven by a single repeating timer provided by the router's OS.

Periodic process synchronization is generally considered to be harmful [7]. To avoid the synchronization due to RSVP periodic refreshes, all the timers need to be randomized during the implementation.

Figure 2 compares various refresh mechanisms. The figure shows that the number of refreshes is greatly 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

```
1: on receiving a trigger message:
        create the corresponding state k;
        if the message is a tear-down message
                k.type ← Tear-Down;
                remove k's reserved resource;
        else
                k.type ← Active;
        k.refresh-timer ← R_f;
        forward the trigger message toward the destination.

2: on receiving an echo-reply for k:
        if k.type = Tear-Down
                delete k.
        else
                k.refresh-timer ← R_s;

3: on expiration of k.refresh-timer:
        if k.refresh-timer < R_c
                if R_c < (k.refresh-timer) · (1 + Δ)
                        k.refresh-timer ← R_c;
                else
                        k.refresh-timer ← (k.refresh-timer) · (1 + Δ);
                construct the corresponding trigger message for k;
                retransmit the trigger message;
        else if k.refresh-timer = R_c and k.type = Tear-Down
                delete k;
        else
                construct the corresponding refresh message for k;
                send out the refresh message;
```

Figure 1: Staged Refresh Timer at a Sending Node - Router Algorithm

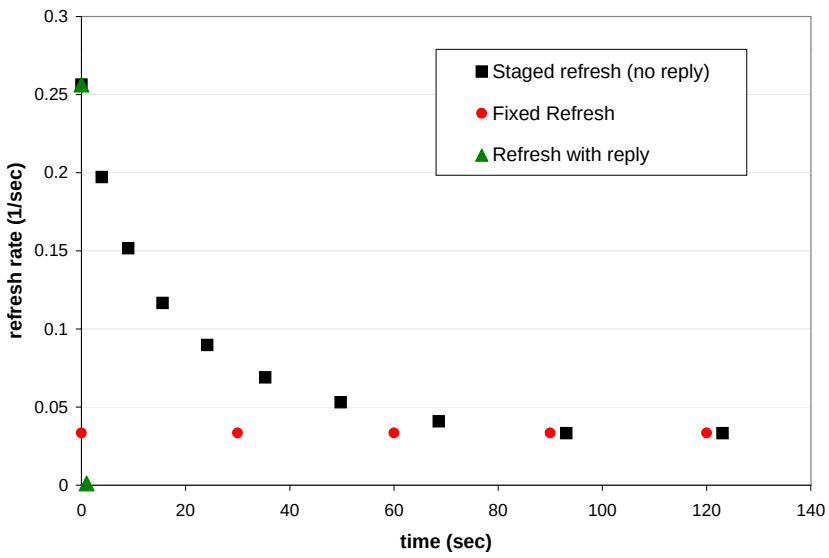

Figure 2: RSVP refresh comparison ($\mathcal{R}_f = 3$ s, $\mathcal{R}_s = 15$ min, $\mathcal{R}_c = 30$ s, $\Delta = 0.3$)

to the fixed refresh interval $\mathcal{R}_c$. While incurring additional overhead, these retransmission increase the likelihood that the reservation state will be established even in a lossy network. In the figure, the sending node transmit $I = 7$ messages before reaching the refresh interval $\mathcal{R}_c$. With a larger value of Δ, $\mathcal{R}_f$ could be decreased to accelerate state establishment.

3.4 Special Considerations

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. To achieve this goal, we have designed the protocol extension in such a way that much of the message manipulation and processing are done at sending nodes, while making no assumption about the capability at receiving nodes.

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.

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. As suggested in the RSVP specification, we set the cleanup timeout value to be 3 refresh interval by default.

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

```
1: on receiving a trigger message carrying refresh value R:
        find/create the corresponding state k;
        if the message is a tear-down message
                free k's reserved resource;
                delete k;
        else
                k.cleanup-timer ← 3 · R;
        if support the staged refresh extension
                reply an echo-reply back to the sending node.

2: on receiving a refresh message for k carrying refresh value R:
        k.cleanup-timer ← 3 · R;

3: on expiration of k.cleanup-timer:
        free k's reserved resource;
        delete k.
```

Figure 3: Staged Refresh Timer at a Receiving Node - Router Algorithm

Figure 3 is the algorithm on processing messages on receiving nodes. Note, the receiving nodes do not need any of the staged refresh timer parameters. The nodes simply require to send an echo-reply back to the sending node upon the receipt of a trigger message.

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-down 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.

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.

