Staged Refresh Timers for RSVP

Ping Pan
IBM T. J. Watson Research Center
30 Saw Mill River Road
Hawthorne, NY 10532
pan@watson.ibm.com

Henning Schulzrinne
Dept. of Computer Science
Columbia University
1214 Amsterdam Avenue
New York, NY 10027
schulzrinne@cs.columbia.edu

March 28, 1997

Abstract

In the current RSVP design [2, 3], there is no reliability mechanism for control message delivery. Instead it uses periodic refresh messages to maintain RSVP states information between routers. This approach have several potential problems in a congested network. If the *first* PATH or RESV message is accidentally lost, it has to wait for the entire duration of a refresh cycle to be retransmitted. If a tear-down messages is dropped due to link congestion, the reservation will not be removed until RSVP *cleanup* timer expires. The above problems may not be acceptable for certain applications. In this paper, we presents an RSVP extension to support fast and reliable message delivery. In order to make the extension to be adapted easily, the new mechanism is designed to be backward compatible. We will also discuss several design issues and the advantages of the new approach.

1 Introduction

After several 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. At end-user level, by interfacing with RSVP, many real-time applications will be widely used in forms of teleconferencing and Internet-phone. In the network, routers and switches will soon all have traffic shaping and scheduling functionality to support various types of service. In the near future, the Internet will be capable of supporting services that at this point can only be provided by telco companies. However, for further deploying and maintaining commercial networks, some of the existing Internet protocols need to be enhanced.

RSVP has been designed to exchange resource reservation information among routers. It has the advantages [6] of 1) supporting both unicast and multicast while independent from any specific routing protocol, 2) accommodating heterogeneous receiver requirements by using *receiver initiated reservation*, and 3) 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 RSVP PATH or RESV message. Each RSVP host or router also maintain a cleanup timer. When a state is not refreshed within a *cleanup timeout* interval, it will be deleted automatically.

However, as shown in Reference[7], in today's Internet backbone, the backbone packet loss rate is around 1-2% on average, and can occasionally be as high as 20% on congested links. Thus, the existing RSVP message delivery method may not be acceptable. For example, when a user is trying to make a reservation, if the *first* reservation request (RESV) message from a user is lost due to network link noise or congestion, the request will not be resent until the next refresh cycle (by default, the duration between refreshes is 30 seconds.) The user at the same time will suffer from poor quality of service.

One way of solving the above problem is to increase refresh frequency. However, it will cause extra bandwidth and processing overhead because of the increase number of refresh messages between routers.

In case of tear-down messages, 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.

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

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

2 RSVP Extension

2.1 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/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 state maintenance purposes. Periodic refreshing is to provide a level of robustness to make sure that

RSVP hosts and routers have correct state information at all time. For example, if the RSVP sub-system on a router is crashed and reset, the way to recover the lost state information is to have neighbor routers sending a copy of RSVP states to the troubled router. Since most state changes can be discovered by neighbor routers through RSVP routing interface[8], and the chance for a RSVP sub-system crash is fairly low in a production network, the refresh period therefore could be very large, (in terms of minutes, if not hours).

2.2 Operation Outline

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

In each RSVP 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 back an echo-reply message back to the sending node to confirm 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 rate. If there is no reply being received on time, the sending node will continue to generate refresh messages to the receiving node with a fast refresh timer.

A staged refresh timer is used in each sending node, in case receiving nodes do not support the new extension. At the time when the first trigger message is sent, a sending node will use a fast refresh timer initially. As time goes on, the sending node will reduce the refresh rate exponentially until is reaches to a threshold. After which, the sending node treats the receiving node as if it does not support the new RSVP extension: it uses a fixed refresh timer for PATH and RESV messages and stops re-transmitting tear-down messages.

The proposed extension requires several modifications in the current RSVP specification: a new bit is defined in the RSVP common header, and four types of message are created for echo-reply. The detailed protocol level modification is shown in Appendix A.

2.3 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 re-transmit a trigger message. After the node sends a trigger message, it will schedule a retransmission after 1/Rf seconds. The value of 1/Rf could be as small as one RTT (round-trip-time) between the sending node and the receiving node.

