Internet Draft
Expiration: July 1998
File: draft-ietf-rap-cops-00.txt

Jim Boyle
MCI
Ron Cohen
Class Data Systems
David Durham
Intel
Shai Herzog
IPHighway
Raju Rajan
IBM
Arun Sastry
Cisco

The COPS (Common Open Policy Service) Protocol

Last Updated: January 20, 1998

Status of this Memo

A revised version of this draft document will be submitted to the RFC editor as a Proposed Standard for the Internet Community. Discussion and suggestions for improvement are requested. This document will expire before June 1998. Distribution of this draft is unlimited.

Abstract

This document describes a simple client/server model for supporting policy control over QoS Signaling Protocols with similar properties as ReSerVation Protocol (RSVP). It is designed to be extensible so that other kinds of policy clients may be supported in the future. The model does not make any assumptions about the decision methods of the policy server, but is based on the server returning responses to policy requests.

1. Introduction

This document describes a simple query and response protocol that can be used to exchange policy information between a policy server (Policy Decision Point or PDP) and its clients (Policy Enforcement Points or PEPs). One policy client is expected to be RSVP routers that must exercise policy-based admission control over RSVP usage [RSVP]. We assume that at least one policy server exists in each controlled administrative domain. The basic model of interaction between a policy server and its clients is compatible with the framework document for policy based admission control [WRK].

A chief objective of our proposal is to begin with a simple but extensible design. The main characteristics of the proposed protocol include:

1. The protocol employs a client/server model where the PEP sends requests, updates, and retractions to the remote PDP and the PDP returns decisions back to the PEP.

2. The protocol uses TCP as its transport protocol for reliable exchange of messages between policy clients and a server. Therefore, no additional mechanisms are necessary for reliable communication between a server and its clients.

3. The protocol is extensible in that it is designed to leverage off self-identifying objects and can support diverse client specific information. Thus, even though the protocol was created for the administration and enforcement of policies in conjunction with RSVP, the protocol may be extended for administration of other (signaling) protocols such as multicast access and network security.

4. The protocol relies on existing protocols for security.

Namely IPSEC [IPSEC] can be used to authenticate and secure the channel between the PEP and the server.

5. The protocol is stateful in two main aspects:
(1) Request/Response state is shared between client and server and (2) State from various events (Request/Response pairs) may be inter-associated. By (1) we mean that requests from the client PEP are installed or remembered by the remote PDP until they are explicitly deleted by the PEP. At the same time, Responses from the remote PDP can be generated asynchronously at any time for a currently installed request state. By (2) we mean that the server may respond to new queries differently because of previously installed, related Request/Response state (e.g., for RSVP, the server may associate state from incoming Path and Resv requests).

1.1. Basic Model

```
+----------------+
|                |
|  Network Node  |            Policy Server
|                |
|   +-----+      |   COPS        +-----+
|   | PEP |<-----|-------------->| PDP |
|   |+----+      |               +-----+
|    ^           |
|    |           |
|    \-->+-----+ |
|        | LDP | |
|        +-----+ |
|                |
+----------------+
```

Figure 1: A COPS illustration.

Figure 1 Illustrates the layout of various policy components in a typical COPS example (taken from [WRK]). Here, COPS is used to communicate policy information between a Policy Enforcement Point (PEP) and a remote Policy Decision Point (PDP).

It is assumed that each participating policy client is functionally consistent with a PEP [WRK]. The PEP may communicate with a policy server (herein referred to as a remote PDP [WRK]) to obtain policy decisions or directives.

The COPS protocol uses a single persistent TCP connection between the PEP and a remote PDP. The remote PDP listens on a well-known port number (COPS=3288), and the PEP is responsible for initiating the connection. The location of the remote PDP can either be configured, or obtained via a service location mechanism [SRVLOC].

Service discovery is outside the scope of this protocol, however.

The PEP uses the TCP connection to send requests to and receive responses from the remote PDP. Communication between the PEP and remote PDP is mainly in the form of a stateful request/response exchange, though the remote PDP may occasionally send an unsolicited response to the PEP to force a change in a previously approved request state. The PEP also has the capacity to report to the remote PDP that it has committed to an accepted request state for purposes of accounting and monitoring. Finally, the PEP is responsible for the deletion (retraction) of a request state that is no longer applicable.

The policy protocol is designed to communicate self-identifying objects which contain the data necessary for identifying request states, establishing the context for a request, identifying the type of request, referencing previously installed requests, relaying policy decisions, reporting errors, and transferring client specific information.

To distinguish between different kinds of clients, the type of client is identified in each message. Different types of clients may have different client specific data and may require different kinds of policy decisions. It is expected that each new client type will have a corresponding extensions draft specifying the specifics of its interaction with this policy protocol.

