Internet Draft
Expires May 22, 1997
File: draft-ietf-rsvp-policy-ext-01.ps

Shai Herzog
IBM T.J. Watson Research Center
November 1996

RSVP Extensions for Policy Control

November 22, 1996

Status of Memo

This document is an Internet-Draft. 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."

To learn the current status of any Internet-Draft, please check the "1id-abstracts.txt" listing contained in the Internet-Drafts Shadow Directories on ds.internic.net (US East Coast), nic.nordu.net (Europe), ftp.isi.edu (US West Coast), or munnari.oz.au (Pacific Rim).

Abstract

This memo describes a set of extensions for supporting generic policy based admission control in RSVP.[1] This document does not advocate particular policy control mechanisms; however, a recommendation for a mechanism built on top of these extensions can be found in [LPM].

[1]This memo could be conceived as an extension to the RSVP functional specifications [RSVPSP].

Contents

1 Introduction

RSVP, by its definition, discriminates between users, by providing some users with better service at the expense of others. Therefore, it is reasonable to expect that RSVP be accompanied by mechanisms for controlling and enforcing access and usage policies. Historically, when RSVP Ver. 1 was developed, the knowledge and understanding of policy issues was in its infancy. As a result, Ver. 1 of the RSVP Functional Specifications[RSVPSP] left a place holder for policy support in the form of POLICY_DATA objects. However, it deliberately refrained from specifying mechanisms, message formats, or providing insight into how policy enforcement should be carried out. This document is intended to fill in this void.

The current RSVP Functional Specification describes the interface to admission (traffic) control that is based *only* on resource availability (capacity). In this document we describe a set of extensions to RSVP for supporting policy based admission control as well, in one atomic operation. The scope of this document is limited to these extensions; a discussion of accounting and access control policies for resource reservation protocols can be found in [Arch] and a recommendation for a mechanism built on top of these extensions can be found in [LPM].

2 Policy Data Object Format

The following replaces section A.13 in [RSVPSP]:

2.1 Base Format

POLICY_DATA class=14

- Type 1 POLICY_DATA object: Class=14, C-Type=1

```
+-------------+-------------+-------------+-------------+
|  Length                   | POLICY_DATA |      1      |
+-------------+-------------+-------------+-------------+
|  Data Offset              | OID                       |
+-------------+-------------+-------------+-------------+
|                                                       |
// Option List                                         //
|                                                       |
+-------------------------------------------------------+
|                                                       |
// Policy Element List                                 //
|                                                       |
+-------------------------------------------------------+
```

Data Offset: 16 bits

The offset in bytes of the data portion (from the first byte of the object header).

OID: 16 bits

This field contains an Object ID or 0 if unused. OIDs must be unique for each object of a local session (within a single RSVP node). The OID value is assigned by the PC module, and is mainly used for fragmenting POLICY_DATA objects. (All fragments of a single POLICY_DATA object must have the same OID, see Section 4.)

Option List

The list of options and their usage is defined in Section 2.2.

Policy Element List

Policy Elements have the following format:

```
+-------------+-------------+-------------+-------------+
|  Length                   |  P-type                   |
+---------------------------+---------------------------+
|                                                       |
// Policy information  (Opaque to RSVP)               //
|                                                       |
+-------------------------------------------------------+
```

The contents of policy elements is opaque to RSVP and its internal format is only known to the Policy Control (PC) module (see [LPM]).

2.2 Policy Data Options

The following objects could appear as options in POLICY_DATA objects:

2.2.1 RSVP Objects

- FILTER_SPEC object (list)

  This list represent the set of senders associated with the POLICY_DATA object. If none is provided, the policy information is assumed to be associated with all the flows of the session.

- RSVP_HOP Object

  The RSVP_HOP object uses the same format as RSVP's neighboring node identifier, however, in policy objects it has a slightly different meaning. Here, it identifies the neighbor/peer policy-capable node that constructed the policy object. When policy is enforced at border nodes, the peer policy-capable node may be several RSVP hops away.

- INTEGRITY Object

  The INTEGRITY object [Bak96], provides guarantees that the object was not compromised.[2]

[2] In this document, we do not define the algorithm for computing the INTEGRITY value. However, in order to guarantee that the policy is associated with the correct flow/reservation, it may be necessary to perform the computation over other RSVP objects like SESSION, FILTER_SPEC list, etc.

