3GPP TSG SA WG3 Security — S3#21b **S3z020013**

31st January - 1st February, 2002

Antwerp, Belgium

Source: Nokia

Title: Proposed additions to 33.200 about COPS usage in Ze interface for Local Security Association and Policy Distribution

Document for: Discussion

Agenda Item: MAP-5.4

This contribution proposes additional chapter 8.2 to the section 8 concerning Local Security Association and Policy Distribution in Ze interface with COPS protocol.

COPS enables already in chapter 8.1 mentioned extended PUSH mechanism for delivering the SA and policy information to MAP-NEs. This model is scalable, as there is signalling traffic only when necessary. Additionally, configurations can be modified directly and not only after scheduled updates. The mechanism also provides possibly existing intrusion detection means to react more quickly.

COPS 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. COPS provides message level security for authentication, replay protection, and message integrity. COPS shall utilize 33.210 NDS/IP mechanisms to make confidentiality possible for delivered MAPSec encryption and integrity keys.

The exact COPS message contents are not defined in this contribute, so COPS extension document is required.

References

[1] IETF RFC 2748: The COPS (Common Open Policy Service) protocol, http://www.ietf.org/rfc/rfc2748.txt

[2] IETF RFC 3084: COPS Usage for Policy Provisioning (COPS-PR), http://www.ietf.org/rfc/rfc3084.txt

[3] IETF Draft (2001): The MAP Security Domain of Interpretation for ISAKMP, draft-arkko-map-doi-04.txt

---

8.2 COPS usage in SA and Policy Distribution

Common Open Policy Service protocol is 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). The optional Local Policy Decision Point (LPDP) can be used to make local policy decisions in the absence of a PDP. In MAPSec the MAP-NE acts as PEP and KAC acts as PDP, LPDP is not utilized.

8.2.1 Establishing COPS connection

Before any COPS transaction can take place between MAP-NE and KAC a communication path has to be set up between them. MAP-NE having PEP functionality is responsible to establish a TCP session to KAC.

For opening COPS connection between MAP-NE and KAC MAP-NE sends *Client-Open* message to KAC. If KAC accepts the client type, it responds with *Client-Accept* message. Mandatory parameters are <PEP identity> in the first message and <Keep Alive timer value> in the second message.

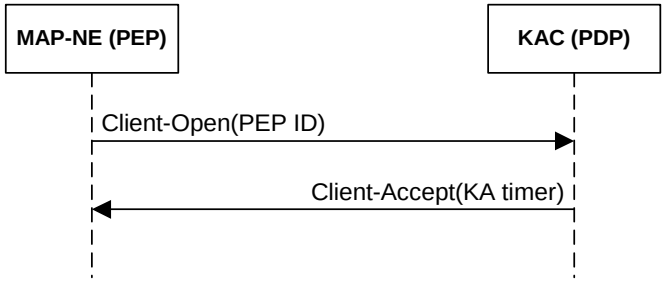


Figure 2: Opening COPS connection

When the connection is established, the MAP-NE sends information about itself to the KAC by sending *Request* message. This message contains in the 'Context' parameter information that it is a configuration request message. With that information KAC knows that it must provide security parameters (policy and SAs) to the MAP-NE. KAC sends *Decision* message(s), which contain the security parameters. MAP-NE responds with Report State message to inform KAC about the status of installation of the security parameters.

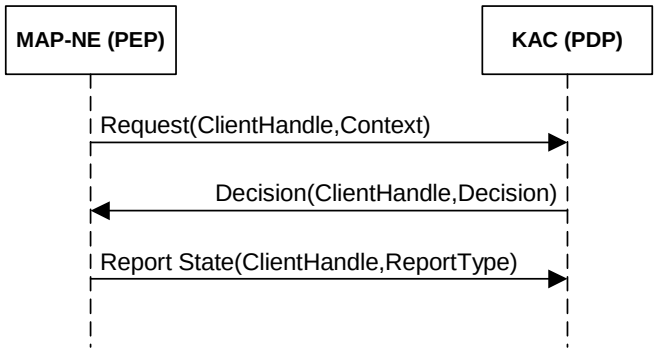


Figure 3: Basic decision request

8.2.2 Security policy management

The initial policy data delivery is done when MAP-NE has registered to KAC.

When policy is changed in KAC, the necessary information has to be delivered to MAP-NEs too. KAC knows all MAP-NEs, which have registered as clients to KAC with Client-Open message, and therefore is able to start delivery of security policy information to all MAP-NEs. KAC uploads the security policy information to MAP-NEs by sending *Decision* message to all registered MAP-NEs. The MAP-NE responds with Report State message to inform KAC about the status of transaction.

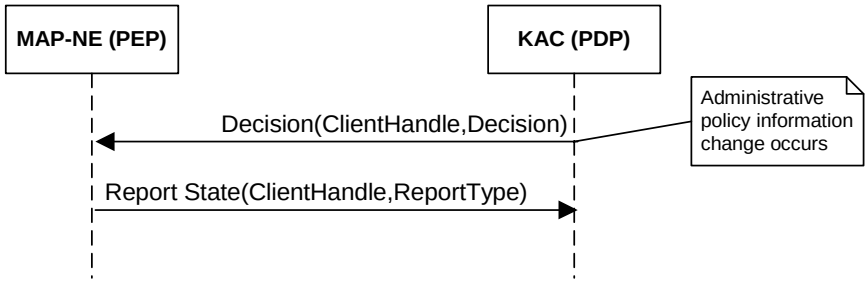


Figure 4: Policy information update

8.2.3 Security association management

KAC may perform an unsolicited download of MAPSec SA to a MAP-NE by sending *Decision* message. <Client Handle> identifies MAP-NE. <Decision> is an installation of configuration data for MAPSec SA towards a target Security Domain. The procedure is similar to KAC initiated policy information update.

8.2.3.1 SA revocation

If SA must be deleted, the initiation must come from KAC. The procedure goes same way as KAC initiated policy information delivery: KAC invokes *Decision* message and Decision parameter contains information that SA must be deleted from SADB.

8.2.4 SA recovery

SA Recovery means a situation where MAP-NE has lost or somehow corrupted all or some of the MAPSec SAs it has received earlier. E.g. MAP-NE might have gone through a reset. Also the cases where MAP-NE has not for some reason received an SA or it notices that SA expires and there is not a new one available are included into *SA recovery*.

To recover from any obscure state MAP-NE has to initialise a *Client-Open* and *Request* procedures to KAC as described in Establishing COPS connection.

8.2.5 Other mandatory COPS procedures

As interval between MAPSec SA renewals may be a long one, to keep COPS connection and TCP session alive a *Keep-Alive* message has to be sent from MAP-NE before KA timer expires. Receiving node has to echo back the same *Keep-Alive* message.

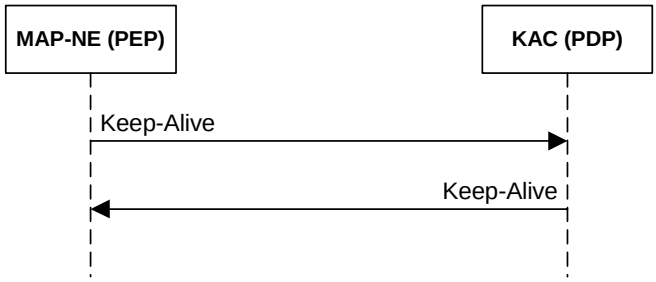


Figure 5: Keep Alive procedure

8.2.6 Manual security association management

If some PLMN supports only manual SA management then it must be possible to manually configure the parameters to KAC. Delivery of SA to MAP-NEs is handled same way as in automatic SA management.

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

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

Copyright Notice

Copyright (C) The Internet Society (2000). All Rights Reserved.

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

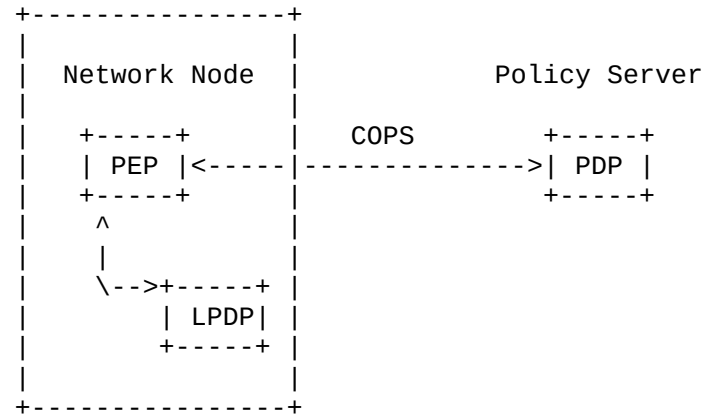


Figure 1: A COPS illustration.

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.

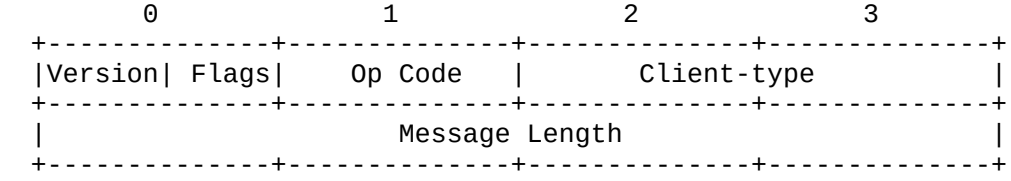
Finally, fault tolerance is a required capability for this protocol, particularly due to the fact it is associated with the security and service management of distributed network devices. Fault tolerance can be achieved by having both the PEP and remote PDP constantly verify their connection to each 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.



