Internet Draft
Expiration: Apr. 1999
File: draft-ietf-rap-cops-rsvp-01.txt

Jim Boyle
Level3
Ron Cohen
Cisco
David Durham
Intel
Shai Herzog
IPHighway
Raju Rajan
IBM
Arun Sastry
Cisco

COPS usage for RSVP

Last Updated: November 18, 1998

Status of this Memo

Abstract

This document describes usage directives for supporting COPS policy services in RSVP environments.

Table of Contents

1. Introduction

The Common Open Policy Service (COPS) protocol is a query response protocol used to exchange policy information between a network policy server and a set of clients [COPS]. COPS is being developed within the RSVP Admission Policy Working Group (RAP WG) of the IETF, primarily for use as a mechanism for providing policy-based admission control over requests for network resources [RAP].

This document is based on and assumes prior knowledge of RAP framework [RAP] and the basic COPS [COPS] protocol. It provides specific usage directives for using COPS in outsourcing policy control decisions by RSVP clients (PEPs) to policy servers (PDPs).

Given the COPS protocol design, client specific functionality is mainly limited to interoperability usage guidelines as well as client specific examples.

2. RSVP values for COPS objects

The format and usage of several COPS objects is affected when used for client type RSVP. This section describes these objects and the usage.

2.1. Context Object (Context)

The semantics of the Context object for RSVP is as follows:

R-Type (Request Type Flag)

0x01 = Incoming-Message request
The arrival of an incoming RSVP message

Allows processing of incoming policy information as well as the decision whether to accept an incoming message. If It is rejected, the message is treated as if it never Arrived.

0x02 = Resource-Allocation request
Applies only for Resv messages.

The decision whether to admit a reservation and commit local resources to it is performed for the merge of all reservations that arrived on a particular interface (potentially from several RSVP Next-Hops).

0x04 = Outgoing-Message request
The forwarding of an outgoing RSVP message.

The Decision whether to allow the forwarding of an outgoing RSVP message as well as providing the relevant outgoing policy information.

M-Type (Message Type)

The M-Type field in the Context Object may have one of the Following values that correspond to supported RSVP messages In COPS:

1 = Path
2 = Resv
3 = PathErr
4 = ResvErr

Note: The PathTear, ResvTear, and the Resv Confirm message types are not supported.

2.2. Client Specific Information (ClientSI)

All objects that were received within an RSVP message that are associated with the RSVP flow are encapsulated inside the Client Specific Information Object without alteration. (See Section 3.1. on multiple flows packed in a single RSVP message). These RSVP objects are simply contained within a single Signaled Client Specific Information Object (RSVP ClientSI) exchanged between the PEP and remote PDP.

2.3. Decision Object (Decision)

COPS allows PDP to control RSVP s response to messages. Beyond traditional accept/deny, PDPs may use the Trigger Error flag to allow a request yet trigger a warning at the same time. To allow resource allocation yet deny forwarding of a message, etc.

Decision Flags

The following decision flags apply to RSVP:

0x01 = Signaled (RSVP) accept (deny if set)

This flag should be interpreted with the decision context flag to figure out what it applies to.

0x08 = Trigger Error (PathErr for Path query, or ResvErr for Resv)

Client Specific Policy Information

This object may include one or more policy elements (as specified for the RSVP Policy Data object [RSVP-EXT] which are assumed to be well understood by the client s LDP. The PEP should consider these as if they arrived in the message Policy Data object.

For example: Given Policy Elements that specify a flow s preemption priority, these elements may be included in an incoming Resv message or may be also be provided by the PDP responding to a query.

Replacement Data

The Replacement object may contain multiple RSVP objects to be replaced (from the original RSVP request). Typical replacement is performed on the Forward Outgoing request (for instance, replacing outgoing Policy Data), but is not limited, and can also be performed on other contexts (such as Allocate Resources). Other examples, may require replacement of the RSVP FlowSpec object for controlling resources across a trusted zone (with PIN nodes).
Currently, RSVP clients are only required to allow replacement of two objects: Policy Data and FlowSpec.