2.2.2 Policy Options

- Fragmentation Option

```
+-------------+------------+------------+-------------+
| Length                   |     0      |      1      |
+--------------------------+------------+-------------+
| Variable Length                                     |
+-----------------------------------------------------+
```

This is required and present only in POLICY_DATA fragment objects (allowing RSVP to distinguish them from unfragmented or token objects). The only current format is:

- Length = 4, no variable length.

  This option is present when semantic fragmentation is used.

- NoChange Option

```
+-------------+------------+------------+-------------+
| Length                   |     0      |      2      |
+--------------------------+------------+-------------+
| 0                        | Previous-OID             |
+--------------------------+--------------------------+
```

This option provide a hint to the receiving node that the policy information is identical to the *Previous-OID* object. While this option may save on input and output processing, it does not reduce the size of the transmitted state; the complete information must be transmitted in full anyhow since with RSVP's soft-state there is no guarantee that the information associated with the Previous-OID is available at the receiving node.

- FilterSpec Option

```
+-------------+------------+------------+-------------+
| Length                   |     0      |      3      |
+--------------------------+------------+-------------+
| Counter                  | hash type  |      0      |
+--------------------------+------------+-------------+
| FILTER_SPEC List 32 bit hash/CRC                    |
+-----------------------------------------------------+
```

This option allows separating FILTER_SPECs from their corresponding policy data. When present, the policy information should not be associated with the session, but instead, with a list of FILTER_SPECs which was previously sent in a separate POLICY_DATA fragment. This option conserves resources across a non-policy cloud: it allows multiple POLICY_DATA objects from multiple rsvp hops to share the same FILTER_SPEC list. The FILTER_SPEC list itself is sent by a special type of POLICY_DATA fragment. This fragment carries the Fragmentation option, however, its OID is always 0; this way, a POLICY_DATA object created at one node can be matched with a FILTER_SPEC list created by another. Instead of using the OID field, this match is based on the counter field, which provides information about the number of FILTER_SPECs in the list. When two FILTER_SPEC lists have the same number of elements, the included CRC becomes the only method for matching policy data and filter lists. The CRC algorithm is a matter for agreement between adjacent policy nodes, (like key management); however, a default hash/CRC (type 0) algorithm can be defined. When this option has a length of 8, no CRC is provided, and the list identification is limited to the counter value.

2.2.3 Options Constraints

- The RSVP_HOP and INTEGRITY options are mutually exclusive since the INTEGRITY object already contains the sending-system address. If neither is present, the policy data is implicitly assumed to have been constructed by the RSVP_HOP indicated in the RSVP message itself (i.e., the neighboring RSVP node is policy-capable).
- If present, the Fragmentation option should appear first.
- If present, the NoChange option should appear first (after the Fragmentation option).
- If present FILTER_SPEC objects must appear as one consecutive list (i.e., no more than one list in each POLICY_DATA object and its fragments).

3 RSVP/Policy Control Interface

Policy control in RSVP is performed through a set of functions that regulate the use of POLICY_DATA objects and advise RSVP about the policy status of reservations. In this section, we describe these services as a set of functions, which conceptually belong in Section 3.10.3 titled "RSVP/Policy Control Interface" of the RSVP functional specification[RSVPSP].

3.1 Policy Control Services

Before we discuss the functions themselves, let us describe some of their common parameters: The session and filter_spec_list describe the set of flows to which an outgoing policy applies. Parameters lih, rsvp_hop and message_type provide network/topology information, and resv_handle and resv_flowspec provide information about the current/desired level of reservation and traffic characteristics.

- Process a received POLICY_DATA object

  ```
  Call:  PC_InPolicy   (session, lih, rsvp_hop, message_type,
                        in_policy_objects, resv_handle,
                        resv_flowspec, timeout)
                        -> RCode
  ```

  Incoming policy objects are checked for syntax, and a policy admission decision takes place (i.e., `PC_AuthCheck()` is called internally). If successful, the reservation can be admitted. Otherwise, the reservation should be rejected in a manner similar to admission control failure. A reservation may be marked as preemptable, which means that admission control may cancel it at any time to make room for another more important reservation. (See the *TC_Preempt()* upcall and the discussion of service preemption in [RSVPSP].) The timeout parameter communicates the lifespan (in seconds) of the state contained in the input policy object. This value should be identical to the one used by RSVP to purge the state created by the RSVP message that carried the policy object.