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)

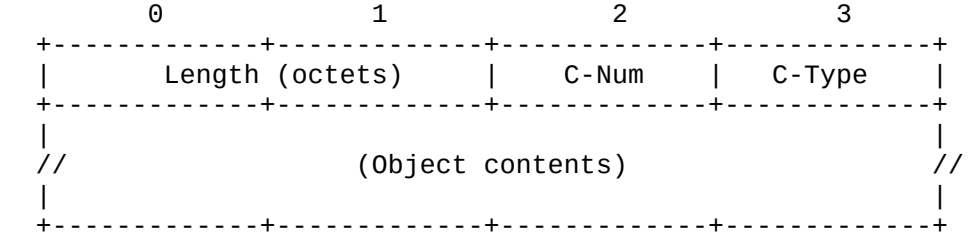
Client-type: 16 bits

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:



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.

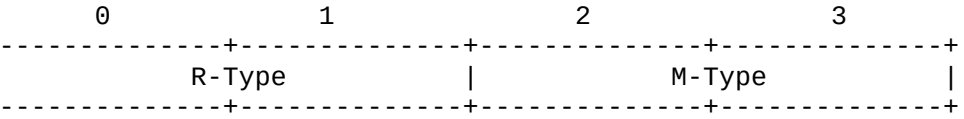
The client handle value is set by the PEP and is opaque to the PDP. The PDP simply performs a byte-wise comparison on the value in this object with respect to the handle object values of other currently installed requests.

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



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)

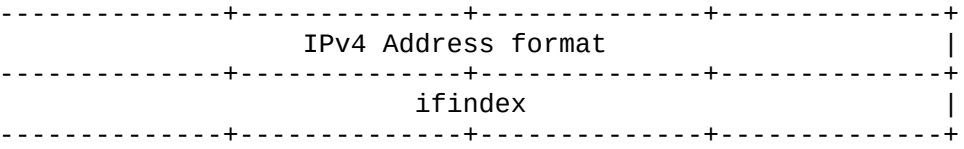
The In-Interface Object is used to identify the incoming interface on which a particular request applies and the address where the received message originated. For flows or messages generated from the PEP's local host, the loop back address and ifindex are used.

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.

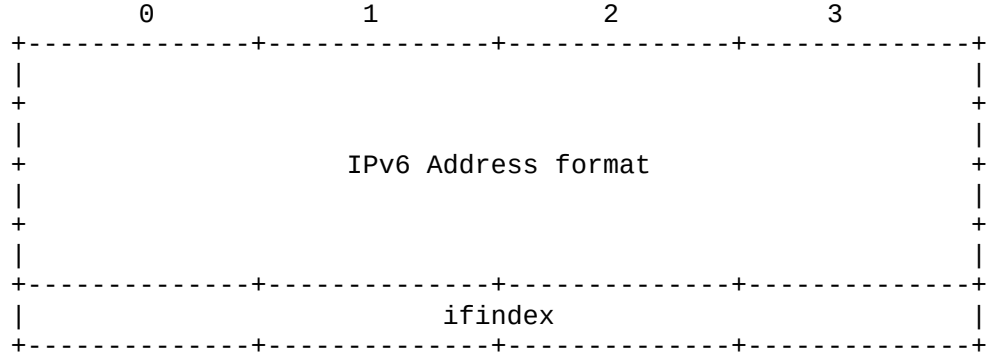
C-Num = 3

C-Type = 1, IPv4 Address + Interface



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



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.

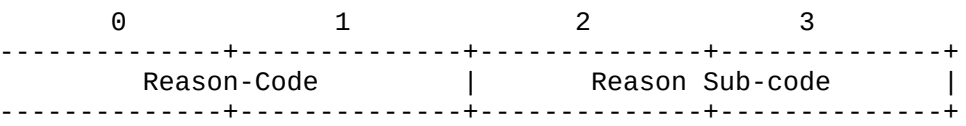
C-Type = 2, IPv6 Address + Interface

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



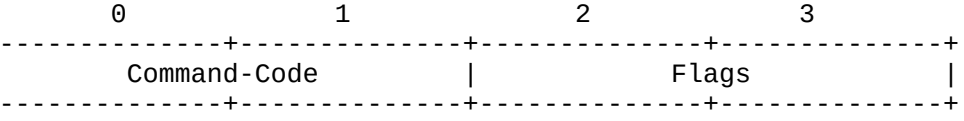
Reason Code:

- 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)



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.

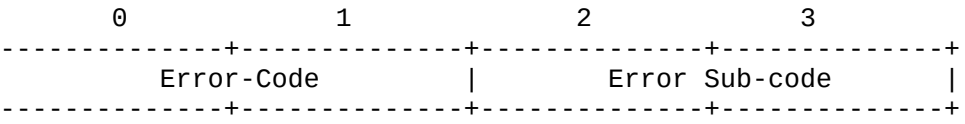
C-Num = 7

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



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.

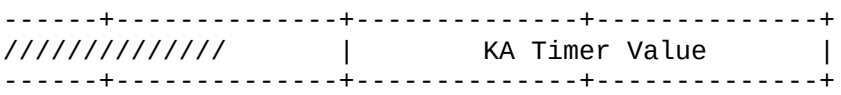
2.2.10 Keep-Alive Timer Object (KATimer)

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.



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.

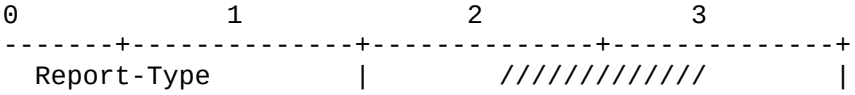
C-Num = 11, C-Type = 1

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



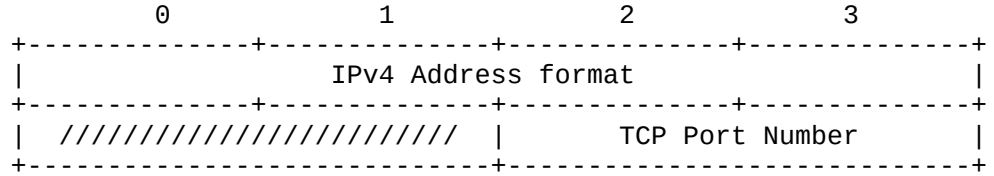
Report-Type:
1 = Success : Decision was successful at the PEP
2 = Failure : Decision could not be completed by PEP
3 = Accounting: Accounting update for an installed state

2.2.13 PDP Redirect Address (PDPRedirAddr)

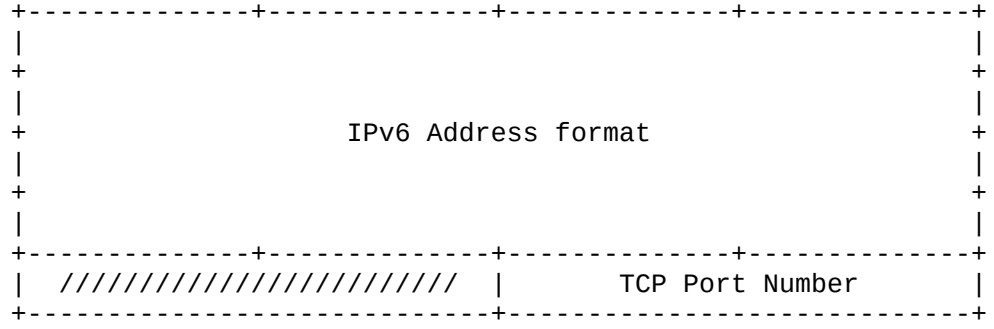
A PDP when closing a PEP session for a particular client-type may optionally use this object to redirect the PEP to the specified PDP server address and TCP port number:

C-Num = 13,

C-Type = 1, IPv4 Address + TCP Port



C-Type = 2, IPv6 Address + TCP Port



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)

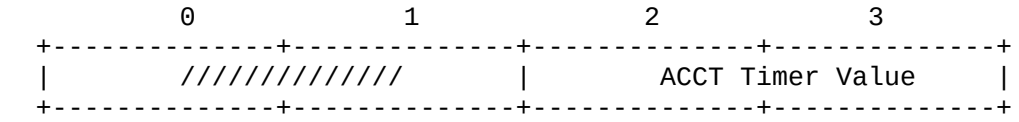
2.2.15 Accounting Timer Object (AcctTimer)

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.



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

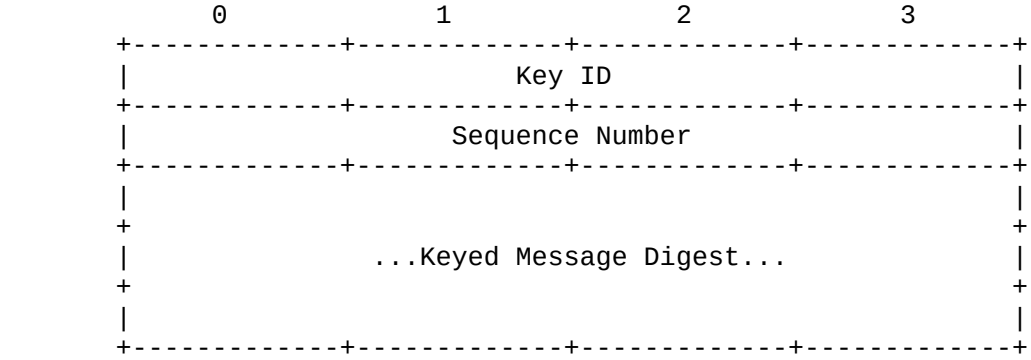
The HMAC integrity object employs HMAC (Keyed-Hashing for Message Authentication) [HMAC] to calculate the message digest based on a key shared between the PEP and its PDP.

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.



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.