  By default, Rf is set to be one refresh every 3 seconds.

- Slow refresh rate, Rs: This is the refresh rate used by the sending node if an echo-reply is received from the receiving node. 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: A sending node will retransmit a trigger message if there is no reply from receiving nodes. And the refresh rate will reduce exponentially. In case of PATH and RESV messages, when it reaches to the rate R, the sending node will use R as the frequency to refresh receiving nodes from then 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 values is (1 + Δ).

  Δ, by default, is set to be 0.30. It could be the same value as Slew.Max that has been defined in RSVP.

2.4 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 again after (1 + Δ)/Rf seconds. (1 + Δ) 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 be equal or larger than 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 rounds that will take for a staged refresh rate to be equal or less than the fixed refresh rate R. For PATH and RESV, a sending node will start to use the fixed timer R after refreshing the receiving nodes I times. Similarly, for tear-down messages, if there is no reply from the receiving nodes after I rounds, a sending node will stop re-transmitting .

In Figure 1, we illustrated how various refreshing mechanism works. As shown, if both sending and receiving node support the new extension, the number of refreshes will is

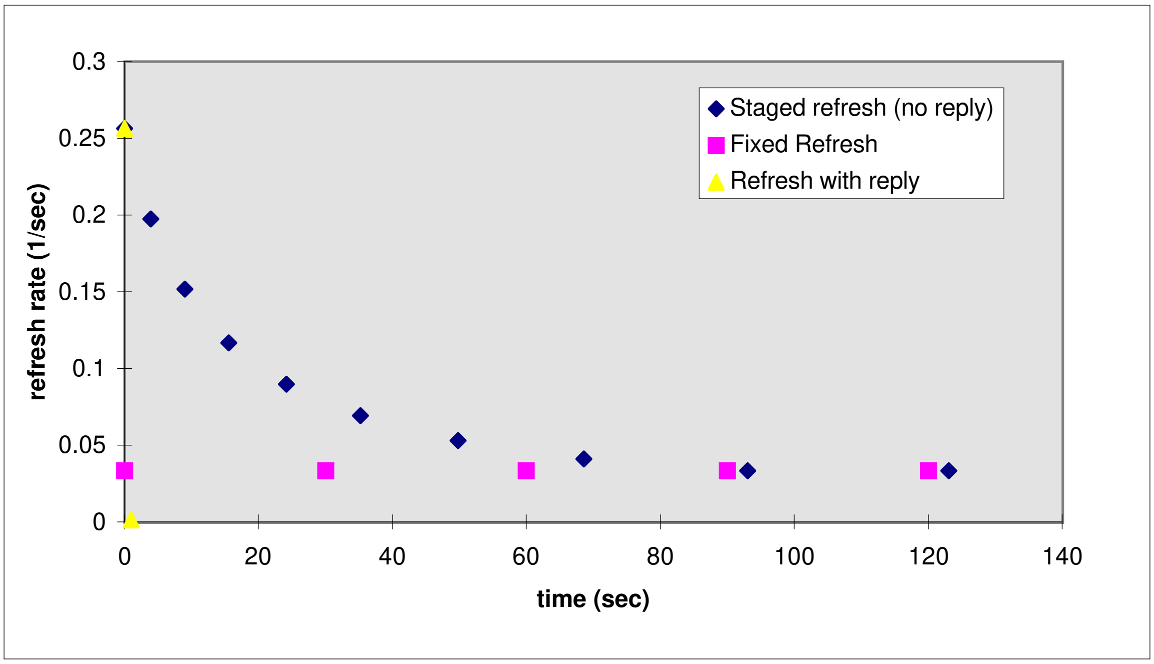

Figure 1: Different types of RSVP refresh mechanisms: Rf $= 1/3$, Rs $= 1/900$, R $= 1/30$, and $\Delta = 0.3$.

reduced dramtically. On the other hand, if the receiving node does not support the new extension, the sending node will converge to a fixed refresh rate eventually. In the example, the sending node starts to use fixed timer after about 60 seconds. To increase the speed of convergence, the sending node can use a larger Δ value.

