                          COPS usage for RSVP

Status of this Memo

   This document specifies an Internet standards track protocol for the
   Internet community, and requests discussion and suggestions for
   improvements. Please refer to the current edition of the "Internet
   Official Protocol Standards" (STD 1) for the standardization state
   and status of this protocol. Distribution of this memo is unlimited.

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 the 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, RSVP directives are mainly limited to
    RSVP applicability, interoperability and usage guidelines, as well as
    client specific examples.

2 RSVP values for COPS objects

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

2.1 Common Header, client-type

    RSVP is COPS client-type 1

2.2 Context Object (Context)

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

   R-Type (Request Type Flag)

   Incoming-Message request

        This context is used when the PEP receives an incoming RSVP
        message. The PDP may decide to accept or reject the incoming
        message and may also apply other decision objects to it. If the
        incoming message is rejected, RSVP should treat it as if it
        never arrived.

   Resource-Allocation request

        This context is used when the PEP is about to commit local
        resources to an RSVP flow (admission control). This context
        applies to Resv messages only. The decision whether to commit
        local resources is made for the merge of all reservations
        associated with an RSVP flow (which have arrived on a
        particular interface, potentially from several RSVP Next-Hops).

   Outgoing-Message request (forwarding an outgoing RSVP message)

        This context is used when the PEP is about to forward an
        outgoing RSVP message. The PDP may decide to allow or deny the
        outgoing message, as well as provide an outgoing policy data
        object.

   M-Type (Message Type)

   The M-Type field in the Context Object identifies the applicable RSVP
   message type. M-Type values are identical to the values used in the
   "msg type" field in the RSVP header [RSVP].

   The following RSVP message types are supported in COPS:

   Path
   Resv
   PathErr
   ResvErr

   Other message types such as PathTear, ResvTear, and Resv Confirm are
   not supported. The list of supported message types can only be
   extended in later versions of RSVP and/or later version of this
   document.

2.3 Client Specific Information (ClientSI)

   All objects that were received in an RSVP message are encapsulated
   inside the Client Specific Information Object (Signaled ClientSI)
   sent from the PEP to the remote PDP (see Section 3.1. on multiple
   flows packed in a single RSVP message).

   The PEP and PDP share RSVP state, and the PDP is assumed to implement
   the same RSVP functional specification as the PEP. In the case where
   a PDP detects the absence of objects required by [RSVP] it should
   return an <Error> in the Decision message indicating "Mandatory
   client-specific info missing". If, on the other hand, the PDP detects
   the absence of optional RSVP objects that are needed to approve the
   Request against current policies, the PDP should return a negative
   <Decision>.

   Unlike the Incoming and Outgoing contexts, "Resource Allocation" is
   not always directly associated with a specific RSVP message. In a
   multicast session, it may represent the merging of multiple incoming
   reservations. Therefore, the ClientSI object should specifically
   contain the SESSION and STYLE objects along with the merged FLOWSPEC,
   FILTERSPEC list, and SCOPE object (whenever relevant).

2.4 Decision Object (Decision)

   COPS provides the PDP with flexible controls over the PEP using
   RSVP's response to messages. While accepting an RSVP message, PDPs
   may provide preemption priority, trigger warnings, replace RSVP
   objects, and much more, using Decision Commands, Flags, and Objects.

   DECISION COMMANDS

   Only two commands apply to RSVP

   Install

      Positive Response:
      Accept/Allow/Admit an RSVP message or local resource allocation.

   Remove

      Negative Response:
      Deny/Reject/Remove an RSVP message or local resource allocation.

DECISION FLAGS

The only decision flag that applies to RSVP:

Trigger Error

   If this flag is set, RSVP should schedule a PathErr, in response
   to a Path message, or a ResvErr (in response of a Resv message).

STATELESS POLICY DATA

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 LPDP. The PEP should consider these as an
addition to the decision already received from the PDP (it can only
add, but cannot override it).

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 provided by the PDP responding to a query.

Stateless objects must be well understood, but not necessarily
supported by all PEPs. For example, assuming a standard policy
element for preemption priority, it is perfectly legitimate for some
PEPs not to support such preemption and to ignore it. The PDP must be
careful when using such objects. In particular, it must be prepared
for these objects to be ignored by PEPs.

Stateless Policy Data may be returned in decisions and apply
individually to each of the contexts flagged in REQ messages. When
applied to Incoming, it is assumed to have been received as a
POLICY_DATA object in the incoming message. When applied to Resource
Allocation it is assumed to have been received on all merged incoming
messages. Last, when applied to outgoing messages it is assumed to
have been received in all messages contributing to the outgoing
message.

