Network Working Group
Request for Comments: 2748
Category: Standards Track

D. Durham, Ed.
Intel
J. Boyle
Level 3
R. Cohen
Cisco
S. Herzog
IPHighway
R. Rajan
AT&T
A. Sastry
Cisco

January 2000

The COPS (Common Open Policy Service) Protocol

Status of this Memo

Copyright Notice

Conventions used in this document

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC-2119].

Abstract

This document describes a simple client/server model for supporting policy control over QoS signaling protocols. 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. The model is designed to be extensible so that other kinds of policy clients may be supported in the future. However, this document makes no claims that it is the only or the preferred approach for enforcing future types of policies.

Table Of Contents

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 an RSVP router 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 this 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.

4. COPS provides message level security for authentication, replay protection, and message integrity. COPS can also reuse existing protocols for security such as IPSEC [IPSEC] or TLS to authenticate and secure the channel between the PEP and the PDP.

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 |
        |   +-----+      |               +-----+
        |    ^           |
        |    |           |
        |    \-->+-----+ |
        |        | LPDP| |
        |        +-----+ |
        |                |
        +----------------+
```

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. The optional Local Policy Decision Point (LPDP) can be used by the device to make local policy decisions in the absence of a 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 PEP is responsible for initiating a persistent TCP connection to a PDP. The PEP uses this 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 successfully completed performing the PDP's decision locally, useful for accounting and monitoring purposes. 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, providing message integrity, and transferring client specific/namespace 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. The COPS Context object identifies the type of request and message (if applicable) that triggered a policy event via its message type and request type fields. 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. The content of a COPS request/decision message depends on the context. A fourth 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 Policy Decision Point (LPDP) [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 an LPDP 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 can be 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 backup/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 deleted state or new 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. Sections 2.3 and 2.5 detail COPS mechanisms for achieving reliability.

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| Flags|    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.

Flags: 4 bits

Defined flag values (all other flags MUST be set to 0):

0x1 Solicited Message Flag Bit

This flag is set when the message is solicited by another COPS message. This flag is NOT to be set (value=0) unless otherwise specified in section 3.

Op Code: 8 bits

The COPS operations:

```
1 = Request                 (REQ)
2 = Decision                (DEC)
3 = Report State            (RPT)
4 = Delete Request State    (DRQ)
5 = Synchronize State Req   (SSQ)
6 = Client-Open             (OPN)
7 = Client-Accept           (CAT)
8 = Client-Close            (CC)
9 = Keep-Alive              (KA)
10= 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). For KA Messages, the client-type in the header MUST always be set to 0 as the KA is used for connection verification (not per client session verification).

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 next 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
            2  = Context
            3  = In Interface
            4  = Out Interface
            5  = Reason code
            6  = Decision
            7  = LPDP Decision
            8  = Error
            9  = Client Specific Info
            10 = Keep-Alive Timer
            11 = PEP Identification
            12 = Report Type
            13 = PDP Redirect Address
            14 = Last PDP Address
            15 = Accounting Timer
            16 = Message Integrity

   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. See Section 2.4 for details.

C-Num = 1

C-Type = 1, Client Handle.

Variable-length field, no implied format other than it is unique from other client handles from the same PEP (a.k.a. COPS TCP connection) for a particular client-type. 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 a particular PEP and installed at the PDP for a client-type. 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 a particular PEP's request for a client-type.

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 = 2, 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 applies and the address where the received message originated. For flows or messages generated from the PEP's local host, the loop back address and ifindex are used.

This Interface object is also used to identify the incoming (receiving) interface via its ifindex. The ifindex may be used to differentiate between sub-interfaces and unnumbered interfaces (see RSVP's LIH for an example). When SNMP is supported by the PEP, this ifindex integer MUST correspond to the same integer value for the interface in the SNMP MIB-II interface index table.

Note: The ifindex specified in the In-Interface is typically relative to the flow of the underlying protocol messages. The ifindex is the interface on which the protocol message was received.

C-Num = 3

C-Type = 1, IPv4 Address + Interface

```
        0             1              2              3
+--------------+--------------+--------------+--------------+
|                   IPv4 Address format                     |
+--------------+--------------+--------------+--------------+
|                          ifindex                          |
+--------------+--------------+--------------+--------------+
```

For this type of the interface object, the IPv4 address specifies the IP address that the incoming message came from.

C-Type = 2, IPv6 Address + Interface

```
       0             1              2             3
+--------------+--------------+--------------+--------------+
|                                                           |
+                                                           +
|                                                           |
+                     IPv6 Address format                   +
|                                                           |
+                                                           +
|                                                           |
+--------------+--------------+--------------+--------------+
|                          ifindex                          |
+--------------+--------------+--------------+--------------+
```

For this type of the interface object, the IPv6 address specifies the IP address that the incoming message came from. The ifindex is used to refer to the MIB-II defined local incoming interface on the PEP as described above.