At each sending node, the following algorithm is used when a RSVP timer is expired for state i:

```
new_refresh_timer = (1 + delta) * current_refresh_timer;

if (new_refresh_timer >= fixed_refresh_timer) {

    if (the message is a tear-down message) {
        clean up state i;
        exit;
    }
    else {
        current_refresh_timer = fixed_refresh_timer;
    }
}
else {
    current_refresh_timer = new_refresh_timer;
```

```
}

retransmit the message;
wake up state i after current_refresh_timer seconds;
exit.
```

3 Special Considerations

3.1 Backward Compatibility

One of the main design objectives for the proposed extension is the support of backward compatibility. 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" the status of receiving nodes. For receiving nodes that do not support the extension, there is no harm to the existing RSVP functionality.

3.2 Computing *Cleanup Timeout* Values

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

In our extension, a sending node inserts the slow refresh timer, 1/Rs, in the *first* PATH or RESV message. If the receiving node supports the extension, the refreshing state will have a *cleanup timeout* interval based on Rs.

In case that a receiving node does not support the new extension, after the sending node realizes that the receiving node does not work with the new extension, it will retransmit RSVP messages with the fixed refresh rate R instead of Rs in TIME_VALUES object. The receiving node will then adjust its *cleanup timeout* interval based on R.

As a result, receiving nodes will always have the accurate timeout information for *soft state* management.

3.3 Handling of Tear-down Messages

In RSVP, PATHTEAR and RESVTEAR are used to tear-down path and reservation state respectively. In the current RSVP design, only one tear-down message is generated from a sending node. If the message is accidentally dropped along the way, the receiving node will have no knowledge of the tear-down. 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. That is, 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 to be *pending*. The state information is saved and to be used for state 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.

3.4 Operation in NBMA Environment

For a multicast RSVP session in a NBMA environment (such as ATM LAN or switched-Ethernet), 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.

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

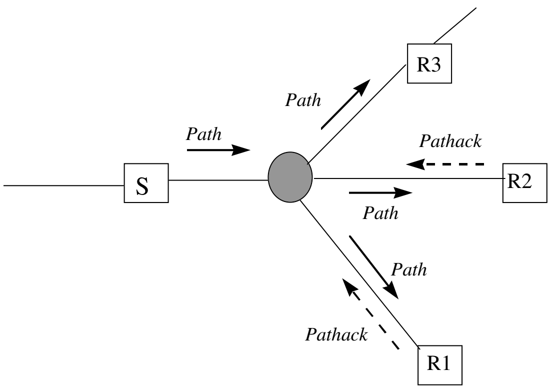

Figure 2: Path message in NBMA

In this case, a sending node should have the option:

1. Query local database such as ARP or MARS server to find out the exact number of the next-hop receivers. It will not switch to a slower refresh rate unless it had received

echo-replies from all known receiving nodes.

2. Since PATH is used for advertisement purposes, turn off the echo-request option for PATH at the egress interface.

However, in case of PATHTEAR message, a sending node has the information about all the receiving nodes that had made reservations. Therefore the following rules should be used:

- A sending node stops re-transmitting PATHTEAR messages once it receives echo-replies from all its reserved 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.

As illustrated in Figure 3, there is no reservation made between S and R3. Therefore after the sending node S receives echo-replies from R1 and R2, it will stop generating PATHTEAR messages.