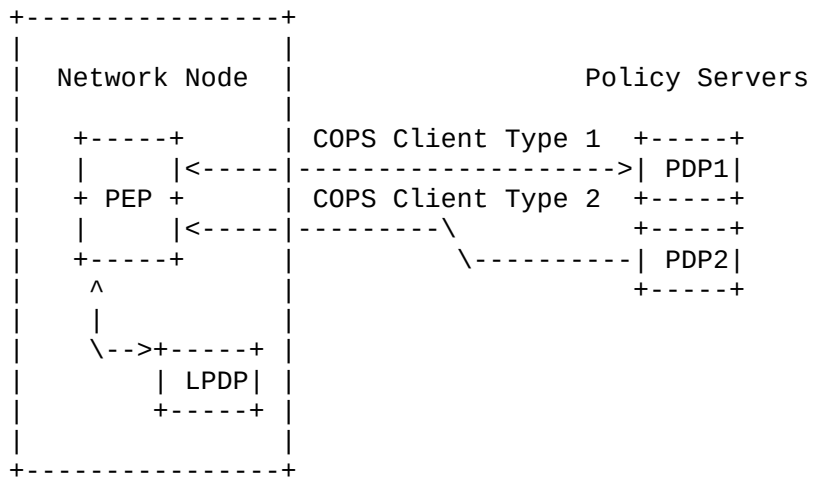


Figure 2: Multiple PDPs illustration.

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

Copyright (C) The Internet Society (2000). All Rights Reserved.

This document and translations of it may be copied and furnished to others, and derivative works that comment on or otherwise explain it or assist in its implementation may be prepared, copied, published and distributed, in whole or in part, without restriction of any kind, provided that the above copyright notice and this paragraph are included on all such copies and derivative works. However, this document itself may not be modified in any way, such as by removing the copyright notice or references to the Internet Society or other Internet organizations, except as needed for the purpose of developing Internet standards in which case the procedures for copyrights defined in the Internet Standards process must be followed, or as required to translate it into languages other than English.

The limited permissions granted above are perpetual and will not be revoked by the Internet Society or its successors or assigns.

This document and the information contained herein is provided on an "AS IS" basis and THE INTERNET SOCIETY AND THE INTERNET ENGINEERING TASK FORCE DISCLAIMS ALL WARRANTIES, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO ANY WARRANTY THAT THE USE OF THE INFORMATION HEREIN WILL NOT INFRINGE ANY RIGHTS OR ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.

Acknowledgement

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

Network Working Group
Request for Comments: 3084
Category: Standards Track

K. Chan
J. Seligson
Nortel Networks
D. Durham
Intel
S. Gai
K. McCloghrie
Cisco
S. Herzog
IPHighway
F. Reichmeyer
PFN
R. Yavatkar
Intel
A. Smith
Allegro Networks
March 2001

COPS Usage for Policy Provisioning (COPS-PR)

Status of this Memo

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

Copyright Notice

Copyright (C) The Internet Society (2001). All Rights Reserved.

Abstract

This document describes the use of the Common Open Policy Service (COPS) protocol for support of policy provisioning (COPS-PR). This specification is independent of the type of policy being provisioned (QoS, Security, etc.) but focuses on the mechanisms and conventions used to communicate provisioned information between PDPs and PEPs. The protocol extensions described in this document do not make any assumptions about the policy data model being communicated, but describe the message formats and objects that carry the modeled policy data.

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].

Table of Contents

Glossary

PRC	Provisioning Class. A type of policy data.
PRI	Provisioning Instance. An instance of a PRC.
PIB	Policy Information Base. The database of policy information.
PDP	Policy Decision Point. See [RAP].
PEP	Policy Enforcement Point. See [RAP].
PRID	Provisioning Instance Identifier. Uniquely identifies an instance of a PRC.

1. Introduction

The IETF Resource Allocation Protocol (RAP) WG has defined the COPS (Common Open Policy Service) protocol [COPS] as a scalable protocol that allows policy servers (PDPs) to communicate policy decisions to network devices (PEPs). COPS was designed to support multiple types of policy clients.

COPS is a query/response protocol that supports two common models for policy control: Outsourcing and Configuration.

The Outsourcing model addresses the kind of events at the PEP that require an instantaneous policy decision (authorization). In the outsourcing scenario, the PEP delegates responsibility to an external policy server (PDP) to make decisions on its behalf. For example, in COPS Usage for RSVP [COPRSVP] when a RSVP reservation message arrives, the PEP must decide whether to admit or reject the request. It can outsource this decision by sending a specific query to its PDP, waiting for its decision before admitting the outstanding reservation.

The COPS Configuration model (herein described as the Provisioning model), on the other hand, makes no assumptions of such direct 1:1 correlation between PEP events and PDP decisions. The PDP may proactively provision the PEP reacting to external events (such as user input), PEP events, and any combination thereof (N:M correlation). Provisioning may be performed in bulk (e.g., entire router QoS configuration) or in portions (e.g., updating a DiffServ marking filter).

Network resources are often provisioned based on relatively static SLAs (Service Level Agreements) at network boundaries. While the Outsourcing model is dynamically paced by the PEP in real-time, the Provisioning model is paced by the PDP in somewhat flexible timing over a wide range of configurable aspects of the PEP.

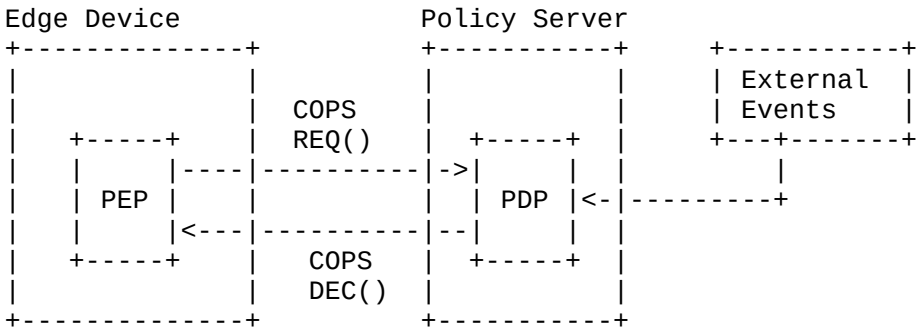


Figure 1: COPS Provisioning Model

In COPS-PR, policy requests describe the PEP and its configurable parameters (rather than an operational event). If a change occurs in these basic parameters, an updated request is sent. Hence, requests are issued quite infrequently. Decisions are not necessarily mapped directly to requests, and are issued mostly when the PDP responds to external events or PDP events (policy/SLA updates).

This document describes the use of the COPS protocol [COPS] for support of policy provisioning. This specification is independent of the type of policy being provisioned (QoS, Security, etc.). Rather, it focuses on the mechanisms and conventions used to communicate provisioned information between PDPs and PEPs. The data model assumed in this document is based on the concept of Policy Information Bases (PIBs) that define the policy data. There may be one or more PIBs for given area of policy and different areas of policy may have different sets of PIBs.

In order to support a model that includes multiple PDPs controlling non-overlapping areas of policy on a single PEP, the client-type specified by the PEP to the PDP is unique for the area of policy being managed. A single client-type for a given area of policy (e.g., QoS) will be used for all PIBs that exist in that area. The client should treat all the COPS-PR client-types it supports as non-overlapping and independent namespaces where instances MUST NOT be shared.

The examples used in this document are biased toward QoS Policy Provisioning in a Differentiated Services (DiffServ) environment. However, COPS-PR can be used for other types of provisioning policies under the same framework.

1.1. Why COPS for Provisioning?

COPS-PR has been designed within a framework that is optimized for efficiently provisioning policies across devices, based on the requirements defined in [RAP]. First, COPS-PR allows for efficient transport of attributes, large atomic transactions of data, and efficient and flexible error reporting. Second, as it has a single connection between the policy client and server per area of policy control identified by a COPS Client-Type, it guarantees only one server updates a particular policy configuration at any given time. Such a policy configuration is effectively locked, even from local console configuration, while the PEP is connected to a PDP via COPS. COPS uses reliable TCP transport and, thus, uses a state sharing/synchronization mechanism and exchanges differential updates only. If either the server or client are rebooted (or restarted) the other would know about it quickly. Last, it is defined as a real-time event-driven communications mechanism, never requiring polling between the PEP and PDP.

1.2. Interaction between the PEP and PDP

When a device boots, it opens a COPS connection to its Primary PDP. When the connection is established, the PEP sends information about itself to the PDP in the form of a configuration request. This information includes client specific information (e.g., hardware type, software release, configuration information). During this phase the client may also specify the maximum COPS-PR message size supported.

In response, the PDP downloads all provisioned policies that are currently relevant to that device. On receiving the provisioned policies, the device maps them into its local QoS mechanisms, and installs them. If conditions change at the PDP such that the PDP detects that changes are required in the provisioned policies currently in effect, then the PDP sends the changes (installs, updates, and/or deletes) in policy to the PEP, and the PEP updates its local configuration appropriately.

If, subsequently, the configuration of the device changes (board removed, board added, new software installed, etc.) in ways not covered by policies already known to the PEP, then the PEP asynchronously sends this unsolicited new information to the PDP in an updated configuration request. On receiving this new information, the PDP sends to the PEP any additional provisioned policies now needed by the PEP, or removes those policies that are no longer required.

2. Policy Information Base (PIB)

The data carried by COPS-PR is a set of policy data. The protocol assumes a named data structure, known as a Policy Information Base (PIB), to identify the type and purpose of unsolicited policy information that is "pushed" from the PDP to the PEP for provisioning policy or sent to the PDP from the PEP as a notification. The PIB name space is common to both the PEP and the PDP and data instances within this space are unique within the scope of a given Client-Type and Request-State per TCP connection between a PEP and PDP. Note that given a device might implement multiple COPS Client-Types, a unique instance space is to be provided for each separate Client-Type. There is no sharing of instance data across the Client-Types implemented by a PEP, even if the classes being instantiated are of the same type and share the same instance identifier.