2.2.4 Out-Interface Object (OUT-Int)

The Out-Interface is used to identify the outgoing interface to which a specific request applies and the address for where the forwarded message is to be sent. For flows or messages destined to the PEP's local host, the loop back address and ifindex are used. The Out-Interface has the same formats as the In-Interface Object.

This Interface object is also used to identify the outgoing (forwarding) interface via its ifindex. The ifindex may be used to differentiate between sub-interfaces and unnumbered interfaces (see RSVP's LIH for an example). When SNMP is supported by the PEP, this ifindex integer MUST correspond to the same integer value for the interface in the SNMP MIB-II interface index table.

Note: The ifindex specified in the Out-Interface is typically relative to the flow of the underlying protocol messages. The ifindex is the one on which a protocol message is about to be forwarded.

C-Num = 4

C-Type = 1, IPv4 Address + Interface

Same C-Type format as the In-Interface object. The IPv4 address specifies the IP address to which the outgoing message is going. The ifindex is used to refer to the MIB-II defined local outgoing interface on the PEP.

C-Type = 2, IPv6 Address + Interface

Same C-Type format as the In-Interface object. For this type of the interface object, the IPv6 address specifies the IP address to which the outgoing message is going. The ifindex is used to refer to the MIB-II defined local outgoing interface on the PEP.

2.2.5 Reason Object (Reason)

This object specifies the reason why the request state was deleted. It appears 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 = 5, C-Type = 1

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

```
Reason Code:
    1 = Unspecified
    2 = Management
    3 = Preempted (Another request state takes precedence)
    4 = Tear (Used to communicate a signaled state removal)
    5 = Timeout (Local state has timed-out)
    6 = Route Change (Change invalidates request state)
    7 = Insufficient Resources (No local resource available)
    8 = PDP's Directive (PDP decision caused the delete)
    9 = Unsupported decision (PDP decision not supported)
    10= Synchronize Handle Unknown
    11= Transient Handle (stateless event)
    12= Malformed Decision (could not recover)
    13= Unknown COPS Object from PDP:
        Sub-code (octet 2) contains unknown object's C-Num
        and (octet 3) contains unknown object's C-Type.
```

2.2.6 Decision Object (Decision)

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

```
        C-Num = 6
        C-Type = 1, Decision Flags (Mandatory)

        0             1              2              3
+--------------+--------------+--------------+--------------+
|        Command-Code         |            Flags            |
+--------------+--------------+--------------+--------------+

    Commands:
        0 = NULL Decision (No configuration data available)
        1 = Install (Admit request/Install configuration)
        2 = Remove (Remove request/Remove configuration)

    Flags:
        0x01 = Trigger Error (Trigger error message if set)
         Note: Trigger Error is applicable to client-types that
         are capable of sending error notifications for signaled
         messages.
```

Flag values not applicable to a given context's R-Type or client-type MUST be ignored by the PEP.

C-Type = 2, Stateless Data

This type of decision object carries additional stateless information that can be applied by the PEP locally. It is a variable length object and its internal format SHOULD be specified in the relevant COPS extension document for the given client-type. This object is optional in Decision messages and is interpreted relative to a given context.

It is expected that even outsourcing PEPs will be able to make some simple stateless policy decisions locally in their LPDP. 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 MUST apply it to the resource allocation event that generated the request.

C-Type = 3, Replacement Data

This type of decision object carries replacement data that is to replace existing data in a signaled message. It is a variable length object and its internal format SHOULD be specified in the relevant COPS extension document for the given client-type. It is optional in Decision messages and is interpreted relative to a given context.

C-Type = 4, Client Specific Decision Data

Additional decision types can be introduced using the Client Specific Decision Data Object. It is a variable length object and its internal format SHOULD be specified in the relevant COPS extension document for the given client-type. It is optional in Decision messages and is interpreted relative to a given context.

C-Type = 5, Named Decision Data

Named configuration information is encapsulated in this version of the decision object in response to configuration requests. It is a variable length object and its internal format SHOULD be specified in the relevant COPS extension document for the given client-type. It is optional in Decision messages and is interpreted relative to both a given context and decision flags.

2.2.7 LPDP Decision Object (LPDPDecision)

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

C-Num = 7

C-Type = (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. The appropriate Error Sub-codes for a particular client-type SHOULD be specified in the relevant COPS extensions document.