The context of each request corresponds to the policy event that triggered it. COPS identifies three types of controlled events: (1) the arrival of an incoming message (2) allocation of local resources, and (3) the forwarding of an outgoing message.
Each of these events may require different decisions to be made.
Context sub types are also defined according to the type of message that triggered the policy event. In RSVP, this subtype is used to define the RSVP signaling message type (e.g., Path, Resv, etc.). The content of a COPS request/response message depends on the context.

The PEP may also have the capability to make a local policy decision via its Local Decision Point (LDP) [WRK], however, the PDP remains the authoritative decision point at all times. This means that any local decision information must always be relayed to the PDP. That is, the PDP must be granted access to all relevant information to make a final policy decision. To facilitate this functionality, the PEP must send its local decision information to the remote PDP via a LDP decision object. The PEP must then abide by the PDP's decision as it is absolute.

Finally, fault tolerance is a required capability for this protocol, particularly due to the fact it is associated with the security and service management of distributed network devices. Fault tolerance is achieved by having both the PEP and remote PDP constantly verify their connection to each other via keep-alive messages. When a failure is detected, the PEP must try to reconnect to the remote PDP or attempt to connect to an new/alternative PDP. Once a connection is reestablished, the remote PDP may request that all the PEP's internal state be resynchronized (all previously installed requests are to be reissued). After failure and before the new connection is fully functional, disruption of service can be minimized if the PEP caches previously communicated decisions and continues to use them for some limited amount of time. (Discussions of specific provisions for such a mechanism are outside of the scope of this draft, and are left to client specific implementations).

2. The Protocol

This section describes the message formats and objects exchanged between the PEP and remote PDP.

2.1 Common Header

Each COPS message consists of the COPS header followed by a number of typed objects.

```
       0              1              2              3
+--------------+--------------+--------------+--------------+
|Version|XXXXXX|    Op Code   |       Client Type           |
+--------------+--------------+--------------+--------------+
|                      Message Length                       |
+--------------+--------------+--------------+--------------+
```

The fields in the header are:
  Version: 4 bits
      COPS version number. Current version is 1.

Op Code: 8 bits
     The COPS operations:
       1 = Request   (REQ)
       2 = Response   (RES)
       3 = Unsolicited Response   (USR)
       4 = Report State   (RPT)
       5 = Delete Request State   (DRQ)
       6 = Synchronize State Req   (SSQ)
       7 = Client-Open   (OPN)
       8 = Client-Accept   (CAT)
       9 = Keep Alive   (KA)

Client Type: 16 bits

The Client Type identifies the policy client. Interpretation of all encapsulated objects is relative to the client type. (See Appendix A for the RSVPv1 client type ID).

Message Length: 32 bits
Size of message in octets, which includes the standard COPS header and all encapsulated objects. Messages must be aligned on 4 octet intervals.

2.2 COPS Specific Object Formats

All the objects follow the same object format; each object consists of one or more 32-bit words with a four octet header, using the following format:

```
       0             1              2             3
+-------------+-------------+-------------+-------------+
|       Length (octets)     |    C-Num    |   C-Type    |
+-------------+-------------+-------------+-------------+
|                                                       |
//                  (Object contents)                   //
|                                                       |
+-------------+-------------+-------------+-------------+
```

Typically, C-Num identifies the class of information contained in the object, and the C-Type identifies the subtype or version of the information contained in the object.

```
   C-num: 8 bits

            1  = Handle
            2  = Handle Reference.
            3  = Context
            4  = In Interface
            5  = Out Interface
            6  = Reason code
            7  = Decision
            8  = LDP Decision
            9  = Protocol Error
            10 = Client Specific Info
            11 = Timer
            12 = PEP Identification
            13 = Report Type

   C-type: 8 bits
            Values defined per C-num.
```

2.2.1 Handle Object (Handle)

Unique value that identifies an installed request state. This identification is used by most COPS operations. The request state corresponding to this handle must be explicitly deleted by the client when no longer applicable.

The handle value is set by the PEP and is opaque to the PDP. The PDP performs a byte-wise comparison on the value in this object with respect to the handle object values for other currently installed requests.

C-Num = 1, C-Type = 1

Variable-length field, no implied format.

2.2.2 Handle Reference Object (HandleRef)

Same C-Type formats as the handle object. This object may appear in requests and is used to associate the current request to previously installed request states. The presence of a reference handle in a request message tells the PDP that it should also consider information in the referenced installed state when making a policy decision for the current request. Handle References are only used for the specific client types that mandate them.

