Internet Draft
Expires September 19, 1997
File: draft-ietf-rsvp-policy-ext-02.ps

Shai Herzog
IBM T.J. Watson Research Center
Apr. 1997

RSVP Extensions for Policy Control

March 19, 1997

Status of Memo

Abstract

This memo presents a set of extensions for supporting generic policy based admission control in RSVP.[1] These extensions include the standard format of POLICY_DATA objects, a generic RSVP/Policy-Control interface, and a description of RSVP's handling of policy events.

This document does not advocate particular policy control mechanisms; however, a Router/Server Policy Protocol description for 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. 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 description of a router-server Policy Protocol for these extensions can be found in [LPM].

2 Policy Data Object Format

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

POLICY_DATA objects are carried by RSVP messages and contain policy information. All policy-capable nodes (at any location in the network) can generate, modify, or remove policy objects in compliance with local policies.[2]

[2] Core nodes can add policy objects to RSVP messages, even when none was provided by senders or receivers. Most likely, this would be based on specific network topology properties (e.g., incoming port ID).

2.1 Base Format

POLICY_DATA class=14

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

```
+-------------+-------------+-------------+-------------+
|  Length                   | POLICY_DATA |      1      |
+---------------------------+-------------+-------------+
|  Data Offset              | Flags       | 0 (reserved)|
+---------------------------+-------------+-------------+
|                                                       |
// 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).

Flags: 8 bits

`0x01` `PCF_Updt` A modified object, don't check against previous one
`0x02` `PCF_Fragment` This is a fragment of a PD object

Reserved: 8 bits

Always 0.

Option List

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

Policy Element List

The contents of policy elements is opaque to RSVP and its internal format is only known to the Local Policy Module (LPM). (See Section 3 and [LPM]).

Policy Elements have the following format:

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

2.2 Policy Data Options

This section describes a set of options that may appear as options in POLICY_DATA objects. All policy options appear as RSVP objects; some use their valid original format while others appear as NULL objects.

2.2.1 RSVP Objects as Policy Options

The following objects retain the same format specified in [RSVPSP] however, they gain different semantics when used inside POLICY_DATA objects.

FILTER_SPEC object (list)

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(s)

The RSVP_HOP object identifies the neighbor/peer policy-capable node that constructed the policy object. When policy is enforced at border nodes, peer policy nodes may be several RSVP hops away from each other.

If an RSVP_HOP object follows either an INTEGRITY or RSVP_HOP objects it identifies the destination policy node.[3] If a destination RSVP_HOP and the address of the receiving node do not match, the entire POLICY_DATA object is ignored.

INTEGRITY Object

The INTEGRITY object provides guarantees that the object was not compromised. It follows the rules from [Bak96], and is calculated over the SESSION object, POLICY_DATA object, and the message type field[4] as if they formed one continuous in-order message, without any alignment. This concatenation is designed to prevent copy and replay attacks of POLICY_DATA objects from other sessions, flows, message types or even other network locations.

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).

2.2.2 Other Options

All options that do not use a valid RSVP object format, should use the NULL RSVP object format with different CType values. This document defines only one such option, however, several other may be considered in future versions. (e.g., Fragmentation, NoChange, etc.).

---

[3] This RSVP_HOP may be used to ensure the POLICY_DATA object is delivered to the targeted policy node. It may be used to emulate unicast delivery in multicast Path messages. It also helps prevent using a policy object in other parts of the network (replay attack).

[4] As it appears in RSVP's common header.

- **Policy Refresh Multiplier**

  Some policies may have looser timing constraints than RSVP, and therefore may allow for lower refresh frequency. If the Policy Refresh Multiplier option is present, policy is refreshed only once in *Multiplier* RSVP refreshes, for *Duplicates* times.