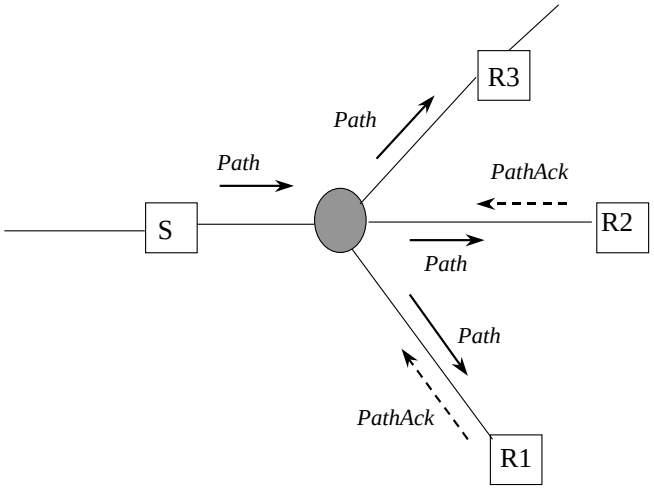

Figure 4: Example: PATH messages in an NBMA network.

For example, as shown in Figure 4, if the receiving node R3 does not support the new RSVP extension, the sending node S should not change to the longer refresh interval $\mathcal{R}_s$, 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 servers [8, 9] 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.
- 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 during the RESV message processing time.

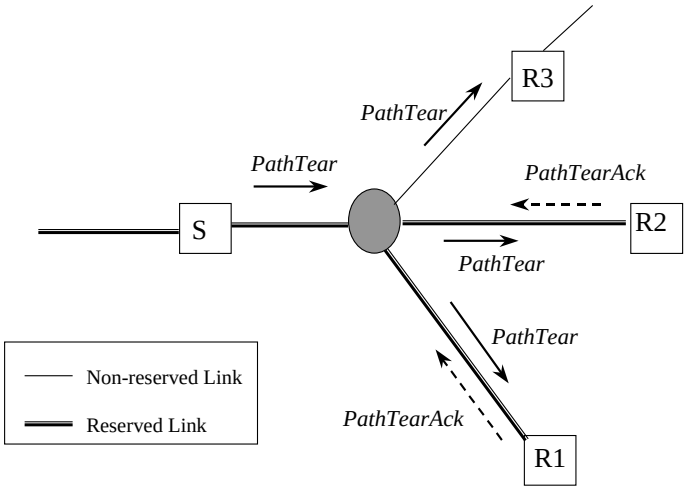

Figure 5: PathTear message in an NBMA network

As shown in Figure 5, 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.

4 Evaluation

	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

Figure 6: Protocol overhead (bytes sent) for different RSVP timer mechanisms

Reduced Protocol Overhead When RSVP trigger messages have been acknowledged by echo-replies, the soft state refresh frequency is reduced. In Table 6, we show the protocol overhead for a single RSVP flow, using the same parameters as before. The RSVP message size is 150 bytes, which is the size of a regular RESV. We also compare this to the mechanism described in [2, p. 57], which simply increases the refresh

interval by a slew factor (here, 0.3). As we have discussed previously, increasing or decreasing refresh interval can have undesirable reliability and scalability effects to RSVP. 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.

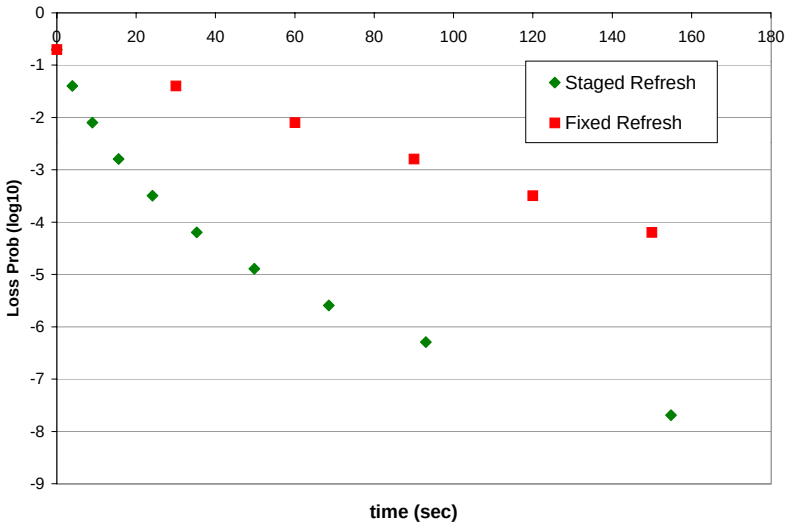

Figure 7: Comparison of message loss probability as a function of time for fixed and staged refresh

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.