C-Num = 8, 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 (Malformed Message)
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
12= Redirect to Preferred Server
13= Unknown COPS Object:
    Sub-code (octet 2) contains unknown object's C-Num
    and (octet 3) contains unknown object's C-Type.
14= Authentication Failure
15= Authentication Required

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-type specific information.

C-Num = 9,

C-Type = 1, Signaled ClientSI.

Variable-length field. All objects/attributes specific to a client's signaling protocol or internal state are 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 PDP server.

2.2.10 Keep-Alive Timer Object (KATimer)

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

C-Num = 10,

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 range of finite timeouts is 1 to 65535 seconds represented as an unsigned two-octet integer. The value of zero implies infinity.

```
          0             1              2             3
   +--------------+--------------+--------------+--------------+
   |        //////////////       |        KA 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 = 11, C-Type = 1

Variable-length field. It is a NULL terminated ASCII string that is also zero padded to a 32-bit word boundary (so the object length is a multiple of 4 octets). The PEPID MUST contain an ASCII string that uniquely identifies the PEP within the policy domain in a manner that is persistent across PEP reboots. For example, it may be the PEP's statically assigned IP address or DNS name. This identifier may safely be used by a PDP as a handle for identifying the PEP in its policy rules.

2.2.12 Report-Type Object (Report-Type)

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

C-Num = 12, C-Type = 1

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

Report-Type:
- 1 = Success   : Decision was successful at the PEP
- 2 = Failure   : Decision could not be completed by PEP
- 3 = Accounting: Accounting update for an installed state

2.2.13 PDP Redirect Address (PDPRedirAddr)

A PDP when closing a PEP session for a particular client-type may optionally use this object to redirect the PEP to the specified PDP server address and TCP port number:

```
      C-Num = 13,

      C-Type = 1, IPv4 Address + TCP Port
               0             1              2             3
      +--------------+--------------+--------------+--------------+
      |                   IPv4 Address format                     |
      +--------------+--------------+--------------+--------------+
      |  /////////////////////////  |       TCP Port Number       |
      +-----------------------------+-----------------------------+

      C-Type = 2, IPv6 Address + TCP Port
               0             1              2             3
      +--------------+--------------+--------------+--------------+
      |                                                           |
      +                                                           +
      |                                                           |
      +                    IPv6 Address format                    +
      |                                                           |
      +                                                           +
      |                                                           |
      +--------------+--------------+--------------+--------------+
      |  /////////////////////////  |       TCP Port Number       |
      +-----------------------------+-----------------------------+
```

2.2.14 Last PDP Address (LastPDPAddr)

When a PEP sends a Client-Open message for a particular client-type the PEP SHOULD specify the last PDP it has successfully opened (meaning it received a Client-Accept) since the PEP last rebooted. If no PDP was used since the last reboot, the PEP will simply not include this object in the Client-Open message.

C-Num = 14,

C-Type = 1, IPv4 Address (Same format as PDPRedirAddr)

C-Type = 2, IPv6 Address (Same format as PDPRedirAddr)

2.2.15 Accounting Timer Object (AcctTimer)

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

C-Num = 15,

C-Type = 1, Accounting timer value

Optional timer value used to determine the minimum interval between periodic accounting type reports. It is used by the PDP to describe to the PEP an acceptable interval between unsolicited accounting updates via Report messages where applicable. It provides a method for the PDP to control the amount of accounting traffic seen by the network. The range of finite time values is 1 to 65535 seconds represented as an unsigned two-octet integer. A value of zero means there SHOULD be no unsolicited accounting updates.

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

2.2.16 Message Integrity Object (Integrity)

The integrity object includes a sequence number and a message digest useful for authenticating and validating the integrity of a COPS message. When used, integrity is provided at the end of a COPS message as the last COPS object. The digest is then computed over all of a particular COPS message up to but not including the digest value itself. The sender of a COPS message will compute and fill in the digest portion of the Integrity object. The receiver of a COPS message will then compute a digest over the received message and verify it matches the digest in the received Integrity object.

C-Num = 16,

C-Type = 1, HMAC digest

The HMAC integrity object employs HMAC (Keyed-Hashing for Message Authentication) [HMAC] to calculate the message digest based on a key shared between the PEP and its PDP.

This Integrity object specifies a 32-bit Key ID used to identify a specific key shared between a particular PEP and its PDP and the cryptographic algorithm to be used. The Key ID allows for multiple simultaneous keys to exist on the PEP with corresponding keys on the PDP for the given PEPID. The key identified by the Key ID was used to compute the message digest in the Integrity object. All implementations, at a minimum, MUST support HMAC-MD5-96, which is HMAC employing the MD5 Message-Digest Algorithm [MD5] truncated to 96-bits to calculate the message digest.

This object also includes a sequence number that is a 32-bit unsigned integer used to avoid replay attacks. The sequence number is initiated during an initial Client-Open Client-Accept message exchange and is then incremented by one each time a new message is sent over the TCP connection in the same direction. If the sequence number reaches the value of 0xFFFFFFFF, the next increment will simply rollover to a value of zero.

The variable length digest is calculated over a COPS message starting with the COPS Header up to the Integrity Object (which MUST be the last object in a COPS message) INCLUDING the Integrity object's header, Key ID, and Sequence Number. The Keyed Message Digest field is not included as part of the digest calculation. In the case of HMAC-MD5-96, HMAC-MD5 will produce a 128-bit digest that is then to be truncated to 96-bits before being stored in or verified against the Keyed Message Digest field as specified in [HMAC]. The Keyed Message Digest MUST be 96-bits when HMAC-MD5-96 is used.