C-num = 2, C-Type = (same as handle object)

2.2.3 Context Object (Context)

Specifies the type of event(s) that triggered the query. Required for request messages.

```
        C-num = 3, C-Type = 1
           0             1               2               3
    +--------------+--------------+--------------+--------------+
    |            R-Type           |            M-Type           |
    +--------------+--------------+--------------+--------------+
```

R-Type (Request Type Flag)

0x01 = Incoming-Message/Admission Control request
0x02 = Resource-Allocation request
0x04 = Outgoing-Message request
0x08 = Configuration request

M-Type (Message Type)

Client Specific 16 bit values of protocol message types

2.2.4 In-Interface Object (IN-Int)

The In-Interface Object is used to identify the incoming interface on which a particular request/response applies. For flows or messages generated from the PEP's local host, the loop back address is used.

Note: In-Interface is typically relative to the flow of the underlying protocol messages. That is, the In-Interface is the interface on which the protocol message was received.

C-Num = 4

```
       C-Type = 1, IPv4 Address
           0             1              2             3
   +--------------+--------------+--------------+--------------+
   |                   IPv4 Address format                     |
   +--------------+--------------+--------------+--------------+

       C-Type = 2, IPv6 Address
           0             1              2             3
   +--------------+--------------+--------------+--------------+
   |                                                           |
   +                                                           +
   |                                                           |
   +                    IPv6 Address format                    +
   |                                                           |
   +                                                           +
   |                                                           |
   +--------------+--------------+--------------+--------------+

       C-Type = 3, Ifindex value
               0             1              2             3
   +--------------+--------------+--------------+--------------+
   |                          ifindex                          |
   +--------------+--------------+--------------+--------------+
```

Ifindex may be used to differ between sub-interfaces and unnumbered interfaces (see RSVP's LIH for an example). When appropriate, this ifindex integer should correspond to the same integer value for the interface in the SNMP MIB-II interface index table.

2.2.5 Out-Interface Object (OUT-Int)

The Out-Interface is used to identify the outgoing interface to which a specific request/response applies. It has the same format as the In-Interface Object.

    C-Num = 5, C-Type = (same C-Type as for In-Interface)

Note: In-Interface is typically relative to the flow of the underlying protocol messages. That is, the Out-Interface is the one on which a protocol message is about to be forwarded.

2.2.6 Reason Object (Reason)

This object specifies the reason why the request state was deleted. It should appear in the delete request (DRQ) message.

C-Num = 6, C-Type = 1

```
        0             1              2             3
+--------------+--------------+--------------+--------------+
|         Reason-Code         |       Reason Sub-code       |
+--------------+--------------+--------------+--------------+
```

Reason Code:
- 1 = Unknown
- 2 = Management
- 3 = Preempted
- 4 = Tear
- 5 = Timeout
- 6 = Route Change
- 7 = Insufficient Resources
- 8 = PDP's Directive
- 9 = Client Specific (details in sub-code)

2.2.7 Decision Object (Decision)

Decision made by the PDP. Must appear in replies. The specific decision objects required in a response to a particular request depend on the type of client.

C-Num = 7

CType = 1, Decision Flags (mandatory!)

```
        0             1              2             3
+--------------+--------------+--------------+--------------+
|                            Flags                          |
+--------------+--------------+--------------+--------------+
```

Flags:
- 0x01 = Reject Incoming (Reject if set)
- 0x02 = Do Not Allocate Resources (Reject if set)
- 0x04 = Drop Outgoing (do not forward message if set)
- 0x08 = Trigger Error (Trigger error message if set)

Ctype = 2, Resource Allocation Data (optional)

It is expected that PEPs would be able to configure simple stateless policy information to be processed locally in their LDP. As this set is well known and implemented ubiquitously, PDPs are aware of it as well (either universally, through configuration, or using the Client-Open message). The PDP may also include this information in its response, and the PEP should apply it to the resource allocation event that generated the request.

Examples of resource allocation information that can be found in other documents are:

Preemption Priority

Priority is used by PEP to decide which of the flows should be preempted, when not enough resources are available on the interface. For RSVP, when preemption is supported, a higher priority reservation can preempt an installed reservation with lower priority.

CType = 3, Replacement Data (Optional)

Format includes a list of client specific data that is to be used in place of information specified in the request. Use of this decision type is optional. For RSVP, this decision is used to change objects carried in RSVP messages. For example, replacing the policy data objects when forwarding a Resv message upstream is possible due to this decision type. If this decision doesn't appear in a response, all objects are passed as if the PDP was not there. To remove an object the decision should carry an empty object of length 4 (header only). Appendix A specifies the list of RSVP objects that can be replaced.

CType = 4, Client Specific Decision Data (Optional)

Proprietary decision types can be introduced using the Client Data Decision Object. Like the Replacement Data object, client specific information is encapsulated within the Client Data Object.