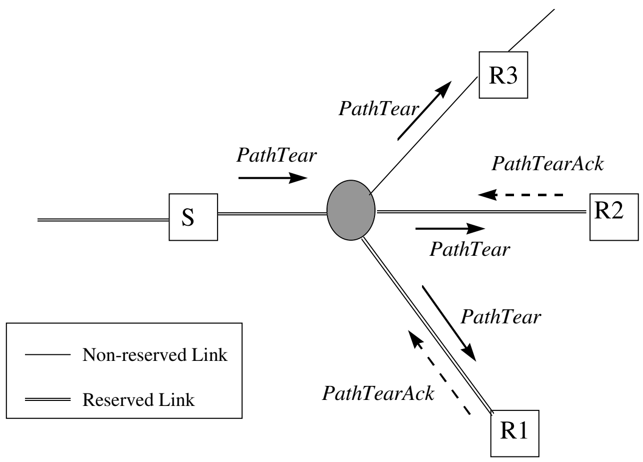

Figure 3: PathTear message in NBMA

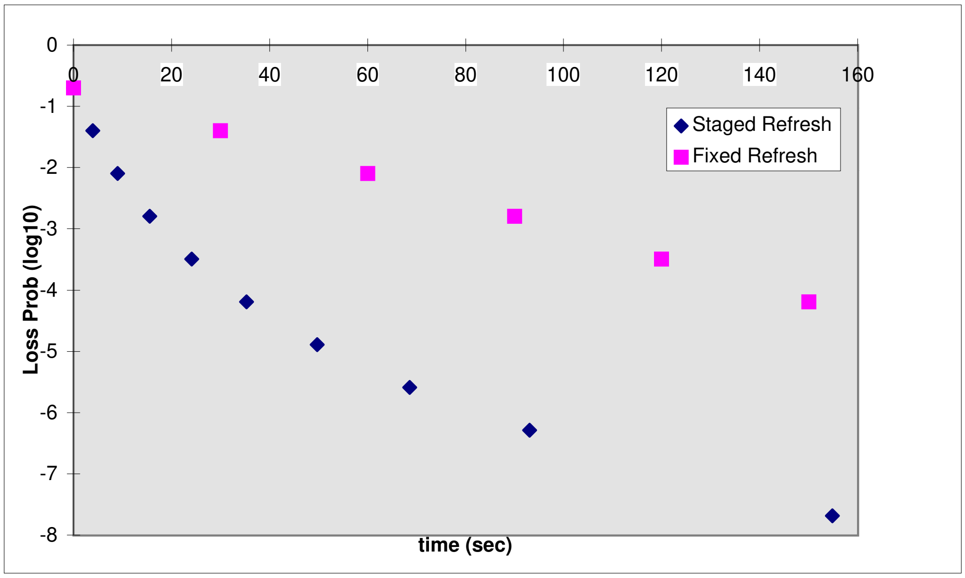

Figure 4: Message loss probability comparison. We use the average link packet loss probability 2% from Reference [7]. Rf = 1/3, R = 1/30 and $\Delta = 0.3$.

4 Advantages

4.1 Reduce Message Loss Probability

Adding feedback mechansim in RSVP message delivery reduce the message loss probability. As shown in Figure 4, within a given interval time after the generation of the *first* trigger message, more RSVP refresh messages are sent over the link. As a result, the message has higher chance to be received by receiving nodes.

In the example, within the first 60 seconds, six refreshes are sent if staged refresh timer is used, compare with only two refreshes in fixed refresh timer. The packet loss probability is reduced to be in the neighborhood of 10^{-4} compare with 10^{-2} over a mildly congested link.

4.2 Simple to Process

The newly defined echo-reply messages are simple to generate and easy to process. As shown in Appendix A, the format of echo-reply messages is 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. On receive, all a sending node needs to do is

1. to locate the RSVP flow from its local state table;

2. to change the flow's refresh timer to slow refresh timer Rs.

The algorithm for staged refresh timer is given in Section 2.4. It takes literally a few dozen lines of C code to implement.

Overall, the processing rules for the new RSVP extension is much simpler comparing with the rules that are used for PATH and RESV message processing[3].

4.3 Low Bandwidth Overhead

After a RSVP flow has been reliably established in a network, its *soft state* refresh interval is changed to a longer period. As a result, in a stable network, the overall number of refresh messages is reduced.

For example, as shown in the table below, 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 a hour. Using staged refresh timer (Rf = 1/3, Rs = 1/900, R = 1/30 and $/Delta = 0.3$), if the receiving node supports the extension, the number of bytes to be transmitted in a hour is only 900 bytes! Also notice, even if the receiving node does not support the new extension, there will be nearly same amount of bytes to be send as the fxied refresh timer case in a hour.

