Internet Draft
Expiration: February 1999
File: draft-ietf-rap-cops-02.txt

Jim Boyle
Level 3
Ron Cohen
Cisco
David Durham
Intel
Shai Herzog
IPHighway
Raju Rajan
IBM
Arun Sastry
Cisco

The COPS (Common Open Policy Service) Protocol

Last Updated: August 6, 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 February 1999. Distribution of this draft is unlimited.

Abstract

This document describes a simple client/server model for supporting policy control over QoS Signaling Protocols and provisioned QoS resource management. 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 methods of the policy server, but is based on the server returning decisions 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 example of a policy client is 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 policy control protocol is to begin with a simple but extensible design. The main characteristics of the COPS protocol include:

1. The protocol employs a client/server model where the PEP sends requests, updates, and deletes 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 without requiring modifications to the COPS protocol itself. The protocol was created for the general administration, configuration, and enforcement of policies whether signaled or provisioned. The protocol may be extended for the administration of a variety of signaling protocols as well as policy configuration on a device.

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/Decision state is shared between client and server
and (2) State from various events (Request/Decision 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,
Decisions 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 Request/Decision state(s) that are
related.

6. Additionally, the protocol is stateful in that it allows the
server to push configuration information to the client, and then
allows the server to remove such state from the client when it
is no longer applicable.

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) within the context of
a particular type of client.

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 PEP uses a TCP connection to send requests to and receive decisions from the remote PDP. Communication between the PEP and remote PDP is mainly in the form of a stateful request/decision exchange, though the remote PDP may occasionally send unsolicited decisions to the PEP to force changes in previously approved request states. 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. The PEP is responsible for notifying the PDP when a request state has changed on the PEP. Finally, the PEP is responsible for the deletion of any state that is no longer applicable due to events at the client or decisions issued by the server.

When the PEP sends a configuration request, it expects the PDP to continuously send named units of configuration data to the PEP via decision messages as applicable for the configuration request. When a unit of named configuration data is successfully installed on the PEP, the PEP should send a report message to the PDP confirming the installation. The server may then update or remove the named configuration information via a new decision message. When the PDP sends a decision to remove named configuration data from the PEP, the PEP will delete the specified configuration and send a report message to the PDP as confirmation.

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/name space 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 usage draft specifying the specifics of its interaction with this policy protocol.

The context of each request corresponds to the type of event that triggered it. COPS identifies three types of outsourcing 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 available to describe the type of message that triggered the policy event. The content of a COPS request/decision message depends on the context. A forth type of request is useful for types of clients that wish to receive configuration information from the PDP. This allows a PEP to issue a configuration request for a specific named device or module that requires configuration information to be installed.

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 the relevant local decision information must 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 a new/alternative PDP. While disconnected, the PEP should revert to making local decisions. Once a connection is reestablished, the PEP is expected to notify the PDP of any events that passed local admission control after the connection was lost. Additionally, 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, typically in the order of minutes. (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| //// |    Op Code   |       Client-type           |
+--------------+--------------+--------------+--------------+
|                      Message Length                       |
+--------------+--------------+--------------+--------------+
```

Global note: //// implies field is reserved, set to 0.

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 = Decision	(DEC)
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)
10= Client-Close	(CC)
11= Synchronize Complete	(SSC)

Client-type: 16 bits

The Client-type identifies the policy client. Interpretation of all encapsulated objects is relative to the client-type. Client-types that set the most significant bit in the client-type field are enterprise specific (these are client-types 0x8000 - 0xFFFF). (See the specific client usage documents for particular client-type IDs).

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)                   //
    |                                                       |
    +-------------+-------------+-------------+-------------+
```

The length is a two-octet value that describes the number of octets (including the header) that compose the object. If the length in octets does not fall on a 32-bit word boundary, padding must be added to the end of the object so that it is aligned to the next 32-bit boundary before the object can be sent on the wire. On the receiving side, a subsequent object boundary can be found by simply rounding up the previous stated object length to the first 32-bit boundary.

