INTERNET-DRAFT
Expiration: September 1999
File: draft-ietf-rsvp-procrules-00.txt

Bob Lindell
Bob Braden
ISI
Lixia Zhang
UCLA

Resource ReSerVation Protocol (RSVP) -- Version 1 Message Processing Rules

February 26, 1999

Status of this Memo

This document is an Internet-Draft and is in full conformance with all provisions of Section 10 of RFC2026.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF), its areas, and its working groups. Note that other groups may also distribute working documents as Internet-Drafts.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet- Drafts as reference material or to cite them other than as ‘‘work in progress.’’

The list of current Internet-Drafts can be accessed at http://www.ietf.org/ietf/1id-abstracts.txt

The list of Internet-Draft Shadow Directories can be accessed at http://www.ietf.org/shadow.html.

Abstract

This memo contains an algorithmic description of the rules used by an RSVP implementation for processing messages. It is intended to clarify the version 1 RSVP protocol specification [RFC 2205]. These rules are decomposed into pieces which are contained inside of an abstract processing architecture.

This document provides an updated version of the previous processing rules [RFC 2209]. It attempts to provide a description that is less dependent on the particular design choices made in the ISI reference implementation of RSVP.

1. Introduction

This memo contains an algorithmic description of the rules used by an RSVP implementation for processing messages. It is intended to clarify the version 1 RSVP protocol specification [RFC 2205]. These rules are decomposed into pieces which are contained inside of an abstract processing architecture.

This document provides an updated version of the previous processing rules [RFC 2209]. It attempts to provide a description that is less dependent on the particular design choices made in the ISI reference implementation of RSVP.

[*Editors note: This is a rough draft of an attempt to rewrite the message processing rules using a more abstract description. It is not intended to replace RFC 2209 at this time, nor is believed to be entirely complete or correct in it's descriptions.*]

2. An Abstract Processing Architecture

In this section we describe an abstract processing architecture to provide a framework to structure the processing rules. The architecture has four major components. The first component, called message prefilters, are a collection of processing rules which are applied to incoming RSVP messages. These prefilters are organized sequentially into a dataflow or pipeline type structure with the output of one filter feeding the input of one or more other prefilters. After a message has been processed through a sequence of message prefilters, the last prefilter will either create, update, or remove state blocks in a soft state repository based on the contents of the message.

The second component of the architecture is the soft state repository. It is a database type component that maintains state blocks for the protocol. Timers can be set on the soft state which expire if the state block is not refreshed. Deletion notification is supplied to the protocol via programmable upcalls. As an optimization, programmable upcalls could be provided that inform tasks when particular state blocks are modified by other tasks in the protocol processing code.

The third component of the architecture is a set of independent tasks which operate

over the soft state repository and either generate updates to the state blocks or generate new RSVP messages. These tasks are ordered in execution relative to other tasks by providing each with a scheduling priority. Task execution is triggered by either a change in particular state blocks or based on some time based scheduling criteria.

The last component of the architecture are the message postfilters. Message postfilters, analogous to the prefilters, have a pipeline type structure and post process outgoing RSVP messages.

[*Editors note: need some pictures here. Need to explain more about the real difference between a filter and a processing element.*]

2.1. Message Prefilters

This section describes the pipeline of message prefilters that are used to processing incoming RSVP messages.

2.1.1. Message Checksum Check

This filter checks the checksum of the RSVP message and drops the message if the checksum value cannot be reproduced from the contents of the message.

- Ignore Zero Checksum

  If the checksum field is set to zero, ignore the checksum and continue with the processing of the message.

- Recompute the Checksum

  Recompute the checksum of the message as described in Section 3.1.1 of RFC 2205. If the computed checksum is not 0xffff, drop the message.

2.1.2. Message Integrity

If a message is carrying an INTEGRITY object, an integrity check is performed. Messages which do not pass the integrity check are dropped. These processing rules are also defined in Section 4.2 of [Baker98].

- Find Key

  Search for a key that matches the same key identifier in the INTEGRITY object and has the same sender address as either the HOP object or the IP sender address.

- Handshake

  Decide if a handshake needs to be performed with the sender. If needed, perform the handshake but drop this processing of this message.

- Recompute the Message Hash

  The integrity hash is computed over the message with the hash field in the INTEGRITY object and the RSVP message header checksum field set to zero. If the computed hash does not match the hash value sent in the message, reject the message.