2.2.8 LDP Decision Object (LDPDecision)

Decision made by the PEP's local decision point (LDP). May appear in requests. These objects correspond to and are formatted the same as the client specific decision objects defined above.

C-Num = 8

CType = (same C-Type as for Decision object)

2.2.9 Error Object (Error)

This object is used to identify a particular COPS protocol error.

C-Num 9, C-Type = 1

```
        0             1              2             3
+--------------+--------------+--------------+--------------+
|          Error-Code         |        Error Sub-code       |
```

```
      +--------------+--------------+--------------+--------------+
```

Error-Code:

1 = Bad handle
2 = Invalid handle reference
3 = Bad message format
4 = Unable to process (server gives up on query)
5 = Mandatory client-specific info missing
6 = Unsupported client type
7 = Mandatory COPS object missing

2.2.10 Client Specific Information Object (ClientSI)

All objects specific to a client's signaling protocol must be encapsulated within one or more Client Information Objects.

Class-Num = 10, C-Type = 1

Variable-length field. The format of the data encapsulated in the ClientSI object is determined by the client type.

2.2.11 Timer Object (Timer)

Times are encoded as 32-bit integer values and are in units of seconds. The time value is treated as a delta.

Class-Num = 11, C-Type = 1 (keep-alive timer value)

```
         0              1              2              3
+--------------+--------------+--------------+--------------+
|                        Timer Value                        |
+--------------+--------------+--------------+--------------+
```

2.2.12 PEP Identification Object (PEPID)

The PEP Identification Object is used to identify the PEP client to the remote PDP. It is required for Client-Open messages.

C-Num = 12, C-Type = 1

Variable-length field (zero padded ASCII symbolic name) configured by local administrators for the PEP. For example, it can be the PEP's main IP address (not to be confused with the actual IP address used in the persistent TCP connection). It may also be the PEP's DNS name, or any other symbol that uniquely identifies each PEP within the policy domain. The choice of configuration bears no significance to the COPS protocol. By default, at least the primary IP address of the PEP represented as a string is expected in the PEPID.

2.2.13 Report-Type Object (Report-Type)

The Type of Report on the request state associated with a handle:

C-Num = 13, C-Type = 1

```
          0             1              2             3
    +--------------+--------------+--------------+--------------+
    |         Report-Type         |        XXXXXXXXXXXXX        |
    +--------------+--------------+--------------+--------------+
```

Report-Type:
 1 = Commit   : State was installed on client (PEP)
 2 = Accounting: Accounting update for an installed state

3. Message Content

This section describes the basic messages exchanged between a PEP and a remote PDP as well as their contents.

3.1 Request (REQ) PEP -> PDP

The PEP establishes a request state handle for which the remote PDP may maintain a state. The remote PDP then uses this handle to refer to the exchanged information and decisions.

Once a stateful handle is established for a new request, any subsequent modifications of the request can be made using the REQ message specifying the previously installed handle.

The format of the Request message is as follows:

```
<Request> ::= <Common Header>
              <Handle>
              <Context>
              [<IN-Int>]
              [<OUT-Int>]
              <ClientSI>
              [<list of HandleRefs>]
              [<LDPDecision>]
```

The context object is used to determine the context within which all the other objects are to be interpreted. It also is used to determine the kind of response to be returned from the policy server. This response might be related to admission control, resource allocation, or object forwarding and substitution.

The interface objects are used to determine the corresponding interface on which a signaling protocol message was received or is about to be sent. They are only used if the client is participating along the path of a signaling protocol.

ClientSI, the client specific information object holds the client type specific data for which a policy decision needs to be made.

The handle reference objects are used to refer to state information currently installed on the PDP that is associated with the current request.

Finally, LDPDecision object holds information regarding the local decision made by the LDP.

3.2 **Response (RES)** PDP -> PEP

The PDP responds to the REQ with a RES message that includes the associated handle and the decision. If there was a protocol error an error object is returned instead.

In order to avoid the issue of keeping track of which Request a particular response belongs to, it is important that, for a given handle, there be at most one outstanding response per query. This essentially means that the PEP should not issue more than one REQ(for a given handle) before it receives a corresponding RES. To avoid deadlock, the client can always timeout after issuing a request. It can then delete the timed-out handle, and try again using a different (new) one.

The format of the Response message is as follows:

```
<Response> ::= <Common Header>
               <Handle>
               <Decision(s)> || <Error>
```

The response may include either an Error object or decision object(s). COPS protocol problems are reported in the Error object (e.g. an error with the format of the original request). Decision object(s) depend on the context of the associated request and the type of client.

3.3 **Unsolicited Response (USR)** PDP -> PEP

The remote PDP can also send an unsolicited response to a PEP to report a different response than the one previously communicated. For example, the PDP may admit a new flow and change its mind to reject it sometime later. The change of mind is communicated using the USR message.