The PIB can be described as a conceptual tree namespace where the branches of the tree represent structures of data or Provisioning Classes (PRCs), while the leaves represent various instantiations of Provisioning Instances (PRIs). There may be multiple data instances (PRIs) for any given data structure (PRC). For example, if one wanted to install multiple access control filters, the PRC might represent a generic access control filter type and each PRI might represent an individual access control filter to be applied. The tree might be represented as follows:

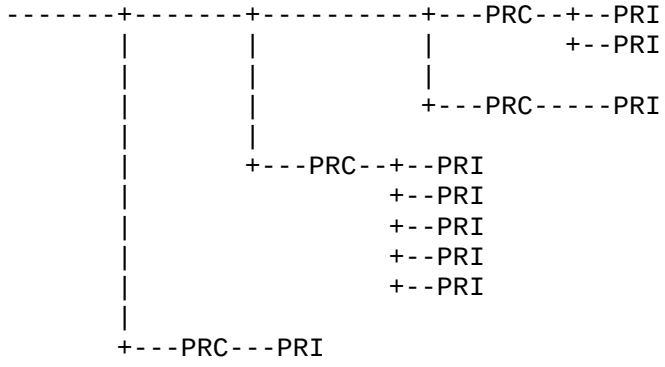


Figure 2: The PIB Tree

Instances of the policy classes (PRIs) are each identified by a Provisioning Instance Identifier (PRID). A PRID is a name, carried in a COPS <Named ClientSI> or <Named Decision Data> object, which identifies a particular instance of a class.

2.1. Rules for Modifying and Extending PIBs

As experience is gained with policy based management, and as new requirements arise, it will be necessary to make changes to PIBs. Changes to an existing PIB can be made in several ways.

(1) Additional PRCs can be added to a PIB or an existing one deprecated.

(2) Attributes can be added to, or deprecated from, an existing PRC.

(3) An existing PRC can be extended or augmented with a new PRC defined in another (perhaps enterprise specific) PIB.

The rules for each of these extension mechanisms is described in this sub-section. All of these mechanisms for modifying a PIB allow for interoperability between PDPs and PEPs even when one party is using a new version of the PIB while the other is using an old version.

Note that the SPPI [SPPI] provides the authoritative rules for updating BER encoded PIBs. It is the purpose of the following section to explain how such changes affect senders and receivers of COPS messages.

2.2. Adding PRCs to, or deprecating from, a PIB

A published PIB can be extended with new PRCs by simply revising the document and adding additional PRCs. These additional PRCs are easily identified with new PRIDs under the module’s PRID Prefix.

In the event that a PEP implementing the new PIB is being configured by a PDP implementing the old PIB, the PEP will simply not receive any instances of the new PRC. In the event that the PEP is implementing the old PIB and the PDP the new one, the PEP may receive PRIs for the new PRC. Under such conditions, the PEP MUST return an error to the PDP, and rollback to its previous (good) state.

Similarly, existing PRCs can be deprecated from a PIB. In this case, the PEP ignores any PRIs sent to it by a PDP implementing the old (non-deprecated) version of the PIB. A PDP implementing the new version of the PIB simply does not send any instances of the deprecated class.

2.2.1. Adding or Deprecating Attributes of a BER Encoded PRC

A PIB can be modified to deprecate existing attributes of a PRC or add new ones.

When deprecating the attributes of a PRC, it must be remembered that, with the COPS-PR protocol, the attributes of the PRC are identified by their order in the sequence rather than an explicit label (or attribute OID). Consequently, an ASN.1 value MUST be sent even for deprecated attributes so that a PDP and PEP implementing different versions of the PIB are inter-operable.

For a deprecated attribute, if the PDP is using a BER encoded PIB, the PDP MUST send either an ASN.1 value of the correct type, or it may send an ASN.1 NULL value. A PEP that receives an ASN.1 NULL for an attribute that is not deprecated SHOULD substitute a default value. If it has no default value to substitute it MUST return an error to the PDP.

When adding new attributes to a PIB, these new attributes must be added in sequence after the existing ones. A PEP that receives a PRI with more attributes than it is expecting MUST ignore the additional attributes and send a warning back to the PDP.

A PEP that receives a PRI with fewer attributes than it is expecting SHOULD assume default values for the missing attributes. It MAY send a warning back to the PDP. If the missing attributes are required and there is no suitable default, the PEP MUST send an error back to the PDP. In all cases the missing attributes are assumed to correspond to the last attributes of the PRC.

2.3. COPS Operations Supported for a Provisioning Instance

A Provisioning Instance (PRI) typically contains a value for each attribute defined for the PRC of which it is an instance and is identified uniquely, within the scope of a given COPS Client-Type and Request-State on a PEP, by a Provisioning Instance Identifier (PRID). The following COPS operations are supported on a PRI:

- Install - This operation creates or updates a named instance of a PRC. It includes two parameters: a PRID object to name the PRI and an Encoded Provisioning Instance Data (EPD) object with the new/updated values. The PRID value MUST uniquely identify a single PRI (i.e., PRID prefix or PRC values are illegal). Updates to an existing PRI are achieved by simply reinstalling the same PRID with the updated EPD data.

- Remove - This operation is used to delete an instance of a PRC. It includes one parameter, a PRID object, which names either the individual PRI to be deleted or a PRID prefix naming one or more complete classes of PRIs. Prefix-based deletion supports efficient bulk policy removal. The removal of an unknown/non-existent PRID SHOULD result in a warning to the PDP (no error).

3. Message Content

The COPS protocol provides for different COPS clients to define their own "named", i.e., client-specific, information for various messages. This section describes the messages exchanged between a COPS server (PDP) and COPS Policy Provisioning clients (PEP) that carry client-specific data objects. All the COPS messages used by COPS-PR conform to the message specifications defined in the COPS base protocol [COPS].

Note: The use of the '*' character represented throughout this document is consistent with the ABNF [RFC2234] and means 0 or more of the following entities.

3.1. Request (REQ) PEP -> PDP

The REQ message is sent by policy provisioning clients to issue a 'configuration request' to the PDP as specified in the COPS Context Object. The Client Handle associated with the REQ message originated by a provisioning client MUST be unique for that client. The Client Handle is used to identify a specific request state. Thus, one client can potentially open several configuration request states, each uniquely identified by its handle. Different request states are used to isolate similarly named configuration information into non-overlapping contexts (or logically isolated namespaces). Thus, an instance of named information is unique relative to a particular client-type and is unique relative to a particular request state for that client-type, even if the information was similarly identified in other request states (i.e., uses the same PRID). Thus, the Client Handle is also part of the instance identification of the communicated configuration information.

The configuration request message serves as a request from the PEP to the PDP for provisioning policy data that the PDP may have for the PEP, such as access control lists, etc. This includes policy the PDP may have at the time the REQ is received as well as any future policy data or updates to this data.

The configuration request message should include provisioning client information to provide the PDP with client-specific configuration or capability information about the PEP. The information provided by

the PEP should include client resources (e.g., queuing capabilities) and default policy configuration (e.g., default role combinations) information as well as incarnation data on existing policy. This information typically does not include all the information previously installed by a PDP but rather should include checksums or shortened references to previously installed information for synchronization purposes. This information from the client assists the server in deciding what types of policy the PEP can install and enforce. The format of the information encapsulated in one or more of the COPS Named ClientSI objects is described in section 5. Note that the configuration request message(s) is generated and sent to the PDP in response to the receipt of a Synchronize State Request (SSQ) message from the PDP. Likewise, an updated configuration request message (using the same Client Handle value as the original request now being updated) may also be generated by the PEP and sent to the PDP at any time due to local modifications of the PEP's internal state. In this way, the PDP will be synchronized with the PEP's relevant internal state at all times.

The policy information supplied by the PDP MUST be consistent with the named decision data defined for the policy provisioning client. The PDP responds to the configuration request with a DEC message containing any available provisioning policy data.

The REQ message has the following format:

```
<Request> ::= <Common Header>
              <Client Handle>
              <Context = config request>
              *(<Named ClientSI>)
              [<Integrity>]
```

Note that the COPS objects IN-Int, OUT-Int and LPDPDecisions are not included in a COPS-PR Request.

3.2. Decision (DEC) PDP -> PEP

The DEC message is sent from the PDP to a policy provisioning client in response to the REQ message received from the PEP. The Client Handle MUST be the same Handle that was received in the corresponding REQ message.

The DEC message is sent as an immediate response to a configuration request with the solicited message flag set in the COPS message header. Subsequent DEC messages may also be sent at any time after the original DEC message to supply the PEP with additional/updated policy information without the solicited message flag set in the COPS message header (as they are unsolicited decisions).

Each DEC message may contain multiple decisions. This means a single message can install some policies and delete others. In general a single COPS-PR DEC message MUST contain any required remove decisions first, followed by any required install decisions. This is used to solve a precedence issue, not a timing issue: the remove decision deletes what it specifies, except those items that are installed in the same message.

The DEC message can also be used by the PDP to command the PEP to open a new Request State or Delete an existing Request-State as identified by the Client-Handle. To accomplish this, COPS-PR defines a new flag for the COPS Decision Flags object. The flag 0x02 is to be used by COPS-PR client-types and is hereafter referred to as the "Request-State" flag. An Install decision (Decision Flags: Command-Code=Install) with the Request-State flag set in the COPS Decision Flags object will cause the PEP to issue a new Request with a new Client Handle or else specify the appropriate error in a COPS Report message. A Remove decision (Decision Flags: Command-Code=Remove) with the Request-State flag set in the COPS Decision Flags object will cause the PEP to send a COPS Delete Request State (DRQ) message for the Request-State identified by the Client Handle in the DEC message. Whenever the Request-State flag is set in the COPS Decision Flags object in the DEC message, no COPS Named Decision Data object can be included in the corresponding decision (as it serves no purpose for this decision flag). Note that only one decision with the Request-State flag can be present per DEC message, and, if present, this MUST be the only decision in that message. As described below, the PEP MUST respond to each and every DEC with a corresponding solicited RPT.