  An authorization to establish reservations must always be performed on outgoing interfaces (`lih`). However, for messages arriving on incoming interfaces (e.g., Path) only the incoming `lih` is known, and therefore only the authorization for accepting a Path message could be checked. Since the policy status of a reservation may change upon receiving an incoming Path message, RSVP should either perform individual `PC_AuthCheck()` calls for each of its outgoing interfaces, or wait until the reservation refresh timer goes off.

- Request an outgoing POLICY_DATA object

  ```
  Call:  PC_OutPolicy (session, filter_spec_list,
                        lih, rsvp_hop, message_type,
                        out_policy_objects,
                        max_pd, avail_pd)
                        -> RCode
  ```

  Before RSVP finalizes an outgoing control message it must query the PC module for policy data objects. RSVP specifies the desired maximal object size (*max_pd*), and the available space within the current RSVP control message (*avail_pd*).[3] The call returns

[3] *avail_pd* must be at least the size of a POLICY_DATA object without a data portion (i.e., 64 bits), since this is a minimal size of any valid policy object.

a linked list of outgoing POLICY_DATA objects which must be sent by RSVP either embedded in the current RSVP message, or in separate ones (see Section 4).

In the case of Path messages, the `rsvp_hop` parameter should be NULL. The outgoing policy object must include policy information for all the next hops over interface `lih`.

- Check the status of an existing reservation

```
Call:  PC_AuthCheck (session, filter_spec_list,
                     lih, message_type, resv_handle,
                     resv_flowspec, ind)
                     -> RCode
```

Authorization checks can be performed on both Path and Resv directions. When the message_type is an upstream type (Resv, Resv Tear, Path Err) the `lih` is assumed to be an outgoing interface and reservation status is checked. However, when the message_type is an downstream type (Path, Path Tear, Resv Err), the `lih` is assumed to be an incoming interface and Path-sending authorization is checked.

Authorization checks are usually event triggered by the arrival of a new message; these are handled transparently by the input processing call `PC_InPolicy()`. However, RSVP itself must verify the status of reservations periodically before refreshing them by calling `PC_AuthCheck()` with RSVP_RESV message type, for each outgoing reserved interface. If the reservation status changes, RSVP must act accordingly (e.g., cancel the reservation, etc.).

- Initialize Policy Control services

```
Call:  PC_Init (void) -> RCode
```

- Synchronize RSVP and policy control state

```
Call:  PC_Branch     (session, filter_spec_list,
                     rsvp_hop, op_type)
                     -> RCode
```

This call affects all the state associated with a particular multicast (or unicast) branch. It is used when routing indicates that this path is no longer in use, or when blockade state changes.

op_type:

1: Block branch (blockade state: ignore branch state)
2: Unblock branch
3: Delete branch

- Delete policy control state

```
Call:  PC_Close (session, filter_spec_list) -> RCode
```

This call purges all the policy state that is associated with the filter_spec_list. It provide the PC module with the opportunity to shut-down on-going operations (e.g., accounting) in an orderly manner before the state is purged.

3.2 PC Success Codes

The return code (RCode) provides policy feedback to RSVP, it is made of three separate return variables:[4]

- Function return value:

  0: Success
  1: Warning
  2: Syntax Error (bad object)
  3: Policy failure

- *PC_errno*:

  An external variable (similar to the *errno* in Unix) which provides specific error (reason) code.

- *PC_flags*:

  An external variable with flags that advise RSVP about required operations:

0x01	`PC_RC_ModState`	New or modified policy
0x02	`PC_RC_SendErr`	Send immediate RSVP error
0x04	`PC_RC_Respond`	Send immediate RSVP response
0x08	`PC_RC_Cancel`	Cancel the reservation
0x10	`PC_RC_Preempt`	Allow preemption of flow

3.3 PC Codes: Required Action by RSVP

The PC success codes, and especially *PC_Flags* advise RSVP about appropriate required actions:

3.3.1 Refreshing Policy State

When flag `PC_RC_ModState` is set, RSVP must immediately send a refresh message. If RSVP schedules an immediate refresh anyway (i.e., because of new or modified Path/Resv state) then the modified policy can be piggybacked on this refresh message. Otherwise, RSVP should either schedule an immediate refresh or send the policy refresh in a vacuous message (see Section 4).

[4]This is only an initial list, we expect that part to change as policy control matures.

3.3.2 Policy Error Signaling

Policy errors are reported inside POLICY_DATA objects and is transparent to RSVP. The PC module is responsible for generating the right contents in outgoing policy error objects and interpreting the incoming ones.