```
+-------------+-------------+-------------+-------------+
|             8             |     0       |      1      |
+-------------+-------------+-------------+-------------+
| Multiplier  | Duplicates  | Reserve (0)               |
+-------------+-------------+---------------------------+
```

  For example, for *Multiplier=16* and *Duplicates=3*, the policy should be refreshed on RSVP's refreshes number 1,2,3,16,17,18,...

3 RSVP/Policy Control Interface

Conceptually, this section belong to Section 3.10.3 titled "RSVP/Policy Control Interface" of the RSVP functional specification[RSVPSP].

Policy control in RSVP is modeled as a set of functions which are provided by a separate component known as Local Policy Module. The LPM controls the use of POLICY_DATA objects and provides authorization information to RSVP.

3.1 Synchronous vs. Asynchronous Policy Control

RSVP must routinely consult the LPM for policy decisions. The consultation can follow one of two models: Synchronous or Asynchronous. In the Synchronous model, when RSVP calls a particular service, it must block until the call is completed. (even if it takes substantial time). In the Asynchronous model, the call never blocks; delayed results are communicated back to RSVP through an upcall. The asynchronous model is harder to support, since RSVP must be able to halt incomplete tasks, save their context, and complete them later, when results become available, however, it has significantly better scaling properties.

Query results may be commonly delayed when policy decisions are performed by an external server (See [LPM]). Consider a case where an average query takes 10ms; a synchronous RSVP/policy implementation would be roughly limited to less than 100 unicast flows and even much less for multicast flows.

Since the two models provide the same functionality, and differ only in performance; each RSVP implementation is free to select the model best fitting its needs. RSVP may choose the synchronous model by specifying a NULL as a `cdp` parameter when calling a service.

3.2 Policy Control Services

- Common Parameters

  The following is a list of common parameters (shared by several policy control functions.

  `session`, `filter_spec_list` and `shr_ind`
  : The set of flows to which the POLICY_DATA object applies, and an indication whether they are shared.

  `rsvp_hop`
  : The peer policy node, as well as the local LIH connecting to it. The (`rsvp_hop` includes the local `lih`),

  `message_type`
  : The direction and type of message that carried the POLICY_DATA object.

  `resv_handle` and `resv_flowspec`
  : Information regarding the current/desired level of reservation and traffic characteristics.

  `cbp` and `giveup_time`
  : A pointer (address) of the Control Block. RSVP provides this address when making service calls. This value is echoed back to RSVP with the completion notification `upcall`. Giveup_time is the maximal period RSVP is willing to wait; If results are still unavailable after this period, RSVP should receive an upcall with failure results (and timer-expired error).

- Call:

```
Call:  PC_InPolicy   (message_type, rsvp_hop, session,
                      shr_ind, filter_spec_list,
                      in_policy_objects,
                      resv_handle,
                      resv_flowspec,
                      refresh_period,
                      cbp, giveup_time)
                      -> RCode
```

  RSVP calls `PC_InPolicy` for all incoming messages; However, it is acceptable for implementations to turn off policy processing for messages other than Path and Resv, when they don't carry any POLICY_DATA objects.[5]

  The LPM verifies any incoming policy objects (if included) and provides an authorization decision.[6] If the incoming message is authorized, RSVP continues its normal

[5]It is highly desirable to authorize Tear and Error messages even when they don't carry policy objects. However, since the risk from relaxed authorization is limited to denial of service for a single flow, the decision is left at the discretion of local administrators.

[6]To prevent code duplication, `PC_AuthCheck()` may be called internally.

processing. If it is rejected, RSVP drops the message entirely (as if it was never received), and sends the appropriate error message (with a policy failure error code). With RSVP's soft-state management, the consequences of dropping the incoming message is that the existing state (Path or Resv) begins to age and would eventually time-out.[7]

Reservations may also be authorized with a warning which marks them as preemptable. A preemptable reservation may be canceled at any time by admission control to make room for another more important reservation. (See *TC_Preempt()* and the discussion of service preemption in [RSVPSP].)

Parameter `refresh-period` has the same value and semantics as in RSVP.