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 "Resources-Allocation Request"). In other
cases, replacement of the RSVP FlowSpec object may be useful for
controlling resources across a trusted zone (with policy ignorant

nodes (PINs). Currently, RSVP clients are only required to allow
replacement of three objects: POLICY_DATA, ERROR_SPEC, and FLOWSPEC,
but could optionally support replacement of other objects.

RSVP object replacement is performed in the following manner:

If no Replacement Data decision appears in a decision message, all
signaled objects are processed 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
anything.

3 Operation of COPS for RSVP PEPs

3.1 RSVP flows

Policy Control is performed per RSVP flow, which is defined by the
atomic unit of an RSVP reservation (TC reservation). Reservation
styles may also impact the definition of flows; a set of senders
which are considered as a single flow for WF reservation are
considered as a set of individual flows when FF style is used.

Multiple FF flows may be packed into a single Resv message. A packed
message must be unpacked where a separate request is issued for each
of the packed flows as if they were individual RSVP messages. Each
COPS Request should include the associated POLICY_DATA objects, which
are, by default, all POLICY_DATA objects in the packed message.
Sophisticated PEPs, capable of looking inside policy objects, may
examine the POLICY_DATA or SCOPE object to narrow down the list of
associated flows (as an optimization).

Please note that the rules governing Packed RSVP message apply
equally to the Incoming as well as the Outgoing REQ context.

3.2 Expected Associations for RSVP Requests

When making a policy decision, the PDP may consider both Resv as well
as its matching Path state (associated state). State association is
straightforward in the common unicast case since the RSVP flow
includes one Path state and one Resv state. In multicast cases this
correspondence may be more complicated, as the match may be many-to-
many. The COPS protocol assumes that the PDP is RSVP knowledgeable
and capable of determining these associations based on the contents
of the Client REQ message and especially the ClientSI 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. After being approved by both, and after the reservation was
successfully installed, the PEP notifies the remote PDP by sending a
report message specifying the Commit type. The Commit type report
message signals when billing should effectively begin and performing
heavier delayed operations (e.g., debiting a credit card) is
permissible by the PDP.

If, instead, a PDP approved reservation 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 (DRQ).

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. When the PEP detects a
change in the corresponding Resv or Path message, it should update
the PDP with the new request-state. PEPs may continue to use the
cached state until receiving the PDP response. This case is very
different from initial admission of a flow; given that valid
credentials and authentication have already been established, the
relatively long RSVP refresh period, and the short PEP-PDP response
time, the tradeoff between expedient updates and attack prevention
leans toward expediency. However, this is really a PEP choice, and is
irrelevant to PDPs.

If the connection is lost between the PEP and the PDP, the cached
RSVP state may be retained for the RSVP timeout period to be used for
previously admitted flows (but cannot be applied to new or updated
state). If the connection can not be reestablished with the PDP or a
backup PDP after the timeout period, the PEP is expected to purge all
its cached decisions. Without applicable cached decision, the PEP
must either reject the flow or resort to its LPDP (if available) for
decisions.

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 in its
LPDPDecision 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 2.2). 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
   represent a substantial portion of expected COPS transactions, and
   can be guaranteed not to cause ambiguity.

   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 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 according to the latest flag. This rule applies both to
   the request (REQ) context as well as to the 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 individual
decisions apply to different sub-groups of the context, the PDP
should send each group of decision objects encapsulated by the
context flags object with the context flags applicable to these
objects set (see the examples in Section 5).

3.7 RSVP Error Reporting

RSVP uses the ERROR_SPEC object in PathErr and ResvErr messages to
report policy errors. While the contents of the ERROR_SPEC object are
defined in [RSVP,RSVP-EXT], the PDP is in the best position to
provide its contents (sub-codes). This is performed in the following
manner: First, the PEP (RSVP) queries the PDP before sending a
PathErr or ResvErr, and then the PDP returns the constructed
ERROR_SPEC in the Replacement Data Decision Object.

4 Security Considerations

This document relies on COPS for its signaling and its security.
Please refer to section "Security Considerations" in [COPS].

Security for RSVP messages is provided by inter-router MD5
authentication [MD5], assuming a chain-of-trust model. A likely
deployment scenario calls for PEPs to be deployed only at the network
edge (boundary nodes) while the core of the network (backbone)
consists of PIN nodes. In this scenario MD5 trust (authentication) is
established between boundary (non-neighboring) PEPs. Such trust can
be achieved through internal signing (integrity) of the Policy Data
object itself, which is left unmodified as it passes through PIN
nodes (see [RSVP-EXT]).