- Prevent Replayed Messages

  If the message is a handshake response, skip this step. Determine if the sequence number for this message will be accepted. The sequence number is used to prevent replay attacks. If the sequence number is not acceptable, reject the message.

- Process a Handshake Response

  If this message is a handshake response, save the sequence number returned by the sender.

2.1.3. Message Loop Prevention (SCOPE)

Prevent message loops by enforcing the scoping present in some RSVP messages.

- Detect SCOPE Object

  If a message arrives with a SCOPE object, check to see if this hop is contained within the scope. If it is not, drop the message.

2.1.4. Message Demultiplexer

This filter determines the message type by looking at the RSVP message header. The message is then demultiplexed, based on the type of the message, to the appropriate message specific filter entity.

2.1.5. Path Message Arrives Or API Sender Request

A path state block (PSB), in the form of soft state, is maintained for the most recent PATH message or API sender request for a given SESSION, SENDER_TEMPLATE, and incoming interface. Figure 1 shows the contents of a PSB, which follows the description in Section 3.1.3 of RFC 2205.

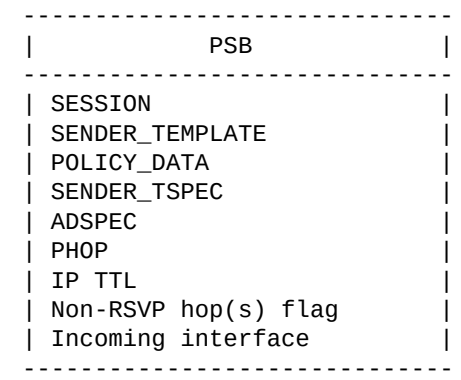

Figure 1: Definition of the Path State Block (PSB) Data Structure

- Find or Create a PSB

  Search for a PSB whose SESSION, SENDER_TEMPLATE, and incoming interface triple matches the PATH message (or API request). If no match can be found, a new PSB should be created. Before creating the PSB it is important to detect errors in port usage with existing PSBs as outlined in Section 3.2 of RFC 2205. Any port usage errors will cause a PATH ERROR message to be sent back to the PHOP (or an application upcall). Additionally, this error will cause the PATH message (or API request) to be dropped and no PSB will be created.

  A PSB will need to maintain the distinction between the receipt of a PATH message or an API request. This can be accomplished by using the PHOP,

the incoming interface field, or by adding an addition field to the PSB.

- Update the Soft State Lifetime Timer

  Update the lifetime of the PSB using the rules described in Section 3.7 of RFC 2205.

- Detect non-RSVP hops

  It is possible to detect the presence of non-RSVP hops by comparing the IP TTL and the message common header TTL fields. This information will be needed when a PATH refresh message is generated.

- Detect redundant PSBs

  It is possible that there are redundant PSBs in existence for the same flow. This can occur if routing causes a PATH message to be received on more than one interface. If this condition is detected, only one of these PSBs should be used to generate PATH refreshes and to compute an updated traffic control state. Attempt to choose the correct PSB by using information from routing. For multicast sessions, this condition is described in Section 3.9 of RFC 2205.

- Detect a route change

  In the absence of route change notification support, it is important to check whether changes in routing will effect the outgoing interface(s) for a given PSB. If it is determined that a change has occurred, execute the *Local Repair or Route Change Notification* event sequence below.

- Detect a modification to the PSB

  Any modification of an existing PSB, including the act of creation, requires that a immediate PATH refresh (or application upcall) is performed. The values of the POLICY_DATA, SENDER_TSPEC, PHOP, and incoming interface are compared between the PATH message (or API request) and the PSB.

- Generate a PATH message refresh or PATH_EVENT API upcall

A new or modified PSB will cause an immediate PATH refresh (or API upcall). Either execute the *PATH Message Refresh* sequence or generate a PATH EVENT upcall, Section 3.11.1 of RFC 2205, if the PATH message has arrived at the destination address.

Call: <Upcall_Proc>(session-id, PATH_EVENT, flags,
sender_tspec, sender_template [, ADSPEC]
[, POLICY_DATA])

- Trigger the Processing Elements

  If the processing elements are not automatically triggered to process the updated state blocks, manual trigger these processes to execute. They will recompute the outgoing traffic control and incoming reservation merge in that order, respectively.