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
         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
         14 = PDP Address
```

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

2.2.1 Handle Object (Handle)

The Handle Object encapsulates a unique value that identifies an installed state. This identification is used by most COPS operations. A state corresponding to a handle must be explicitly deleted when it is no longer applicable.

C-Num = 1

C-Type = 1, Client Handle.

Variable-length field, no implied format other than it is unique from other client handles. It is always initially chosen by the PEP and then deleted by the PEP when no longer applicable. The client handle is used to refer to a request state initiated by the PEP and installed at the PDP. A PEP will specify a client handle in its Request messages, Report messages and Delete messages sent to the PDP. In all cases, the client handle is used to uniquely identify the PEP request.

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

2.2.2 Context Object (Context)

Specifies the type of event(s) that triggered the query. Required for request messages. Admission control, resource allocation, and forwarding requests are all amenable to client-types that outsource their decision making facility to the PDP. For applicable client-types a PEP can also make a request to receive named configuration information from the PDP. This named configuration data may be in a form useful for setting system attributes on a PEP, or it may be in the form of policy rules that are to be directly verified by the PEP.

Multiple flags can be set for the same request. This is only allowed, however, if the set of client specific information in the combined request is identical to the client specific information that would be specified if individual requests were made for each specified flag.

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

2.2.3 In-Interface Object (IN-Int)

The In-Interface Object is used to identify the incoming interface on which a particular request/decision 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.

2.2.4 Out-Interface Object (OUT-Int)

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

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

Note: Out-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.

2.2.5 Reason Object (Reason)

This object specifies the reason why the request state was deleted. It should appear in the delete request (DRQ) message. The Reason Sub-code field is reserved for more detailed client-specific reason codes defined in the corresponding documents.

C-Num = 6, C-Type = 1

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

Reason Code:
1 = Unspecified
2 = Management
3 = Preempted
4 = Tear
5 = Timeout
6 = Route Change
7 = Insufficient Resources
8 = PDP's Directive
9 = Unsupported decision
10= Synchronize Handle Unknown
11= Transient Handle (stateless event)

2.2.6 Decision Object (Decision)

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

C-Num = 7

CType = 1, Decision Flags (Mandatory)

A flag bit set to 1 implies a negative decision for that flag. Not setting any flags generally implies a positive decision. Flag values not applicable to a given request type MUST be ignored by the PEP.

```
        0             1              2             3
```

```
       +--------------+--------------+--------------+--------------+
       |                           Flags                           |
       +--------------+--------------+--------------+--------------+
```

Flags:

0x01 =	Allow Incoming (Reject if set)
0x02 =	Allocate Resources (Reject if set)
0x04 =	Forward Outgoing (do not forward if set)
0x08 =	Trigger Error (Trigger error message if set)
0x10 =	NULL Configuration (No configuration data if set)
0x20 =	Install Configuration (Nothing to install if set)
0x40 =	Remove Configuration (Nothing to remove if set)
0x80 =	Enable Configuration (Nothing to enable if set)
0x100=	Disable Configuration (Nothing to disable if set)
0x200=	Solicited Decision (Initial decision after a new/updated request if set)

Ctype = 2, Resource Allocation Data

It is expected that even outsourcing PEPs will be able to make some simple stateless policy decisions 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 decision, and the PEP should apply it to the resource allocation event that generated the request.

As an example, reservations may be admitted by a PDP contingent on some type of per-session preemption priority. A RSVP PEP could have a set of stateless policy rules for when to preempt other reservations in favor of a new one (e.g. higher-priority pre-empts any of lower priority). The PDP would need to include appropriate priority information for each reservation in its decisions that the PEP can use to apply its rules.

CType = 3, Replacement Data

This object is typically applicable as a decision for an outgoing request. 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 decision message, all signaled 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).

CType = 4, Client Specific Decision Data

Additional decision types can be introduced using the Client

Specific Decision Data Object. Like the Replacement Data object, client specific information is encapsulated within the Client Data Object.

Ctype = 5, Named Decision Data

Named configuration information should be encapsulated in this version of the decision object in response to configuration requests.

2.2.7 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 objects)

2.2.8 Error Object (Error)

This object is used to identify a particular COPS protocol error. The error sub-code field contains additional detailed client specific error codes.

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
8 = Client Failure
9 = Communication Failure
10= Unspecified
11= Shutting down

2.2.9 Client Specific Information Object (ClientSI)

The various types of this object are required for requests, and used in reports and opens when required. It contains client-related information.