Replacement is performed in the following manner:
If Replacement Data decision doesn't appear in a decision message, all signaled objects are passed as if the PDP was not there. When an object of a certain C-Num appears it replaces ALL the instances of C-Num objects in the RSVP message. If it appears empty (with a length of 4) it simply removes all instances of C-Num objects without adding a thing.

3. Operation of COPS for Policy Control Over RSVP

3.1. RSVP flows

Policy Control is performed per RSVP flow. An RSVP flow corresponds to an atomic unit of reservation as identified by RSVP (TC reservation). It should be noted that RSVP allows multiple flows to be packed (which is different from merged) into a single FF Resv message. To support such messages a separate COPS request must be issued for each of the packed flows as if they were individual RSVP messages.

3.2. Expected Associations for RSVP Requests

RSVP signaling requires the participation of both senders and receivers. RSVP processing rules define what is the subset of the Path state that matches each Resv state. In the common unicast case, the RSVP session includes one Path state and one Resv state. In multicast cases the correspondence might be many to many. Since the decision to admit a reservation for a session may depend on information carried both in Path and Resv messages, we term the Path States that match with a single Resv state as its associated states. It is assumed that the PDP is capable of determining these associations based on the RSVP message processing rules given the RSVP objects expressed in the COPS Client Specific Information

Object.

For example, the PDP should be able to recognize activation and deactivation of RSVP blockade state following discrete events like the arrival of a ResvErr message (activate the blockade state) as well as the change in the outgoing Resv message.

3.3. RSVP's Capacity Admission Control: Commit and Delete

In RSVP, the admission of a new reservation requires both an administrative approval (policy control) and capacity admission control. Once local admission control accepts the reservation, the PEP notifies the remote PDP by sending a report message specifying the Commit type. The Commit type report message is to be used to signify when billing should effectively begin, and performing heavier operations (e.g., debiting a credit card) is permissible.

If instead a reservation approved by the PDP fails admission due to lack of resources, the PEP must issue a no-commit report and fold back and send an updated request to its previous state (previously installed reservation). If no state was previously installed, the PEP should issue a delete.

3.4. Policy Control Over PathTear and ResvTear

PathTear and ResvTear messages are not controlled by this policy architecture. This relies on two assumptions: First, that MD-5 authentication verifies that the Tear is received from the same node that sent the initial reservation, and second, that it is functionally equivalent to that node holding-off refreshes for this reservation. When a ResvTear or PathTear is received at the PEP, all affected states installed on the PDP should either be deleted or updated by the PEP.

3.5. PEP Caching COPS Decisions

Because COPS is a stateful protocol, refreshes for RSVP Path and Resv messages need not be constantly sent to the remote PDP. Once a decision has been returned for a request, the PEP can cache that decision and apply it to future refreshes. The PEP is only responsible for updating a request state if there is a change detected in the corresponding Resv or Path message.

If the connection is lost between the PEP and the PDP, the cached RSVP state may be retained for the RSVP timeout period. If no connection can be reestablished with the PDP or a backup PDP after the timeout period, the RSVP PEP is expected to default back to using its LDP results. Additionally, the LDP is to be used for the admission control of any new RSVP messages that may have arrived while connectivity was lost.

Once a connection is reestablished to a new (or the original) PDP

the PDP may issue a SSQ request. In this case, the PEP must reissue

requests that correspond to the current RSVP state (as if all the state has been updated recently). It should also include as LDP the current (cached) decision regarding each such state.

3.6. Using Multiple Context Flags in a single query

RSVP is a store-and-forward control protocol where messages are processed in three distinctive steps (input, resource allocation, and output). Each step requires a separate policy decision as indicated by context flags (see Section 2.1). In many cases, setting multiple context flags for bundling two or three operations together in one request may significantly optimize protocol operations.