A COPS-PR DEC message MUST be treated as a single "transaction", i.e., either all the decisions in a DEC message succeed or they all fail. If they fail, the PEP will rollback to its previous good state, which is the last successful DEC transaction, if any. This allows the PDP to delete some policies only if other policies can be installed in their place. The DEC message has the following format:

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

<Decision> ::= <Context>
               <Decision: Flags>
               [<Named Decision Data: Provisioning >]
```

Note that the Named Decision Data (Provisioning) object is included in a COPS-PR Decision when it is an Install or Remove decision with no Decision Flags set. Other types of COPS decision data objects (e.g., Stateless, Replacement) are not supported by COPS-PR client-types. The Named Decision Data object MUST NOT be included in the decision if the Decision Flags object Command-Code is NULL (meaning there is no configuration information to install at this time) or if the Request-State flag is set in the Decision Flags object.

For each decision in the DEC message, the PEP performs the operation specified in the Command-Code and Flags field in the Decision Flags object on the Named Decision Data. For the policy provisioning clients, the format for this data is defined in the context of the Policy Information Base (see section 5). In response to a DEC message, the policy provisioning client MUST send a RPT message, with the solicited message flag set, back to the PDP to inform the PDP of the action taken.

3.3. Report State (RPT) PEP -> PDP

The RPT message is sent from the policy provisioning clients to the PDP to report accounting information associated with the provisioned policy, or to notify the PDP of changes in the PEP (Report-Type = ' Accounting') related to the provisioning client.

RPT is also used as a mechanism to inform the PDP about the action taken at the PEP in response to a DEC message. For example, in response to an 'Install' decision, the PEP informs the PDP if the policy data is installed (Report-Type = 'Success') or not (Report-Type = 'Failure'). Reports that are in response to a DEC message MUST set the solicited message flag in their COPS message header. Each solicited RTP MUST be sent for its corresponding DEC in the order the DEC messages were received. In case of a solicited failure, the PEP is expected to rollback to its previous (good) state as if the erroneous DEC transaction did not occur. The PEP MUST always respond to a DEC with a solicited RPT even in response to a NULL DEC, in which case the Report-Type will be 'Success'.

Reports can also be unsolicited and all unsolicited Reports MUST NOT set the solicited message flag in their COPS message header. Examples of unsolicited reports include 'Accounting' Report-Types, which were not triggered by a specific DEC messages, or 'Failure' Report-Types, which indicate a failure in a previously successfully installed configuration (note that, in the case of such unsolicited failures, the PEP cannot rollback to a previous "good" state as it becomes ambiguous under these asynchronous conditions what the correct state might be).

The RPT message may contain provisioning client information such as accounting parameters or errors/warnings related to a decision. The data format for this information is defined in the context of the policy information base (see section 5). The RPT message has the following format:

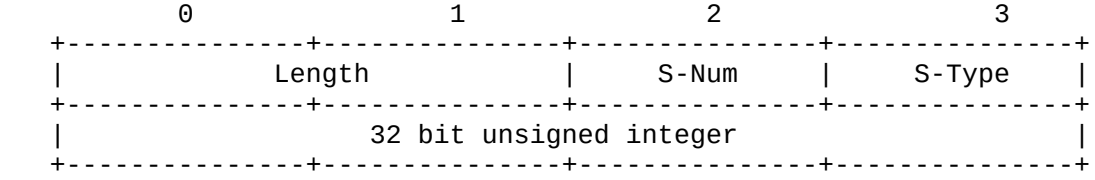
```
<Report State> ::= <Common Header>
                   <Client Handle>
                   <Report Type>
                   *(<Named ClientSI>)
                   [<Integrity>]
```

4. COPS-PR Protocol Objects

The COPS Policy Provisioning clients encapsulate several new objects within the existing COPS Named Client-specific information object and Named Decision Data object. This section defines the format of these new objects.

COPS-PR classifies policy data according to "bindings", where a binding consists of a Provisioning Instance Identifier and the Provisioning Instance data, encoded within the context of the provisioning policy information base (see section 5).

The format for these new objects is as follows:



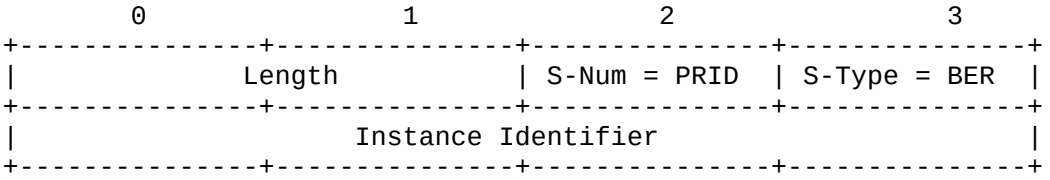
S-Num and S-Type are similar to the C-Num and C-Type used in the base COPS objects. The difference is that S-Num and S-Type are used only for COPS-PR clients and are encapsulated within the existing COPS Named ClientSI or Named Decision Data objects. The S-Num identifies the general purpose of the object, and the S-Type describes the specific encoding used for the object. All the object descriptions and examples in this document use the Basic Encoding Rules as the encoding type (S-Type = 1). Additional encodings can be defined for the remaining S-Types in the future (for example, an additional S-Type could be used to carry XML string based encodings [XML] as an EPD of PRI instance data, where URNs identify PRCs [URN] and XPointers would be used for PRIDs).

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 stated object length of the current object to the next 32-bit boundary. The values for the padding MUST be all zeros.

4.1. Complete Provisioning Instance Identifier (PRID)

S-Num = 1 (Complete PRID), S-Type = 1 (BER), Length = variable.

This object is used to carry the identifier, or PRID, of a Provisioning Instance. The identifier is encoded following the rules that have been defined for encoding SNMP Object Identifier (OID) values. Specifically, PRID values are encoded using the Type/Length/Value (TLV) format and initial sub-identifier packing that is specified by the binary encoding rules [BER] used for Object Identifiers in an SNMP PDU.



For example, a (fictitious) PRID equal to 1.3.6.1.2.2.8.1 would be encoded as follows (values in hex):

```
06 07 2B 06 01 02 02 08 01
```

The entire PRID object would be encoded as follows:

```
00 0D                       - Length
01                          - S-Num
01                          - S-Type (Complete PRID)
06 07 2B 06 01 02 02 08 01  - Encoded PRID
00 00 00                    - Padding
```

NOTE: When encoding an xxxTable's xxxEntry Object-Type as defined by the SMI [V2SMI] and SPPI [SPPI], the OID will contain all the sub-identifiers up to and including the xxxEntry OID but not the columnar identifiers for the attributes within the xxxEntry's SEQUENCE. The last (suffix) identifier is the INDEX of an instance of an entire

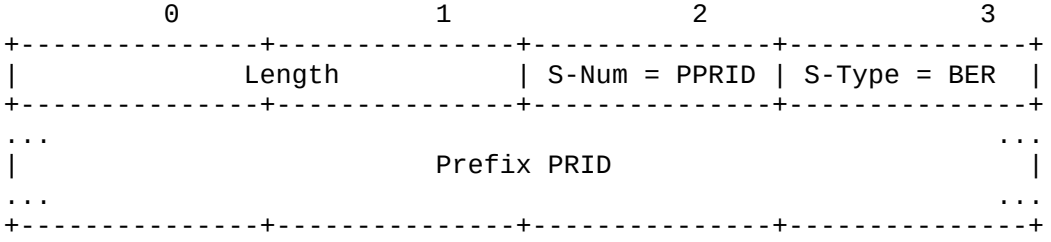
xxxEntry including its SEQUENCE of attributes encoded in the EPD (defined below). This constitutes an instance (PRI) of a class (PRC) in terms of the SMI.

A PRID for a scalar (non-columnar) value's OID is encoded directly as the PRC where the instance identifier suffix is always zero as there will be only one instance of a scalar value. The EPD will then be used to convey the scalar value.

4.2. Prefix PRID (PPRID)

Certain operations, such as decision removal, can be optimized by specifying a PRID prefix with the intent that the requested operation be applied to all PRIs matching the prefix (for example, all instances of the same PRC). PRID prefix objects MUST only be used in the COPS protocol <Remove Decision> operation where it may be more optimal to perform bulk decision removal using class prefixes instead of a sequence of individual <Remove Decision> operations. Other COPS operations, e.g., <Install Decision> operations always require individual PRID specification.

S-Num = 2 (Prefix PRID), S-Type = 1 (BER), Length = variable.



Continuing with the previous example, a prefix PRID that is equal to 1.3.6.1.2.2 would be encoded as follows (values in hex):

06 05 2B 06 01 02 02

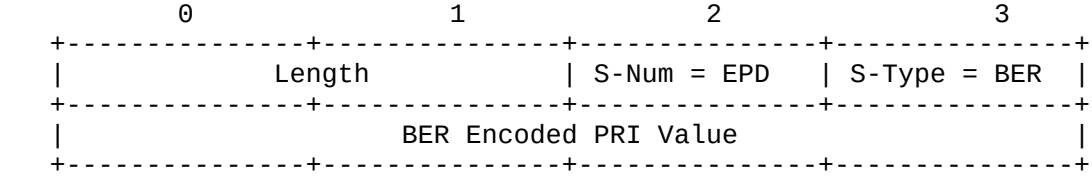
The entire PPRID object would be encoded as follows:

```
00 0B                       - Length
02                          - S-Num = Prefix PRID
01                          - S-Type = BER
06 05 2B 06 01 02 02        - Encoded Prefix PRID
00                          - Padding
```

4.3. Encoded Provisioning Instance Data (EPD)

S-Num = 3 (EPD), S-Type = 1 (BER), Length = variable.

This object is used to carry the encoded value of a Provisioning Instance. The PRI value, which contains all of the individual values of the attributes that comprise the class (which corresponds to the SMI's xxxEntry Object-Type defining the SEQUENCE of attributes comprising a table [V2SMI][SPPI]), is encoded as a series of TLV sub-components. Each sub-component represents the value of a single attribute and is encoded following the BER. Note that the ordering of non-scalar (multiple) attributes within the EPD is dictated by their respective columnar OID suffix when defined in [V2SMI]. Thus, the attribute with the smallest columnar OID suffix will appear first and the attribute with the highest number columnar OID suffix will be last.



As an example, a fictional definition of an IPv4 packet filter class could be described using the SMI as follows:

```
ipv4FilterIpFilter OBJECT IDENTIFIER ::= { someExampleOID 1 }

-- The IP Filter Table

ipv4FilterTable OBJECT-TYPE
    SYNTAX         SEQUENCE OF Ipv4FilterEntry
    MAX-ACCESS     not-accessible
    STATUS         current
    DESCRIPTION
        "Filter definitions.  A packet has to match all fields in
        a filter.  Wildcards may be specified for those fields
        that are not relevant."