	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

5 Conclusion and Future Works

We believe that RSVP message delivery mechanism requires some type 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 router software platform, and will be tested on several types of router/switch products.

6 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-ietf-rsvp-spec-14.ps, November 1996.

[3] R. Braden, and L. Zhang. Resource ReSerVation Protocol (RSVP) - Version 1 Message Processing Rules. Internet draft-ietf-rsvp-procrules-00.txt, October 1996.

[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

[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

[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-ietf-rsvp-routing-01.txt, November 1996.

A Message Format

A.1 Common Header

For PATH, RESV, PATHTEAR and RESVTEAR messages, there is an additional echo-request flag in the RSVP common header. Four additional new messages have also been defined to support feedback.

The format of the RSVP common-header is:

```
      0             1              2             3
+-------------+-------------+-------------+-------------+
| Vers | Flags|  Msg Type   |       RSVP Checksum       |
+-------------+-------------+-------------+-------------+
|  Send_TTL   | (Reserved)  |        RSVP Length        |
+-------------+-------------+-------------+-------------+

Flags: 4 bits

        0x01:         echo-request flag.

Msg Type: 8 bits

         8 = PathAck
         9 = PathTearAck
        10 = ResvAck
        11 = ResvTearAck
```

The echo-request flag only apples to Path, PathTear, Resv and ResvTear messages.

A.2 Echo-Reply Messages

A.2.1 PathAck Messages

The format of a PathAck message is as follows:

```
<PathAck Message> ::= <Common Header> [ <INTEGRITY> ]

                      <SESSION> <RSVP_HOP>

                      [ <sender descriptor> ]

<sender descriptor> ::= <SENDER_TEMPLATE> [ <SENDER_TSPEC> ] [ <ADSPEC> ]
```

The RSVP_HOP object of each PathAck message contains the IP address of the interface through which the Path message was received and the LIH (Logical Interface Handle) which was carried in the Path message.

In case of serial interface, (such as PPP and Frame Relay), using PathAck message can reduce the number of Path messages. However, in NBMA environment, for a RSVP multicast session, a PATH message sending node may not have the knowledge of all its next-hop RSVP nodes.

In this case, a sending node should have the option:

1. Query local database such as ARP or MARS server to find out the exact number of the next-hops, and retransmit PATH message accordingly.
2. Turn off the echo-request option for PATH message.

A.2.2 PathTearAck Messages

The format of a PathTearAck message is as follows:

```
<PathTearAck Message> ::= <Common Header> [ <INTEGRITY> ]

                          <SESSION> <RSVP_HOP>

                          [ <sender descriptor> ]

<sender descriptor> ::= <SENDER_TEMPLATE> [ <SENDER_TSPEC> ] [ <ADSPEC> ]
```

The RSVP_HOP object of each PathTearAck message contains the IP address of the interface through which the PathTear message was received and the LIH of which was carried in the PathTear message.

A upstream router will continue to generate PathTear messages with Rf frequency until is has received echo-replies from all reserved RSVP next-hop nodes.

A.2.3 ResvAck Messages

```
<ResvAck Message> ::= <Common Header> [<INTEGRITY>]

                      <SESSION> <RSVP_HOP>

                      [ <SCOPE> ] <STYLE>

                      <flow descriptor list>

<flow descriptor list> ::=  (see RSVP specification)[1]
```

The RSVP_HOP (i.e., the PHOP) object contains the IP address of the interface through which the Resv message was received. It also carries the corresponding LIH.

A.2.4 ResvTearAck Messages

```
<ResvTearAck Message> ::= <Common Header> [<INTEGRITY>]
```

```
<SESSION> <RSVP_HOP>

[ <SCOPE> ] <STYLE>

<flow descriptor list>

<flow descriptor list> ::=  (see RSVP specification)[1]
```

The RSVP_HOP (i.e., the PHOP) object contains the IP address of the interface through which the ResvTear message was received. It also carries the corresponding LIH.