The following rules apply for setting multiple Context flags:

a. Multiple context flags can be set only in two generic cases which are guaranteed not to cause ambiguity and represent substantial portion of expected COPS transactions.

   Unicast FF:

   [Incoming + Allocation + Outgoing]

   Multicast with only one Resv message received on the interface

   [Incoming + Allocation]

b. Context events are ordered by time since every message processing must first be processed as Incoming, then as Resource allocation and only then as Outgoing. When multiple context flags are set, all ClientSI objects included in the request are assumed to be processed to the latest flag. This rule applies both to request (REQ) context as well as to decision (DEC) context.

   For example: when combining Incoming + Allocation for an incoming Resv message, the Flowspec included in the ClientSI would be the one corresponding to the Resource-Allocation context (TC).

c. Each decision is bound to a context object, which determines which portion of the request context it applies to. When different decisions apply to different sub-groups of context the PDP should send each group of decision objects encapsulated or separated by the context flags object with the context flags applicable to these objects set. (See the examples in Section 4).

3.7. Trusted zones and secure policy tunneling

Security for RSVP messages is provided by inter-router MD5 authentication [MD5], assuming a chain-of-trust model.
A possible deployment scenario calls for PEPs to be deployed at the network edge (boundary nodes) while PINs are deployed in the core of the network (backbone). In this case, MD5 trust (authentication) must be established between boundary (non-neighboring) PEPs, which is achieved through internal signing of the Policy Data object. [RSVP-EXT].

4. Illustrative Examples, Using COPS for RSVP

This section details both typical unicast and multicast scenarios.

4.1. Unicast Flow Example

This section details the steps in using COPS for controlling a Unicast RSVP flow. It details the contents of the COPS messages with respect to the following figure.

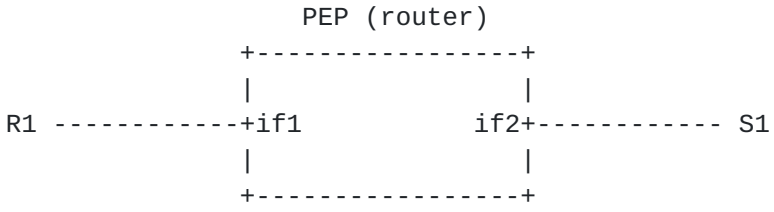

Figure 1: Unicast Example: a single PEP view

The PEP router has two interfaces (if1, if2). Sender S1 sends to receiver R1.

A Path message arrives from S1:

```
PEP --> PDP   REQ := <Handle A> <Context: in & out, Path>
                     <In-Interface if2> <Out-Interface if1>
                     <ClientSI: all objects in Path message>

PDP --> PEP   DEC := <Handle A> <Context: in & out, Path>
                     <Decision flags: Accept>
```

A Resv message arrives from R1:

```
PEP --> PDP   REQ := <Handle B>
                     <Context: in & allocation & out, Resv>
                     <In-Interface if1> <Out-Interface if2>
                     <ClientSI: all objects in Resv message>
```

```
       PDP --> PEP   DEC := <Handle B>
                            <Context: in, Resv>
                            <Decision flags: accept>
                            <Context: allocation, Resv>
                            <Decision flags: accept>
                            <Decision: ClientSI, Priority=7>
                            <Context: out, Resv>
                            <Decision flags: accept>
                            <Decision replace: POLICY-DATA1>

       PEP --> PDP   RPT := <Handle B>
                            <Report type: commit>
```

Notice that the Decision was split because of the need to specify different decision objects for different context flags.

