Internet Draft
Expires December 12, 1996
File: draft-ietf-rsvp-policy-ext-00.ps

Shai Herzog
USC/ISI
June 1996

RSVP Extensions for Policy Control

June 12, 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 RSVP extensions required to support policy based admission control in RSVP. The contents of this document should be considered as an extension to the RSVP functional specifications document [BRA96].

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 infantile stages. As a result, Ver. 1 of the RSVP Functional Specifications[BRA96] 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 any insight about how policy enforcement should be carried out. The current RSVP Functional Specification defines an admission decision 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 [HER96b] and a discussion of a prototype mechanism for enforcing these policies can be found in [HER96a]. We expect that this document would merge into the RSVP specification document [BRA96] as it matures.

2 Object Format

In this section we describe the internal format of two types of policy objects. This section defines RSVP objects, and thus, conceptually belongs with the RSVP spec, however, due to the early stage and dynamic nature of this work, this unification is currently deferred.

2.1 POLICY_DATA objects

```
+-------------+-------------+-------------+-------------+
|    Length                 | POLICY_DATA |      1      |
+-------------+-------------+-------------+-------------+
|    Flags    |              Reserved                   |
+-------------+-----------------------------------------+
|                                                       |
//   FILTER_SPEC Objects (optional) ...                //
|                                                       |
+-------------------------------------------------------+
|                                                       |
//   POLICY_ELEMENT or {INTEGRITY,POLICY_ELEMENT}      //
|    objects (optional) ...                             |
|                                                       |
+-------------------------------------------------------+
```

POLICY_DATA objects begin with the standard RSVP object header. The header is followed by a 32 bit word, out of which the first eight bits are used for flags and the rest is reserved for future use. The only flag currently defined is `PDF_REPORT`. This flag applies only to Resv messages, and if set, it informs RSVP that a Resv-Report message is required.[1] Next comes an optional list of FILTER_SPEC. This list can associate the policies included in the object with a set of specific senders (flows). If no FILTER_SPEC objects are listed, the policy is considered to be associated with the *all* the flows of the session. The POLICY_DATA object ends with a list of POLICY_ELEMENTs.

2.2 POLICY_ELEMENT objects

We introduce POLICY_ELEMENT as a new class of RSVP object. These objects carry policy specific information, and their internal representation is opaque to RSVP.

```
+-------------+-------------+-------------+-------------+
|     Length                |      20     |    CType    |
+---------------------------+-------------+-------------+
|     Variable length opaque data                       |
+-------------------------------------------------------+
```

Individual POLICY_ELEMENT objects may be preceded by INTEGRITY objects. In this document, we do not define the algorithm for computing the INTEGRITY value. However, in order to guarantee that the POLICY_ELEMENT object 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.

2.3 Semantic fragmentation of POLICY_DATA objects

POLICY_DATA objects may need to be fragmented since their size can potentially exceed the size of the MTU by orders of magnitude. Large objects mainly result from the concatenation of multiple POLICY_DATA objects arriving from multiple previous hops. Semantic fragmentation provides an excellent solution for this case; if these objects are large as a result of concatenation, we see no reason why the original, self contained objects could not be forwarded individually. The relationship between these self contained objects is not trivial and usually depends on the underlying policy. Consider the case where N receivers share a reservation; in one access control policy, access may be granted based on a sufficient subset, while in some accounting policies, accurate results may depend on obtaining the complete set. However, when fragments are lost we only have two options: one is to use the received subset, and the other one is to drop the whole message. These options should be preserved such that the policy module would be aware of fragments (POLICY_DATA objects) loss, and specific policies can choose their response to the loss. The practical implications are that semantic fragmentation POLICY_DATA objects turns out to be quite simple, using a single rule:

A POLICY_DATA object consisting of a list of objects L can be broken into N sub-lists (L_i), each creating an object conceptually identical to the original one,[2] except that L_i replaces L

[1] This flag may become obsolete if/when a Report Request bit is added to the Resv message format.

[2] Checksums, and similar values may change as a result of the object split.

in each object $(i = 1..N)$.

This rule applies recursively; for example, consider the following POLICY_DATA object:

$[H, F, S_1, S_2, S_3, Pe_1, I_2, Pe_2, Pe_3]$

Where H, F represent the header and flags. F_i represents a filter for source i, Pe_j represent a POLICY_ELEMENT object j and I_j represent an INTEGRITY object for Pe_j.