```
          0             1              2             3
    +-------------+-------------+-------------+-------------+
    |                        Key ID                         |
    +-------------+-------------+-------------+-------------+
    |                    Sequence Number                    |
    +-------------+-------------+-------------+-------------+
    |                                                       |
    +                                                       +
    |                 ...Keyed Message Digest...            |
    +                                                       +
    |                                                       |
    +-------------+-------------+-------------+-------------+
```

2.3 Communication

The COPS protocol uses a single persistent TCP connection between the PEP and a remote PDP. One PDP implementation per server MUST listen on a well-known TCP port number (COPS=3288 [IANA]). The PEP is responsible for initiating the TCP connection to a PDP. 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.

If a single PEP can support multiple client-types, it may send multiple Client-Open messages, each specifying a particular client-type to a PDP over one or more TCP connections. Likewise, a PDP residing at a given address and port number may support one or more client-types. Given the client-types it supports, a PDP has the ability to either accept or reject each client-type independently. If a client-type is rejected, the PDP can redirect the PEP to an alternative PDP address and TCP port for a given client-type via COPS. Different TCP port numbers can be used to redirect the PEP to another PDP implementation running on the same server. Additional provisions for supporting multiple client-types (perhaps from independent PDP vendors) on a single remote PDP server are not provided by the COPS protocol, but, rather, are left to the software architecture of the given server platform.

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 Servers
|                |
|   +-----+      | COPS Client Type 1  +-----+
|   |     |<-----|-------------------->| PDP1|
|   + PEP +      | COPS Client Type 2  +-----+
|   |     |<-----|---------\           +-----+
|   +-----+      |          \----------| PDP2|
|    ^           |                     +-----+
|    |           |
|    \-->+-----+ |
|        | LPDP| |
|        +-----+ |
|                |
+----------------+
```

Figure 2: Multiple PDPs illustration.

When a TCP connection is torn down or is lost, the PDP is expected to eventually clean up any outstanding request state related to request/decision exchanges with the PEP. When the PEP detects a lost connection due to a timeout condition it SHOULD explicitly send a Client-Close message for each opened client-type containing an <Error> object indicating the "Communication Failure" Error-Code. Additionally, the PEP SHOULD continuously attempt to contact the primary PDP or, if unsuccessful, any known backup PDPs. Specifically the PEP SHOULD keep trying all relevant PDPs with which it has been configured until it can establish a connection. If a PEP is in communication with a backup PDP and the primary PDP becomes available, the backup PDP is responsible for redirecting the PEP back to the primary PDP (via a <Client-Close> message containing a <PDPRedirAddr> object identifying the primary PDP to use for each affected client-type). Section 2.5 details synchronization behavior between PEPs and PDPs.

2.4 Client Handle Usage

The client handle is used to identify a unique request state for a single PEP per client-type. 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 for a particular Client-Type over a particular TCP connection 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 by the PEP to identify a new request state. Handles referring to different request states MUST be unique within the context of a particular TCP connection and client-type.

2.5 Synchronization Behavior

When disconnected from a PDP, 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 have passed local admission control. Additionally, the remote PDP may request that all the PEP's internal state be resynchronized (all previously installed requests are to be reissued) by sending a Synchronize State message.

After a failure and before a 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 appropriate length of time. Specific rules for such behavior are to be defined in the appropriate COPS client-type extension specifications.

A PEP that caches state from a previous exchange with a disconnected PDP MUST communicate this fact to any PDP with which it is able to later reconnect. This is accomplished by including the address and TCP port of the last PDP for which the PEP is still caching state in the Client-Open message. The <LastPDPAddr> object will only be included for the last PDP with which the PEP was completely in sync. If the service interruption was temporary and the PDP still contains the complete state for the PEP, the PDP may choose not to synchronize all states. It is still the responsibility of the PEP to update the PDP of all state changes that occurred during the disruption of service including any states communicated to the previous PDP that had been deleted after the connection was lost. These MUST be explicitly deleted after a connection is reestablished. If the PDP issues a synchronize request the PEP MUST pass all current states to the PDP followed by a Synchronize State Complete message (thus completing the synchronization process). If the PEP crashes and loses all cached state for a client-type, it will simply not include a <LastPDPAddr> in its Client-Open message.