Time Passes, the PDP changes its decision:

```
       PDP --> PEP   DEC := <Handle B>
                            <Context: allocation, Resv>
                            <Decision flags: accept>
                            <Decision: ClientSI, Priority=3>
```

Because the priority is too low, the PEP preempts the flow:

```
       PEP --> PDP   DRQ := <Handle B>
                            <Reason Code: Preempted>
```

Time Passes, the sender S1 ceases to send Path messages:

```
       PEP --> PDP   DRQ := <Handle A>
                            <Reason: Timeout>
```

4.2. Shared Multicast Flows

This section details the steps in using COPS for controlling a multicast RSVP flow. It details the contents of the COPS messages with respect to the following figure.

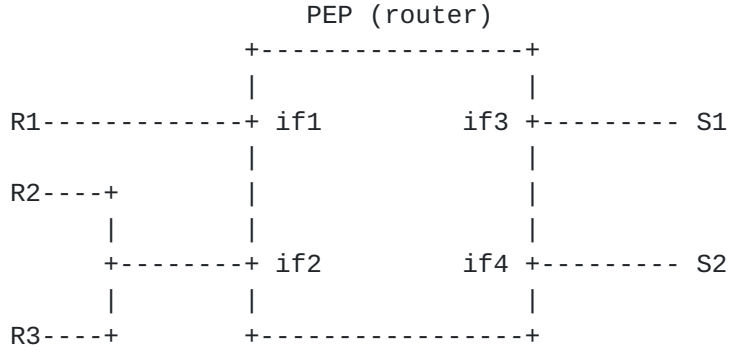

Figure 2: Multicast example: a single PEP view

Figure 2 shows an RSVP PEP (router) which has two senders (S1, S2) and three receivers (R1, R2, R3) for the same multicast session. Interface if2 is connected to a shared media.
In this example, we assume that the multicast membership is already in place, no previous RSVP messages were received, and the first to arrive is a Path message on interface if3 from sender S1:

```
    PEP --> PDP   REQ := <Handle A> <Context: in, Path>
                         <In-interface if3>
                         <ClientSI: all objects in incoming Path>

    PDP --> PEP   DEC := <Handle A> <Context: in, Path>
                         <Decision flags: accept>
```

The PEP consults its forwarding table, and finds two outgoing interface for the path (if1, if2). The exchange below is for interface if1, another exchange would likewise be completed for if2 using the new handle B2.

```
    PEP --> PDP   REQ := <Handle B1> <Context: out, Path>
                         <Out-interface if1>
                         <clientSI: all objects in outgoing Path>

    PDP --> PEP   DEC := <Handle B1>
                         <Context: out, Path>
                         <Decision flags: accept>
                         <Decision Replacement: POLICY-DATA1>
```

Here, the PDP decided to allow the forwarding of the Path message and provided the appropriate policy-data object for interface if1.

Next, a WF Resv message from receiver R2 arrives on interface if2.

```
    PEP --> PDP   REQ := <Handle C> <Context: in & allocation, Resv>
                         <In-interface if2>
                         <ClientSI: all objects in Resv message
                          including RSpec1 >

    PDP --> PEP   DEC := <Handle C>
                         <Context: in, Resv>
                         <Decision flags: accept>
                         <Context: allocation, Resv>
                         <Decision flags: accept>
                         <Decision ClientSI: priority=5>

    PEP --> PDP   RPT := <handle C> <Commit>
```

Here, the PDP approves the reservation and assigned it preemption priority of 5. The PEP responded with a commit report.

The PEP needs to forward the Resv message upstream toward S1:

```
    PEP --> PDP   REQ := <Handle E> <Context: out, Resv>
                         <out-interface if3>
                         <Client info: all objects in outgoing Resv>

    PDP --> PEP   DEC := <Handle E>
                         <Context: out, Resv>
                         <Decision flags: accept>
                         <Decision replacement: POLICY-DATA2>
```

Note: The Context object is part of this DEC message even though it may look redundant since the REQ specified only one context flag.

Next, a new WF Resv message from receiver R3 arrives on interface if2 with a higher RSpec (Rspec2). Given two reservations arrived on if2, it cannot perform a request with multiple context flags, and must issue them separately.