  Note: If PHOP or Fwd_Flowspecs change, do a merge. A change in the PHOP requires local repair, which is described in Section 3.6 of RFC 2205.

2.1.6. Resv Message Arrives Or API Reserve Request

A reservation state block (RSB), in the form of soft state, is maintained for the most recent RESV message or API reserve request for a given SESSION and NHOP. Figure 2 shows the contents of a RSB which includes the following information derived from the receipt of a RESV message.

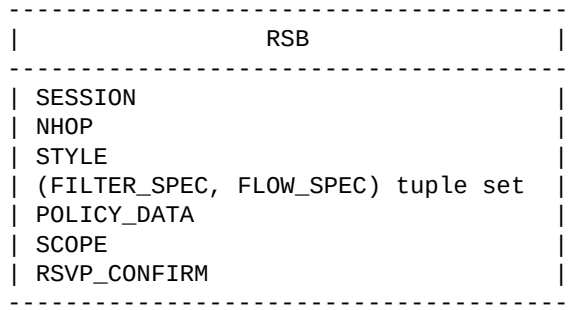

Figure 2: Definition of the Reservation State Block (RSB) Data Structure

- Find or Create a RSB

  Search for a RSB whose SESSION and NHOP matches the RESV message (or API request). If no match can be found, a new RSB should be created. Before creating the RSB it is important to detect errors which prevent the creation of an RSB. These errors are port usage with existing PSBs as outlined in Section 3.2 of RFC 2205, no corresponding PSBs for the reservation, a conflicting style with an existing RSB, or unknown style. Any of these errors will cause a RESV ERROR message to be sent back to the NHOP (or an application upcall). Additionally, this error will cause the RESV message (or API request) to be dropped and no RSB will be created.

  A RSB will need to maintain the distinction between the receipt of a RESV message or an API request. This can be accomplished by using the NHOP, the incoming interface field, or by adding an addition field to the RSB.

- Update the Soft State Lifetime Timer

  Update the lifetime of the RSB using the rules described in Section 3.7 of RFC 2205.

- Detect a modification to the RSB

  Any modification of an existing RSB, including the act of creation, requires an immediate merge recalculation. This recalculation may cause an immediate RESV refresh upstream (or application upcall).

  The values of the POLICY_DATA, RESV_CONFIRM, SCOPE, STYLE, and (FLOW_SPECS, FILTER_SPECS) tuples are compared between the RESV message (or API request) and the RSB. Currently, a change is STYLE can only occur is there is only a single RSB for this session.

- Trigger the Processing Elements

  If the processing elements are not automatically triggered to process the updated state blocks, manual trigger these processes to execute. They will recompute the outgoing traffic control and incoming reservation merge in that order, respectively.

2.1.7. PATH Tear Message Arrives or API Sender Tear

A PATH tear message is simply forwarded downstream towards receivers removing path state as it is processed at each hop.

- Find the PSB

  Search for a PSB whose SESSION, SENDER_TEMPLATE, PHOP, and incoming interface matches the PATH tear message. If no match can be found, drop the message and stop executing this sequence of steps.

- Forward the PATH Tear Message

  If the message is not at the destination address, forward the PATH tear message towards the destination if this was not a redundant PSB.

- Remove the PSB

  Garbage collect any state associated with and including this PSB.

- Trigger the Processing Elements

  If the processing elements are not automatically triggered to process the updated state blocks, manual trigger these processes to execute. They will recompute the outgoing traffic control and incoming reservation merge in that order, respectively.

  Note: If the Fwd_Flowspecs change, do a merge.

2.1.8. RESV Tear Message Arrives or API Reserve Tear

A RESV tear message is simply forwarded upstream towards senders removing reservation state as it is processed at each hop. In more general terms, it can modify the reservation or completely tear it down. When a reservation tear arrives at a merge point, the tear may not be forwarded. Af the merge point, the tear may be completed, or require that a new RESV message may be generated upstream.

- Find the RSB

Search for a RSB whose SESSION and NHOP matches the RESV tear message. If no match can be found, drop the message and stop executing this sequence of steps.

- Modify the RSB

  Update the RSB to remove the requested (FLOW_SPECS, FILTER_SPECS) tuples from the RSB. A complete tear results in an RSB with the null set. Recomputing the merge will garbage collect the RSB if necessary and any associated state such as a TCSB.