3. Message Content

This section describes the basic messages exchanged between a PEP and a remote PDP as well as their contents. As a convention, object ordering is expected as shown in the BNF for each COPS message unless otherwise noted. The Integrity object, if included, MUST always be the last object in a message. If security is required and a message was received without a valid Integrity object, the receiver MUST send a Client-Close message for Client-Type=0 specifying the appropriate error code.

3.1 Request (REQ) PEP -> PDP

The PEP establishes a request state client handle for which the remote PDP may maintain state. The remote PDP then uses this handle to refer to the exchanged information and decisions communicated over the TCP connection to a particular PEP for a given client-type.

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 Message> ::=  <Common Header>
                       <Client Handle>
                       <Context>
                       [<IN-Int>]
                       [<OUT-Int>]
                       [<ClientSI(s)>]
                       [<LPDPDecision(s)>]
                       [<Integrity>]

<ClientSI(s)> ::= <ClientSI> | <ClientSI(s)> <ClientSI>

<LPDPDecision(s)> ::= <LPDPDecision> |
                      <LPDPDecision(s)> <LPDPDecision>

<LPDPDecision> ::= [<Context>]
                   <LPDPDecision: Flags>
                   [<LPDPDecision: Stateless Data>]
                   [<LPDPDecision: Replacement Data>]
                   [<LPDPDecision: ClientSI Data>]
                   [<LPDPDecision: Named Data>]
```

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. The ordering of multiple ClientSIs is not important.

Finally, LPDPDecision object holds information regarding the local decision made by the LPDP.

Malformed Request messages MUST result in the PDP specifying a Decision message with the appropriate error code.

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 grouped relative to a Context object and Decision Flags object type pair. If there was a protocol error an error object is returned instead.

It is required that the first decision message for a new/updated request will have the solicited message flag set (value = 1) in the COPS header. 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 message flag set. The PDP MUST always issue decisions for requests on a particular handle in the order they arrive and all requests MUST have a corresponding decision.

To avoid deadlock, the PEP can always timeout after issuing a request that does not receive a decision. It MUST then delete the timed-out handle, and may try again using a new handle.

The format of the Decision message is as follows:

```
<Decision Message> ::= <Common Header>
                       <Client Handle>
                       <Decision(s)> | <Error>
                       [<Integrity>]

<Decision(s)> ::= <Decision> | <Decision(s)> <Decision>

<Decision> ::= <Context>
               <Decision: Flags>
               [<Decision: Stateless Data>]
               [<Decision: Replacement Data>]
               [<Decision: ClientSI Data>]
               [<Decision: Named Data>]
```

The Decision message may include either an Error object or one or more context plus associated decision objects. COPS protocol problems are reported in the Error object (e.g. an error with the format of the original request including malformed request messages, unknown COPS objects in the Request, etc.). The applicable Decision object(s) depend on the context and the type of client. The only ordering requirement for decision objects is that the required Decision Flags object type MUST precede the other Decision object types per context binding.

3.3 Report State (RPT) PEP -> PDP

The RPT message is used by the PEP to communicate to the PDP its success or failure in carrying out the PDP's decision, or to report an accounting related change in state. The Report-Type specifies the kind of report and the optional ClientSI can carry additional information per Client-Type.

For every DEC message containing a configuration context that is received by a PEP, the PEP MUST generate a corresponding Report State message with the Solicited Message flag set describing its success or failure in applying the configuration decision. In addition, outsourcing decisions from the PDP MAY result in a corresponding solicited Report State from the PEP depending on the context and the type of client. RPT messages solicited by decisions for a given Client Handle MUST set the Solicited Message flag and MUST be sent in the same order as their corresponding Decision messages were received. There MUST never be more than one Report State message generated with the Solicited Message flag set per Decision.

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 appropriate client specific information object.

```
<Report State> ::== <Common Header>
                    <Client Handle>
                    <Report-Type>
                    [<ClientSI>]
                    [<Integrity>]
```

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>
                      [<Integrity>]
```

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.

Malformed Decision messages MUST trigger a DRQ specifying the appropriate erroneous reason code (Bad Message Format) and any associated state on the PEP SHOULD either be removed or re-requested. If a Decision contained an unknown COPS Decision Object, the PEP MUST delete its request specifying the Unknown COPS Object reason code because the PEP will be unable to comply with the information contained in the unknown object. In any case, after issuing a DRQ, the PEP may retry the corresponding Request again.