    ::= { ipv4FilterIpFilter 1 }

ipv4FilterEntry OBJECT-TYPE
    SYNTAX         Ipv4FilterEntry
    MAX-ACCESS     not-accessible
    STATUS         current
    DESCRIPTION
        "An instance of the filter class."
```

```
        INDEX { ipv4FilterIndex }

        ::= { ipv4FilterTable 1 }

    Ipv4FilterEntry ::= SEQUENCE {
            ipv4FilterIndex        Unsigned32,
            ipv4FilterDstAddr      IpAddress,
            ipv4FilterDstAddrMask  IpAddress,
            ipv4FilterSrcAddr      IpAddress,
            ipv4FilterSrcAddrMask  IpAddress,
            ipv4FilterDscp         Integer32,
            ipv4FilterProtocol     Integer32,
            ipv4FilterDstL4PortMin Integer32,
            ipv4FilterDstL4PortMax Integer32,
            ipv4FilterSrcL4PortMin Integer32,
            ipv4FilterSrcL4PortMax Integer32,
            ipv4FilterPermit       TruthValue
    }

    ipv4FilterIndex OBJECT-TYPE
        SYNTAX         Unsigned32
        MAX-ACCESS     read-write
        STATUS         current
        DESCRIPTION
         "An integer index to uniquely identify this filter among all
          the filters."

        ::= { ipv4FilterEntry 1 }

    ipv4FilterDstAddr OBJECT-TYPE

        SYNTAX         IpAddress
        MAX-ACCESS     read-write
        STATUS         current
        DESCRIPTION
         "The IP address to match against the packet's destination IP
         address."

        ::= { ipv4FilterEntry 2 }

    ipv4FilterDstAddrMask OBJECT-TYPE
        SYNTAX         IpAddress
        MAX-ACCESS     read-write
        STATUS         current
        DESCRIPTION
         "A mask for the matching of the destination IP address.
         A zero bit in the mask means that the corresponding bit in
```

```
         the address always matches."

        ::= { ipv4FilterEntry 3 }

    ipv4FilterSrcAddr OBJECT-TYPE
        SYNTAX          IpAddress
        MAX-ACCESS      read-write
        STATUS          current
        DESCRIPTION
            "The IP address to match against the packet's source IP
            address."

        ::= { ipv4FilterEntry 4 }

    ipv4FilterSrcAddrMask OBJECT-TYPE
        SYNTAX          IpAddress
        MAX-ACCESS      read-write
        STATUS          current
        DESCRIPTION
            "A mask for the matching of the source IP address."

        ::= { ipv4FilterEntry 5 }

    ipv4FilterDscp OBJECT-TYPE
        SYNTAX          Integer32 (-1 | 0..63)
        MAX-ACCESS      read-write
        STATUS          current
        DESCRIPTION
            "The value that the DSCP in the packet can have and
            match.  A value of -1 indicates that a specific
            DSCP value has not been defined and thus all DSCP values
            are considered a match."

        ::= { ipv4FilterEntry 6 }

    ipv4FilterProtocol OBJECT-TYPE
        SYNTAX          Integer32 (0..255)
        MAX-ACCESS      read-write
        STATUS          current
        DESCRIPTION
            "The IP protocol to match against the packet's protocol.
            A value of zero means match all."

        ::= { ipv4FilterEntry 7 }

    ipv4FilterDstL4PortMin OBJECT-TYPE
        SYNTAX          Integer32 (0..65535)
        MAX-ACCESS      read-write
```

```
        STATUS         current
        DESCRIPTION
            "The minimum value that the packet's layer 4 destination
            port number can have and match this filter."

        ::= { ipv4FilterEntry 8 }

    ipv4FilterDstL4PortMax OBJECT-TYPE
        SYNTAX         Integer32 (0..65535)
        MAX-ACCESS     read-write
        STATUS         current
        DESCRIPTION
            "The maximum value that the packet's layer 4 destination
            port number can have and match this filter."

        ::= { ipv4FilterEntry 9 }

    ipv4FilterSrcL4PortMin OBJECT-TYPE
        SYNTAX         Integer32 (0..65535)
        MAX-ACCESS     read-write
        STATUS         current
        DESCRIPTION
            "The minimum value that the packet's layer 4 source port
            number can have and match this filter."

        ::= { ipv4FilterEntry 10 }

    ipv4FilterSrcL4PortMax OBJECT-TYPE
        SYNTAX         Integer32 (0..65535)
        MAX-ACCESS     read-write
        STATUS         current
        DESCRIPTION
            "The maximum value that the packet's layer 4 source port
            number can have and match this filter."

        ::= { ipv4FilterEntry 11 }

    ipv4FilterPermit OBJECT-TYPE
        SYNTAX         TruthValue
        MAX-ACCESS     read-write
        STATUS         current
        DESCRIPTION
            "If false, the evaluation is negated.  That is, a
            valid match will be evaluated as not a match and vice
            versa."

        ::= { ipv4FilterEntry 12 }
```

A fictional instance of the filter class defined above might then be encoded as follows:

```
02 01 08          :ipv4FilterIndex/Unsigned32/Value = 8
40 04 C0 39 01 05 :ipv4FilterDstAddr/IpAddress/Value = 192.57.1.5
40 04 FF FF FF FF :ipv4FilterDstMask/IpAddress/Value=255.255.255.255
40 04 00 00 00 00 :ipv4FilterSrcAddr/IpAddress/Value = 0.0.0.0
40 04 00 00 00 00 :ipv4FilterSrcMask/IpAddress/Value = 0.0.0.0
02 01 FF          :ipv4FilterDscp/Integer32/Value = -1 (not used)
02 01 06          :ipv4FilterProtocol/Integer32/Value = 6 (TCP)
05 00             :ipv4FilterDstL4PortMin/NULL/not supported
05 00             :ipv4FilterDstL4PortMax/NULL/not supported
05 00             :ipv4FilterSrcL4PortMin/NULL/not supported
05 00             :ipv4FilterSrcL4PortMax/NULL/not supported
02 01 01          :ipv4FilterPermit/TruthValue/Value = 1 (true)
```

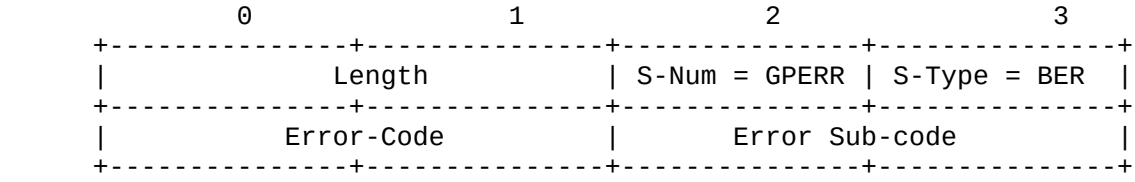
The entire EPD object for this instance would then be encoded as follows:

```
00 30                         - Length
03                            - S-Num = EPD
01                            - S-Type = BER
02 01 08                      - ipv4FilterIndex
40 04 C0 39 01 05             - ipv4FilterDstAddr
40 04 FF FF FF FF             - ipv4FilterDstMask
40 04 00 00 00 00             - ipv4FilterSrcAddr
40 04 00 00 00 00             - ipv4FilterSrcMask
02 01 FF                      - ipv4FilterDscp
02 01 06                      - ipv4FilterProtocol
05 00                         - ipv4FilterDstL4PortMin
05 00                         - ipv4FilterDstL4PortMax
05 00                         - ipv4FilterSrcL4PortMin
05 00                         - ipv4FilterSrcL4PortMax
02 01 01                      - ipv4FilterPermit
```

Note that attributes not supported within a class are still returned in the EPD for a PRI. By convention, a NULL value is returned for attributes that are not supported. In the previous example, source and destination port number attributes are not supported.

4.4. Global Provisioning Error Object (GPERR)

S-Num = 4 (GPERR), S-Type = 1 (for BER), Length = 8.



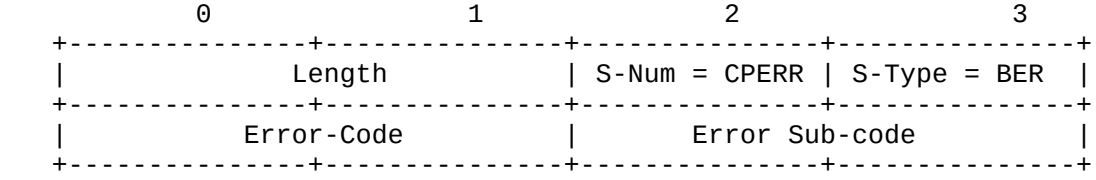
The global provisioning error object has the same format as the Error object in COPS [COPS], except with C-Num and C-Type replaced by the S-Num and S-Type values shown. The global provision error object is used to communicate general errors that do not map to a specific PRC.

The following global error codes are defined:

availMemLow(1)
availMemExhausted(2)
unknownASN.1Tag(3) - The erroneous tag type SHOULD be specified in the Error Sub-Code field.
maxMsgSizeExceeded(4) - COPS message (transaction) was too big.
unknownError(5)
maxRequestStatesOpen(6)- No more Request-States can be created by the PEP (in response to a DEC message attempting to open a new Request-State).
invalidASN.1Length(7) - An ASN.1 object length was incorrect.
invalidObjectPad(8) - Object was not properly padded.
unknownPIBData(9) - Some of the data supplied by the PDP is unknown/unsupported by the PEP (but otherwise formatted correctly). PRC specific error codes are to be used to provide more information.
unknownCOPSPRObject(10)- Sub-code (octet 2) contains unknown object's S-Num and (octet 3) contains unknown object's S-Type.
malformedDecision(11) - Decision could not be parsed.

4.5. PRC Class Provisioning Error Object (CPERR)

S-Num = 5 (CPERR), S-Type = 1 (for BER), Length = 8.



The class-specific provisioning error object has the same format as the Error object in COPS [COPS], except with C-Num and C-Type replaced by the S-Num and S-Type values shown. The class-specific error object is used to communicate errors relating to specific PRCs and MUST have an associated Error PRID Object.

The following Generic Class-Specific errors are defined:

priSpaceExhausted(1) -	no more instances may currently be installed in the given class.
priInstanceInvalid(2) -	the specified class instance is currently invalid prohibiting installation or removal.
attrValueInvalid(3) -	the specified value for identified attribute is illegal.
attrValueSupLimited(4) -	the specified value for the identified attribute is legal but not currently supported by the device.
attrEnumSupLimited(5) -	the specified enumeration for the identified attribute is legal but not currently supported by the device.
attrMaxLengthExceeded(6) -	the overall length of the specified value for the identified attribute exceeds device limitations.
attrReferenceUnknown(7) -	the class instance specified by the policy instance identifier does not exist.
priNotifyOnly(8) -	the class is currently only supported for use by request or report messages prohibiting decision installation.
unknownPrc(9) -	attempt to install a PRI of a class not supported by PEP.
tooFewAttrs(10) -	recvd PRI has fewer attributes than required.
invalidAttrType(11) -	recvd PRI has an attribute of the wrong type.

deletedInRef(12)	-	deleted PRI is still referenced by other (non) deleted PRIs
priSpecificError(13)	-	the Error Sub-code field contains the PRC specific error code

Where appropriate (errors 3, 4, 5, 6, 7 above) the error sub-code SHOULD identify the OID sub-identifier of the attribute associated with the error.

4.6. Error PRID Object (ErrorPRID)

S-Num = 6 (ErrorPRID), S-Type = 1 (BER), Length = variable.

This object is used to carry the identifier, or PRID, of a Provisioning Instance that caused an installation error or could not be installed or removed. The identifier is encoded and formatted exactly as in the PRID object as described in section 4.1.

5. COPS-PR Client-Specific Data Formats

This section describes the format of the named client specific information for the COPS policy provisioning client. ClientSI formats are defined for Decision message's Named Decision Data object, the Request message's Named ClientSI object and Report message's Named ClientSI object. The actual content of the data is defined by the policy information base for a specific provisioning client-type (see below).

5.1. Named Decision Data

The formats encapsulated by the Named Decision Data object for the policy provisioning client-types depends on the type of decision. Install and Remove are the two types of decisions that dictate the internal format of the COPS Named Decision Data object and require its presence. Install and Remove refer to the 'Install' and 'Remove' Command-Code, respectively, specified in the COPS Decision Flags Object when no Decision Flags are set. The data, in general, is composed of one or more bindings. Each binding associates a PRID object and a EPD object. The PRID object is always present in both install and remove decisions, the EPD object MUST be present in the case of an install decision and MUST NOT be present in the case of a remove decision.

The format for this data is encapsulated within the COPS Named Decision Data object as follows:

```
<Named Decision Data> ::= <<Install Decision> |
                           <Remove Decision>>

<Install Decision>    ::= *(<PRID> <EPD>)

<Remove Decision>     ::= *(<PRID>|<PPRID>)
```

Note that PRID objects in a Remove Decision may specify PRID prefix values. Explicit and implicit deletion of installed policies is supported by a client. Install Decision data MUST be explicit (i.e., PRID prefix values are illegal and MUST be rejected by a client).

5.2. ClientSI Request Data

The provisioning client request data will use same bindings as described above. The format for this data is encapsulated in the COPS Named ClientSI object as follows:

```
<Named ClientSI: Request> ::= <*(<PRID> <EPD>)>
```

5.3. Policy Provisioning Report Data

The COPS Named ClientSI object is used in the RPT message in conjunction with the accompanying COPS Report Type object to encapsulate COPS-PR report information from the PEP to the PDP. Report types can be 'Success' or 'Failure', indicating to the PDP that a particular set of provisioning policies has been either successfully or unsuccessfully installed/removed on the PEP, or 'Accounting'.

5.3.1. Success and Failure Report-Type Data Format

Report-types can be 'Success' or 'Failure' indicating to the PDP that a particular set of provisioning policies has been either successfully or unsuccessfully installed/removed on the PEP. The provisioning report data consists of the bindings described above and global and specific error/warning information. Specific errors are associated with a particular instance. For a 'Success' Report-Type, a specific error is an indication of a warning related to a specific policy that has been installed, but that is not fully implemented (e.g., its parameters have been approximated) as identified by the ErrorPRID object. For a 'Failure' Report-Type, this is an error code specific to a binding, again, identified by the ErrorPRID object. Specific errors may also include regular <PRID><EPD> bindings to

carry additional information in a generic manner so that the specific errors/warnings may be more verbosely described and associated with the erroneous ErrorPRID object.

Global errors are not tied to a specific ErrorPRID. In a 'Success' RPT message, a global error is an indication of a general warning at the PEP level (e.g., memory low). In a 'Failure' RPT message, this is an indication of a general error at the PEP level (e.g., memory exhausted).

In the case of a 'Failure' Report-Type the PEP MUST report at least the first error and SHOULD report as many errors as possible. In this case the PEP MUST roll-back its configuration to the last good transaction before the erroneous Decision message was received.

The format for this data is encapsulated in the COPS Named ClientSI object as follows:

<Named ClientSI: Report> ::= <[<GPERR>] *(<report>)>

<report> ::= <ErrorPRID> <CPERR> *(<PRID><EPD>)

5.3.2. Accounting Report-Type Data Format

Additionally, reports can be used to carry accounting information when specifying the 'Accounting' Report-Type. This accounting report message will typically carry statistical or event information related to the installed configuration for use at the PDP. This information is encoded as one or more <PRID><EPD> bindings that generally describe the accounting information being reported from the PEP to the PDP.

The format for this data is encapsulated in the COPS Named ClientSI object as follows:

<Named ClientSI: Report> ::= <*(<PRID><EPD>)>

NOTE: RFC 2748 defines an optional Accounting-Timer (AcctTimer) object for use in the COPS Client-Accept message. Periodic accounting reports for COPS-PR clients are also obligated to be paced by this timer. Periodic accounting reports SHOULD NOT be generated by the PEP more frequently than the period specified by the COPS AcctTimer. Thus, the period between new accounting reports SHOULD be greater-than or equal-to the period specified (if specified) in the AcctTimer. If no AcctTimer object is specified by the PDP, then there are no constraints imposed on the PEP's accounting interval.

6. Common Operation

This section describes, in general, typical exchanges between a PDP and Policy Provisioning COPS client.

First, a TCP connection is established between the client and server and the PEP sends a Client-Open message specifying a COPS- PR client-type (use of the ClientSI object within the Client-Open message is currently undefined for COPS-PR clients). If the PDP supports the specified provisioning client-type, the PDP responds with a Client-Accept (CAT) message. If the client-type is not supported, a Client-Close (CC) message is returned by the PDP to the PEP, possibly identifying an alternate server that is known to support the policy for the provisioning client-type specified.

After receiving the CAT message, the PEP can send requests to the server. The REQ from a policy provisioning client contains a COPS 'Configuration Request' context object and, optionally, any relevant named client specific information from the PEP. The information provided by the PEP should include available client resources (e.g., supported classes/attributes) and default policy configuration information as well as incarnation data on existing policy. The configuration request message from a provisioning client serves two purposes. First, it is a request to the PDP for any provisioning configuration data which the PDP may currently have that is suitable for the PEP, such as access control filters, etc., given the information the PEP specified in its REQ. Also, the configuration request effectively opens a channel that will allow the PDP to asynchronously send policy data to the PEP, as the PDP decides is necessary, as long as the PEP keeps its request state open (i.e., as long as the PEP does not send a DRQ with the request state's Client Handle). This asynchronous data may be new policy data or an update to policy data sent previously. Any relevant changes to the PEP's internal state can be communicated to the PDP by the PEP sending an updated REQ message. The PEP is free to send such updated REQ messages at any time after a CAT message to communicate changes in its local state.

After the PEP sends a REQ, if the PDP has Policy Provisioning policy configuration information for the client, that information is returned to the client in a DEC message containing the Policy Provisioning client policy data within the COPS Named Decision Data object and specifying an "Install" Command-Code in the Decision Flags object. If no filters are defined, the DEC message will simply specify that there are no filters using the "NULL Decision" Command-Code in the Decision Flags object. As the PEP MUST specify a Client Handle in the request message, the PDP MUST process the Client Handle and copy it in the corresponding decision message. A DEC message

MUST be issued by the PDP with the Solicited Message Flag set in the COPS message header, regardless of whether or not the PDP has any configuration information for the PEP at the time of the request. This is to prevent the PEP from timing out the REQ and deleting the Client Handle.

The PDP can then add new policy data or update/delete existing configurations by sending subsequent unsolicited DEC message(s) to the PEP, with the same Client Handle. Previous configurations installed on the PEP are updated by the PDP by simply re-installing the same instance of configuration information again (effectively overwriting the old data). The PEP is responsible for removing the Client handle when it is no longer needed, for example when an interface goes down, and informing the PDP that the Client Handle is to be deleted via the COPS DRQ message.

For Policy Provisioning purposes, access state, and access requests to the policy server can be initiated by other sources besides the PEP. Examples of other sources include attached users requesting network services via a web interface into a central management application, or H.323 servers requesting resources on behalf of a user for a video conferencing application. When such a request is accepted, the edge device affected by the decision (the point where the flow is to enter the network) needs to be informed of the decision. Since the PEP in the edge device did not initiate the request, the specifics of the request, e.g., flowspec, packet filter, and PHB to apply, needs to be communicated to the PEP by the PDP. This information is sent to the PEP using the Decision message containing Policy Provisioning Named Decision Data objects in the COPS Decision object as specified. Any updates to the state information, for example in the case of a policy change or call tear down, is communicated to the PEP by subsequent unsolicited DEC messages containing the same Client Handle and the updated Policy Provisioning request state. Updates can specify that policy data is to be installed, deleted, or updated (re-installed).

PDPs may also command the PEP to open a new Request State or delete an exiting one by issuing a decision with the Decision Flags object's Request-State flag set. If the command-code is "install", then the PDP is commanding the PEP to create a new Request State, and therefore issue a new REQ message specifying a new Client Handle or otherwise issue a "Failure" RPT specifying the appropriate error condition. Each request state represents an independent and logically non-overlapping namespace, identified by the Client Handle, on which transactions (a.k.a., configuration installations, deletions, updates) may be performed. Other existing Request States will be unaffected by the new request state as they are independent (thus, no instances of configuration data within one Request State

can be affected by DECs for another Request State as identified by the Client Handle). If the command-code is "Remove", then the PDP is commanding the PEP to delete the existing Request-State specified by the DEC message's Client Handle, thereby causing the PEP to issue a DRQ message for this Handle.

The PEP MUST acknowledge a DEC message and specify what action was taken by sending a RPT message with a "Success" or "Failure" Report-Type object with the Solicited Message Flag set in the COPS message header. This serves as an indication to the PDP that the requestor (e.g., H.323 server) can be notified whether the request has been accepted by the network or not. If the PEP needs to reject the DEC operation for any reason, a RPT message is sent with a Report-Type with the value "Failure" and optionally a Client Specific Information object specifying the policy data that was rejected. Under such solicited report failure conditions, the PEP MUST always rollback to its previously installed (good) state as if the DEC never occurred. The PDP is then free to modify its decision and try again.

The PEP can report to the PDP the current status of any installed request state when appropriate. This information is sent in a Report-State (RPT) message with the "Accounting" flag set. The request state that is being reported is identified via the associated Client Handle in the report message.

Finally, Client-Close (CC) messages are used to cancel the corresponding Client-Open message. The CC message informs the other side that the client-type specified is no longer supported.

7. Fault Tolerance

When communication is lost between PEP and PDP, the PEP attempts to re-establish the TCP connection with the PDP it was last connected to. If that server cannot be reached, then the PEP attempts to connect to a secondary PDP, assumed to be manually configured (or otherwise known) at the PEP.

When a connection is finally re-established with a PDP, the PEP sends a OPN message with a <LastPDPAddr> object providing the address of the most recent PDP for which it is still caching decisions. If no decisions are being cached on the PEP (due to reboot or TTL timeout of state) the PEP MUST NOT include the last PDP address information. Based on this object, the PDP may request the PEP to re-synch its current state information (by issuing a COPS SSQ message). If, after re-connecting, the PDP does not request synchronization, the client can assume the server recognizes it and the current state at the PEP is correct, so a REQ message need not be sent. Still, any state changes which occurred at the PEP that the PEP could not communicate

to the PDP due to communication having been lost, MUST be reported to the PDP via the PEP sending an updated REQ message. Whenever re-synchronization is requested, the PEP MUST reissue any REQ messages for all known Request-States and the PDP MUST issue DEC messages to delete either individual PRIDs or prefixes as appropriate to ensure a consistent known state at the PEP.

While the PEP is disconnected from the PDP, the active request-state at the PEP is to be used for policy decisions. If the PEP cannot re-connect in some pre-specified period of time, all installed Request-States are to be deleted and their associated Handles removed. The same holds true for the PDP; upon detecting a failed TCP connection, the time-out timer is started for all Request-States associated with the PEP and these states are removed after the administratively specified period without a connection.

8. Security Considerations

The COPS protocol [COPS], from which this document derives, describes the mandatory security mechanisms that MUST be supported by all COPS implementations. These mandatory security mechanisms are used by the COPS protocol to transfer opaque information from PEP to PDP and vice versa in an authenticated and secure manner. COPS for Policy Provisioning simply defines a structure for this opaque information already carried by the COPS protocol. As such, the security mechanisms described for the COPS protocol will also be deployed in a COPS-PR environment, thereby ensuring the integrity of the COPS-PR information being communicated. Furthermore, in order to fully describe a practical set of structured data for use with COPS-PR, a PIB (Policy Information Base) will likely be written in a separate document. The authors of such a PIB document need to be aware of the security concerns associated with the specific data they have defined. These concerns MUST be fully specified in the security considerations section of the PIB document along with the required security mechanisms for transporting this newly defined data.

9. IANA Considerations

COPS for Policy Provisioning follows the same IANA considerations for COPS objects as the base COPS protocol [COPS]. COPS-PR has defined one additional Decision Flag value of 0x02, extending the COPS base protocol only by this one value. No new COPS Client- Types are defined by this document.

COPS-PR also introduces a new object number space with each object being identified by its S-Num and S-Type value pair. These objects are encapsulated within the existing COPS Named ClientSI or Named Decision Data objects [COPS] and, therefore, do not conflict with any

assigned numbers in the COPS base protocol. Additional S-Num and S-Type pairs can only be added to COPS-PR using the IETF Consensus rule as defined in [IANA]. These two numbers are always to be treated as a pair, with one or more S-Types defined per each S-Num. This document defines the S-Num values 1-6 and the S-Type 1 for each of these six values (note that the S-Type value of 2 is reserved for transport of XML encoded data). A listing of all the S-Num and S-Type pairs defined by this document can be found in sections 4.1-4.6.

Likewise, additional Global Provisioning error codes and Class-Specific Provisioning error codes defined for COPS-PR can only be added with IETF Consensus. This document defines the Global Provisioning error code values 1-11 in section 4.4 for the Global Provisioning Error Object (GPERR). This document also defines the Class-Specific error code values 1-13 in section 4.5 for the Class Provisioning Error Object (CPERR).

10. Acknowledgements

This document has been developed with active involvement from a number of sources. The authors would specifically like to acknowledge the valuable input given by Michael Fine, Scott Hahn, and Carol Bell.

11. References

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

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

[COPRSVP] Boyle, J., Cohen, R., Durham, D., Herzog, S., Raja, R. and A. Sastry, "COPS usage for RSVP", RFC 2749, January 2000.

[ASN1] Information processing systems - Open Systems Interconnection, "Specification of Abstract Syntax Notation One (ASN.1)", International Organization for Standardization, International Standard 8824, December 1987.

[BER] Information processing systems - Open Systems Interconnection - Specification of Basic Encoding Rules for Abstract Syntax Notation One (ASN.1), International Organization for Standardization. International Standard 8825, (December, 1987).

[RFC2475] Blake, S., Black, D., Carlson, M., Davies, E., Wang, Z. and W. Weiss, "An Architecture for Differentiated Service," RFC 2475, December 1998.

[SPPI] McCloghrie, K., Fine, M., Seligson, J., Chan, K., Hahn, S., Sahita, R., Smith, A. and F. Reichmeyer, "Structure of Policy Provisioning Information SPPI", Work in Progress.

[V2SMI] McCloghrie, K., Perkins, D., Schoenwaelder, J., Case, J., Rose, M. and S. Waldbusser, "Structure of Management Information Version 2(SMIv2)", STD 58, RFC 2578, April 1999.

[RFC2234] Crocker, D. and P. Overell, "Augmented BNF for Syntax Specifications: ABNF", RFC 2234, November 1997.

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

[URN] Moats, R., "Uniform Resource Names (URN) Syntax", RFC 2141, May 1997.

[XML] World Wide Web Consortium (W3C), "Extensible Markup Language (XML)," W3C Recommendation, February, 1998, http://www.w3.org/TR/1998/REC-xml-19980210.

12. Authors' Addresses

Kwok Ho Chan
Nortel Networks, Inc.
600 Technology Park Drive
Billerica, MA 01821

Phone: (978) 288-8175
EMail: khchan@nortelnetworks.com

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

Phone: (503) 264-6232
Email: david.durham@intel.com

Silvano Gai
Cisco Systems, Inc.
170 Tasman Dr.
San Jose, CA 95134-1706

Phone: (408) 527-2690
EMail: sgai@cisco.com

Shai Herzog
IPHighway Inc.
69 Milk Street, Suite 304
Westborough, MA 01581

Phone: (914) 654-4810
EMail: Herzog@iphighway.com

Keith McCloghrie

Phone: (408) 526-5260
EMail: kzm@cisco.com

Francis Reichmeyer
PFN, Inc.
University Park at MIT
26 Landsdowne Street
Cambridge, MA 02139

Phone: (617) 494 9980
EMail: franr@pfn.com

John Seligson
Nortel Networks, Inc.
4401 Great America Parkway
Santa Clara, CA 95054

Phone: (408) 495-2992
Email: jseligso@nortelnetworks.com

Raj Yavatkar

Phone: (503) 264-9077
EMail: raj.yavatkar@intel.com

Andrew Smith
Allegro Networks
6399 San Ignacio Ave.
San Jose, CA 95119, USA

EMail: andrew@allegronetworks.com

13. Full Copyright Statement

Copyright (C) The Internet Society (2001). All Rights Reserved.

This document and translations of it may be copied and furnished to others, and derivative works that comment on or otherwise explain it or assist in its implementation may be prepared, copied, published and distributed, in whole or in part, without restriction of any kind, provided that the above copyright notice and this paragraph are included on all such copies and derivative works. However, this document itself may not be modified in any way, such as by removing the copyright notice or references to the Internet Society or other Internet organizations, except as needed for the purpose of developing Internet standards in which case the procedures for copyrights defined in the Internet Standards process must be followed, or as required to translate it into languages other than English.

The limited permissions granted above are perpetual and will not be revoked by the Internet Society or its successors or assigns.

This document and the information contained herein is provided on an "AS IS" basis and THE INTERNET SOCIETY AND THE INTERNET ENGINEERING TASK FORCE DISCLAIMS ALL WARRANTIES, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO ANY WARRANTY THAT THE USE OF THE INFORMATION HEREIN WILL NOT INFRINGE ANY RIGHTS OR ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.

Acknowledgement

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