The format for an USR is the same as that for a RES and similarly, it dependents on the context of the original request.

3.4 **Report State (RPT)** PEP -> PDP

This message is used by the PEP to communicate the change in status of a previously installed request state to the server. A commit report indicates to the PDP that a particular policy directive has been acted upon. (In RSVP this would mean that the reservation successfully passed capacity admission control).

The Report State may also be used to provide periodic updates of client specific information for accounting and state monitoring purposes depending on the type of the client. In such cases the accounting report type should be specified utilizing the client specific information object.

```
<Report State> ::== <Common Header>
                    <Handle>
                    <Report-Type>
                    [ <Client Specific Information> ]
```

3.5 **Delete Request State (DRQ)** PEP -> PDP

This message indicates to the remote PDP that the request state must be deleted. This will be used by the remote PDP to initiate the appropriate housekeeping actions. The reason code object is interpreted with respect to the client type.

The format of the Delete Request State message is as follows:

```
<Delete Request>  ::= <Common Header>
                      <Handle>
                      <Reason>
```

3.6 **Synchronize State Request (SSQ)** PDP -> PEP

The format of the Synchronize State Query message is as follows:

```
<Synchronize State> ::= <Common Header>
                        [<Handle>]
```

This message indicates that the remote PDP wishes the client (which appears in the common header) to re-send its state. If the optional Handle is present, only the state associated with this handle is synchronized. Otherwise, all the client state should be synchronized with the PDP.

The client performs state synchronization by re-issuing request queries of the specified client type for the existing state in the PEP.

3.7 **Client-Open (OPN)** PEP -> PDP

The Client-Open message can be used to provide the characteristics of the connection, suggested time intervals for the keep-alive messages, and information on the locally known policy elements.

```
<Client-Open>  ::= <Common Header>
                   <PEPID>
                   [<Timer>]
```

The PEPID is a symbolic, variable length name that identifies the specific client to the PDP. Values for the PEPID are configurable by administrators of administrative domains and are of direct significance to the COPS protocol. By default, the PEPID specifies the primary IP address in the form of a string for the PEP in question.

If included, the timer corresponds to PEP's preference for the maximum intermediate time between the generation of messages for connection verification.

3.8 **Client-Accept (CAT)** PDP -> PEP

The Client-Accept message is used to respond to the Client-Open message. This message will return to the PEP either a timer object indicating the expected time interval between keep-alive messages, or an error object indicating that an error occurred (e.g. requested client type is not supported by the remote PDP).

```
<Client-Accept>  ::= <Common Header>
                     <Timer> || <Error>
```

If the PDP refuses the client, it will return an Error object to describe the reason.

The timer corresponds to maximum acceptable intermediate time between the generation of messages by the PDP and PEP. The timer value is determined by the PDP taking into account the client's preference established with the OPN message. A timer value of 0xFFFFFFFF implies no secondary connection verification is necessary.

3.9 **Keep-Alive (KA)** PEP -> PDP, PDP -> PEP

The keep-alive message only needs to be transmitted when there has been no activity between the client and server for a period approaching half that of the minimum timer value negotiated with the OPN & CAT messages. It is a validation for each side that the other is still functioning.

```
<Keep-Alive>  ::= <Common Header>
```

Both client and server may assume the connection is insufficient for the client type with the minimum time value (specified in the CAT message) if no communication activity is detected for a period exceeding the timer period. For the PEP, such detection implies the remote PDP or connection is down and the PEP should now attempt to use an alternative/backup PDP.

4. Common Operation

This section describes the typical exchanges between remote PDP servers and PEP clients.

After a connection is established between the PEP and a remote PDP, the PEP will send one or more Client-Open messages to the remote PDP, one for each client type supported by the PEP. The open message should contain the common header noting one client type supported by the PEP. The remote PDP will then respond with a Client-Accept message echoing back each of the client types the PEP supports that it can support as well. If a specific client type is not supported by the PDP, the corresponding Client-Accept message sent back to the PEP will include an error object specifying the client type is not supported. The PDP will include the timer interval between keep-alive messages in its Client-Accept.

When the PEP receives an event that requires a new policy decision it sends a request message to the remote PDP. The remote PDP then makes a decision and sends a response back to the PEP. Since the request is stateful, the request will be remembered, or installed, on the remote PDP. The unique handle, specified in both the request and its corresponding response identifies this request state. The PEP is responsible for deleting this request state once the request is no longer applicable.

The PEP may update a previously installed request state by reissuing a request for the previously installed handle. The remote PDP is then expected to make new decisions and send a response back to the PEP. Likewise, the server may change a previously issued decision on any currently installed request state at any time by issuing an asynchronous response. At all times the PEP module is expected to abide by the PDP's decisions.