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>]
                        [<Integrity>]
```

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 MUST 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 MUST 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, the last PDP to which the PEP connected for the given client-type, 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>
                   [<ClientSI>]
                   [<LastPDPAddr>]
                   [<Integrity>]
```

The PEPID is a symbolic, variable length name that uniquely identifies the specific client to the PDP (see Section 2.2.11).

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).

The PEP may also provide a Last PDP Address object in its Client-Open message specifying the last PDP (for the given client-type) for which it is still caching decisions since its last reboot. A PDP can use this information to determine the appropriate synchronization behavior (See section 2.5).

If the PDP receives a malformed Client-Open message it MUST generate a Client-Close message specifying the appropriate error code.

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>]
                     [<Integrity>]
```

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 and is specified in seconds. A timer value of 0 implies no secondary connection verification is necessary.

The optional ACCT Timer allows the PDP to indicate to the PEP that periodic accounting reports SHOULD NOT exceed the specified timer interval per client handle. This allows the PDP to control the rate at which accounting reports are sent by the PEP (when applicable).

In general, accounting type Report messages are sent to the PDP when determined appropriate by the PEP. The accounting timer merely is used by the PDP to keep the rate of such updates in check (i.e. Preventing the PEP from blasting the PDP with accounting reports). Not including this object implies there are no PDP restrictions on the rate at which accounting updates are generated.

If the PEP receives a malformed Client-Accept message it MUST generate a Client-Close message specifying the appropriate error code.

3.8 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>
                    [<PDPRedirAddr>]
                    [<Integrity>]
```

The Error object is included to describe the reason for the close (e.g. the requested client-type is not supported by the remote PDP or client failure).

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

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

The keep-alive message MUST be transmitted by the PEP within the period defined by the minimum of all KA Timer values specified in all received CAT messages for the connection. A KA message MUST be generated randomly between 1/4 and 3/4 of this minimum KA timer interval. When the PDP receives a keep-alive message from a PEP, it MUST echo a keep-alive back to the PEP. This message provides validation for each side that the connection is still functioning even when there is no other messaging.

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

```
<Keep-Alive>  ::= <Common Header>
                  [<Integrity>]
```

Both client and server MAY assume the TCP 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.

3.10 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. The Client Handle object only needs to be included if the corresponding Synchronize State Message originally referenced a specific handle.

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

4. Common Operation

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

4.1 Security and Sequence Number Negotiation

COPS message security is negotiated once per connection and covers all communication over a particular connection. If COPS level security is required, it MUST be negotiated during the initial Client-Open/Client-Accept message exchange specifying a Client-Type of zero (which is reserved for connection level security negotiation and connection verification).

If a PEP is not configured to use COPS security with a PDP it will simply send the PDP Client-Open messages for the supported Client-Types as specified in section 4.3 and will not include the Integrity object in any COPS messages.

Otherwise, security can be initiated by the PEP if it sends the PDP a Client-Open message with Client-Type=0 before opening any other Client-Type. If the PDP receives a Client-Open with a Client-Type=0 after another Client-Type has already been opened successfully it MUST return a Client-Close message (for Client-Type=0) to that PEP. This first Client-Open message MUST specify a Client-Type of zero and MUST provide the PEPID and a COPS Integrity object. This Integrity object will contain the initial sequence number the PEP requires the

PDP to increment during subsequent communication after the initial Client-Open/Client-Accept exchange and the Key ID identifying the algorithm and key used to compute the digest.

Similarly, if the PDP accepts the PEP's security key and algorithm by validating the message digest using the identified key, the PDP MUST send a Client-Accept message with a Client-Type of zero to the PEP carrying an Integrity object. This Integrity object will contain the initial sequence number the PDP requires the PEP to increment during all subsequent communication with the PDP and the Key ID identifying the key and algorithm used to compute the digest.

If the PEP, from the perspective of a PDP that requires security, fails or never performs the security negotiation by not sending an initial Client-Open message with a Client-Type=0 including a valid Integrity object, the PDP MUST send to the PEP a Client-Close message with a Client-Type=0 specifying the appropriate error code. Similarly, if the PDP, from the perspective of a PEP that requires security, fails the security negotiation by not sending back a Client-Accept message with a Client-Type=0 including a valid Integrity object, the PEP MUST send to the PDP a Client-Close message with a Client-Type=0 specifying the appropriate error code. Such a Client-Close message need not carry an integrity object (as the security negotiation did not yet complete).