- Trigger the Processing Elements

  If the processing elements are not automatically triggered to process the updated state blocks, manual trigger these processes to execute. They will recompute the outgoing traffic control and incoming reservation merge in that order, respectively.

  Note: If the Fwd_Flowspecs change, do a merge.

2.1.9. PATH Error Message Arrives

A PATH error message is simply forwarded upstream towards a sender until it finally generates the appropriate application upcall.

- Find the PSB

  Search for a PSB whose SESSION, SENDER_TEMPLATE, and incoming interface triple matches the PATH error message. If no match can be found, drop the message and stop executing this sequence of steps.

- Generate a PATH_ERROR API upcall

  Generate a PATH ERROR upcall, Section 3.11.1 of RFC 2205, if the PATH error message has arrived at the destination address.

Call: <Upcall_Proc>(session-id, PATH_ERROR,
Error_code, Error_value, Node_Addr,
Sender_Template [, Policy_Data])

- Forward the PATH Error Message

  If the message is not at the destination address, forward the PATH error message to the PHOP in the PSB.

2.1.10. RESV Error Message Arrives

A blockade state block (BSB), in the form of soft state, is maintained for the most recent RESV error message received for a given SESSION, PHOP, and optionally (FILTER_SPEC, FLOW_SPEC) tuple set depending on style. Figure 3 shows the contents of a BSB which includes the following information derived from the receipt of a RESV error message.

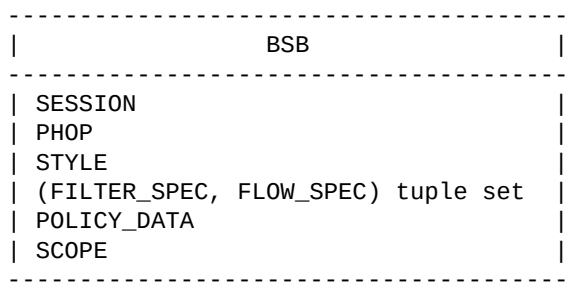

Figure 3: Definition of the Blockade State Block (BSB) Data Structure

- Find or Create a BSB

  If the error code indicates an admissional control failure, search for a BSB whose SESSION, PHOP matches the RESV error message. If no match can be found, a new BSB should be created.

- Update the Soft State Lifetime Timer

  Update the lifetime of the BSB using the rules described in Section 3.5 of RFC 2205.

- Compute the intersection with RSBs

  Compute the intersection of the FILTER_SPEC set with all RSBs which have the same SESSION. The remaining steps in this sequence will be executed separately for each RSB.

- Ignore Smaller Reservations

  If the error code indicates an admissional control failure, skip any RSB that is strictly less than the FLOW_SPEC.

- Generate RESV_Error API Upcalls

  Generate a RESV ERROR upcall, Section 3.11.1 of RFC 2205, if the RSB originated from an API request. If the FLOWSPEC in the RESV error message is strictly greater than the RSB FLOWSPEC, then turn on the Not-Guilty flag in the ERROR_SPEC.

  Call: <Upcall_Proc>(session-id, RESV_ERROR,
  Error_code, Error_value, Node_Addr,
  Error_flags, Flowspec, Filter_Spec_List
  [, Policy_data])

- Forward the RESV Error Message

  If RSB did not originate from the API, forward the RESV error message to the NHOP in the RSB.

2.1.11. Confirmation Message Arrives

Reservation confirmations are unicasted messages which must be delivered hop by hop to the destination to allow the inclusion of the optional message integrity support.

- Generate a RESV CONFIRM upcall, Section 3.11.1 of RFC 2205, if the confirmation message has arrived at the destination address.

Call: <Upcall_Proc>(session-id, RESV_CONFIRM,
Error_code, Error_value, Node_Addr,
LUB-Used, nlist, Flowspec,
Filter_Spec_List, NULL, NULL)

- Otherwise, forward the confirmation message to the IP address contained in RESV_CONFIRM object.

2.2. Processing Elements

Processing elements are a set of independent tasks which operate over the soft state repository and either generate updates to the state blocks or generate new RSVP messages. These tasks are ordered in execution relative to other tasks by providing each with a scheduling priority. Task execution is triggered by either a change in particular state blocks or based on some time based scheduling criteria.

2.2.1. Outgoing Interface Merge

A traffic control merge is performed for each distinct reservation request on an outgoing interface with respect to a given session. Each successful merge will create or modify a traffic control state block (TCSB). Figure 4 shows the contents of a TCSB which includes the following information derived from a combination of PSB and RSB information.