```
           C-Num = 10,

           C-Type = 1, Signaled ClientSI.
```

Variable-length field. All objects/attributes specific to a client's signaling protocol or internal state must be encapsulated within one or more signaled Client Specific Information Objects. The format of the data encapsulated in the ClientSI object is determined by the client-type.

C-Type = 2, Named ClientSI.

Variable-length field. Contains named configuration information useful for relaying specific information about the PEP, a request, or configured state to the server.

2.2.10 Timer Object (Timer)

Times are encoded as 2 octet integer values and are in units of seconds. The timer value is treated as a delta.

C-Num = 11,

C-Type = 1, Keep-alive timer value

Timer object used to specify the maximum time interval over which a COPS message must be sent or received. The value of zero implies infinity.

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

C-Type = 2, Accounting timer value

Optional timer value used to determine the minimum interval between periodic accounting type reports. The value of zero implies infinity.

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

2.2.11 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 for the COPS protocol, but does for policy at the PDP that may need to uniquely identify individual PEPs. By default, at least the primary IP address of the PEP represented as a string is expected in the PEPID.

2.2.12 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         |        /////////////        |
+--------------+--------------+--------------+--------------+
```

```
Report-Type:
    1 = Commit    : PEP's local resources now allocated
    2 = Accounting: Accounting update for an installed state
    3 = No Commit : PEP's resource allocation failure
    4 = Installed : Named configuration installed
    5 = Removed   : Named configuration removed
    6 = Enabled   : Named configuration enabled
    7 = Disabled  : Named configuration disabled
    8 = Inst&Enab : Named config. installed and enabled
```

2.2.13 PDP Address (PDPAddr)

A PDP when closing a PEP session for a particular client-type may optionally use this object to redirect the PEP to another PDP server via this object:

C-Num = 14,

C-Type = 1, IPv4 Address (4 octets, as shown for In-interface)

C-Type = 2, IPv6 Address (16 octets, as shown for In-interface)

2.3 Communication

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.

It is possible a single PEP may have open connections to multiple PDPs. This is the case when there are physically different PDPs supporting different client-types as shown in figure 2.

```
       +----------------+
       |                |
       |  Network Node  |            Policy Server
       |                |
       |   +-----+      | COPS Client Type 1  +-----+
       |   |PEP1 |<-----|-------------------->| PDP1|
       |   +-----+  |   | COPS Client Type 2  +-----+
       |    ^ | PEP2|<--|---------\           +-----+
       |    | +-----+   |          \----------| PDP2|
       |    \-->+-----+ |                     +-----+
       |        | LDP | |
       |        +-----+ |
       |                |
       +----------------+
```

Figure 2: Multiple PDPs illustration.

When a TCP connection is torn down or is lost, both the PEP and PDP is expected to clean up any outstanding state related to any pervious request/decision exchanges. Additionally, the PEP should continuously attempt to contact the primary PDP or, if unsuccessful, any known backup PDPs. If a PEP is in communication with a backup PDP and the primary PDP becomes available, the backup PDP should redirect the PEP back to the primary PDP (via a close/redirect message for the affected client-type).

2.4 Client Handle Usage

The client handle is used to identify a unique request state. Client handles are chosen by the PEP and are opaque to the PDP. The PDP simply uses the request handle to uniquely identify the request state and generically tie its decisions to a corresponding request. Client handles are initiated in request messages and are then used by subsequent request, decision, and report messages to reference the same request state. When the PEP is ready to remove a local request state, it will issue a delete message to the PDP for the corresponding client handle. A handle MUST be explicitly deleted by the PEP before it can be used to identify a new request state.