The security initialization can fail for one of several reasons: 1. The side receiving the message requires COPS level security but an Integrity object was not provided (Authentication Required error code). 2. A COPS Integrity object was provided, but with an unknown/unacceptable C-Type (Unknown COPS Object error code specifying the unsupported C-Num and C-Type). 3. The message digest or Key ID in the provided Integrity object was incorrect and therefore the message could not be authenticated using the identified key (Authentication Failure error code).

Once the initial security negotiation is complete, the PEP will know what sequence numbers the PDP expects and the PDP will know what sequence numbers the PEP expects. ALL COPS messages must then include the negotiated Integrity object specifying the correct sequence number with the appropriate message digest (including the Client-Open/Client-Accept messages for specific Client-Types). ALL subsequent messages from the PDP to the PEP MUST result in an increment of the sequence number provided by the PEP in the Integrity object of the initial Client-Open message. Likewise, ALL subsequent messages from the PEP to the PDP MUST result in an increment of the sequence number provided by the PDP in the Integrity object of the initial Client-Accept message. Sequence numbers are incremented by one starting with the corresponding initial sequence number. For example, if the sequence number specified to the PEP by the PDP in the initial Client-Accept was 10, the next message the PEP sends to the PDP will provide an Integrity object with a sequence number of 11... Then the next message the PEP sends to the PDP will have a sequence number of 12 and so on. If any subsequent received message contains the wrong sequence number, an unknown Key ID, an invalid message digest, or is missing an Integrity object after integrity was negotiated, then a Client-Close message MUST be generated for the Client-Type zero containing a valid Integrity object and specifying the appropriate error code. The connection should then be dropped.

4.2 Key Maintenance

Key maintenance is outside the scope of this document, but COPS implementations MUST at least provide the ability to manually configure keys and their parameters locally. The key used to produce the Integrity object's message digest is identified by the Key ID field. Thus, a Key ID parameter is used to identify one of potentially multiple simultaneous keys shared by the PEP and PDP. A Key ID is relative to a particular PEPID on the PDP or to a particular PDP on the PEP. Each key must also be configured with lifetime parameters for the time period within which it is valid as well as an associated cryptographic algorithm parameter specifying the algorithm to be used with the key. At a minimum, all COPS implementations MUST support the HMAC-MD5-96 [HMAC][MD5] cryptographic algorithm for computing a message digest for inclusion in the Keyed Message Digest of the Integrity object which is appended to the message.

It is good practice to regularly change keys. Keys MUST be configurable such that their lifetimes overlap allowing smooth transitions between keys. At the midpoint of the lifetime overlap between two keys, senders should transition from using the current key to the next/longer-lived key. Meanwhile, receivers simply accept any identified key received within its configured lifetime and reject those that are not.

4.3 PEP Initialization

Sometime after a connection is established between the PEP and a remote PDP and after security is negotiated (if required), the PEP will send one or more Client-Open messages to the remote PDP, one for each client-type supported by the PEP. The Client-Open message MUST contain the address of the last PDP with which the PEP is still caching a complete set of decisions. If no decisions are being cached from the previous PDP the LastPDPAddr object MUST NOT be included in the Client-Open message (see Section 2.5). Each Client-Open message MUST at least contain the common header noting one client-type supported by the PEP. The remote PDP will then respond with separate Client-Accept messages for each of the client-types requested by the PEP that the PDP can also support.

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 and port. Otherwise, the PDP will send a Client-Accept specifying the timer interval between keep-alive messages and the PEP may begin issuing requests to the PDP.

4.4 Outsourcing Operations

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 (unique per TCP connection and client-type), 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 an unsolicited 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.

4.5 Configuration Operations

In the configuration scenario, as in the outsourcing 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 command. 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 can 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.

4.6 Keep-Alive Operations

The Keep-Alive message is used to validate the connection between the client and server is still functioning even when there is no other messaging from the PEP to PDP. The PEP MUST generate a COPS KA message randomly within one-fourth to three-fourths the minimum KA Timer interval specified by the PDP in the Client-Accept message. On receiving a Keep-Alive message from the PEP, the PDP MUST then respond to this Keep-Alive message by echoing a Keep-Alive message back to the PEP. If either side does not receive a Keep-Alive or any other COPS message within the minimum KA Timer interval from the other, the connection SHOULD be considered lost.

4.7 PEP/PDP Close

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. When the PEP detects a lost connection due to a keep-alive timeout condition it SHOULD explicitly send a Client-Close message for each opened client-type specifying a communications failure error code. Then the PEP MAY proceed to terminate the connection to the PDP and attempt to reconnect again or try a backup/alternative PDP. When the PDP is shutting down, it SHOULD also explicitly send a Client-Close to all connected PEPs for each client-type, perhaps specifying an alternative PDP to use instead.