5 Illustrative Examples, Using COPS for RSVP

This section details both typical unicast and multicast scenarios.

5.1 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 Figure 1.

```
                          PEP (router)
                    +----------------+
                    |                |
          R1 -----------+if1       if2+----------- S1
                    |                |
                    +----------------+
```
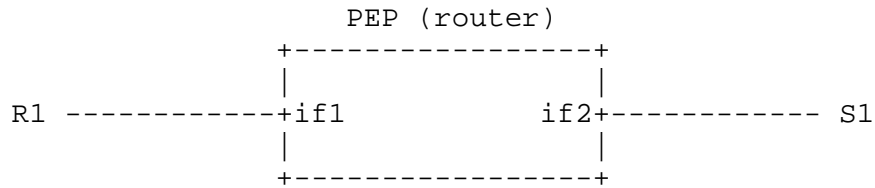
Figure 1: Unicast Example: a single PEP view

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: Command, Install>
```

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: command, Install>
                        <Context: allocation, Resv>
                        <Decision: command, Install>
                        <Decision: Stateless, Priority=7>
                        <Context: out, Resv>
                        <Decision: command, Install>
                        <Decision: replacement, 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: command, Install>
                          <Decision: Stateless, 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>
```

5.2 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 Figure 2.

```
                         PEP (router)
                    +----------------+
                    |                |
        R1------------+ if1      if3 +--------- S1
                    |                |
        R2----+     |                |
              |     |                |
              +--------+ if2      if4 +--------- S2
              |     |                |
        R3----+     +----------------+
```
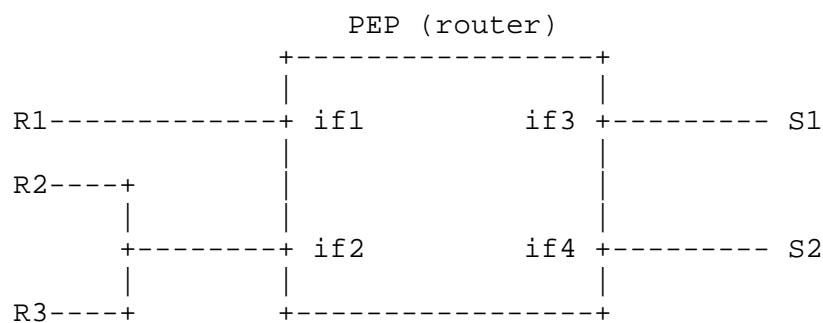
             Figure 2: Multicast example: a single PEP view

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: command, Install>
```

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: command, Install>
                              <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: command, Install>
                              <Context: allocation, Resv>
                              <Decision: command, Install>
                              <Decision: Stateless, 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: command, Install>
                         <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: command, Install>

     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: command, Install>
                         <Decision: Stateless, 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: command, Install>

      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: command, Install>
                          <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.

6 References

   [RSVP-EXT] Herzog, S., "RSVP Extensions for Policy Control", RFC
              2750, January 2000.

   [RAP]   Yavatkar, R., Pendarakis, D. and R. Guerin, "A Framework
              for Policy Based Admission Control", RFC 2753, January
              2000.

   [COPS]   Boyle, J., Cohen, R., Durham, D., Herzog, S., Raja, R. and
              A. Sastry, "The COPS (Common Open Policy Service)
              Protocol", RFC 2748, January 2000.

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

7 Author Information and Acknowledgments

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

   Jim Boyle
   Level 3 Communications
   1025 Eldorado Boulevard
   Broomfield, CO 80021

   Phone: 720.888.1192
   EMail: jboyle@Level3.net

   Ron Cohen
   CISCO Systems
   4 Maskit St.
   Herzeliya Pituach 46766 Israel

   Phone: 972.9.9700064
   EMail: ronc@cisco.com

   David Durham
   Intel
   2111 NE 25th Avenue
   Hillsboro, OR 97124

   Phone: 503.264.6232
   EMail: David.Durham@intel.com

   Raju Rajan
   AT&T Labs Research
   180 Park Ave., P.O. Box 971
   Florham Park, NJ 07932

   Phone: 973.360.7229
   EMail: raju@research.att.com

Shai Herzog
IPHighway, Inc.
55 New York Avenue
Framingham, MA 01701

Phone: 508.620.1141
EMail: herzog@iphighway.com

Arun Sastry
Cisco Systems
4 The Square
Stockley Park
Uxbridge, Middlesex UB11 1BN
UK

Phone: +44-208-756-8693
EMail: asastry@cisco.com

8 Full Copyright Statement

Acknowledgement