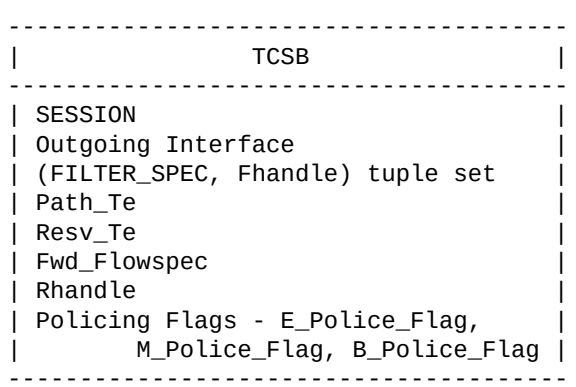

Figure 4: Definition of the Traffic Control State Block (TCSB) Data Structure

- Find all RSBs.

  For a given outgoing interface and SESSION, find all matching RSBs. Since we do not currently merge different styles, all RSBs will carry the same style.

- Compute the union of all FILTER_SPECs

  If the style is shared, compute the union of all senders using the FILTER_SPEC objects in the RSBs. A wildcard style trivially defines the union as all senders.

  If the style is distinct, we will divide the complete set of RSBs (for this SESSION and outgoing interface) into disjoint sets and each set will be merged into a separate TCSB. For each distinct FILTER_SPEC set in all the RSBs, construct the disjoint sets of RSBs which share the same FILTER_SPEC set. The remaining steps in this sequence will be executed separately for each set of RSBs.

- Compute the intersection with PSBs

  Compute the intersection of the FILTER_SPEC set with all PSBs which have the same SESSION and outgoing interface. Information from routing can provide the outgoing interfaces for a PSB using both SESSION and SENDER_TEMPLATE information. If the intersection is null, generate a RESV ERROR message for all RSBs (which are not marked torn) indicating either no sender or path information. Torn down RSBs will have a null (FLOW_SPECS, FILTER_SPECS) tuple set. Remove these RSBs (including those torn) and continue executing the steps below with no effective FLOW_SPEC so that the corresponding TCSBs will be either removed or marked as torn.

- Compute Path_Te and Resv_Te

  Compute the effective SENDER_TSPEC and FLOW_SPEC for the outgoing interface as defined in Section 2.2 of RFC 2205. This is defined as the LUB of all matching PSBs and RSBs, respectively.

- Set Policing Flags

  Set the Policing flags as defined in Section 3.8 of RFC 2205. Set the

TC_B_Police_flag on if Resv_Te is smaller than, or incomparable to, any FLOWSPEC in the RSBs. Set the TC_E_Police_flag on if any of these PSBs have their E_Police flag on in the SESSION object. Set TC_M_Police_flag on if it is a shared style and there is more than one PSB in the set.

- Find, Create, or Remove a TCSB

  Search for a TCSB whose SESSION, STYLE, and outgoing interface match the merge. If no match can be found, a new TCSB should be created and a subsequent call traffic control as follows:

  Call: TC_AddFlowspec(Interface, Resv_Te,
  Path_Te, police_flags) -> Rhandle, Fwd_Flowspec

  recording the Rhandle and Fwd_Flowspec in the TCSB. If this call fails, generate a RESV ERROR message for all RSBs indicating "Admission control failed" with the InPlace flag off. Remove the TCSB, remove any RESV_CONFIRM object(s) from the RSBs, and end this sequence of steps.

  If a match is found, but there is no effective FLOW_SPEC from the previous step, remove the flow using a call to traffic control, as described in Section 3.11.2 of RFC 2205. If there was no matching PSBs for this TCSB in the step above (Compute the intersection with PSBs), remove the TCSB. Otherwise mark the TCSB as torn by setting the Rhandle to null. It will be removed at a later point in the merge. Continue this sequence of steps so that an application upcall will be generated if necessary.

  Call: TC_DelFlowspec(Interface, Rhandle)

  If the TCSB needs to be modified, update traffic control using the following call:

  Call: TC_ModFlowspec(Interface, Rhandle, Resv_Te,
  Path_Te, police_flags) -> Fwd_Flowspec

  and record Rhandle and Fwd_Flowspec in the TCSB. If this call fails, generate a RESV Error message for all RSBs with a FLOW_SPEC larger than the value in the TCSB indicating "Admission control failed" with the

InPlace flag on. Leave the TCSB unmodified, remove any RESV_CONFIRM object(s) from the RSBs, and end this sequence of steps.

