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

Copyright Notice

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.

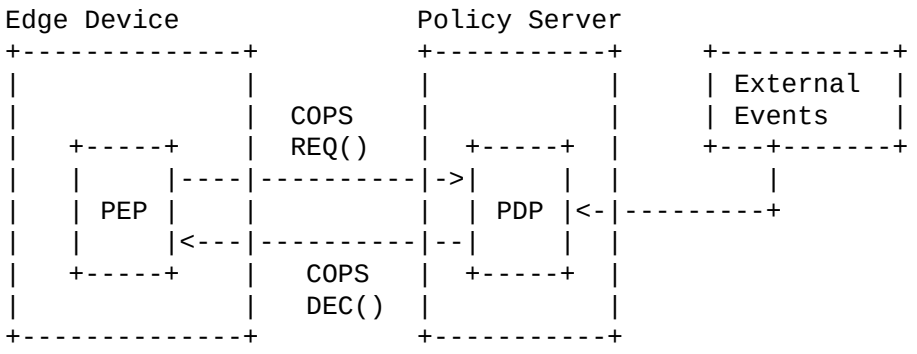


Figure 1: COPS Provisioning Model

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:

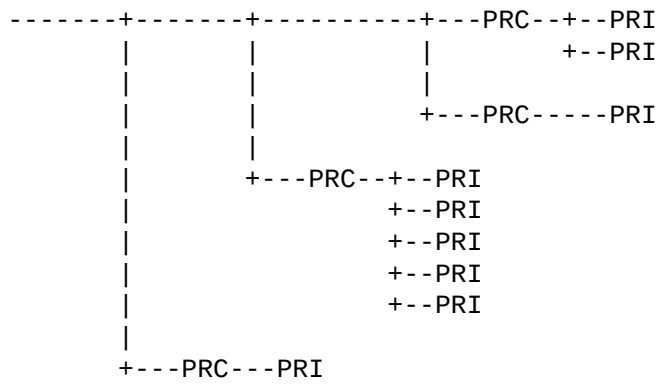


Figure 2: The PIB Tree

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:

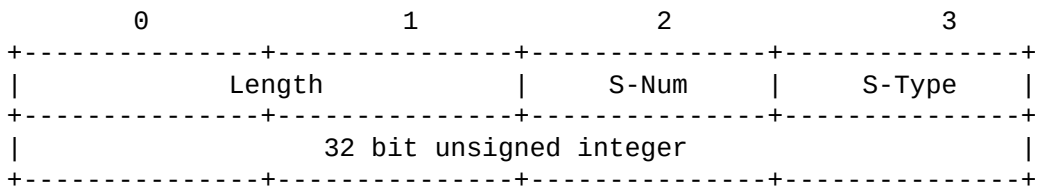
```
<Report State> ::= <Common Header>
                   <Client Handle>
                   <Report Type>
                   *(<Named ClientSI>)
                   [<Integrity>]
```

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:



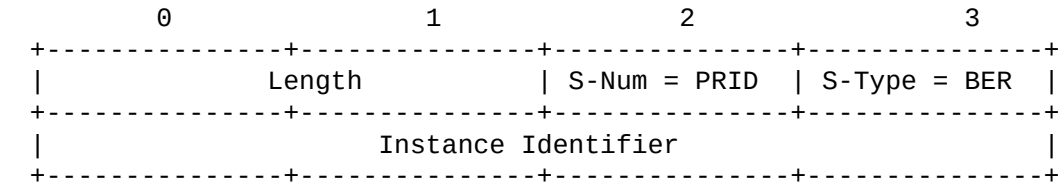
S-Num and S-Type are similar to the C-Num and C-Type used in the base COPS objects. The difference is that S-Num and S-Type are used only for COPS-PR clients and are encapsulated within the existing COPS Named ClientSI or Named Decision Data objects. The S-Num identifies the general purpose of the object, 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

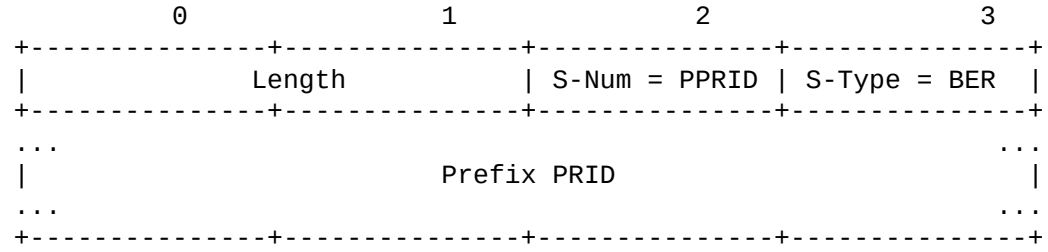
xxxEntry including its SEQUENCE of attributes encoded in the EPD (defined below). This constitutes an instance (PRI) of a class (PRC) in terms of the SMI.

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
```

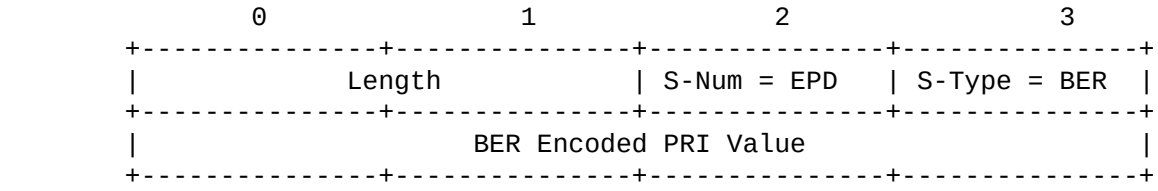
The entire PPRID object would be encoded as follows:

```
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)
```

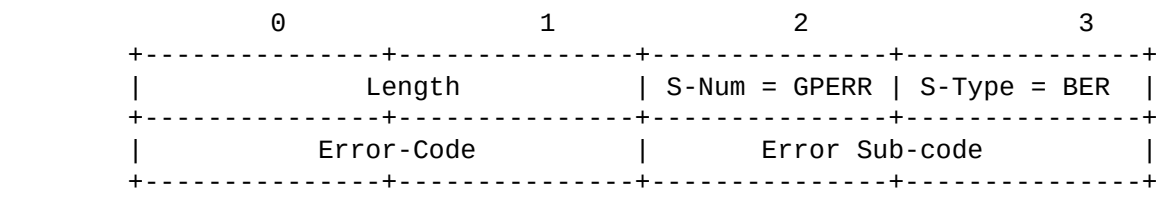
The entire EPD object for this instance would then be encoded as follows:

```
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.



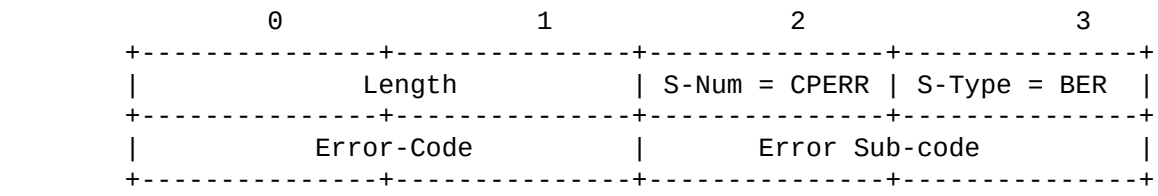
The global provisioning error object has the same format as the Error object in COPS [COPS], except with C-Num and C-Type replaced by the S-Num and S-Type values shown. The global provision error object is used to communicate general errors that do not map to a specific PRC.

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

Acknowledgement

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