Handles referring to different request states must be unique.

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 client 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 PEP is responsible for notifying the PDP whenever its local state changes so the PDP's state will be able to accurately mirror the PEP's state.

The format of the Request message is as follows:

```
              <Request> ::= <Common Header>
                            <Client Handle>
                            <Context>
                            [<IN-Int>]
                            [<OUT-Int>]
                            <ClientSI(s)>
                            [<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 decision to be returned from the policy server. This decision might be related to admission control, resource allocation, object forwarding and substitution, or configuration.

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 typically used if the client is participating along the path of a signaling protocol or if the client is requesting configuration data for a particular interface.

ClientSI, the client specific information object holds the client-type specific data for which a policy decision needs to be made. In the case of configuration, the named clientSI may include named information about the module, interface, or functionality to be configured.

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

3.2 Decision (DEC) PDP -> PEP

The PDP responds to the REQ with a DEC message that includes the associated client handle and one or more decision objects. If there was a protocol error an error object is returned instead.

It is assumed that the first decision for a new/updated request will set the solicited decision flag. This avoids the issue of keeping track of which updated request (that is, a request reissued for the same handle) a particular decision corresponds. It is important that, for a given handle, there be at most one outstanding solicited decision per request. This essentially means that the PEP should not issue more than one REQ (for a given handle) before it receives a corresponding DEC with the solicited decision flag set.

To avoid deadlock, the client can always timeout after issuing a request. It must then delete the timed-out handle, and possibly try again using a different (new) handle.

The format of the Decision message is as follows:

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

The decision may include either an Error object or one or more decision objects. 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 and the type of client.

3.3 Report State (RPT) PEP -> PDP

This message is used by the PEP to communicate a change in the status of a previously installed state to the PDP. A commit or no-commit report-type indicates to the PDP that a particular policy directive has or has not been acted upon as is relevant for accounting purposes. (In RSVP this would mean that a reservation passed or failed local capacity admission control. For a configuration decision, it would mean the decision data either could or could not be installed by the PEP).

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>
```

```
                                  <Client Handle>
                                  <Report-Type>
                                  [<ClientSI(s)>]
```

3.4 Delete Request State (DRQ) PEP -> PDP

When sent from the PEP this message indicates to the remote PDP that the state identified by the client handle is no longer available/relevant. This information will then be used by the remote PDP to initiate the appropriate housekeeping actions. The reason code object is interpreted with respect to the client-type and signifies the reason for the removal.

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

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

Given the stateful nature of COPS, it is important that when a request state is finally removed from the PEP, a DRQ message for this request state is sent to the PDP so the corresponding state may likewise be removed on the PDP. Request states not explicitly deleted by the PEP will be maintained by the PDP until either the client session is closed or the connection is terminated.

3.5 Synchronize State Request (SSQ) PDP -> PEP

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

```
           <Synchronize State> ::= <Common Header>
                                   [<Client 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 Client Handle is present, only the state associated with this handle is synchronized. If the PEP does not recognize the requested handle, it should immediately send a DRQ message to the PDP for the handle that was specified in the SSQ message. If no handle is specified in the SSQ message, all the active 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. When synchronization is complete, the PEP must issue a synchronize state complete message to the PDP.

3.6 Client-Open (OPN) PEP -> PDP

The Client-Open message can be used by the PEP to specify to the PDP the client-types the PEP can support, a *suggested* time interval for keep-alive messages, and/or minimum time intervals for accounting updates, and/or client specific feature negotiation. A Client-Open message can be sent to the PDP at any time and multiple

Client-Open messages for the same client-type are allowed (in case of global state changes).

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

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 and/or the minimum time interval between periodic accounting reports.

Finally, a named ClientSI object can be included for relaying additional global information about the PEP to the PDP when required (as specified in the appropriate extensions document for the client-type).

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

The Client-Accept message is used to positively respond to the Client-Open message. This message will return to the PEP a timer object indicating the maximum time interval between keep-alive messages. Optionally, a timer specifying the minimum allowed interval between accounting report messages may be included when applicable.