- Update Filters

  If the TCSB is new, add the FILTER_SPECs using TC_AddFilter. If the FILTER_SPECs have changed on an existing TCSB, make an appropriate set of TC_DelFilter and TC_AddFilter calls to transform the old set into the new set.

  Call: TC_AddFilter(Interface, RHandle,
  Session , FilterSpec) -> FHandle

  Call: TC_DelFilter(Interface, FHandle)

- Send Confirmation messages

  If the TCSB was not created in the previous steps, this node is a merge point for reservations and all received confirmations should be acknowledged at this time. For all RSBs which have a RESV_CONFIRM object, send a confirmation message and subsequently remove the RESV_CONFIRM from the RSB.

- Generate RESV_EVENT API Upcalls

  A new, modified, or removed TCSB will cause a RESV_EVENT API upcall to applications for all PSBs that were part of the TCSB merge and originated from an API request. Generate a RESV EVENT upcall, Section 3.11.1 of RFC 2205, of the form:

  Call: <Upcall_Proc>(session-id, RESV_EVENT,
  style, Flowspec, Filter_spec_list [,
  POLICY_DATA])

2.2.2. Incoming Interface Reservation Merge

A reservation merge is performed for each distinct PHOP on an incoming interface with respect to a given session. Changes in the result of the merge should be immediately propagated upstream with a RESV message.

- Iterate over each distinct SESSION and PHOP

  Execute the following steps for each distinct SESSION and PHOP pair found in the PSBs.

- Construct the union of all SENDER_TEMPLATES

  Construct a SENDER_TEMPLATE set as the union of all senders in the PSBs.

- Compute the intersection with TCSBs

  Compute the intersection of the SENDER_TEMPLATE set with all TCSBs which have the same SESSION and outgoing interface. Information from routing can provide the outgoing interfaces for a PSB using both SESSION and SENDER_TEMPLATE information. If the intersection is null, no RESV message is sent.

- Generate a RESV tear message

  If the intersection contains only TCSBs that are marked torn, generate and send a RESV tear message to the PHOP containing the FILTER_SPECSs listed in the TCSBs. Remove the torn TCSBs and continue at the iteration step above.

- Removed blockaded TCSBs

  Compute the intersection of the remaining TCSBs with the blockade state blocks.

- Compute the merged FLOW_SPECs

  If the intersection with the BSBs includes all of the TCSBs, the merged FLOW_SPECs will be computed as the the GLB of the TCSBs. This is

described in Section 3.5 of RFC 2205. If the GLB FLOW_SPEC is also blockaded by the BSBs, do not send any RESV message. Otherwise, the FLOW_SPECs are computed using the LUB of the non-blockaded TCSBs as follows.

If the style is shared, compute the merged FLOW_SPEC as the LUB of either the Fwd_Flowspec or Resv_Te of all the TCSBs which are not blockaded. See Section 2.2 of RFC 2205. If the style is also explicit, construct a FILTER_SPEC set as the union of all FILTER_SPEC sets in the TCSBs.

If the style is distinct, we will compute a separate merged FLOW_SPEC for each distinct FILTER_SPEC across all the TCSBs. Construct a (FILTER_SPEC, FLOW_SPEC) tuple set containing an entry for each distinct FILTER_SPEC and the corresponding FLOW_SPEC computed as the LUB of either the Fwd_Flowspec or Resv_Te.

- Send the RESV Message

  Look at the RSBs for a RESV_CONFIRM object and add this to the message if it exists.

2.2.3. PATH Message Refresh

This is easy. It just regurgitates a PSB.

2.2.4. RESV Message Refresh

This is just a incoming interface merge.

2.2.5. Local Repair or Route Change Notification

This is just a delayed PATH refresh.

2.3. Message Postfilters

This section describes the pipeline of message postfilters that are used to processing outgoing RSVP messages.

2.3.1. Message Multiplexer

In general, this multiplexer groups message types together to send through the appropriate postfilters. Current, this multiplexer separates those messages which may need loop prevention support from all others. Messages needing loop prevention support are sent through that filter, all other are sent to the last stage of the pipeline for integrity and checksum generation.