Generic error signaling involves the following steps:

- While in `PC_AuthCheck()` or `PC_InPolicy()`, the PC module detects an error, and reports it to RSVP using a return code.
- RSVP performs its standard error handling, by initiating either a PathErr or ResvErr message.
- Before sending the error message, RSVP queries the PC module for an outgoing object `PC_OutPolicy()`.
- The PC modules provides an outgoing object with specific error information. (Setting flag `PC_RC_SendErr`).
- RSVP sends the error message with the embedded (error) policy object.

3.3.3 RSVP Response

When flag `PC_RC_Respond` is set, RSVP must generate a message in the reverse direction to the current one. (i.e., Path vs. Resv and PathErr vs. ResvErr). The reverse message may be a standard one. It may also be a vacuous message if no RSVP state need be transmitted in the reverse direction at this time. A common case for such a response is when an acknowledgment for some specific policy is required.

3.4 Default Handling of Policy Data Objects

It is generally assumed that policy enforcement (at least in its initial stages) is likely to concentrate on border nodes between autonomous systems. This would mean that policy objects transmitted at one edge of an autonomous cloud may need to traverse a non-policy-capable, RSVP cloud before reaching the other edge. The minimal requirement from a non-policy-capable RSVP node is to forward POLICY_DATA objects embedded in the appropriate outgoing messages, as-is (without modifications) according to the following rules:

- POLICY_DATA objects are to be forwarded as is, in RSVP messages with the same type as the ones with which they arrive.

- POLICY_DATA objects may be syntactically fragmented at any time to fit inside the outgoing message.[5][6][7]
- Multicast merging (splitting) nodes:

  In the upstream direction, POLICY_DATA objects are concatenated into a merged list. If the list is too large it is up to RSVP to fragment the outgoing message.[8]

4 Syntactic Fragmentation of large Policy Data objects

RSVP's extensions for policy control provide support for a wide range of policies; at least in the initial phases, we could assume that they would be limited to very basic policies, carried by small size POLICY_DATA objects. However, we it is important to ensure that our approach would be capable of handling policies that are more complex, and of large sizes.

In the current version of the RSVP spec [RSVPSP], each RSVP message must occupy exactly one IP datagram. If it exceeds the MTU, such a datagram will be fragmented by IP and reassembled at the recipient node. Future versions of the RSVP protocol may provide more flexible solutions; the most likely direction will be to perform "semantic fragmentation" (see Section 3.2 in [RSVPSP]). Unfortunately neither the current or proposed fragmentation solutions would be adequate for policy objects; When using IP fragmentation, large POLICY_DATA objects would increase the overall RSVP message size, the number of fragments, and as a result, the risk of loosing complete RSVP messages. Even if RSVP adopts a future proposal for semantic fragmentation, it is hard to see how POLICY_DATA objects, being semantically opaque, could be fragmented effectively by RSVP.

If the prevailing goal is to have as little as possible adverse effect on RSVP, fragmentation and reassembly of POLICY_DATA objects should be separated from RSVP. We introduce a third approach called "syntactic fragmentation". With this approach, RSVP would be aware of the syntax but not the semantics of policy fragmentation. (A detailed fragmentation discussion can be found in [LPM].)

The basic building blocks of syntactic fragmentation are:

[5] The available space in an outgoing message may be smaller than in the incoming message due to state merging, change of MTU or other reasons.

[6] The minimal POLICY_DATA object size is 64 bits (without the data portion). Bigger objects are considered as complete objects that may be fragmented.

[7] Syntactic fragmentation is achieved by breaking the object into two asymmetric objects: the full size POLICY_DATA object and a token object. The full size object will undergo IP fragmentation (see [LPM]).

[8] Notice here that because this is a set of semantically independent POLICY_DATA objects, RSVP can fragment the list effectively.

Vacuous RSVP Messages

Vacuous RSVP messages are a method for using RSVP signaling to carry policy information without jeopardizing important RSVP state. Vacuous messages carry fragments and only the minimal RSVP information that is required to properly route and process these messages; however, any information contained in them is merely a duplicate of information sent by other non-vacuous RSVP messages. As a result, a lost vacuous message has no adverse effect on RSVP's signaling.