The PEP may also notify the remote PDP of the local status of an installed request using the report message where appropriate. The report message is to be used to signify when billing should effectively begin, or to produce periodic updates for monitoring and accounting purposes depending on the client. This message can carry client specific information when needed.

Finally, to validate the connection between the client and server is still functioning, the keep-alive message is used. If no COPS message is generated within one half the minimum timer value interval, a keep-alive message needs to be generated. Both the PEP and remote PDP are expected to follow this procedure.

5. Security

The security of RSVP messages is provided by inter-router MD5 authentication [MD5]. This assumes a chain-of-trust model for inter PEP authentication. Security between the client (PEP) and server (PDP) is provided by IPSEC [IPSEC].

To ensure the client (PEP) is communicating with the correct policy server (PDP) involves two issues: authentication of the policy client and server using a shared secret, and consistent proof that the connection remains valid. The shared secret requires manual configuration of keys, which is a maintenance issue. IPSEC AH may be used for the validation of the connection; IPSEC ESP may be used to provide both validation and secrecy.

6. Open issues

6.1 Bi-directional Connection Establishment:

Currently, only the PEP is supposed to connect with the PDP. It might be useful to have the PDP proactive in establishing connections with its PEPs. Such would potentially simplify PEP configuration and allow a primary PDP that has failed to notify its clients that it is functional again.

6.2 Client Type Close/Redirect:

Is there a need for a Close message per client type so the PEP and PDP can notify each other in case of a capability change? If there is a close, should the PDP be able to tell the PEP which PDP server it should now use (redirect)?

6.3 Division of Labor Negotiation:

How can (and is there a need for) the PEP to notify the remote PDP of its LDP's capabilities (e.g. the LDP can directly authenticate user information)?

6.4 Group ID:

Is there a need for a Group ID for identifying the group a client belongs akin to how the PEPID identifies an individual client?

6.5 RSVP Object Replacement:

Should the PDP be capable of directing the RSVP PEP to replace other objects than the Policy Data object (e.g. FlowSpec)? If so, for which request types?

6.6 RSVP Priority Element Definition (other work).

7. References

[RSVP] Braden, R. ed. et al., "Resource ReSerVation Protocol (RSVP) Version 1 - Functional Specification", RFC 2205, September 1997.

[WRK] Yavatkar, R. et al., "A Framework for Policy-Based Admission Control", Internet-Draft, draft-ietf-rap-framework-00.txt, November 1997.

[SRVLOC] Guttman, E. et al., "Service Location Protocol", Internet-Draft, draft-ietf-svrloc-protocol-v2-01.txt, October 1997.

[INSCH] Shenker, S., Wroclawski, J., "General Characterization Parameters for Integrated Service Network Elements", RFC 2215, September 1997.

[IPSEC] Atkinson, R., "Security Architecture for the Internet Protocol", RFC1825, August 1995.

[MD5] Baker, F., "RSVP Cryptographic Authentication", Internet-Draft, draft-ietf-rsvp-md5-05.txt, August 1997.

[RSVPPR] Braden, R., Zhang, L., "Resource ReSerVation Protocol (RSVP) - Version 1 Message Processing Rules", RFC 2209, September 1997.

8. Author Information and Acknowledgments

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

Jim Boyle
MCI
2100 Reston Parkway
Reston, VA 20191
703.715.7006
jboyle@mci.net

Ron Cohen
Class Data Systems
13 Hasadna St.
Ra'anana 43650 Israel
972.9.7462020
ronc@classdata.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
2055 Gateway Pl., Suite 400
San Jose, CA 95110
408.390.3045
herzog@iphighway.com

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

Appendix A. COPS Extensions for Use with RSVP

A.1 Overview of COPS extensions for RSVP

Building on the foundations described in the previous sections this section describes the specific functionality required for this protocol to support policy control over RSVP.

Setting the client type in the COPS common header to 1 indicates an RSVP client capable of performing admission control and policy data substitution using RSVP V1 objects.

A.2 COPS objects for use with RSVP

The COPS objects defined in section 2.2 are applicable to RSVP policy control, and their use is described in the text that follows.

The message type information found in the RSVP message header is represented by the M-Type in the COPS Context Object.

All objects contained within RSVP messaging are expected to be encapsulated in the Client Specific Information Object without alteration. Multiple RSVP objects may be contained within a single Client Specific Information Object exchanged between the PEP and remote PDP.

Finally, for the COPS outgoing message responses, RSVP objects may be returned to the PEP from the remote PDP via the Replacement Data Decision Object. This object may contain multiple RSVP objects, but is primarily concerned with returning the Policy Data object. Objects included in the Replace Data Decision Object are to replace their corresponding object in the RSVP message (typically for outgoing RSVP messages).