5 Other Proposals

There have been several other proposals [10, 11, 12] in recent years on improving scalability and reliability of soft-state-driven protocols.

The mechanisms introduced in [11, 12] require both sending and receiving nodes to operate a set of new refresh reduction algorithms simultaneously. Since RSVP has already been deployed in several networks, both proposals present a backward compatibility problem and may cause deployment difficulties.

However, two other refresh reduction approaches, *message bundling* and *summary refresh*, described in [10], are backward compatible and can solve specific problems in RSVP.

5.1 Message Bundling

Message bundling is to pack multiple RSVP messages between two neighboring nodes into a single datagram. The goal is to reduce total number of RSVP messages that routers have to process. This solution is particularly useful to relive the processing burden on routers that are based on UNIX operating system.

Socket interface in UNIX is a processing bottleneck. Although BSD, a popular version of UNIX, has been re-designed over the years to improve virtual memory and I/O system interface [13], processing overhead between kernel and user space through sockets remains to be significant.

To demonstrate the problem, we ran a simple test on an Intel Celeron 500 MHz PC running FreeBSD 3.4. This PC was connected to two other PC's through 10 Mb/s Ethernet interfaces; we confirmed that it could route traffic between the interfaces at media speed. Our goal is to measure the message processing time at both kernel and user space. RSVP PATH messages are generally encapsulated with IP Router Alert option. Since the processing time should be the same for all IP option types, and we want to have the flexibility to evaluate the performance variation base on packet size, we thus used the *ping -R* command to generate IP record route option packets. We used a modified FreeBSD kernel that can intercept passing IP option packets from the wire and direct them to the user space quickly through raw sockets [14]. At the user space, we used a system function call, *ipodump*, to reinject IP option packets back into the network immediately after intercepting them. Two simple timing checks were added in the kernel, one at the beginning of the IP input, the other at the end of IP output routine. Using this, we can measure the processing delay at socket interface.

The results of our measurements are shown in Figure 5.1. The processing delay does not depend significantly on the packet size. Out of the 16 sets of data we have collected, the delay for processing 1-byte ICMP packets with IP record route option was 116.4 μsec, compared with 132.6 μs for 1400-byte packets, *i.e.*, only a 12% increase. We believe this is due to the efficient memory management in BSD.

However, the average packet processing time in the user space, including reading from and writing to sockets, was 46.28 μs. In comparison, an intercepted packet spent 124.64 μs on average traversing kernel and user space. That is, the BSD socket interface contributed as much as 78.36 μs or 62.87% of the process!

The test results thus imply the following: *for BSD routers, increasing message size does not impact the performance very much, but reducing the sheer number of messages on the wire can greatly relive the processing overhead.*

5.2 Summary Refresh

In RSVP, the refresh messages have in the same format as the trigger messages. Given RSVP flow information can be quite descriptive and large, so it seems to be a waste of link bandwidth to retransmit the same

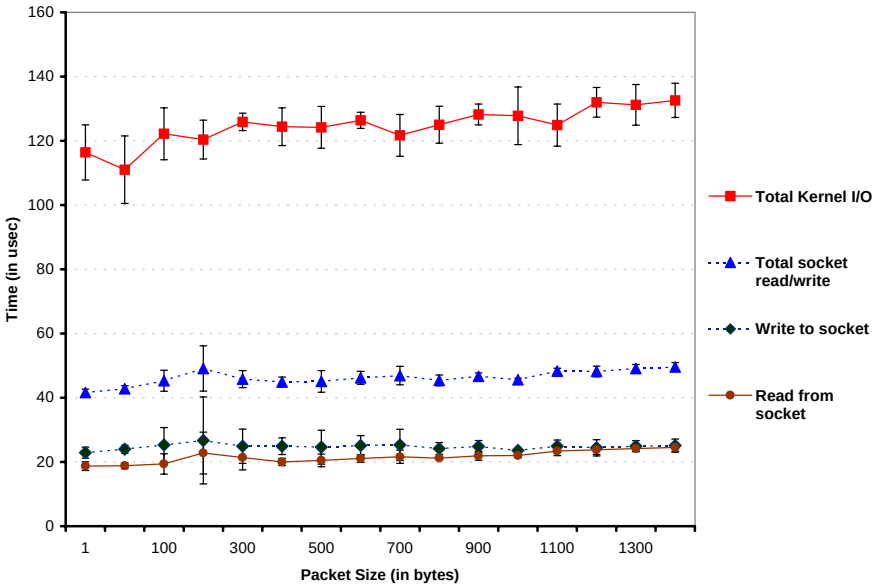

Figure 8: Comparing the kernel and user processing overhead

information over and over again at each refresh cycle. Summary Refresh is simply to assign an identification number to each RSVP state during trigger message processing. At each refresh cycle, the sending nodes only transmit those identification numbers to update reservation states. By using the summary refresh mechanism, each RSVP refresh message can reduce approximately 50% of its size.