- ```
  Call:  PC_OutPolicy  (message_type, rsvp_hop_list, session,
                        shr_ind, filter_spec_list,
                        max_pd, avail_pd,
                        cbp, giveup_time,
                        out_policy_objects)
                        -> RCode
  ```

  Before RSVP finalizes an any outgoing RSVP message it calls `PC_OutPolicy()` to prepare outgoing objects for the a specified flow. RSVP specifies the desired maximal object size (*max_pd*), and the available space within the current RSVP control message (*avail_pd*).[8] The filter_spec_list includes the set of filters corresponding to the reserved sources.

  When the filter_spec_list includes multiple filters (either because of a shared reservation or an aggregated policy over multiple FF) individual outgoing hops may be associated with different sets of filter_specs. RSVP must build the filter_spec_list as a union of all filter_spec lists over all outgoing hops. (All reserved sources) The LPM computes individual per-hop filter_spec lists as the intersection of this list with the set of sources upstream to a specific previous hop. (Previous-hop information is obtained from incoming Path messages.) A NULL filter_spec_list represents *all* sources (i.e., WF).

  The call returns a list of outgoing POLICY_DATA objects for each rsvp_hop.

- ```
  Call:  PC_AuthCheck  (message_type, session,
                        shr_ind, filter_spec_list,
                        resv_desc list,
                        full_list_ind,
                        cbp, giveup_time)
                        -> RCode
  ```

[7] An incoming messages may fail authorization simply because it lacks a particular policy object which was lost in transit. This approach is consistent with RSVP's state management rules.

[8] *avail_pd* must be at least the size of a POLICY_DATA object without a data portion or the call would fail.

Parameter resv_desc provides a list of reservation descriptions. This description is made of three components: `lih`, `resv_handle`, and `resv_flowspec`.

In the upstream direction (e.g., Resv) authorization may need to be checked on multiple LIHs (all reservations for a flow). In such cases, status queries can be perform separately for each LIH, once for all LIHs, or anything in between. full_list_indication must be set at the last of `PC_AuthCheck()` query of the series.[9]

Authorization can be verified on both the 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 triggered by the arrival of a new message; these are handled transparently by the input processing call `PC_InPolicy()`. In a synchronous, when an upcall mechanism is not supported, RSVP must verify the status of reservations before refreshing them.

- ```
  Call:  PC_Init       (K, upcall,...  )
                       -> RCode
  ```

  This call initializes the LPM and sets specific RSVP/policy configuration parameters. `K` is the soft-state multiplier for refresh-period (see [RSVPSP]) and `upcall` registers a routine that may be called by the LPM to notify RSVP on policy changes. (See next item)

- ```
  Call:  upcall        (event_type, cbp, message_type,
                       lih, rsvp_hop list, session,
                       shr_ind, filter_spec_list,
                       out_policy_objects,
                       RCode)
  ```

  `Event_type` determines the original call type (if applicable). `cbp` is an echo of the cbp provided by RSVP when making the original call. RSVP may use this pointer to locate the original context of the call.

  RCode uses the same values specified in this document, as it would for the original calls. Notice that the upcall parameters are a superset of the parameters used by all the non-blocking PC calls.

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

  This call affects all the state associated with a particular multicast (or unicast) branch. It is used for route changes, blockade state other instantaneous state change performed

[9] When policies are interdependent across LIHs (as when the cost is shared among downstream receivers), full_list_ind notifies the server that the list of reserved LIH is complete and that it can safely compute the status of these reservations.
  ```

by RSVP. When applicable parameters are NULL, an aggregate of the state is affected (across all values of the NULL-ed parameter). For example, PC_DelState(NULL, session, NULL, NULL, PC_delete) would purge all the policy state associated with *session*.

Parameter *op_type* is the requested type of state change:

```
PC_Delete :        Purge state.
PC_Block  :        Blockade (ignore) state
PC_Unblock:        Unblock state.
```

3.3 PC Success Codes

The return code (RCode) provides policy feedback to RSVP, it is made of three separate return variables:[10]

- Function return value:

  0: Success

  The call was completed successfully. For `PC_AuthCheck()` and `PC_InPolicy()` it also signals the authorization of the RSVP operation (e.g., Reservation, Path, Tear, etc.) RSVP need not check the PC_flags or PC_errno.

  1: Pending

  The requested results are delayed. Should these results become available or the giveup_time expires, the notification upcall would signal RSVP.

  2: Warning

  Same as success except that there is a non-fatal warning and RSVP must check the PC_flags for further instructions.

  3: Policy failure

  Policy authorization for the RSVP operation has failed. RSVP should invoke its standard error reporting mechanism (PathErr or ResvErr).