```
           <Client-Accept>  ::= <Common Header>
                                <KA Timer>
                                [<ACCT Timer>]
```

If the PDP refuses the client, it will instead issue a Client-Close message.

The KA 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 0 implies no secondary connection verification is necessary.

The optional accounting timer allows the PDP to indicate to the PEP that periodic accounting reports should not exceed the specified timer interval. This allows the PDP to control the rate at which accounting reports are sent by the PEP (when applicable).

3.8 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 of all timer values negotiated with the OPN & CAT messages. It is a validation for each side that the connection is still functioning.

Note: The client-type in the header should always be set to 0 as the KA is used for connection verification (not per client session verification).

```
            <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.

3.9 Client-Close (CC) PEP -> PDP, PDP -> PEP

The Client-Close message can be issued by either the PDP or PEP to notify the other that a particular type of client is no longer being supported.

```
            <Client-Close>  ::= <Common Header>
                                [<Error>]
                                [<PDPAddr>]
```

An Error object is optionally included to describe the reason for the close due to an error condition (e.g. requested client-type is not supported by the remote PDP or client failure).

A PDP may optionally include a PDP-Address object in order to inform the PEP of the alternate PDP it should use for the client-type specified in the common header.

3.10 Synchronize State Complete (SSC) PEP -> PDP

The Synchronize State Complete is sent by the PEP to the PDP after the PDP sends a synchronize state request to the PEP and the PEP has finished synchronization. It is useful so that the PDP will know when all the old client state has been successfully re-requested and, thus, the PEP and PDP are completely synchronized.

```
          <Synchronize State Complete>  ::= <Common Header>
```

4. Common Operation

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

Sometime 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, at least 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 PDP will instead respond with a Client-Close specifying the client-type is not supported and will possibly suggest an alternate PDP address. Otherwise, the PDP will specify the timer interval between keep-alive messages in its Client-Accept and the PEP can begin issuing its requests to the PDP.

In the outsourcing scenario, when the PEP receives an event that requires a new policy decision it sends a request message to the remote PDP. What specifically qualifies as an event for a particular client-type should be specified in the specific document for that client-type. The remote PDP then makes a decision and sends a decision message 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 decision identifies this request state. The PEP is responsible for deleting this request state once the request is no longer applicable.

The PEP can 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 decision message back to the PEP. Likewise, the server may change a previously issued decision on any currently installed request state at any time by issuing another decision message. At all times the PEP module is expected to abide by the PDP's decisions and notify the PDP of any state changes.

Likewise, in the configuration scenario, the PEP will make a configuration request to the PDP for a particular interface, module, or functionality that may be specified in the named client specific information object. The PDP will then send potentially several decisions containing named units of configuration data to the PEP. The PEP is expected to install and use the configuration locally. A particular named configuration can be updated by simply sending additional decision messages for the same named configuration. When the PDP no longer wishes the PEP to use a piece of configuration information, it will send a decision message specifying the named configuration and a decision flags object with the remove configuration flag set. The PEP should then proceed to remove the corresponding configuration and send a report message to the PDP that specifies it has been deleted.

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

The keep-alive message is used to validate the connection between the client and server is still functioning when there is no other messaging between the PEP and PDP. The PEP must generate a COPS message within one half the negotiated minimum timer interval or else a keep-alive message must be generated. Likewise, the PDP must either have sent a COPS message to every connected PEP within half the negotiated minimum timer interval or a keep-alive must be issued. If either side does not receive a keep-alive or any other COPS message within the negotiated timer interval from the other, the connection should be considered lost.

Finally, Client-Close messages are used to negate the effects of the corresponding Client-Open messages, notifying the other side that the specified client-type is no longer supported/active.

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

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.

[UserID]Yadav, S., Pabbati, R., Ford, P., Herzog, S., "User Identity Representation for RSVP", Internet-Draft, draft-ietf-rap-user-identity-00.txt, March 1998.

8. 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@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