POLICY_DATA fragments

A POLICY_DATA object is broken into a series of fragments $PD_1, ..., PD_n, PDE$, where all the fragments are conceptually linked by having the same OID value in their header. The OID serves a similar purpose to IPv6's Fragment Identification field in the Fragment header; its value should be selected in a way that will prevent two instances of policy objects (of the same session and RSVP_HOP) from having the same OID value. OID selection is a responsibility of the sending node, and can be achieved by various strategies. A possible approach could be to use the low 16 bit value in seconds of the Real-Time system clock.[9].

Objects $PD_1, ..., PD_n$ contain policy data fragments, and the Fragmentation option. PDE is a special token object with a minimal size (64 bit header only). The PDE's minimal size allows it to be embedded in the standard outgoing RSVP control message, while the other fragments must be sent by separate "vacuous" RSVP messages.

Sending Fragments

Prior to sending an RSVP message M of type RSVP_XXXX, RSVP calls `PC_OutPolicy()` to obtain the list of outgoing POLICY_DATA objects.

For efficient processing, fragment objects must appear first on this list. If the list contains any fragment objects, RSVP halts its normal processing to send these fragments in vacuous RSVP_XXXX messages. Once these fragments have been sent, RSVP continues its regular processing by placing the PDE objects in M, and sending the standard RSVP_XXXX message out. In some cases, multiple PDE objects ($PDE_1 ... PDE_k$) may be embedded in the outgoing message M. However, if their overall size exceeds the available space in the outgoing message, RSVP could apply its own fragmentation rules, but should never send them in vacuous messages (whose sole function is to carry fragments).

Receiving Fragments

On the receiving side, when an RSVP message arrives with a POLICY_DATA object with a fragmentation option, it should be handed over to the PC module (using `PC_InPolicy()`) regardless of the success of the RSVP message-syntax checks or policy control return codes. Token objects do not contain the fragmentation option, and therefore are indistinguishable (to RSVP) from unfragmented ones.

[9] Using the system clock provides protection against system crash/recovery problems. Such OID values would wraps around only after 2^16 seconds (over 8 hour) which is enough to guarantee that all old fragments have either been timed-out or lost.

5 API Considerations

Section 3.10.1 in [RSVPSP] defines the Application/RSVP interface. Because the API design is operating system specific, this section should be only considered as a suggestion.

Supporting policy control requires the following considerations:

- The SENDER and RESERVE calls accept policy data parameters. There are at least two different approaches for providing these parameters: the first places the burden on applications (or their proxy servers) to build complete POLICY_DATA objects and pass them as API parameters. The second require that applications merely provide general guidelines that are later converted to POLICY_DATA by the API processing code. We recommend using hybrid approach where applications provide partial POLICY_DATA objects through the API, and API processing adds additional, system related information (e.g., INTEGRITY object, FILTER_SPEC list, etc.). This hybrid approach provides applications with the flexibility to specify new policy parameterization without having to change the API, and at the same time relieves applications from the burden of specifying routine, system related details.
- State merging at the API level should be handled with care; It is essential that each API client (application) have its own separate state. An analogy to shared medium may be appropriate: to distinguish between reservations over a shared medium, RSVP maintains reservation state for each outgoing interface, as well as individual next hops. In the API case, local clients may be perceived as all belonging to a single virtual interface (local-host) however, each of them is subscribed with a separate upcall procedure address ("next-hop").

6 Acknowledgment

This document incorporates inputs from Lou Berger, Bob Braden, Deborah Estrin, Roch Guérin and Scott Shenker, and feedback from many RSVP collaborators.

References

[Bak96] F. Baker. RSVP Cryptographic Authentication *Internet-Draft*, draft-ietf-rsvp-md5-02.txt, 1996.

[RSVPSP] R. Braden, L. Zhang, S. Berson, S. Herzog, and S. Jamin, Resource ReSerVation Protocol (RSVP) Version 1 Functional Specification. *Internet-Draft*, draft-ietf-RSVPSP-14.[ps,txt], Nov. 1996.

[LPM] S. Herzog Local Policy Modules (LPM): Policy Enforcement for Resource Reservation Protocols. *Internet-Draft*, draft-ietf-rsvp-policy-lpm-01.[ps,txt], Nov. 1996.

[Arch] S. Herzog Accounting and Access Control Policies for Resource Reservation Protocols. *Internet-Draft*, draft-ietf-rsvp-policy-arch-01.[ps,txt], Nov. 1996.

Author's Address

Shai Herzog
IBM T. J. Watson Research Center,
P.O. Box 704
Yorktown Heights, NY 10598

Phone: (914) 784-6059
Email: herzog@watson.ibm.com