- *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`

  The incoming POLICY_DATA object contains an update. RSVP must immediately initiate update forwarding procedures.

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

`0x02 PC_RC_Resp`

RSVP must immediately initiate a message. The type of requested message is placed in the PC_errno variable and corresponds to message type values in the RSVP common header.

`0x04 PC_RC_Preempt`

Only for Resv incoming objects. RSVP should inform the local admission control that the reservation is of lower priority and can be preempted, if necessary.

3.4 RSVP's Policy Actions

The PC success codes, and especially *PC_Flags* advise RSVP about appropriate required actions. This section describes these actions in greater detail.

3.4.1 Pending Results and Asynchronous Notification

For various reasons the LPM may need to consult an external entity (e.g., server) for partial policy processing. (For a description of a router/server protocol see [LPM]). For efficiency reasons, RSVP must not be forced to wait idly while external policy processing takes place. Instead, A configurable option permits calls to PC_AuthCheck() or PC_OutPolicy() to terminate with a "pending" return value whenever results are delayed (for any reason).

The following steps take place when RSVP receives a pending return value:

- RSVP halts the current operation, saves its state (linked to the cbp), and moves to the next task.
- Once results are available or the giveup_time expires[11] the LPM "wakes up" RSVP by calling the notification upcall.
- The wakeup call provides results, context, and the cbp (see Section 3.2).
- RSVP resumes the previously halted operation and uses the provided context parameters as if they were returned by the original (previously pending) call.

3.4.2 Error Signaling

Policy errors are reported by either ResvErr or PathErr messages with a policy failure error code (specified in [RSVPSP]). Policy error message must include a POLICY_DATA object;

[11] If results are still unavailable at giveup_time, the answer is assumed to be failure (for AuthCheck) or no output (for OutPolicy).

the object contains details of the error type and reason. If none is provided, the error message should not be sent.

If a multicast reservation fails, RSVP should not attempt to discover which reservation caused the failure (as it would do for blockade state). Instead, it should attempt to deliver the policy ResvErr to ALL downstream hops. The LPM would limit the error distribution by providing outgoing objects only to "culprit" next-hops; if the LPM performs local repair[12] it can prevent the further distribution of ResvErr or PathErr messages.

The LPM should internally implement blockade state style mechanism for similar reasons as RSVP (preventing an unauthorized reservation from forcing other valid reservations to fail). This document does not define this mechanism and assumes it would be policy-implementation specific.

3.4.3 Policy Response

The LPM can initiate an immediate outgoing RSVP message (Path, Resv, etc.) by setting the flag `PC_RC_Response` and providing the number of the requested RSVP message in the `PC_errno` variable.[13] This mechanism is useful when the appropriate RSVP message is either scheduled for a significantly later time, or not at all. When the `PC_RC_Response` flag is set, RSVP, should schedule the requested outgoing message as if its refresh timer has expired; for non-refreshed messages like ResvErr, RSVP should act as if they were just received.

This mechanism allows policies that require an immediate confirmation by scheduling a reverse-direction message with a confirmation POLICY_DATA object.

3.5 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. Consequently, policy objects transmitted at one edge of an autonomous cloud may traverse intermediate non-policy-capable RSVP nodes. 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 the same type of RSVP messages in which they arrived.

[12] Local repair can be performed by substituting the failed POLICY_DATA object with a different one.

[13] The value of the PC_errno corresponds to message type values in the RSVP common header.

- Multicast merging (splitting) nodes:

  In the upstream direction:

  Applicable incoming POLICY_DATA objects are concatenated, and the list is forwarded with the upstream message.

  On the downstream direction:

  A copy of all applicable incoming objects is forwarded with each downstream message.

The same rules apply to unrecognized policies (sub-objects) within the POLICY_DATA object. However, since that can only occur in a policy-capable node, it is the responsibility of the LPM and not RSVP.

4 Acknowledgment

This document incorporates inputs from Lou Berger, Bob Braden, Deborah Estrin, Roch Guérin, Dimitrios Pendarakis, Raju Rajan, and Scott Shenker. It also reflects feedback from many other 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 ReSer-Vation 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