5. Security Considerations

The COPS protocol provides an Integrity object that can achieve authentication, message integrity, and replay prevention. All COPS implementations MUST support the COPS Integrity object and its mechanisms as described in this document. To ensure the client (PEP) is communicating with the correct policy server (PDP) requires authentication of the PEP and PDP using a shared secret, and consistent proof that the connection remains valid. The shared secret minimally requires manual configuration of keys (identified by a Key

ID) shared between the PEP and its PDP. The key is used in conjunction with the contents of a COPS message to calculate a message digest that is part of the Integrity object. The Integrity object is then used to validate all COPS messages sent over the TCP connection between a PEP and PDP.

Key maintenance is outside the scope of this document beyond the specific requirements discussed in section 4.2. In general, it is good practice to regularly change keys to maintain security. Furthermore, it is good practice to use localized keys specific to a particular PEP such that a stolen PEP will not compromise the security of an entire administrative domain.

The COPS Integrity object also provides sequence numbers to avoid replay attacks. The PDP chooses the initial sequence number for the PEP and the PEP chooses the initial sequence number for the PDP. These initial numbers are then incremented with each successive message sent over the connection in the corresponding direction. The initial sequence numbers SHOULD be chosen such that they are monotonically increasing and never repeat for a particular key.

Security between the client (PEP) and server (PDP) MAY be provided by IP Security [IPSEC]. In this case, the IPSEC Authentication Header (AH) SHOULD be used for the validation of the connection; additionally IPSEC Encapsulation Security Payload (ESP) MAY be used to provide both validation and secrecy.

Transport Layer Security [TLS] MAY be used for both connection-level validation and privacy.

6. IANA Considerations

The Client-type identifies the policy client application to which a message refers. Client-type values within the range 0x0001-0x3FFF are reserved Specification Required status as defined in [IANA-CONSIDERATIONS]. These values MUST be registered with IANA and their behavior and applicability MUST be described in a COPS extension document.

Client-type values in the range 0x4000 - 0x7FFF are reserved for Private Use as defined in [IANA-CONSIDERATIONS]. These Client-types are not tracked by IANA and are not to be used in standards or general-release products, as their uniqueness cannot be assured.

Client-type values in the range 0x8000 - 0xFFFF are First Come First Served as defined in [IANA-CONSIDERATIONS]. These Client-types are tracked by IANA but do not require published documents describing their use. IANA merely assures their uniqueness.

Objects in the COPS Protocol are identified by their C-Num and C-Type values. IETF Consensus as identified in [IANA-CONSIDERATIONS] is required to introduce new values for these numbers and, therefore, new objects into the base COPS protocol.

Additional Context Object R-Types, Reason-Codes, Report-Types, Decision Object Command-Codes/Flags, and Error-Codes MAY be defined for use with future Client-types, but such additions require IETF Consensus as defined in [IANA-CONSIDERATIONS].

Context Object M-Types, Reason Sub-Codes, and Error Sub-codes MAY be defined relative to a particular Client-type following the same IANA considerations as their respective Client-type.

7. References

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

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

[SRVLOC] Guttman, E., Perkins, C., Veizades, J. and M. Day, "Service Location Protocol , Version 2", RFC 2608, June 1999.

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

[IPSEC] Atkinson, R., "Security Architecture for the Internet Protocol", RFC 2401, August 1995.

[HMAC] Krawczyk, H., Bellare, M. and R. Canetti, "HMAC: Keyed-Hashing for Message Authentication", RFC 2104, February 1997.

[MD5] Rivest, R., "The MD5 Message-Digest Algorithm", RFC 1321, April 1992.

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

[TLS] Dierks T. and C. Allen, "The TLS Protocol Version 1.0", RFC 2246, January 1999.

[IANA] http://www.isi.edu/in-notes/iana/assignments/port-numbers

[IANA-CONSIDERATIONS] Alvestrand, H. and T. Narten, "Guidelines for Writing an IANA Considerations Section in RFCs", BCP 26, RFC 2434, October 1998.

8. Author Information and Acknowledgments

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

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

Phone: 720.888.1192
EMail: jboyle@Level3.net

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

Phone: +972.9.9700064
EMail: ronc@cisco.com

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

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

Raju Rajan
AT&T Shannon Laboratory
180 Park Avenue
P.O. Box 971
Florham Park, NJ 07932-0971

EMail: rajan@research.att.com

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

Phone: 508.620.1141
EMail: herzog@iphighway.com

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

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

9. Full Copyright Statement

Acknowledgement

Funding for the RFC Editor function is currently provided by the Internet Society.