First, we fragment the source list, creating two objects:

$[H, F, S_1, S_2, Pe_1, I_2, Pe_2, Pe_3]$, and $[H, F, S_3, Pe_1, I_2, Pe_2, Pe_3]$

Then we fragment the policy element list, creating four objects:

$$\begin{array}{lcl} [H, F, S_1, S_2, Pe_1, Pe_3], & & [H, F, S_1, S_2, I_2, Pe_2], \\ [H, F, S_3, Pe_1, Pe_3], & \text{and} & [H, F, S_3, I_2, Pe_2] \end{array}$$

The result of this semantic fragmentation allowed us to replace a single large object with four smaller ones, while maintaining identical semantics. Semantic fragmentation imposes an added burden on state management since the absence of a policy element is ambiguous; a missing policy information may be the result of an actual change in the policy, or a result of a fragment loss. This implies the need for a timeout mechanism for each {POLICY_ELEMENT, FILTER_SPEC} object pair. Moreover, if one wanted to purge state before the timeout period, a teardown mechanism would be required as well.

Notice that we consider individual POLICY_ELEMENT objects to be atomic and do not attempt to further fragment them. This decision is based on our desire to keep things simple, and our believe that at present, IP fragmentation should be sufficient for such individual objects.

3 New RSVP message: Reservation Report

Irrespective of specific mechanisms or implementation, the basic building blocks of access control and accounting must be bi-directional in order to allow both source and receiver based policies, and both advertising and feedback. Which RSVP messages should encapsulate these upstream and downstream objects? The choice for upstream message is natural; the reservation message. The downstream direction, however, is more problematic: Path messages flow downstream, but are routed according to the multicast group membership, and therefore cannot accurately be delivered to a specific next hop.[3] This makes Path messages less likely to be used for access control, and especially for accounting.

[3] The same problem existed for the original design of ResvErr, until it was changed to a unicast delivery along the multicast tree.

We proposed a new RSVP message type: *Reservation Report* (Rept). Reservation Report messages are sent unicast, downstream, according to the Next_Hop object carried by Resv messages; although Reservation Report messages follow the multicast tree, their unicast delivery provides accurate delivery to the appropriate next hop nodes and only to these nodes[4] Although we propose this new message for policy control, it may prove useful for other, more general functions of the RSVP protocol. A reservation may have different forms of responses to it: A negative response (ResvErr), a positive response (ack), and a more advanced form of a Reservation Report, like the one proposed here. An integrated approach may incorporate all three responses in the same message type while leaving room for future types.

4 Default handling of policy data objects

It is generally agreed upon that policies may be enforced on a much coarser resolution than RSVP deployment. We do not expect (or desire) every RSVP node to function as a policy node, nor do we expect each node to be able to recognize and handle all types of policies (i.e., both intranet and internet policies). The question is: what should be RSVP's default handling of unrecognized POLICY_DATA objects ? We consider two potential cases; first, when the RSVP node is **not** a policy node at all, and second, when the incoming POLICY_DATA object includes objects of an unknown type. Both cases are handled in a similar manner: RSVP is required to stored and forward the set of unknown object without any modification, merging or other manipulation. Notice that the internal format of POLICY_DATA objects is a list of POLICY_ELEMENT objects; If a node is a merging point in the multicast tree, the default output is simply the concatenation of the lists of incoming objects encapsulated in a single POLICY_DATA object.

5 API modifications

The current version of the RSVP functional specification [BRA96] is silent about possible policy interaction between applications (end-users) and the first hop RSVP node. Furthermore, it does not provide guidelines for an API design that would be consistent with future policy control mechanisms. For example, some policies (e.g., accounting) require RSVP to be able to relate an incoming message to individual local receivers. (e.g., split incoming costs between the local receivers). An API which is not designed to maintain information about local receivers, but instead merges their state, would be unable to support user-based accounting.

[4] Consider the case of multipoint links or network clouds: a single copy of a Path message may be delivered to an unknown number of next hops, while the copy of a Report message is guaranteed to reach only the targeted node.

This section is a place holder for a more detailed API design that will be developed in future versions of this document.

6 Acknowledgment

This document incorporates inputs from Deborah Estrin, Scott Shenker and Bob Braden and feedback from RSVP collaborators.

References

[BRA96] R. Braden, L. Zhang, S. Berson, S. Herzog, and S. Jamin, Resource ReSerVation Protocol (RSVP) Version 1 Functional Specification. *Internet-Draft*, draft-ietf-rsvp-spec-12.txt.

[HER96a] Local Policy Modules (LPM): Policy Enforcement for Resource Reservation Protocols. *Internet-Draft*, draft-ietf-rsvp-policy-lpm-00.[ps,txt].

[HER96b] Accounting and Access Control Policies for Resource Reservation Protocols. *Internet-Draft*, draft-ietf-rsvp-policy-arch-00.[ps,txt].