A.3. Operation of COPS for Policy Control Over RSVP

A.3.1 RSVP values for the 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 Previous 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 = PathErr

Note: At this point, PathTear, ResvTear, and the Resv Confirm message types are not supported.

A.3.2 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.

A.3.4 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.

A.3.5 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 notify the PDP by issuing a delete message.

A.3.6 Policy Control Over Path and Resv Tear

Path and Resv Tear 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 Resv or Path Tear is received at the PEP, all affected states installed on the PDP should either be deleted or updated by the PEP.

A.3.7 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.

A.3.8 Data Expected in Request Messages for RSVP Support

The information required in a RSVP request for each applicable message type and request type combination is outlined below:

```
In, Path -
    <handle><context: in, Path><in-interface>
    <client info: all objects in Path message>
Out, Path -
    <handle><context: out, Path><out-interface>
    <client info: all objects in outgoing Path message>
In & Out (unicast combined request), Path -
    <handle><context: in & out, Path><in-interface>
    <out-interface>
    <client info: all objects in Path message>

In, Resv -
    <handle><context: in, Resv><in-interface>
    <client info: all objects in Resv message>
Merge, Resv -
    <handle><context: merge, Resv><in-interface>
    <client info: all objects in merged Resv message including
     the merged FLOWSPEC object>
Out, Resv -
    <handle><context: out, Resv><out-interface>
    <client info: all objects in outgoing Resv message>
In & Merge (combined request, PEP can merge), Resv -
    <handle><context: in & merge, Resv><in-interface>
    <client info: all objects in Resv message>
In & Merge & Out (unicast combined request), Resv -
    <handle><context: in & merge & out, Resv><in-interface>
    <out-interface>
    <client info: all objects in Resv message>

In, PathErr -
    <handle><context: in, PathErr><in-interface>
    <client info: all objects in PathErr message>
Out, PathErr -
    <handle><context: out, PathErr><out-interface>
    <client info: all objects in outgoing PathErr message>
In & Out (unicast combined request), PathErr -
    <handle><context: in & out, PathErr><in-interface>
    <out-interface>
    <client info: all objects in PathErr message>


In, ResvErr -
    <handle><context: in, ResvErr><in-interface>
```

```
        <client info: all objects in ResvErr message>
    Out, ResvErr -
        <handle><context: out, ResvErr><out-interface>
        <client info: all objects in outgoing ResvErr message>
    In & Out (unicast combined request), ResvErr
```

```
          <handle><context: in & out, ResvErr><in-interface>
          <out-interface>
          <client info: all objects in ResvErr message>
```

A.3.9 Expected Decisions for RSVP Requests

The expected decision information relative to a request for each applicable message type and request type combination is outlined below:

```
      In, Path -
          <handle><Decision Flags>
      Out, Path -
          <handle><Decision Flags><Decision Replacement: policy data>
      In & Out (combined request), Path -
          <handle><Decision Flags><Decision Replacement: policy data>

      In, Resv -
          <handle><Decision Flags>
      Merge, Resv -
          <handle><Decision Flags><Decision Priority>
      Out, Resv -
          <handle><Decision Flags><Decision Replacement: policy data>
      In & Merge (combined request, PEP can merge), Resv -
          <handle><Decision Flags><Decision Priority>
      In & Merge & Out (unicast combined request), Resv -
          <handle><Decision Flags><Decision Priority>
          <Decision Replacement: policy data>

      In, PathErr -
          <handle><Decision Flags>
      Out, PathErr -
          <handle><Decision Flags><Decision Replacement: policy data>
      In & Out (combined request), PathErr -
          <handle><Decision Flags><Decision Replacement: policy data>

      In, ResvErr -
          <handle><Decision Flags>
      Out, ResvErr -
          <handle><Decision Flags><Decision Replacement: policy data>
      In & Out (combined request), ResvErr -
          <handle><Decision Flags><Decision Replacement: policy data>
```

A.4 Illustrative Examples, Using COPS for RSVP

A.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.

```
                         PEP (router)
                      +-----------------+
                      |                 |
       R1 ------------+if1           if3+------------ S1
                      |       if2       |
                      +--------+--------+
                               |
                               |
                              PDP (server)
```

figure 1: Unicast Example: a single router view

The PEP router has three interfaces (1,2,3). Sender S1 sends to receiver R1.