2.3.2. Message Loop Prevention (SCOPE)

2.3.3. Message Integrity

2.3.4. Message Checksum

This filter computes the checksum of the RSVP message.

- Compute the Checksum

  Compute the checksum of the message by setting the checksum field in the message header to zero and computing the checksum as described in Section 3.1.1 of RFC 2205.

- Update RSVP Message Header Set the computed checksum value into the RSVP message header.

3. Commentary on the RSVP Version 1 Specification

In this section, we provide clarification or suggested revisions to the specification in RFC 2205.

3.1. Reservation Confirmations

Flowspecs are not always orderable. Change the spec to forward the first confirmation. All others that arrive at the merge are confirmed after passing local admission control.

3.2. Changing ADSPECs

Changes in the ADSPEC would normally cause the immediate refresh of a PATH message. Assuming that ADSPECs change frequently along heavily used paths, each router will be frequently refreshing PATH messages. This has an additive effect for downstream routers causing an excessive number of PATH messages to be generated. Should probably have a hold down timer.

3.3. Fwd_Flowspecs

The merging rules currently use the Fwd_Flowspec is provided by local traffic control, rather than the receivers flowspec. This can cause RESV messages to be generated when a PATH message arrives which causes an update to traffic control. Is this desirable? In general, PATH type messages should not trigger RESV type messages and vis versa.

3.4. Blockade State and KR Problems

Mention Mohit's new KR Internet Draft.

4. References

[Baker98] Baker, F., Lindell, R., Talwar, M., *RSVP Cryptographic Authentication*, Work in Progress.

[RFC 2205] Braden, R., Ed., Zhang, L., Berson, S., Herzog, S., and S. Jamin, *Resource ReSerVation Protocol (RSVP) -- Version 1 Functional Specification*, RFC 2205, September 1997.

[RFC 2207] Berger, L. and T. O'Malley, *RSVP Extensions for IPSEC IPv4 Data Flows*, RFC 2207, September 1997.

[RSVP93] Zhang, L., Deering, S., Estrin, D., Shenker, S., and D. Zappala, *RSVP: A New Resource ReSerVation Protocol*, IEEE Network, September 1993.

5. Security Considerations

6. Author's Addresses

Bob Lindell
USC Information Sciences Institute
4676 Admiralty Way
Marina del Rey, CA 90292

Phone: (310) 822-1511
EMail: lindell@ISI.EDU

Bob Braden
USC Information Sciences Institute
4676 Admiralty Way
Marina del Rey, CA 90292

Phone: (310) 822-1511
EMail: braden@ISI.EDU

Lixia Zhang
UCLA Computer Science Department
4531G Boelter Hall
Los Angeles, CA 90095-1596 USA

Phone: 310-825-2695
EMail: lixia@cs.ucla.edu

Table of Contents

1. Introduction 2
2. An Abstract Processing Architecture 2
2.1. Message Prefilters 3
2.1.1. Message Checksum Check 3
2.1.2. Message Integrity 3
2.1.3. Message Loop Prevention (SCOPE) 4
2.1.4. Message Demultiplexer 5
2.1.5. Path Message Arrives Or API Sender Request 5
2.1.6. Resv Message Arrives Or API Reserve Request 7
2.1.7. PATH Tear Message Arrives or API Sender Tear 9
2.1.8. RESV Tear Message Arrives or API Reserve Tear 9
2.1.9. PATH Error Message Arrives 10
2.1.10. RESV Error Message Arrives 11
2.1.11. Confirmation Message Arrives 12
2.2. Processing Elements 13
2.2.1. Outgoing Interface Merge 13
2.2.2. Incoming Interface Reservation Merge 17
2.2.3. PATH Message Refresh 18
2.2.4. RESV Message Refresh 18
2.2.5. Local Repair or Route Change Notification 18
2.3. Message Postfilters 19
2.3.1. Message Multiplexer 19
2.3.2. Message Loop Prevention (SCOPE) 19
2.3.3. Message Integrity 19
2.3.4. Message Checksum 19
3. Commentary on the RSVP Version 1 Specification 19
3.1. Reservation Confirmations 20
3.2. Changing ADSPECs 20
3.3. Fwd_Flowspecs 20
3.4. Blockade State and KR Problems 20
4. References 20
5. Security Considerations 21
6. Author’s Addresses 21