Both *bundling* and *summary refresh* use the same mechanism as the one introduced in staged refresh timer to support backward compatibility.

6 Summary and Availability

RSVP message delivery mechanism requires some degree of reliability guarantee to make RSVP useful for individual applications. One way of improving reliability is to grant some minimal bandwidth 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.

We have presented a mechanism called *staged refresh timer* 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 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], PIM [15], RTP [16], SRM [17], and so forth.

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 and message delivery, RSVP remains a rather complex protocol, and can be very slow to setup end-to-end reservations:

- Experience has shown that state management is greatly simplified by requiring only one message (in either direction) to establish state, rather than going through several intermediate states. State establishment messages should be idempotent. However, in RSVP, no matter how we reduce the number of messages on the wire, it requires at least two messages, PATH and RESV, to make a single reservation.

- RSVP uses *blockade states* to handle admission control failure. Since the proposed mechanism cannot recover from the failure conditions quickly, end-users may experience long reservation delays and frequent reservation rejections.

- Finally, resource reservation requires the network routers to install per-flow "states", and the network providers to manage these "states" individually. The Internet size has been growing pretty much according to the Moore's Law. Hence, unless we introduce reservation hierarchy in the Internet, it's doubtful that providers and the routers can accommodate the volume and the requirements from all reservation clients.

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, Ed., L. Zhang, S. Berson, S. Herzog, and S. Jamin, "Resource ReSerVation protocol (RSVP) – version 1 functional specification," Request for Comments 2205, Internet Engineering Task Force, Sept. 1997.

[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] S. Casner, H. Schulzrinne, *et al.*, "Frequently asked questions (faq) on the multicast backbone (mbone)." http://www.cs.columbia.edu/ hgs/internet/mbone-faq.html.

[5] L. Mathy, D. Hutchison, and S. Simpson, "Modelling and improving flow establishment in RSVP," in *Proc. of Protocols for High Speed Networks (PfHSN)*, Aug. 1999.

[6] G. Varghese and A. Lauck, "Hashed and hierarchical timing wheels: Efficient data structures for implementing a timer facility," *IEEE/ACM Transactions on Networking*, vol. 5, pp. 824–834, Dec. 1997.

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

[8] G. Armitage, "Support for multicast over UNI 3.0/3.1 based ATM networks," Request for Comments 2022, Internet Engineering Task Force, Nov. 1996.

[9] R. Talpade and M. Ammar, "Multicast server architectures for MARS-based ATM multicasting," Request for Comments 2149, Internet Engineering Task Force, May 1997.

[10] L. Berger, D. Gan, G. Swallow, P. Pan, F. Tommasi, and S. Molendini, "RSVP refresh overhead reduction extensions," Internet Draft, Internet Engineering Task Force, June 2000. Work in progress.

[11] P. Sharma, D. Estrin, S. Floyd, V. Jacobson, P. Sharma, D. Estrin, S. Floyd, and V. Jacobson, "Scalable timers for soft state protocols," in *Proceedings of the Conference on Computer Communications (IEEE Infocom)*, (Kobe, Japan), p. 222, Apr. 1997.

[12] S. Raman and S. McCanne, "A model, analysis, and protocol framework for soft state-based communication," *SIGCOMM'99*, Aug. 1999.

[13] M. McKusick, B. K., M. Karels, and J. S. Quarterman, *The Design and Implementation of the 4.4BSD UNIX Operating System*. Massachusetts: Addison-Wesley Publishing Company, 1996.

[14] P. Pan and H. Schulzrinne, "PF_IPOPTION: A kernel extension for IP option packet processing," Technical Memorandum 10009669-02TM, Bell Labs, Lucent Technologies, Murray Hill, NJ, June 2000.

[15] S. Deering, D. Estrin, D. Farinacci, V. Jacobson, C.-G. Liu, and L. Wei, "An architecture for wide-area multicast routing," in *SIGCOMM Symposium on Communications Architectures and Protocols*, (London, UK), pp. 126–135, Sept. 1994.

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

[17] S. Floyd, V. Jacobson, C.-G. Liu, S. McCanne, and L. Zhang, “Reliable multicast framework for light-weight sessions and application level framing,” in *SIGCOMM Symposium on Communications Architectures and Protocols*, (Cambridge, Massachusetts), pp. –, Sept. 1995.