A Path message arrives from S1:

```
PEP --> PDP   REQ := <Handle A><Context in&out, Path>
                     <In-Interface if3> <Out-Interface if1>
                     <ClientSI: all objects in Path message>

PDP --> PEP   RES := <Handle A><Decision accept>
```

A Resv message arrives from R1:

```
PEP --> PDP   REQ := <Handle B><Context in&merge&out, Resv>
                     <In-Interface if1> <Out-Interface if3>
                     <ClientSI: all objects in Resv message>

PDP --> PEP   RES := <Handle B>
                     <Decisions: accept, Priority=7,
                      Replace: POLICY.DATA1>

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

Time Passes, the PDP changes its decision:

```
PDP --> PEP   USR := <Handle B>
```

<Decisions: accept, Priority=3,
 Replace: POLICY.DATA2>

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>
```

A.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.

```
                             -----------------
                  r1        |                 |
            H1-------------|i1               |      r4
                            |              o1 |---------------- S1
                  r2        |     Router      |
            H2 ------------|i2               |
                  |         |              o2 |---------------- S2
                  | r3      |                 |
                  |          -----------------
                 H3
```

figure 1: 2 senders and 3 receivers

Figure 1 shows an RSVP router which has two senders and three receivers for the same multicast session. Interface i2 is connected to a shared media.

First detailed is the request message content for a Path sent by sender S1, assuming that both receivers have already joined the multicast session, but haven't sent a Resv message as yet. Assume sender S2 has not yet sent a path message. The Path message arrives on interface o1:

```
    PEP -----> PDP    REQ   := <handle A><context in, Path>
                               <in-interface o1><client info: all
                               objects in Path message>
    PDP -----> PEP    RES   := <handle A><Decision accept>
```

Here the PDP decides to allow the Path message. Next, the Router consults its forwarding table, and finds two outgoing interfaces, i1 and i2, for the path. The exchange below is for interface i1, another exchange would likewise be completed for i2 using the new handle B2.

```
PEP -----> PDP    REQ   := <handle B1><context out, Path>
                           <out-interface i1><client info: all
                           objects in outgoing Path message>
```

```
PDP -----> PEP    RES   := <handle B1><Decision forward>
                           <Decision replacement object:
                            policy object>
```

Here, the PDP decided to allow the forwarding of the Path message via interface i1, and determined the appropriate policy objects for the message going out on this interface.

Next, the receiver r2 sends a Resv message of WF style. The Resv arrives on interface i2. Here the PEP queries the PDP which decides to accept this reservation with priority 5 as shown below.

```
PEP -----> PDP    REQ   := <handle C><context in, Resv>
                           <in-interface i2><client info: all
                           objects in Resv message>
PDP -----> PEP    RES   := <handle C><Decision accept>
```

This assumes the PEP is not itself capable of merging priority information, and, thus, must make another query for the incoming interface merge.

```
PEP -----> PDP    REQ   := <handle D><context merge, Resv>
                           <in-interface i2><client info: all
                           objects in merged Resv message>
PDP -----> PEP    RES   := <handle D><Decision Priority: 5>
```

After PEP successfully admitted the reservation it sends a report message that signals to the PDP that it can start an accounting log for this reservation.

```
PEP -----> PDP    RPT   := <handle D>
                           <commit>
```

The reservation r2 needs to be sent upstream towards sender S1 out interface o1. An outgoing Resv request is made which carries the associated handle of the Path message for which this Resv is being forwarded.

```
PEP -----> PDP    REQ   := <handle E><context out,Resv>
                           <out-interface o1><client info: all
                           objects in outgoing Resv message>
PDP -----> PEP    RES   := <handle E><Decision forward><Decision
                           replacement object: policy object>
```

Next, receiver H3 sends the Resv message r3. The PEP sends an incoming request for handle F and the PDP decides to accept the Resv (as before). The new reservation also requires the PEP to update the merged request (handle D) due to the modified flowspec. The PDP now gives this request priority 7. If accepted by local admission control, a report is again sent.

```
PEP -----> PDP    REQ   := <handle D><context merge, Resv>
                           <in-interface i2><client info: all
```

```
                                 objects in merged Resv message w/
                                 new merged FLOWSPEC>
       PDP -----> PEP    RES   := <handle D><Decision priority 7>
       PEP -----> PDP    RPT   := <handle D>
                                 <commit>
```

Now the outgoing request for handle E is reissued for the merged (R2 & R3) outgoing Resv to be sent towards sender S1 due to a modified flowspec.

```
       PEP -----> PDP    REQ   := <handle E><context out,Resv>
                                 <out-interface o1><client info: all
                                 objects in outgoing Resv message w/
                                 new merged FLOWSPEC>
       PDP -----> PEP    RES   := <handle E><Decision forward><Decision
                                 replacement object: policy object>
```

When S2 joins the session by sending a Path message, incoming and outgoing Path requests are issued for the new Path. The two incoming Resv requests may then be reissued for handle C and handle E if there is a change in their shared sender filter list (for SE filters) specifying the new sender. A new outgoing Resv request would then be issued for the Resv to be sent to s2 out interface o2.