The PEP re-issues an updated handle C REQ with a new context object <Context: in , Resv>, and receives a DEC for handle C.

```
    PEP --> PDP   REQ := <Handle F> <Context: in , Resv>
                         <In-interface if2>
                         <ClientSI: all objects in Resv message
                          including RSpec2 >

    PDP --> PEP   DEC := <Handle F> <Context: in , Resv>
                         <Decision flags: accept>

    PEP --> PDP   REQ := <Handle G> <Context: allocation, Resv>
                         <In-interface if2>
                         <ClientSI: all objects in merged Resv
                          including RSpec2 >

    PDP --> PEP   DEC := <Handle G>
                         <Context: allocation, Resv>
                         <Decision flags: accept>
                         <Decision ClientSI: priority=5>

    PEP --> PDP   RPT := <handle G> <Commit>
```

Given the change in incoming reservations, the PEP needs to forward a new outgoing Resv message upstream toward S1. This repeats exactly the previous interaction of Handle E, except that the ClientSI objects now reflect the merging of two reservations.

If an ResvErr arrives from S1, the PEP maps it to R3 only (because it has a higher flowspec: Rspec2) the following takes place:

```
PEP --> PDP   REQ := <Handle H> <Context: in, ResvErr>
                     <In-interface if3>
                     <ClientSI: all objects in incoming ResvErr>

PDP --> PEP   DEC := <Handle H> <Context: in, ResvErr>
                     <Decision flags: accept>

PEP --> PDP   REQ := <Handle I> <Context: out, ResvErr>
                     <Out-interface if2>
                     <clientSI: all objects in outgoing ResvErr>

PDP --> PEP   DEC := <Handle I>
                     <Context: out, ResvErr>
                     <Decision flags: accept>
                     <Decision Replacement: POLICY-DATA3>
```

When S2 joins the session by sending a Path message, incoming and outgoing Path requests are issued for the new Path. A new outgoing Resv request would be sent to S2.

5. References

[RSVP-EXT] Herzog, S. "RSVP Extensions for Policy Control", Internet-Draft, draft-ietf-rap-rsvp-ext-00.txt, Apr. 1998.

[RAP] Yavatkar, R., et al., "A Framework for Policy Based Admission Control",IETF <draft-ietf-rap-framework-00.txt>, November, 1997.

[COPS] Boyle, J., Cohen, R., Durham, D., Herzog, S., Raja,n R., Sastry, A., "The COPS (Common Open Policy Service) Protocol", IETF <draft-ietf-rap-cops-02.txt>, Aug. 1998.

[RSVP] Braden, R., Zhang, L., Berson, S., Herzog, S., and Jamin, S., "Resource Reservation Protocol (RSVP) Version 1 Functional Specification", IETF RFC 2205, Proposed Standard, September 1997.

6. Author Information and Acknowledgments

Special thanks to Timothy O'Malley our WG Chair, Raj Yavatkar, Russell Fenger, Fred Baker, Laura Cunningham, Roch Guerin, Ping Pan, and Dimitrios Pendarakis for their valuable contributions.

Jim Boyle
Level 3 Communications
1450 Infinite Drive13
Louisville, CO 80027
303.926.3100
email: jboyle@l3.net

Ron Cohen
CISCO Systems
Hasadna St.
Ra'anana 43650 Israel
972.9.7462020
ronc@cisco.com

David Durham
Intel
2111 NE 25th Avenue
Hillsboro, OR 97124
503.264.6232
David_Durham@mail.intel.com

Raju Rajan
IBM T.J. Watson Research Cntr
P.O. Box 704
Yorktown Heights, NY 10598
914.784.7260
raju@watson.ibm.com

Shai Herzog
IPHighway
400 Kelby St., Suite 1500
Fort-Lee, NJ 07024
201.585.0800
herzog@iphighway.com

Arun Sastry
Cisco Systems
506210 W Tasman Drive
San Jose, CA 95134
408.526.7685
asastry@cisco.com